**Predict The Flight Ticket Price**

1.Problem Definition

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travellers saying that flight ticket prices are so unpredictable. Here we will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

In this context, the use of Regression models to predict what is the price on the basis of given information.

This data set presents flight ticket prices. The data set contains approximately 10,683 entries. Given the size of the data set, the model should only be expected to provide modest improvement in indentification of Price.

2.Data Analysis

In order to start with exercise, I have used Flight ticket prices Dataset The dataset includes features like Airline, Date\_of\_Journey, Source, Destination etc. It includes the data of 10683 tickets.

FEATURES:

Airline: The name of the airline.

Date\_of\_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep\_Time: The time when the journey starts from the source.

Arrival\_Time: Time of arrival at the destination.

Duration: Total duration of the flight.

Total\_Stops: Total stops between the source and destination.

Additional\_Info: Additional information about the flight

Price: The price of the ticket

As here, we can see that No of numerical features: 1 ['Price'], and No of categorical features: 10 ['Airline', 'Date\_of\_Journey', 'Source', 'Destination', 'Route', 'Dep\_Time', 'Arrival\_Time', 'Duration', 'Total\_Stops', 'Additional\_Info']

Here, the shape of data is (10683, 11) where we have more no of categorical data which is need to be coded because machine learning algorithms only understand numerical data. So to train n test need to convert data from categorical to numerical.

There is only single entry is missing in Route and Total\_stops.

3.EDA Concluding Remarks

But before we begin any problem, we must understand what the problem actually is there in come the analysis part. Since we have a large volume of data, we must apply statistical analytical tools to understand the various factors at play. This is called exploratory analysis.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. It is a good practice to understand the data first and try to gather as many insights from it.

By checking the statistics summary ,  
According to dataset : Number of rows in each column are same, means there are no null values in the data set.Null values are handled. The mean and median of price is different and the STD and mean are not very close to each other.Data is skewed. By checking the difference between the 75% and max value there are outliers present. Unique shows the no of unique values in categorical data.

The maximum ticket price is Rs.79512 and minimum is Rs.1759.

All the flight are from 5 different cities to all other 6 different cities except the same city. As from the data we can clearly see that the price of the flights tickets are highly related to the route , airline ,stops and additional information about the flight.

4.Pre-processing Pipeline

Now preparing the data for modeling and training. For this need to do two steps:

**1.Checking and removing outliers.**

# checking outliers

plt.figure(figsize=(10,5))

df.plot(kind='box',subplots=**True**,layout=(2,6))

plt.show()

# Removing outliers

**from** **scipy.stats** **import** zscore *# removing outliers*

z=abs(zscore(df))

print(df.shape)

df\_final=df.loc[(z<3).all(axis=1)]

print(df\_final.shape)

df=df\_final

(10683, 11)

(10578, 11)

#spliting the data in target and independent variables.

y=df["Price"]

dfx=df.drop(columns=['Price'], axis=1)

**2.Checking and handling skewness to make data in form to fit in model building**

*#Checking skewness*

dfx.skew()

Airline 0.731709

Date\_of\_Journey -0.061918

Source -0.439204

Destination 1.269454

Route -0.511368

Dep\_Time 0.194141

Arrival\_Time -0.608273

Duration -0.216095

Total\_Stops 0.618208

Additional\_Info -1.533587

dtype: float64

*# handling skewness*

**from** **sklearn.preprocessing** **import** PowerTransformer

pt=PowerTransformer(method='yeo-johnson')

d=pt.fit\_transform(dfx)

d=pd.DataFrame(d,columns=dfx.columns)

x=d

5.Building Machine Learning Models

Now that we are done with the basic pre-processing steps, we can go ahead and build simple machine learning models over this data. We will try nine models here –KNeighborsRegressor, SVR, LinearRegression, Lasso, Ridge, DecisionTreeRegressor, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor to predict the Flight Ticket Price.

To compare the performance of the models, we will create a validation set (or test set). Here I have randomly split the data into two parts using the train\_test\_split() function, such that the validation set holds 33% of the data points while the train set has 64%

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=.33,random\_state=42)

print(x\_train.shape,x\_test.shape)

print(y\_train.shape,y\_test.shape)

(7087, 10) (3491, 10)

(7087,) (3491,)

maxrscore=0

**for** r\_state **in** range(42,100):

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=.33,random\_state=r\_state)

reg=linear\_model.LinearRegression()

reg.fit(x\_train,y\_train)

y\_pred= reg.predict(x\_test)

r2s=r2\_score(y\_test,y\_pred)

**if** r2s > maxrscore:

maxrscore=r2s

fr\_state=r\_state

print("max r2 score corresponding to ",fr\_state," is ",maxrscore)

max r2 score corresponding to 86 is 0.32960883734409196

parameter = {'kernel':('linear', 'rbf','poly'), 'C':[1, 10]}

svr = SVR()

grid = GridSearchCV( estimator=SVR(), param\_grid = parameter)

grid.fit(x, y)

print(grid)

