Reducer Results

August 24, 2021

[ ]:

```
```

[2]:

```python
#!/usr/bin/env python3
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys
import numpy as np
import time
import pandas as pd
from IPython.display import display
from datetime import datetime, timedelta
from sklearn.cluster import MiniBatchKMeans, KMeans
from pandarallel import pandarallel # use: pip install pandarallel
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor # use: conda install xgboost
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import math
from prettytable import PrettyTable
import gc
import fileinput
start_time = time.time()
i = 0

def model_evaluation(algorithem_name, X_Test, y_pred, y_true):

    # R2 and Adjasted R2
    r2 = r2_score(y_true, y_pred)
    adj_r2 = 1-(1-r2)*((len(X_Test)-1)/(len(X_Test)-X_Test.shape[1]-1))
    # MSE and RMSE
    mse = mean_squared_error(y_true, y_pred)
    rmse = math.sqrt(mse)
    # print in table
    table = PrettyTable()
    print('\n' + algorithem_name + ':')
    table.field_names = ['Experiment', 'Value']
    table.add_row(['R2', r2])
    table.add_row(['Adjusted R2', adj_r2])
    table.add_row(['MSE',mse])
    table.add_row(['RMSE', rmse])
    return table


def measure_time(label = ''):
    global start_time
    global i
    print('{}\t took {}s'.format(label, round(time.time() - start_time)))
    start_time = time.time()
    i+=1

def read_mapper_output(file):
    for line in file:
        yield line

def main(separator='\t'):

    df_train = []
    df_test = []
    names=['pickup_time', 'pickup_longitude', 'pickup_latitude']
    c=['tpep_pickup_datetime',
           'pickup_longitude',
           'pickup_latitude']
    measure_time('initialisation')



    # input comes from STDIN (standard input)
    gc.disable()

    with fileinput.input(files=('/home/Aziz/MapperOutput_2015-01.csv',
                                '/home/Aziz/MapperOutput_2015-02.csv',
                                '/home/Aziz/MapperOutput_2015-03.csv',
                                '/home/Aziz/MapperOutput_2016-01.csv',
                                '/home/Aziz/MapperOutput_2016-02.csv',
                                '/home/Aziz/MapperOutput_2016-03.csv'
                               )) as f:
        for line in f:
            #data = read_mapper_output(sys.stdin)
            for row in f:
                #print(row)
                key, row = row.split(separator)
                row = row.split(',')
                if key=='train':
                    df_train.append(row)
                elif key=='test':
                    df_test.append(row)


#     data = read_mapper_output(sys.stdin)
#     for row in data:
#         print(row)
#         key, row = row.split(separator)
#         row = row.split(',')
#         if key=='train':
#             df_train.append(row)
#         elif key=='test':
#             df_test.append(row)
#     gc.enable()
    measure_time('split rows') # measuring time for debugging purposes



    # converting rows of data into pandas dataframes
    df_train = pd.DataFrame(df_train, columns=names)
    df_test = pd.DataFrame(df_test, columns=names)
    # for debug only
    print('No. rows in training dataset:  %d' % df_train.shape[0])
    print('No. rows in test dataset:  %d' % df_test.shape[0])
    measure_time('convert rows to dataframes') # printing time for debugging␣
 ↪only



    ## GEOGRAPHICAL SEGMENTATION BY CLUSTERING
    #Clustering pickups, Getting clusters
    coord = df_train[["pickup_latitude", "pickup_longitude"]].values
    regions = MiniBatchKMeans(n_clusters = 30, batch_size = 10000).fit(coord)
    measure_time('training cluster model') # printing time for debugging only



    #predecting clusters
    df_train["pickup_cluster"] = regions.predict(df_train[["pickup_latitude",␣
 ↪"pickup_longitude"]])
    df_test["pickup_cluster"] = regions.predict(df_test[["pickup_latitude",␣
 ↪"pickup_longitude"]])
```

```
    measure_time('predicting clusters') # printing time for debugging only



    # Replacing mins and sec with 0
    pandarallel.initialize()
    df_train['pickup_time'] = df_train.pickup_time.parallel_apply(lambda x : pd.
↪to_datetime(x).replace(minute=0, second=0))
    df_test['pickup_time'] = df_test.pickup_time.parallel_apply(lambda x : pd.
↪to_datetime(x).replace(minute=0, second=0))
    measure_time('reformat time') # printing time for debugging only



    ## CALCULATING TAXI-DEMAND
    # Group by Cluster_id and time
    df_train_2 = df_train.groupby(['pickup_time','pickup_cluster']).size().
↪reset_index(name='count')
    df_test_2 = df_test.groupby(['pickup_time','pickup_cluster']).size().
↪reset_index(name='count')
    measure_time('grouping and calculate pickups') # printing time for␣
↪debugging only



