Selected Models and saving those for further use

In [1]:

```
#import libraries....
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.sparse import hstack
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,f1_score

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
import lightgbm as lgb
from sklearn.model_selection import StratifiedKFold
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder

import datetime

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
#load the data with all created features
data = pd.read_csv("data_with_advanced_features.csv")
data.drop("Unnamed: 0", inplace=True, axis=1)
```

In [3]:

```
#label encoding of seller_id
label = LabelEncoder()
seller = label.fit_transform(data.seller_id)
data["seller_id"] = seller

#save the encoder
filename="seller_id_encode.pkl"
pickle.dump(label,open(filename,"wb"))

#label encoding of product id
label = LabelEncoder()
product = label.fit_transform(data.product_id)
data["product_id"] = product

# save the encoder
filename="product_id_encode.pkl"
pickle.dump(label,open(filename,"wb"))
```

Creating binary classifier system

In [4]:

```
#creating class labels
binary = []
for i in range(len(data)):
    if data.review_score[i]==5:
        binary.append(1)
    else:
        binary.append(0)

data["binary_target"] = binary
```

In [5]:

```
#target variable is review_score
Y = data["binary_target"]
X = data
```

Train test split

In [6]:

```
#train test split with test size 25% and 75% of data as train
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.25,stratify=Y,random_state=10)
```

Featurization

In [7]:

```
#payment_type
vec = CountVectorizer()

vec.fit(x_train["payment_type"].values)

x_tr_pay_type = vec.transform(x_train.payment_type.values)
x_te_pay_type = vec.transform(x_test.payment_type.values)

#save as pickle file
filename = "count_vect_payment_1.pkl"
pickle.dump(vec,open(filename,"wb"))
```

In [8]:

```
#order_item_id
x_train.order_item_id = x_train.order_item_id.astype(str)
x_test.order_item_id = x_test.order_item_id.astype(str)

vec = CountVectorizer(vocabulary=range(1,22))

vec.fit(x_train["order_item_id"])

x_tr_id = vec.transform(x_train.order_item_id)
x_te_id = vec.transform(x_test.order_item_id)

#save as pickle file
filename = "count_vect_item_1.pkl"
pickle.dump(vec,open(filename,"wb"))
```

In [9]:

```
#product_category_name
vec = CountVectorizer()

vec.fit(x_train["product_category_name"].values)

x_tr_cat = vec.transform(x_train.product_category_name.values)
x_te_cat = vec.transform(x_test.product_category_name.values)

#save as pickle file
filename = "count_vect_cat_1.pkl"
pickle.dump(vec,open(filename,"wb"))
```

Binary features

In [10]:

```
x_tr_same_state = x_train.same_state.values.reshape(-1,1)
x_te_same_state = x_test.same_state.values.reshape(-1,1)

x_tr_same_city = x_train.same_city.values.reshape(-1,1)
x_te_same_city = x_test.same_city.values.reshape(-1,1)

x_tr_late_shipping = x_train.late_shipping.values.reshape(-1,1)
x_te_late_shipping = x_test.late_shipping.values.reshape(-1,1)

x_tr_high_freight = x_train.high_freight.values.reshape(-1,1)
x_te_high_freight = x_test.high_freight.values.reshape(-1,1)
```

Numerical features

In [11]:

```
#data to be standardized
tr = x_train[["payment_sequential","payment_installments","payment_value","seller_id","product_id","seller
              "bs_share","cust_share",
          "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_descripti
           "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","delivery
              "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_of_
              "num_of_sellers_for_cust","total_order_for_seller",
           "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_approved
           "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]

te = x_test[["payment_sequential","payment_installments","payment_value","seller_id","product_id","seller_
              "bs_share","cust_share",
          "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_descripti
           "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","delivery
              "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_of_
              "num_of_sellers_for_cust","total_order_for_seller",
           "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_approved
           "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]
```

In [12]:

```
norm = StandardScaler()

norm.fit(tr.values)

x_tr_num = norm.transform(tr.values)
x_te_num = norm.transform(te.values)

#save as pickle file
filename = "std_num_1.pkl"
pickle.dump(norm,open(filename,"wb"))
```

In [13]:

```
#horizontal stacking of all the features
train = hstack((x_tr_pay_type,x_tr_id,x_tr_cat,x_tr_num,x_tr_same_state,
                   x_tr_same_city,x_tr_late_shipping,x_tr_high_freight)).toarray()

test = hstack((x_te_pay_type,x_te_id,x_te_cat,x_te_num,x_te_same_state,
                 x_te_same_city,x_te_late_shipping,x_te_high_freight)).toarray()
```

In [14]:

```
#reset the index of target variable
y_trains = y_train.reset_index()
y_train = y_trains["binary_target"]

y_tests = y_test.reset_index()
y_test = y_tests["binary_target"]
```

Logistic Regression

In [15]:

```
best_param = 0.01
model = LogisticRegression(C=best_param,class_weight="balanced")
model.fit(train,y_train)
```

Out[15]:

```
LogisticRegression(C=0.01, class_weight='balanced')
```

In [23]:

```
#saving the logistic model as pickle file
filename = "binary_model.pkl"
pickle.dump(model,open(filename,"wb"))
```

In [ ]:

Custom Ensemble for (1,2,3,4)