*#print(grid.best\_score)*

print(grid.best\_estimator\_.kernel)

print(grid.best\_params\_)

s=grid.best\_estimator\_.kernel

GridSearchCV(cv=None, error\_score=nan,

estimator=SVR(C=1.0, cache\_size=200, coef0=0.0, degree=3,

epsilon=0.1, gamma='scale', kernel='rbf',

max\_iter=-1, shrinking=True, tol=0.001,

verbose=False),

iid='deprecated', n\_jobs=None,

param\_grid={'C': [1, 10], 'kernel': ('linear', 'rbf', 'poly')},

pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,

scoring=None, verbose=0)

linear

{'C': 10, 'kernel': 'linear'}

parameter = {'n\_neighbors':(3,5,7,9,11), 'weights':['uniform','distance']}

knn = KNeighborsRegressor()

grid = GridSearchCV( estimator=knn, param\_grid = parameter)

grid.fit(x, y)

print(grid)

*#print(grid.best\_score)*

print(grid.best\_estimator\_.n\_neighbors)

print(grid.best\_estimator\_.weights)

print(grid.best\_params\_)

k=grid.best\_estimator\_.n\_neighbors

w=grid.best\_estimator\_.weights

GridSearchCV(cv=None, error\_score=nan,

estimator=KNeighborsRegressor(algorithm='auto', leaf\_size=30,

metric='minkowski',

metric\_params=None, n\_jobs=None,

n\_neighbors=5, p=2,

weights='uniform'),

iid='deprecated', n\_jobs=None,

param\_grid={'n\_neighbors': (3, 5, 7, 9, 11),

'weights': ['uniform', 'distance']},

pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,

scoring=None, verbose=0)

5

uniform

{'n\_neighbors': 5, 'weights': 'uniform'}

KNR=KNeighborsRegressor(n\_neighbors=k,weights=w)

SV=SVR(kernel=s)

LR=LinearRegression()

LAR=Lasso()

RR=Ridge()

DT=DecisionTreeRegressor(random\_state=fr\_state)

RFR=RandomForestRegressor(random\_state=fr\_state)

ABR=AdaBoostRegressor(random\_state=fr\_state)

GBR=GradientBoostingRegressor(random\_state=fr\_state)

models = []

models.append(('KNeighborsRegressor', KNR))

models.append(('SVR', SV))

models.append(('LinearRegression', LR))

models.append(('LassoRegression', LAR))

models.append(('RidgeRegressor', RR))

models.append(('DecisionTreeRegressor', DT))

models.append(('RandomForestRegressor', RFR))

models.append(('AdaBoostRegressor', ABR))

models.append(('GradientBoostingRegressor',GBR))

Model = []

rmse = []

cvs=[]

r2score=[]

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=.33,random\_state=fr\_state)

**for** name,model **in** models:

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*',name,'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('**\n**')

Model.append(name)

model.fit(x\_train,y\_train)

print(model)

y\_pred=model.predict(x\_test)

print('**\n**')

sc = cross\_val\_score(model, x, y, cv=10, scoring='r2').mean()

print('Cross\_Val\_Score = ',sc)

cvs.append(sc)

print('**\n**')

print("error:")

r2s=r2\_score(y\_test,y\_pred)

print("r2 score is: ",r2s)

r2score.append(r2s)

print('**\n**')

rmse1=np.sqrt(mean\_squared\_log\_error(y\_test,y\_pred))

print("root Mean squared log error: ",rmse1)

rmse.append(rmse1)

print('**\n**')

6.Concluding Remarks

In this session we will check dissimilarity between different these nine algorithms:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Model** | **Cross\_val\_score** | **r2\_score** | **root\_mean\_squared\_log\_error** |
| 0 | KNeighborsRegressor | 0.759780 | 0.752343 | 0.223944 |
| 1 | SVR | 0.242679 | 0.255872 | 0.402324 |
| 2 | LinearRegression | 0.302619 | 0.329609 | 0.392572 |
| 3 | LassoRegression | 0.302619 | 0.329511 | 0.392685 |
| 4 | RidgeRegressor | 0.302619 | 0.329602 | 0.392584 |
| 5 | DecisionTreeRegressor | 0.856280 | 0.849242 | 0.177286 |
| 6 | RandomForestRegressor | 0.906449 | 0.900194 | 0.146596 |
| 7 | AdaBoostRegressor | 0.568019 | 0.565924 | 0.350418 |
| 8 | GradientBoostingRegressor | 0.828680 | 0.833782 | 0.196047 |

As it can been seen from above comparison table the applied models can successfully predict

Cross\_val\_score is max (0.906449) with Random Forest Regressor algorithm. Thus, Random Forest Regressor is the best model, as it always predicts a high area in accuracy and a better R2score.

**Random Forest Regressor**

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.  
Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

With R2score of 0.900194 and a Cross\_val\_score of 0.906449 which makes this the most suitable model.

Basically, with the help of our model, the Price of the Flight ticket can be predicted just by collecting the needed data then run through the model.

A special thanks to Data Trained for creating the platform to learn Data Science and Artificial Intelligence by creating world-class learning.

Click here for the link to the notebook:

https://github.com/jainnvandana/evaluation-projects/blob/main/project12\_flightprice.ipynb