Homework 5: Aakar Jinwala adj329 Aalap Parikh adp459

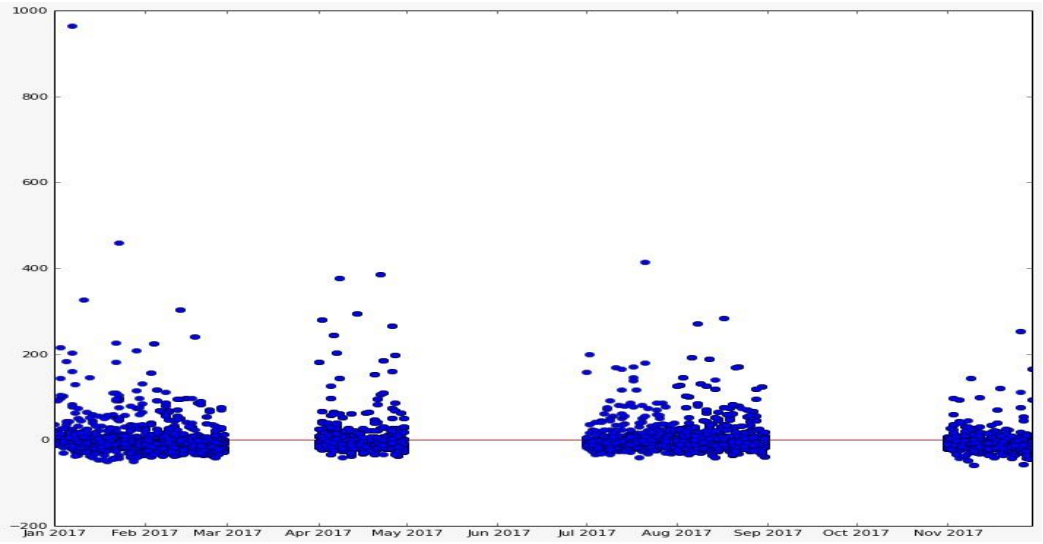
A) Feature Selection:

Features that we **have removed** from our attribute list:

FL_NUM | AIRLINE_ID | UNIQUE_CARRIER	When tested they are not having any positive impact
ORIGIN_CITY_NAME | ORIGIN_STATE_ABR ORIGIN_CITY_MARKET_ID DEST_CITY_NAME | DEST_STATE_ABR DEST_CITY_MARKET_ID	These all attributes are co-related with ORIGIN and DEST, When we have tested number of distinct values in these attributes, we can say ORIGIN & DEST are superset of all these features.
DISTANCE	Correlated with Distance group. Also distance group has lower values, hence becomes easier to use in training.
FL_DATE	Extracted month from the date.
FIRST_DEP_TIME	It has very less information.

So, our final model contains:
Continues Attributes:
TAXI_OUT | TAXI_IN | ACTUAL_ELAPSED_TIME | DISTANCE
Categorical Attributes:
CRS_DEP_TIME | DAY_OF_WEEK | ORIGIN | DEST

We would like to explain, why we extracted only Month from FL_DATE



As from above image, flights are not delayed during the months of Mar, May, June,Sep, Oct. So this signifies some relationship between Month and delay.

B) Preprocessing Tasks:

a) Since, the feature **DISTANCE** had **‘,’** it was necessary to remove it as it was throwing an error.
b) For **all continuous variables**, we have converted them to numeric, which helps in comparison
c) For all continuous variables, we have performed feature scaling as it is important for kNN, Multi layer Perceptrons. It does not make that great difference for Decision Tree Regressors.
d) Converted the target feature ARR_DELAY to float for the training set data.

C) For categorical features:

a) For ORIGIN and DEST, we have first converted them to integers by performing Label encoding.
b) For rest of the categorical features, we have type casted them to 32 bit integer.
c) Lastly, used get_dummies in Pandas for all the categorical features.

