赛题代码

一、导入所需模块

```
import numpy as np
import pandas as pd
import copy
import gc
import time
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
from sklearn.decomposition import PCA

import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.cluster import DBSCAN,KMeans
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error as MSE
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from xgboost import plot_importance, plot_tree
import lightgbm as lgb

import torch
import torch.nn as nn
from torch.utils.data import DataLoader,Dataset
from torch.nn import functional as F

from joblib import dump, load
%matplotlib inline
```

二、读取数据

```
train=pd.read_csv("path/train.zip")
test=pd.read_csv("path/test.zip")

B=pd.read_csv("path/Addition.csv")
train=pd.concat([train,B],axis=1)
```

```
B=pd.read_csv("path/Addition_test.csv")
test=pd.concat([test,B],axis=1)
```

注意这里的聚类特征由KMeans构建：

```
# coord=train[['pickup_latitude','pickup_longitude']].values
# coord1=test[['pickup_latitude','pickup_longitude']].values
# c=np.vstack((coord,coord1))
# # 下车点聚类
# # 不做DBSCAN了，有机会再试
# coord=train[['dropoff_latitude','dropoff_longitude']].values
# coord1=test[['dropoff_latitude','dropoff_longitude']].values
# d=np.vstack((coord,coord1))
# f=np.vstack((c,d))
# Cluster=KMeans(n_clusters=18, random_state=0, n_init=15).fit(f)
# dump(Cluster,"cluster.pkl")
```

保存后直接读取结果就是了

```
test_C=load("cluster.pkl")
train['in_Cluster']=test_C.predict(train[['pickup_latitude','pickup_longitude
']].values)
train['out_Cluster']=test_C.predict(train[['dropoff_latitude','dropoff_longit
ude']].values)
test['in_Cluster']=test_C.predict(test[['pickup_latitude','pickup_longitude']
].values)
test['out_Cluster']=test_C.predict(test[['dropoff_latitude','dropoff_longitud
e']].values)
```

天气数据预处理

```
weather = pd.read_csv("wdn2016.csv")
weather['precipitation'] = pd.to_numeric(weather['precipitation'],
errors='coerce')
weather['snow fall'] = pd.to_numeric(weather['snow fall'], errors='coerce')
weather['snow depth'] = pd.to_numeric(weather['snow depth'], errors='coerce')
weather = weather.fillna(0)
```

