Final

April 30, 2021

[1]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import category_encoders as ce
import xgboost as xgb
import pickle
import joblib
import os
import numpy as np
from multipledispatch import dispatch
from sklearn.metrics import f1_score
```

[2]:
```
def read_pickle_dictionary(filename):
    '''
    Load serialized dictionary
    '''
    with open(os.path.join(filename), 'rb') as handle:
        file_dict = pickle.load(handle)

    return file_dict

def deserialize_model( filename):
    """
    DeSerialize trained model
    """
    return joblib.load(filename)


def standardize(X_train, X_test ,test_preprocessing_object , flag ='train'):
    """
    This function standardize test and train columns if flag is train␣
 ↪standization will fit and trainsform
    if test it will just standardize.
    returns train ,test and  test_preprocessing_object if flag is train else␣
 ↪test data and test_preprocessing_object
    X_train : train data
    X_test  :test data
```

```
    test_preprocessing_object : dictionary containing object of columns␣
↪transformer for different column
    flag : string either train or test
    """
    std_columns␣
↪=['ur_pr_reordered','order_number','ttl_cnt_product_user','Avg_no_prod_perOrder','days_sinc
    scaler_objects =[]

    if(flag == 'train'):
        for col in std_columns:

            scaler = StandardScaler()
            scaler.fit(X_train.loc[:,col].values.reshape(-1,1))
            X_train.loc[:,col] =scaler.transform(X_train.loc[:,col].values.
↪reshape(-1,1))
            X_test.loc[:,col] =scaler.transform(X_test.loc[:,col].values.
↪reshape(-1,1))
            scaler_objects.append({col:scaler})
            del scaler
            gc.collect()
        test_preprocessing_object['std']=scaler_objects
    elif(flag =='test'):
        for item in test_preprocessing_object['std']:
            for col_item in item.items():
                col =col_item[0]
                scaler=col_item[1]

                X_test.loc[:,col] =scaler.transform(X_test.loc[:,col].values.
↪reshape(-1,1))
    if(flag=='train'):
        return X_train.copy() , X_test.copy() ,test_preprocessing_object
    else:
        return X_test.copy() ,test_preprocessing_object

def response_code_test( X_test ,response_dict):
    """
    This function takes data and does fit transform based on column wise,
    according to column specific encoder stored in reponse_dctionary
    X_test : test data
    response_dict : dictionary containing encoder object
    return transformed test data
    """

    response_column =['max_hour_of_day' ,'reordered_last','max_dow']
    for col in response_column:
        encoder =response_dict[col]
        X_test.loc[:,col] =encoder.transform(X_test.loc[:,col])
```

```
    return X_test.copy()


def merge_products(x):
    """
    x : string input
    This function merge group to a list
    returns list of strings
    """
    return " ".join(list(x.astype('str')))
```

[3]:
```
def suggestProduct(test_sub, pred , thresh):
    """
    This suggests products based on if Prediction is reordered =1

    returns Dataframe containing order_id and group of product_id seperated by␣
 ↪space

    test_sub : dataframe containing order_id and product_id

    pred:prediction probality of positive class

    thresh : optimal threshold to convert probality to class
    """

    # if pobality is greter than threshold predict 1 else 0
    test_sub["Pred"] = np.where(pred>=thresh ,1,0)
    # select all cases where prediction is 1
    test_sub = test_sub.loc[test_sub["Pred"].astype('int')==1]
    #group by order_id and create lsit of products
    test_sub = test_sub.groupby("order_id")["product_id"].
 ↪aggregate(merge_products).reset_index()
    test_sub.columns = ["order_id", "products"]


    return test_sub['products'].values

@dispatch(list,int)
def validate_order_id( orderIds,id_):
    """
    This function checks if querried order id is there in list of order_id
    returns True is order id is present and false if not present
```

```
    """

    flag =False
    for order_id in orderIds:
        if id_ == order_id:
            flag=True
            break
    return flag


@dispatch(list,list)
def validate_order_id( orderIds,id_list):

    """
    This function returns list of valid order_id  querried by user
    """

    valid_order_ids =[]
    for querry_id in id_list:
        for order_id in orderIds:
            if order_id ==querry_id:
                valid_order_ids.append(querry_id)
                break

    return valid_order_ids
```

[4]:
```
@dispatch(int)
def final(orderNumber):
    """
    This function takes orderNumber and suggest Product user is most Likely to␣
 ↪buy
    orderNumber :Integer
    retruns : None or string of product Id seperated by space

    """
    # Read data set
    test =pd.read_parquet('data/test.gzip')
    # get all the order_id in dataset
    orderIds =list(test.order_id.values)
    product_suggestion ='None'