For the purpose of consistency in building the models, we have first combined both the datasets before applying **label encoding** and later separated the combined datasets, such that both train and test dataset have same number of attributes.

```
# for dummy and label encoding
len_train = len(train)
train_test = pd.concat([train, test])
train_test = label_and_dummy_endcoding(train_test)
train, test = train_test.iloc[:len_train, :], train_test.iloc[len_train:,:]
```

D) Selection of learning algorithm:

a) Since KNN uses some distance metric, it is not suitable to be used here, as we have so many categorical features, and therefore we have a very high dimensional space.
b) The most suitable choices for this problem would be Decision Trees, Random Forest and Multi-Layer Perceptron since we are dealing with numerically encoded categorical features.
c) Decision trees gave us a very low training error on setting high depth and low leaf split. But it performed poorly on cross validation, and hence isn’t expected to generalize very well.
d) Ensemble Random Forests performed reasonably good compared to Decision Trees. We kept number of trees = 10 and a maximum depth of 8, and achieved a good amount of generalization during cross validation; i.e. the difference between the training error and test error was reasonable.
e) Multi-layer perceptron was working fairly better than these two. With 2 hidden layers, ReLU activation functions and regularizing L2 norm, achieved good results with cross

validation error. The best average CV error is reported to be equal to 2066 at 4 hidden layers (5,4,3,2) and learning rate = 0.1.

Here is a comparison of how MLP compares with other models:

Model	**Training Error**	**Average test error for 5-fold Cross Validation**
Decision Tree: Max depth: 8 min_samples_split : 3	1435.70203429	2,578.89
Random Forests: Number of estimators: 10 Max depth: 8	1290.49484208	2,146.17
Multi Layer Perceptron	**2048.14**	**2,056.8**

E) Fine Tuning the Multi-Layer Perceptron

Hyperparameters Tuning for Multi-Layer Perceptron:
Learning rate: 0.001, 0.01, 0.1, 1.
Below 0.01, the optimizer converges very slowly.

Experimenting with Weights updation algorithm:

We tried using Stochastic gradient descent and Adam optimizers, out of which the latter converges much faster.

Experimenting with size of hidden layer:

Here are the errors and the corresponding hidden layers that we've tried putting:

Hidden Layers (excluding output layer)	Training Error	Average test error for 5-fold Cross Validation
N = 2 : 200,150	8.22	3491.8
N = 5 : 20,15,10,5,2	2059.5393	2,064.8
N = 4 : 7,5,4,2	2056.9859	2,064.55
N = 3 : 4,3,2	2052.288	2,058.3
N = 3: 50, 20, 10	**2048.14**	**2,056.8**
N = 2 : 3,2	2053.527	2,063.2

As, we can see from the above table, MLP is overfitting when we are having very large sizes for hidden layers. For N = 2, with two hidden layers with 200 and 150, while training error is very less.

Experimenting with Momentum:

```
regressor = MLPRegressor(hidden_layer_sizes=(3,2),solver='adam',alpha = 0.1,
                         max_iter = 1000, shuffle = False, activation = 'relu', random_state = None, verbose = False, \
                         momentum = 0.4, early_stopping = False)

```

Momentum rate	Average test error for 5-fold Cross Validation
0.1	2066.29396469
0.2	2062.47198906
0.4	**2060.69426256**
1	2069.62008229

As, we can see from above with the increase in value of momentum, Cross Validation error decreases, but with very high value such as 1, error increases again

Experimenting with Alpha: It is L2 penalty (regularization term) parameter.

Alpha rate	Average test error for 5-fold Cross Validation
0.0001	2069.43874839
0.001	2062.25605002
0.01	**2059.09156853**
1	2069.05362012

Considering the above outputs, we are predicting the values of test set with following parameters in our final output program:

```
# Muti layer Percepton Fine Tuning
print("Muti layer Percepton")
from sklearn.neural_network import MLPRegressor
regressor = MLPRegressor(hidden_layer_sizes=(50, 20, 10,),solver='adam',alpha = 0.01,
                         max_iter = 1000, shuffle = False, activation = 'relu', random_state = None, verbose = False, \
                         momentum = 0.4, early_stopping = False)

```

And we are getting

```
Muti layer Percepton
2048.75589689
[-1860.22987083 -2161.94055876 -2130.09557839 -2250.04539813 -1894.8955472 ]
-2059.44139066
```

References:

1) http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
2) https://stackoverflow.com/questions/41335718/keep-same-dummy-variable-in-training-and-testing-data?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
3) https://www.analyticsvidhya.com/blog/2015/11/easy-methods-deal-categorical-variables-predictive-modeling/
4) https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/