    # Converting pickup counts to demand percentage
    df_train_2['count'] = df_train_2['count'].parallel_apply(lambda x :  (x /␣
↪df_train_2['count'].max()))
    df_test_2['count'] = df_test_2['count'].parallel_apply(lambda x :  (x /␣
↪df_test_2['count'].max()))
    measure_time('pickups to percentage') # printing time for debugging only



    # Getting month, days, hours, day of week
    df_train_2['month'] = pd.DatetimeIndex(df_train_2['pickup_time']).month
    df_train_2['day'] = pd.DatetimeIndex(df_train_2['pickup_time']).day
    df_train_2['dayofweek'] = pd.DatetimeIndex(df_train_2['pickup_time']).
↪dayofweek
    df_train_2['hour'] = pd.DatetimeIndex(df_train_2['pickup_time']).hour

    df_test_2['month'] = pd.DatetimeIndex(df_test_2['pickup_time']).month
    df_test_2['day'] = pd.DatetimeIndex(df_test_2['pickup_time']).day
    df_test_2['dayofweek'] = pd.DatetimeIndex(df_test_2['pickup_time']).
↪dayofweek
    df_test_2['hour'] = pd.DatetimeIndex(df_test_2['pickup_time']).hour
    measure_time('reformat time') # printing time for debugging only



    # X and y for training
    X_train = df_train_2[['pickup_cluster', 'month', 'day', 'hour',␣
↪'dayofweek']]
    y_train = df_train_2['count']

    # X and y for testing
    X_test = df_test_2[['pickup_cluster', 'month', 'day', 'hour', 'dayofweek']]
    y_test = df_test_2['count']
    measure_time('spliting to train and test') # printing time for debugging␣
↪only


    # split training data into training and validation data
    #X_train, X_validation, y_train, y_validation = train_test_split(X_train,␣
↪y_train,
    #                                                 test_size=0.33,␣
↪random_state=42)
    #measure_time('spliting train and validation') # printing time for␣
↪debugging only




    # Linear regression
    LReg = LinearRegression()
    LReg.fit(X_train, y_train)
    LReg_y_pred = LReg.predict(X_test)

    # RandomForest regression
    RFRegr = RandomForestRegressor()
    RFRegr.fit(X_train, y_train)
    RFRegr_y_pred = RFRegr.predict(X_test)

    # XGB regression
    GBRegr = XGBRegressor(n_estimators=1000, max_depth=7, eta=0.1, subsample=0.
↪7, colsample_bytree=0.8)
    GBRegr.fit(X_train, y_train)
    GBRegr_y_pred = GBRegr.predict(X_test)
    measure_time('training models') # printing time for debugging only



    # evaluation
    #print(model_evaluation('y True',X_Test=X_test, y_pred=y_test,␣
↪y_true=y_test))
    print(model_evaluation('Linear Regression',X_Test=X_test,␣
↪y_pred=LReg_y_pred, y_true=y_test))
    print(model_evaluation('RandomForest',X_Test=X_test, y_pred=RFRegr_y_pred,␣
↪y_true=y_test))
    print(model_evaluation('XGB',X_Test=X_test, y_pred=GBRegr_y_pred,␣
↪y_true=y_test))
    measure_time('printing results') # printing time for debugging only



    #print("--- %s seconds --- 0" % (time.time() - start_time))
    #display(X_train)
    #display(y_train)
    #display(X_validation)
    #display(y_validation)
    #display(X_test)
    #display(y_test)



if __name__ == "__main__":
    main()
```

```
initialisation   took 0s
split rows       took 97s
No. rows in training dataset:  37561189
No. rows in test dataset:  33722144
convert rows to dataframes       took 67s
training cluster model   took 36s
predicting clusters      took 69s
INFO: Pandarallel will run on 24 workers.
INFO: Pandarallel will use Memory file system to transfer data between the main
process and workers.
reformat time    took 597s
grouping and calculate pickups   took 4s
pickups to percentage    took 24s
reformat time    took 0s
spliting to train and test       took 0s
```

```
/opt/conda/anaconda/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
training models  took 13s
```

```
Linear Regression:
+-------------+----------------------+
|  Experiment |        Value         |
+-------------+----------------------+
|      R2     | 0.12358378989809549  |
| Adjusted R2 | 0.12351541283175582  |
|     MSE     | 0.022728725137356843 |
|     RMSE    | 0.15076048931121458  |
+-------------+----------------------+

RandomForest:
+-------------+-----------------------+
|  Experiment |         Value         |
+-------------+-----------------------+
|      R2     |   0.9121364300505471  |
| Adjusted R2 |   0.9121295750276915  |
|     MSE     | 0.0022786284734918785 |
|     RMSE    |  0.04773498165383411  |
+-------------+-----------------------+

XGB:
+-------------+----------------------+
|  Experiment |        Value         |
+-------------+----------------------+
|      R2     |  0.9190748065302935  |
| Adjusted R2 |  0.919068492832237   |
|     MSE     | 0.002098690619661767 |
|     RMSE    | 0.045811468211156114 |
+-------------+----------------------+
printing results         took 0s
```

[48]:

```
#sys.stdin = open('/home/Aziz/MapperOutput.txt','r')
```