    # check if order If order Id querred is valid
    if(validate_order_id( orderIds,orderNumber)):
        # filter dataset based on orderId
        test=test[test.order_id ==orderNumber]
```

```
        # store all the product user bought for particular order id
        test_temp = test[["order_id", "product_id"]]

        # drop unnecessarory columns
        test.drop(columns =["order_id",'ur_pr_count' ,"user_id" ,'product_id'␣
↪,'department_id' ,'aisle_id' ,'ur_pr_count'] , inplace =True)
        # standardizing test data
        test_preprocessing_object= read_pickle_dictionary(os.path.
↪join('final_model_pkl' ,'test_preprocessing_object_dict.pkl' ))
        test , test_preprocessing_object=standardize(None, test␣
↪,test_preprocessing_object , flag ='test')
        # Target code encodding
        reponse_dict= read_pickle_dictionary(os.path.join('final_model_pkl'␣
↪,'reponse_dict.pkl' ))
        test=response_code_test( test ,reponse_dict)
        # read pickled  model for prediction
        xgboost=deserialize_model('final_model_pkl/xgboost.pkl')
        # predict probality
        predict_xg_test =xgboost.predict_proba(test)[:,1]
        # threshold for prediction
        threshold=0.692886
        product_name =suggestProduct(test_temp ,predict_xg_test ,threshold)
        # if suggested product is not empty take that as suggested product else␣
↪None
        if(product_name.shape[0]>0):
            product_suggestion =product_name[0]

    return product_suggestion
```

[5]:
```
@dispatch(list)
def final(orderNumber_list):
    """
    This function takes list orderNumber and return F1 score
    orderNumber :list of integer


    """

     # Read data set
    train =pd.read_parquet('data/train.gzip')
    # get all the order_id in dataset
    orderIds =list(train.order_id.values)


    y_predicted_final =[]
```

```
    y_orignal_final =[]

    # check if order If order Id querred is valid
    validated_order_id=validate_order_id( orderIds,orderNumber_list)

    # filter dataset based on orderId
    train=train.loc[train.order_id.isin(validated_order_id)]
    # getting Y original value
    y_orignal = train.reordered
    y_orignal_final.extend(y_orignal)

    # preparing train data
    train.drop(columns =["order_id",'ur_pr_count' ,"user_id" ,'product_id'␣
↪,'department_id' ,'aisle_id' ,'ur_pr_count' ,'reordered'] , inplace =True)
    # standardizing train data

    test_preprocessing_object= read_pickle_dictionary(os.path.
↪join('final_model_pkl' ,'test_preprocessing_object_dict.pkl' ))
    train , test_preprocessing_object=standardize(None, train␣
↪,test_preprocessing_object , flag ='test')
    # Target code encodding
    reponse_dict= read_pickle_dictionary(os.path.join('final_model_pkl'␣
↪,'reponse_dict.pkl' ))
    train=response_code_test( train ,reponse_dict)

    # read pickled  model for prediction
    xgboost=deserialize_model('final_model_pkl/xgboost.pkl')
    # predict probality
    predict_xg_test =xgboost.predict_proba(train)[:,1]

    threshold=0.692886
    # threshold for prediction
    y_predicted =np.where(predict_xg_test>=threshold ,1,0)
    y_predicted_final.extend(y_predicted)


    return  f1_score(y_orignal_final ,y_predicted_final)
```

0.1 1. final method is overloaded

0.2 final(int) - Take Order number and returns string which contains list of product_id seperarted by space

0.3 final(list[int]) Take list of Order number and returns F1_score(y_true , y_pred)

[6]:
```
test =pd.read_parquet('data/test.gzip')
orderIds_te =set(test.order_id.values)

# Read data set
train =pd.read_parquet('data/train.gzip')
# get all the order_id in dataset
orderIds_tr =set(train.order_id.values)

print("Train and test order id id mutually exclusive:",orderIds_te.
 ↪intersection(orderIds_tr))
```

```
Train and test order id id mutually exclusive: set()
```

1 final(int)

Filter all the row based on order_id than pass data it through standardization and Target encoding

Load pickled model, make prediction

For all the prediction that is marked 1 or reodered , use associated product_id as suggestion

[7]:
```
# loading test data to get order_id
```

[8]:
```
test =pd.read_parquet('data/test.gzip')
orderIds_te =list(set(test.order_id.values))
```

[9]:
```
# randomly take 20 order id and make prediction
for i in range(20):
    order_id=np.random.choice(orderIds_te ,1)
    print("Order id: {} suggested product: {}  ".format(order_id,␣
 ↪final(int(order_id[0]))))
```