三、数据处理

这部分主要是特征工程，我们将其集成到一个方法里方便调用：

```
def process(train,is_Train=True):


##############################################################################
############################
    ############                          时间特征
################################

##############################################################################
############################

    def rush_hour_f(row):
        rhour = row['real_hour']
        if (6 <= rhour) & (rhour <= 10):
            return 1
        if (10 < rhour) & (rhour < 16):
            return 2
        if (16 <= rhour) & (rhour <= 20):
            return 3
        return 0

    encoder.fit(train['store_and_fwd_flag'])
    train['store_and_fwd_flag'] =
encoder.transform(train['store_and_fwd_flag'])

    train['pickup_datetime'] = pd.to_datetime(train['pickup_datetime'])
    train['month'] = train['pickup_datetime'].dt.month
    train['weekday'] = train['pickup_datetime'].dt.weekday
    train['hour'] = train['pickup_datetime'].dt.hour
    train['minute'] = train['pickup_datetime'].dt.minute
    train['minute_of_day'] = train['hour'] * 60 + train['minute']
    train['real_hour'] = train['minute_of_day'] / 60
        #for weather
    train['year'] = train['pickup_datetime'].dt.year
    train['day'] = train['pickup_datetime'].dt.day
    train['rush_hour'] = train.apply(rush_hour_f, axis=1)

    cal = calendar()
    holidays = cal.holidays(start=pd.Timestamp(2015,12,31),
end=pd.Timestamp(2017,1,1))
    H=[[i.month,i.day] for i in holidays]
    train['is_weekend']=train['weekday'].apply(lambda x:1 if x>4 else 0)
    for i,j in H:
        # 这里特别值得注意，查询返回的是一个视图
        # 对视图的修改并不会返回到原始上
        # 因此，我们需要做的是，采用iloc获得条件查询的index并进行修改
        train.iloc[train[(train['month']==i)&(train['day']==j)].index,-1]=1
    encoder.fit(train['is_weekend'])
    train['is_weekend'] = encoder.transform(train['is_weekend'])


##############################################################################
```

```
############################
    ############                        空间特征
################################

#############################################################################################
############################
    AVG_EARTH_RADIUS =6369
    def ft_haversine_distance(lat1, lng1, lat2, lng2):
    # Haversine距离，用于测量大地距离
        lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
        lat = lat2 - lat1
        lng = lng2 - lng1
        d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng
* 0.5) ** 2
        h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
        return h

    def ft_degree(lat1,log1,lat2,log2):
        # 用于计算坐标方位
        lng_delta_rad = np.radians(log2 - log1)
        lat1, lng1, lat2, lng2 = map(np.radians, (lat1,log1,lat2,log2))
        y = np.sin(lng_delta_rad) * np.cos(lat2)
        x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) *
np.cos(lng_delta_rad)
        return np.degrees(np.arctan2(y, x))
    train['distance']=ft_haversine_distance(train['pickup_latitude'].values,

train['pickup_longitude'].values,

train['dropoff_latitude'].values,

train['dropoff_longitude'].values)
    train['direction'] = ft_degree(train['pickup_latitude'].values,
                                   train['pickup_longitude'].values,
                                   train['dropoff_latitude'].values,
                                   train['dropoff_longitude'].values)

    if is_Train:
        train = train[(train.trip_duration < 1000000)]
        train = train[train['pickup_longitude'].between(-75, -73)]
        train = train[train['pickup_latitude'].between(40, 42)]
        train = train[train['dropoff_longitude'].between(-75, -73)]
        train = train[train['dropoff_latitude'].between(40, 42)]
        duration = train['trip_duration']
        train['speed'] = train.distance / duration * 2236.936292
        train['trip_duration'] = np.log(train['trip_duration'].values)


#############################################################################################
############################
    ############                        附带信息特征
################################
```

```
#############################################################################
############################
        #NYS max speed limit 55mph

    weather["date"] = pd.to_datetime(weather["date"],format='%d-%m-%Y')
    weather['year'] = weather['date'].dt.year
    weather_2016 = weather[weather["year"] == 2016]
    weather_2016.drop(["year"], axis=1, inplace=True)


train['date']=pd.to_datetime(train[['year','month','day']],errors='coerce')
    left_merge = pd.merge(left=train, right=weather_2016, on="date",
how="left")
    train = left_merge.loc[:, left_merge.columns != 'date']

    train['in_Cluster']=train['in_Cluster'].astype(str)
    train['out_Cluster']=train['out_Cluster'].astype(str)
    train['node']=train['in_Cluster']+"/"+train['out_Cluster']

    encoder.fit(train['node'])
    train['node']=encoder.transform(train['node'])
    encoder.fit(train['in_Cluster'])
    train['in_Cluster']=encoder.transform(train['in_Cluster'])
    encoder.fit(train['out_Cluster'])
    train['out_Cluster']=encoder.transform(train['out_Cluster'])

    return train
```

```
train,test=process(train),process(test,False) # 这样就好了
```

四、机器学习模型

```
def xgb_train(x,y):
    xgb_regressor = lgb.LGBMRegressor(
    n_estimators=500,
    learning_rate=0.1,
    max_depth=25,
    subsample=0.9,
    random_state=42,
    colsample_bytree=0.8,
    eta=0.05
    )
    xgb_regressor.fit(x[features], y)
    print("Score in Train
%.10f"%np.sqrt(MSE(xgb_regressor.predict(x[features]),y)))
    print("Score in Val
%.10f"%np.sqrt(MSE(xgb_regressor.predict(testXsplit[features]),testYsplit)))
    return xgb_regressor
```

```
def predict(model,name='sub'):
    y_pred = a.predict(test[features])
    submission = pd.DataFrame({'id': test.id, 'trip_duration':
np.exp(y_pred)})
    submission.fillna( submission['trip_duration'].mean(),inplace=True)
    submission.to_csv("%s.csv"%name,index=False)
```

```
columns=[ 'vendor_id',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'month', 'weekday',
          'hour', 'minute', 'minute_of_day',
       'real_hour', 'rush_hour', 'is_weekend', 'year', 'day',
          'distance','humidity', 'pressure', 'temperature',
       'wind_direction', 'wind_speed',
       'direction', 'maximum temperature', 'minimum temperature',
       'average temperature', 'precipitation', 'snow fall', 'snow
depth','node','in_Cluster','out_Cluster','speed','trip_duration'
         ]
features=columns[:-2]
```

```
x_var=train[train.speed<37]
y_var=x_var.trip_duration
```

```
trainXsplit, testXsplit, trainYsplit, testYsplit = train_test_split(x_var,
y_var, test_size=0.3, random_state=42)
trainXsplit.shape, trainYsplit.shape, testXsplit.shape, testYsplit.shape
```

```
lgb_params = {
    #'metric' : 'rmse',
    'learning_rate': 0.1,
    'max_depth': 25,
    'num_leaves': 1000,
    'objective': 'regression',
    'feature_fraction': 0.9,
    'bagging_fraction': 0.5,
    'max_bin': 1000
#     ,'verbosity': 2
    }

lgb_df = lgb.Dataset(x_var[features], y_var)
# lgb_model = lgb.train(lgb_params, lgb_df, num_boost_round=1500)
# dump(lgb_model,"model/model1.pkl")
lgb_model=load("model/model1.pkl")

y_pred=lgb_model.predict(test[features])
submission=pd.DataFrame({'id': test.id, 'trip_duration':
np.round(np.exp(y_pred))})
```

```
submission.fillna( submission['trip_duration'].mean(),inplace=True)
submission.to_csv("result/lgbm.csv",index=False)
```

