multiple-linear-regression-model

June 12, 2023

1 Multiple Linear Regression

1.0.1 References

- https://statsandr.com/blog/multiple-linear-regression-made-simple/ (teory for multiple linear regression)
- https://machinelearningmastery.com/making-predictions-with-multilinear-regression-in-pytorch/#:~:text=The%20multilinear%20regression%20model%20is,predict%20the%20target%20variable%2 (Implementation Ideologies of multiple Linear Regression)
- http://www.sthda.com/english/articles/40-regression-analysis/163-regression-with-categorical-variables-dummy-coding-essentials-in-r/ (Linear Regression using Categorical Variables)

1.0.2 Rationale For Using this Approach

The dataset provides multiple parameters that could be related to the travel duration. This approach generates a linear combination of parameters with weights to generate the output duration thus allowing the use of more than one parameter (as would have been the case for simple linear regression). This will form a baseline machine learning model to evaluate other models used later on.

1.0.3 Import Libraries

[2]:
```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
import torch
from datetime import datetime
from torch.utils.data import TensorDataset, DataLoader
```

1.0.4 Read Data and Pre-Process

[3]:
```
#Read into Dataframe
taxi_data = pd.read_csv("kaggle_data/train.csv")

#Calculate and Create Time Column
def travel_time(polyline):
    return max(polyline.count("[") - 2, 0) * 15
```

```python
def parse_timestamp(taxi_data):
    date_time = datetime.fromtimestamp(taxi_data["TIMESTAMP"])
    return date_time.year, date_time.month, date_time.day, date_time.hour,␣
 ↪date_time.weekday()

taxi_data["LEN"] = taxi_data["POLYLINE"].apply(travel_time)

taxi_data[["YR", "MON", "DAY", "HR", "WK"]] = taxi_data[["TIMESTAMP"]].
 ↪apply(parse_timestamp, axis=1, result_type="expand")

mean_duration = taxi_data["LEN"].mean()
standard_deviation = taxi_data["LEN"].std()
median = taxi_data["LEN"].median()
taxi_data = taxi_data[taxi_data["LEN"] < mean_duration + 3*standard_deviation]
```

1.0.5 Input Feature Decisions

[4]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#Mapping Call Type Letters to Numbers
letter_to_num = {
    "A" : 1,
    "B" : 2,
    "C" : 3
}
num_to_letter = {
    1 : "A",
    2 : "B",
    3 : "C"
}

duration = taxi_data["LEN"].tolist()

hour = taxi_data["HR"].tolist()
month = taxi_data["MON"].tolist()
week = taxi_data["WK"].tolist()
day = taxi_data["DAY"].tolist()
calltype = taxi_data["CALL_TYPE"].tolist()
taxi = taxi_data["TAXI_ID"].tolist()

for count in range(0, len(calltype), 1):
    calltype[count] = (letter_to_num[calltype[count]])

inputs = []

#Combine Input Vectors
for count in range(0, len(hour), 1):
```

```
    inputs.append([hour[count], month[count], week[count], day[count],␣
↪calltype[count], taxi[count]])
```

1.0.6 Create Dataset

[5]:
```
inputs = torch.tensor(inputs, dtype=torch.float32).to(device)
target = torch.tensor(duration, dtype=torch.float32).to(device)

dataset = TensorDataset(inputs, target)
```

1.0.7 Create the Model

[4]:
```
class MLR(torch.nn.Module):
    # Object Constructor
    def __init__(self, input_features, output_features):
        super().__init__()
        self.linear = torch.nn.Linear(input_features, 1)
        self.dropout = torch.nn.Dropout(0.5)
        self.linear2 = torch.nn.Linear(1, output_features)
        self.relu = torch.nn.ReLU()
        self.norm = torch.nn.BatchNorm1d(num_features = 6)

    # define the forward function for prediction
    def forward(self, x):
        x = self.norm(x)
        x = self.dropout(self.relu(self.linear(x)))
        y_pred = self.dropout(self.relu(self.linear2(x)))
        return y_pred

predict = MLR(6, 1)#.to(device)

print(predict)
```

```
MLR(
  (linear): Linear(in_features=6, out_features=1, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=1, out_features=1, bias=True)
  (relu): ReLU()
  (norm): BatchNorm1d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

[7]:
```
# Define optimizer (this will perform your parameter updates use)
lr = 1e-7
opt = torch.optim.Adam(predict.parameters(), lr=lr)
```

1.0.8 Train Set

[8]:
```
batch_size = 64

train_err = []
parameters = []

trainloader = DataLoader(dataset, batch_size, shuffle=True)
```