```
Order id: [2277305] suggested product: 41950 1446 25919 22883 31181 16136 9146
Order id: [2468092] suggested product: 15902 34270 16557
Order id: [2105909] suggested product: 27323 33768 30827 36557 44271
Order id: [905492] suggested product: 15613 47766 49235 27966 1463 19057 11182
19677 3952 6532 21334 30949 28226 21775 9426
Order id: [1182327] suggested product: 13176 24852 22935 28985 37646 5077 4210
32655 15712 611
```

```
Order id: [841721] suggested product: 5025 16797 24852 260 4920 7969 20114 45
33090 46327
Order id: [3366061] suggested product: 5876 6210 34658 35451 35917
Order id: [1108984] suggested product: 47766 38650 37022 12395
Order id: [1098688] suggested product: 30618
Order id: [1933382] suggested product: 24852 33647 11182 38456 22656 27850 11471
8732 23033
Order id: [922821] suggested product: 24852 37158 5077 17559 39758 32463 25783
14381 34320 15263 20191 22175
Order id: [2541621] suggested product: 21903 21137 22825 27104 33198 47209 37646
27845 26800 21616 48109 3957 19816 19613 36717 30949 30561 15943
Order id: [1068037] suggested product: 24852 10580
Order id: [552610] suggested product: 21288 13966 13634 33494 38978
Order id: [1631737] suggested product: 44632 46979 22935 30233 20754 34270 31257
Order id: [418310] suggested product: 44142 19348 43867 4317 27022 43331 5198
26425 34146 42153
Order id: [2868837] suggested product: 13176 40706 24964 8174 41950 24799 44628
48364 14129
Order id: [646853] suggested product: 26914 32689 4100
Order id: [2220216] suggested product: 43352 41178 12508 46779 26528
Order id: [2867464] suggested product: 21137 24852 46906 21174 3957 21289 39825
```

2 final([int])

Using train data as test data does not have target value to calulate f1 score

Filter all the row based on list of order_id than pass data it through standardization and Target encoding

Load pickled model, make prediction

Use predicted and orignal target value to calculate f1 score

[10]:
```
# Read data set
train =pd.read_parquet('data/train.gzip')
# get all the order_id in dataset
orderIds_tr =list(set(train.order_id.values))

# randomly take 20 order id and make prediction
for i in range(20):
    np.random.randint(1,5)

    order_ids=list(np.random.choice(orderIds_tr ,np.random.randint(1,5)))
    print("Order list : " ,order_ids)
    print("F1_score(y_true , y_pred) : ",final(order_ids))
```

```
Order list :  [774701]
F1_score(y_true , y_pred) :  0.15384615384615383
```

```
Order list :  [120316, 1886837, 3381434, 1261929]
F1_score(y_true , y_pred) :  0.39999999999999997
Order list :  [3320875, 2208104]
F1_score(y_true , y_pred) :  0.3225806451612903
Order list :  [2385852]
F1_score(y_true , y_pred) :  0.0
Order list :  [2784628, 505149, 653655, 2477494]
F1_score(y_true , y_pred) :  0.37777777777777777
Order list :  [1813819, 2857532]
F1_score(y_true , y_pred) :  0.5128205128205129
Order list :  [757660, 1612762, 2762360, 346468]
F1_score(y_true , y_pred) :  0.5416666666666666
Order list :  [3227691, 3025511, 3008881]
F1_score(y_true , y_pred) :  0.375
Order list :  [434445, 2735175, 1649476, 2385603]
F1_score(y_true , y_pred) :  0.5000000000000001
Order list :  [652399, 1173294, 246988, 294698]
F1_score(y_true , y_pred) :  0.3076923076923077
Order list :  [3159731]
F1_score(y_true , y_pred) :  0.2222222222222222
Order list :  [397809]
F1_score(y_true , y_pred) :  0.6666666666666666
Order list :  [2726867, 1792242, 2961638, 3085804]
F1_score(y_true , y_pred) :  0.32
Order list :  [948071, 730748, 691102]
F1_score(y_true , y_pred) :  0.5217391304347826
Order list :  [33785, 1150572, 1282426, 518144]
F1_score(y_true , y_pred) :  0.5494505494505495
Order list :  [2386179, 3346023, 3323369]
F1_score(y_true , y_pred) :  0.6285714285714287
Order list :  [948267, 3036539, 333318, 1834363]
F1_score(y_true , y_pred) :  0.6363636363636365
Order list :  [732189, 1253975]
F1_score(y_true , y_pred) :  0.25
Order list :  [2902493, 927600, 1639895]
F1_score(y_true , y_pred) :  0.4324324324324324
Order list :  [3235324, 1984264, 151224]
F1_score(y_true , y_pred) :  0.6250000000000001
```