五、深度学习模型

```
class NetR(nn.Module):
    def
__init__(self,in_features=48,n_hidden1=48,n_hidden2=64,n_hidden3=32,out_featu
res=1):
        super(NetR, self).__init__()
        self.flatten=nn.Flatten()
        self.hidden1=nn.Sequential(
            nn.Linear(in_features,n_hidden1,False),
            nn.BatchNorm1d(n_hidden1),
            nn.GELU()
        )
        self.hidden2=nn.Sequential(
            nn.Linear(in_features,n_hidden1),
            nn.Dropout(0.5),
            nn.BatchNorm1d(n_hidden1),
           nn.GELU()
        )
        self.hidden3=nn.Sequential(
            nn.Linear(n_hidden1,n_hidden2),
            nn.BatchNorm1d(n_hidden2),
            nn.GELU()
        )
        self.hidden5=nn.Sequential(
            nn.Linear(n_hidden2,n_hidden3),
            nn.BatchNorm1d(n_hidden3),
            nn.GELU()

        )
        self.out=nn.Sequential(nn.Linear(n_hidden3,out_features))

    def forward(self,x):
        x1=self.hidden1(x)
        x2=self.hidden2(x)
        x3=self.hidden3(x2+x1)

        o=self.hidden5(x3)

        return self.out(o)
class MLP1(nn.Module):
    def __init__(self, in_dim):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(in_dim, 256),
            nn.LayerNorm(256),
            nn.GELU(),
```

```
            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.GELU(),
            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.GELU(),
            nn.Linear(64, 32),
            nn.GELU(),
            nn.Linear(32, 20),
            nn.GELU(),
            nn.Linear(20, 18),
            nn.GELU(),
            nn.Linear(18, 16),
            nn.GELU(),
            nn.Linear(16, 14),
            nn.GELU(),
            nn.Linear(14, 12),
            nn.GELU(),
            nn.Linear(12, 10),
            nn.GELU(),
            nn.Linear(10, 8),
            nn.GELU(),
            nn.Linear(8, 6),
            nn.GELU(),
            nn.Linear(6, 4),
            nn.GELU(),
            nn.Linear(4, 2),
            nn.GELU(),
            nn.Linear(2, 1),

        )

    def forward(self, x):
        return self.layers(x)


class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim=64, output_dim=1, num_layers=2):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers,
batch_first=True)
        self.fc = nn.Sequential(
            nn.Linear(hidden_dim, 128),
            nn.BatchNorm1d(128),
            nn.GELU(),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.GELU(),
            nn.Linear(256, 16),
            nn.GELU(),
            nn.Linear(16, 1),

        )
```

```
    def forward(self, x):
        # LSTM需要的输入是 (batch_size, seq_length, num_features)
        lstm_out, _ = self.lstm(x)
        # 仅使用最后一个时间步的输出
        out = self.fc(lstm_out[:, -1, :])
        return out
```

```
ef train_(model,x_train_tensor,y_train_tensor,path="1.pth",epoch=200):

    criterion=nn.MSELoss()
    # optimizer=torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9,
weight_decay=1e-5, nesterov=True)
    optimizer =torch.optim.AdamW(model.parameters(), lr=0.001)

    device="cuda" if torch.cuda.is_available() else "cpu"
    # device="cpu"
    if torch.cuda.device_count() > 1:
        print(f"Let's use {torch.cuda.device_count()} GPUs!")
        model = nn.DataParallel(model)

    model.to(device)
    criterion.to(device)
    Best_=np.inf
    for e in tqdm.tqdm(range(epoch)):
        model.train()
#         for idx,(x,y) in enumerate(train_Data):


x_train_tensor,y_train_tensor=x_train_tensor.to(device),y_train_tensor.to(dev
ice)
        optimizer.zero_grad()
        y_pre=model(x_train_tensor)
        loss=torch.sqrt(criterion(y_pre.squeeze(),y_train_tensor))
        loss.backward()
        optimizer.step()


        if (loss.item())< Best_:
            Best_=loss.item()
            state={
                'epoch':e,
                'best_':loss.item(),
                "model_state_dict":model.state_dict(),
                'optimizer_state_dict':optimizer.state_dict()}
            torch.save(state,path)
        if e%100==0:
            print(f"epoch: {e+1}/{epoch}")
            print(f"Score: {loss.item()}")
```

```
class LoadData(Dataset):
    def __init__(self,x,y):
        self.x=x
        self.y=y

    def __getitem__(self, idx):
        return self.x[idx],self.y[idx]

    def __len__(self):
        return  len(self.x)
```