[9]:
```
def train(epochs, model, optimize):
    for epoch in range(epochs):
        for x, y in trainloader:
            model.train()
            prediction = model(x)
            loss = torch.sqrt(torch.nn.functional.mse_loss(prediction, torch.
 ↪unsqueeze(y, 1)))
            #print(prediction)
            #print(y)
            optimize.zero_grad()
            loss.backward()
            optimize.step()

        print("Epoch: " + str(epoch) + "\t" + "Loss: " + str(loss.tolist()))
```

[10]:
```
epochs = 10
train(epochs, predict, opt)
```

```
Epoch: 0        Loss: 996.1138305664062
Epoch: 1        Loss: 840.6067504882812
Epoch: 2        Loss: 728.8763427734375
Epoch: 3        Loss: 845.4779052734375
Epoch: 4        Loss: 865.6972045898438
Epoch: 5        Loss: 716.5923461914062
Epoch: 6        Loss: 970.431396484375
Epoch: 7        Loss: 803.6483154296875
Epoch: 8        Loss: 716.9827880859375
Epoch: 9        Loss: 708.0394287109375
```

2 PREDICT

[18]:
```
#Read into Dataframe
test_data = pd.read_csv("kaggle_data/test_public.csv")
test_data['ORIGIN_STAND'] = taxi_data['ORIGIN_STAND'].fillna(0)
test_data[["YR", "MON", "DAY", "HR", "WK"]] = taxi_data[["TIMESTAMP"]].
 ↪apply(parse_timestamp, axis=1, result_type="expand")

test_hour = test_data["HR"].tolist()
```

```
test_month = test_data["MON"].tolist()
test_week = test_data["WK"].tolist()
test_day = test_data["DAY"].tolist()
test_calltype = test_data["CALL_TYPE"].tolist()
test_taxi = test_data["TAXI_ID"].tolist()
test_origin = test_data["ORIGIN_STAND"].tolist()

for count in range(0, len(test_calltype), 1):
    test_calltype[count] = (letter_to_num[test_calltype[count]])

test_inputs = []
for count in range(0, len(test_hour), 1):
    test_inputs.append([test_hour[count], test_month[count], test_week[count],␣
 ↪test_day[count], test_calltype[count], test_taxi[count]])

test_tensor = torch.tensor(test_inputs, dtype=torch.float32).to(device)

test_dataset = TensorDataset(test_tensor)
testloader = DataLoader(test_dataset, batch_size, shuffle=True)
```

[23]:
```
test_ids = test_data["TRIP_ID"].tolist()

prediction = predict(test_tensor)
    #loss = torch.sqrt(torch.nn.functional.mse_loss(prediction, torch.
 ↪unsqueeze(y, 1)))
    #optimize.zero_grad()
    #loss.backward()
    #optimize.step()
prediction = prediction.tolist()

for i in range (0, len(test_ids), 1):
    print(str(test_ids[i])+","+str(prediction[i][0]))
```

```
T1,0.0
T2,0.0
T3,0.0
T4,0.0
T5,0.0
T6,0.0
T7,0.0
T8,1.8661231994628906
T9,0.0
T10,1.4627022743225098
T11,0.0
T12,0.0
T13,1.1850254535675049
T14,0.0
```

```
T15,0.0
T16,0.0
T17,0.0
T18,0.0
T19,0.0
T20,0.0
T21,0.6327923536300659
T22,0.0
T23,2.746330976486206
T24,0.0
T25,0.0
T26,0.0
T27,0.0
T28,0.0
T29,0.0
T30,1.686988115310669
T31,0.0
T32,0.0
T33,0.0
T34,0.0
T35,0.0
T36,0.0
T37,0.0
T38,0.0
T39,0.0
T40,0.0
T41,0.0
T42,0.0
T43,0.0
T44,0.0
T45,0.0
T46,0.0
T47,0.0
T48,0.0
T49,0.0
T50,3.3593828678131104
T51,0.0
T52,1.6764670610427856
T53,0.0
T54,0.004072587005794048
T55,2.378579616546631
T56,1.9153804779052734
T57,0.0
T58,0.0
T59,0.0
T60,0.0
T61,3.2462127208709717
T62,0.0
```