标准化

```
std=StandardScaler()

features_toTrian=['vendor_id', 'passenger_count', 'pickup_longitude',
'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
'month',
       'weekday', 'hour', 'minute', 'rush_hour',
       'is_weekend', 'distance', 'humidity', 'pressure',
       'temperature', 'wind_direction', 'wind_speed', 'direction',
       'precipitation', 'snow fall', 'snow depth', 'node', 'in_Cluster',
       'out_Cluster']

x_to_train=copy.deepcopy(x_var[features_toTrian])
```

One-Hot编码

```
def get_dummy(data,col):
    val=pd.get_dummies(data[col],dtype=np.uint8,prefix=col)
    data=pd.concat([data,val],axis=1)
    data.drop(col,axis=1,inplace=True)
    return data

oneHot=[
    'vendor_id',
    'store_and_fwd_flag',
    'rush_hour',
    'is_weekend',
    'node', 'in_Cluster',
    "weekday"
]

for i in oneHot:
    x_to_train=get_dummy(x_to_train,i)
```

转为Tensor

```
x_train=std.fit_transform(x_to_train)
x_train_tensor=torch.from_numpy(x_train).float()
mean_y,std_y=y_var.mean(),y_var.std()
y_var=(y_var-mean_y)/std_y
y_train_tensor=torch.from_numpy(y_var.values).float()
m1=MLP1(in_dim=x_train_tensor.shape[1])
train_(m1,x_train_tensor=x_train_tensor,y_train_tensor=y_train_tensor,path="m
odel/dl1.pth",epoch=3000)
```

六、模型融合与提交

最终提交采用交叉验证方式，我们在这个项目上尝试了不少Tricks，结果都不尽人意，深度学习的效果也不算特别好，可能是因为异常值特别的多的原因吧。提交的代码为：

```
def get_oof_ligGBM(x_train,y_train,x_test,p=None):

    lgb_params = {
'learning_rate': 0.1,
'max_depth': 16,
'num_leaves':1000,
'objective': 'regression',
'feature_fraction': 0.8,
'bagging_fraction': 0.6,
'max_bin': 1000
,'verbosity': -1,
    'metric':'rmse',
} if p==None else p

    oof_train=np.zeros((x_train.shape[0],))
    oof_test=np.zeros((x_test.shape[0],))


    for i,(train_index,valid_index) in enumerate(kf.split(x_train,y_train)):
        trn_x,trn_y=x_train.iloc[train_index],y_train.iloc[train_index]
        val_x,val_y=x_train.iloc[valid_index],y_train.iloc[valid_index]
        dtrn=lgb.Dataset(trn_x,trn_y)
        dval=lgb.Dataset(val_x,val_y)
        bst=lgb.train(lgb_params, dtrn, num_boost_round=1500,valid_sets=
[dtrn,dval],early_stopping_rounds=100,verbose_eval=100)

        oof_train[valid_index]=bst.predict(val_x)
        oof_test+=bst.predict(x_test)/5
    return oof_train.reshape(-1,1),oof_test.reshape(-1,1)
```

\

```
def makeRes(pred,p):
    submission = pd.DataFrame({'id': ori_test.id, 'trip_duration': np.nan})
    submission.iloc[test_index,-1]=pred
    submission.fillna(submission.trip_duration.mean(),inplace=True)
#     submission.fillna(0,inplace=True)
    submission.to_csv("%s.csv"%p,index=False)
```

```
x,y=x_clear[features],x_clear.trip_duration

%%time
pred_train3,pred_test3=get_oof_ligGBM(x,y,test[features])

makeRes(np.exp(pred_test3),'model_1')
```

最终结果为

LB	分数	排名
私榜	0.36536	36/1255
公榜	0.36862	39/1255