T63,0.0
T64,1.5748587846755981
T65,3.1728198528289795
T66,0.0
T67,0.0
T68,0.0
T69,0.0
T70,0.0
T71,0.0
T72,0.0
T73,0.0
T74,0.0
T75,0.0
T76,0.0
T77,0.0
T78,0.0
T79,0.0
T80,0.0
T81,1.7906768321990967
T82,0.0
T83,0.0
T84,0.0
T85,0.0
T86,5.034870147705078
T87,0.0
T88,0.0
T90,0.0
T91,4.17775821685791
T92,0.0
T93,0.0
T94,0.0
T95,0.0
T96,0.0
T98,0.0
T99,0.0
T100,0.0
T101,5.032817363739014
T102,0.0
T103,0.0
T104,0.0
T107,0.0
T109,0.0
T110,0.0
T111,0.0
T112,0.0
T113,0.0
T114,0.0
T115,5.512697696685791

T116,0.0
T117,0.0
T118,0.0
T119,4.944796562194824
T120,0.0
T121,0.9294309020042419
T122,0.0
T123,0.0
T124,0.0
T125,0.0
T126,0.0
T127,2.014962673187256
T128,0.0
T129,0.0
T130,0.0
T131,0.0
T132,0.0
T133,3.7356011867523193
T134,0.0
T135,0.0
T136,0.477792888879776
T137,0.0
T138,0.0
T139,0.0
T140,0.0
T141,0.6758896112442017
T142,0.0
T143,0.0
T144,4.957370758056641
T145,0.0
T146,0.0
T147,2.5923447608947754
T148,5.382846832275391
T149,0.0
T151,0.0
T152,4.402044296264648
T153,0.0
T154,0.0
T155,0.0
T156,0.0
T157,0.0
T158,0.0
T159,0.0
T160,0.0
T161,0.0
T162,0.0
T163,2.4163029193878174
T164,0.0

```
T166,0.0
T167,0.0
T168,0.0
T169,0.0
T170,0.0
T171,0.0
T172,0.0
T173,0.0
T174,0.0
T175,0.0
T176,0.0
T177,0.0
T178,0.0
T179,0.0
T180,0.0
T181,0.0
T182,0.0
T183,0.0
T184,0.0
T185,4.638904571533203
T186,0.0
T187,0.0
T188,0.7020785808563232
T189,0.481899619102478
T190,1.6105284690856934
T191,0.0
T192,0.0
T193,0.0
T194,0.0
T195,0.0
T196,0.0
T197,0.0
T198,0.0
T199,3.0912137031555176
T200,0.0
T201,0.0
T202,0.0
T203,0.0
T204,0.0
T205,0.0
T206,0.0
T207,0.0
T208,0.0
T209,0.0
T210,0.0
T211,0.0
T212,0.0
T213,0.0
```

```
T214,0.0
T215,0.0
T216,0.0
T217,0.0
T218,0.8026736974716187
T219,0.0
T220,0.0
T221,0.0
T222,0.0
T223,0.0
T224,0.0
T225,0.0
T226,0.0
T227,0.0
T228,0.0
T229,0.0
T230,0.0
T231,0.0
T232,0.0
T233,2.4414517879486084
T234,0.0
T235,2.8281912803649902
T236,0.0
T237,0.0
T238,0.0
T239,0.0
T240,0.0
T241,0.0
T242,0.0
T243,0.0
T244,0.0
T245,1.0405468940734863
T246,0.0
T247,0.0
T248,0.0
T249,0.0
T250,0.0
T251,0.0
T252,0.0
T253,0.0
T254,0.0
T255,0.0
T256,0.0
T257,0.8906944990158081
T258,0.0
T259,0.0
T260,0.0
T261,0.0
```

T262,0.0
T263,0.0
T264,0.0
T265,0.0
T266,0.0
T267,0.0
T268,0.0
T269,0.0
T270,0.0
T271,0.0
T272,0.0
T273,0.0
T274,0.0
T275,0.0
T276,0.0
T277,0.0
T278,0.0
T279,0.0
T280,0.0
T281,0.0
T282,0.0
T283,0.0
T284,0.19910281896591187
T285,0.0
T286,0.5229436755180359
T287,0.0
T288,0.0
T289,0.0
T290,0.0
T291,0.0
T292,0.0
T293,0.0
T294,0.0
T295,0.0
T296,0.0
T297,0.2735089361667633
T298,2.422717332839966
T299,0.0
T300,5.328442573547363
T301,0.5208902955055237
T302,2.8261382579803467
T303,0.0
T304,0.0
T305,1.4041913747787476
T306,0.0
T307,0.0
T308,0.0
T309,0.0

```
T310,0.0
T311,0.0
T312,2.1093978881835938
T313,0.0
T314,0.0
T315,0.0
T316,0.0
T317,0.0
T318,0.0
T319,3.894961357116699
T320,2.2371950149536133
T321,0.0
T322,0.0
T323,2.924679756164551
T324,0.0
T325,0.0
T326,0.0
T327,0.0
```

[ ]:

```

```