In [25]:

```
#load the data with all created features
data = pd.read_csv("data_with_advanced_features.csv")
data.drop("Unnamed: 0", inplace=True, axis=1)

#label encoding of seller_id
label = LabelEncoder()
seller = label.fit_transform(data.seller_id)
data["seller_id"] = seller

filename = "seller_encode_2.pkl"
pickle.dump(label,open(filename,"wb"))

#label encoding of product id
label = LabelEncoder()
product = label.fit_transform(data.product_id)
data["product_id"] = product

filename = "product_encode_2.pkl"
pickle.dump(label,open(filename,"wb"))
```

In [26]:

```
data = data[data["review_score"]!=5]
Y = data["review_score"]
X = data

######### train test split with test size 25% and 75% of data as train ##############
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=10)
#########################################################################################################

######## payment_type ##########
vec = CountVectorizer()
vec.fit(x_train["payment_type"].values)
x_tr_pay_type = vec.transform(x_train.payment_type.values)
x_te_pay_type = vec.transform(x_test.payment_type.values)

# save as pickle file
filename = "countvec_pay_2.pkl"
pickle.dump(vec,open(filename,"wb"))


###### order_item_id ###########
x_train.order_item_id = x_train.order_item_id.astype(str)
x_test.order_item_id = x_test.order_item_id.astype(str)

vec = CountVectorizer(vocabulary=range(1,22))
vec.fit(x_train["order_item_id"])
x_tr_id = vec.transform(x_train.order_item_id)
x_te_id = vec.transform(x_test.order_item_id)

# save as pickle file
filename = "countvec_item_2.pkl"
pickle.dump(vec,open(filename,"wb"))

######### product_category_name ############
vec = CountVectorizer()
vec.fit(x_train["product_category_name"].values)
x_tr_cat = vec.transform(x_train.product_category_name.values)
x_te_cat = vec.transform(x_test.product_category_name.values)

# save as pickle file
filename = "countvec_cat_2.pkl"
pickle.dump(vec,open(filename,"wb"))

########## Binary features #####################
x_tr_same_state = x_train.same_state.values.reshape(-1,1)
x_te_same_state = x_test.same_state.values.reshape(-1,1)

x_tr_same_city = x_train.same_city.values.reshape(-1,1)
x_te_same_city = x_test.same_city.values.reshape(-1,1)

x_tr_late_shipping = x_train.late_shipping.values.reshape(-1,1)
x_te_late_shipping = x_test.late_shipping.values.reshape(-1,1)

x_tr_high_freight = x_train.high_freight.values.reshape(-1,1)
x_te_high_freight = x_test.high_freight.values.reshape(-1,1)

################################################################################
############# data to be standardized #########################################
tr = x_train[["payment_sequential","payment_installments","payment_value","seller_id","product_id","seller
              "bs_share","cust_share",
          "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_descripti
           "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","delivery
              "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_of_
              "num_of_sellers_for_cust","total_order_for_seller",
           "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_approved
           "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]

te = x_test[["payment_sequential","payment_installments","payment_value","seller_id","product_id","seller_
              "bs_share","cust_share",
          "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_descripti
           "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","delivery
              "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_of_
              "num_of_sellers_for_cust","total_order_for_seller",
           "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_approved
           "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]
```

```
norm = StandardScaler()

norm.fit(tr.values)

x_tr_num = norm.transform(tr.values)
x_te_num = norm.transform(te.values)

# save as pickle file
filename = "std_num_2.pkl"
pickle.dump(norm,open(filename,"wb"))
##########################################################################################################

#horizontal stacking of all the features
train = hstack((x_tr_pay_type,x_tr_id,x_tr_cat,x_tr_num,x_tr_same_state,
                   x_tr_same_city,x_tr_late_shipping,x_tr_high_freight)).toarray()

test = hstack((x_te_pay_type,x_te_id,x_te_cat,x_te_num,x_te_same_state,
                 x_te_same_city,x_te_late_shipping,x_te_high_freight)).toarray()

#reset the index of target variable
y_trains = y_train.reset_index()
y_train = y_trains["review_score"]

y_tests = y_test.reset_index()
y_test = y_tests["review_score"]
```

Custom ensemble with Logistic regression

In [27]:

```
def custom_ensemble(x_tr,y_tr,x_te,n_estimators,estimator,meta_clf):
    """This function creates the custom ensemble model and returns predicted  target variable of test set"

    ########### SPlitting train data into 50-50 as d1 and d2 ############
    kf = StratifiedKFold(n_splits=2)

    d1 = x_tr[list(kf.split(x_tr,y_tr))[1][0]]
    d1_y = y_tr[list(kf.split(x_tr,y_tr))[1][0]]

    d2 = x_tr[list(kf.split(x_tr,y_tr))[1][1]]
    d2_y = y_tr[list(kf.split(x_tr,y_tr))[1][1]]
    ######################################################################
    d1_y = np.array(d1_y)
    d2_y = np.array(d2_y)
    ######################################################################
    ### Creating base learners and training them using samples of d1 ####

    models=[]

    for i in range(n_estimators):
        ind = np.random.choice(19387,size=(20000),replace=True)
        sample = d1[ind]
        sample_y = d1_y[ind]

        estimator.fit(sample,sample_y)
        models.append(estimator)

    # save as pickle file
    filename="base_models.pkl"
    pickle.dump(models,open(filename,"wb"))
    ########### Predictions from base learners for d2 set ###############
    predictions = []
    for model in models:

        pred = model.predict(d2)
        predictions.append(pred)

    predictions = np.array(predictions).reshape(-1,n_estimators)

    ########## meta classifier on predictions of base learners ##########

    meta_clf.fit(predictions,d2_y)

    # save as pickle file
    filename="meta_clf.pkl"
    pickle.dump(meta_clf,open(filename,"wb"))
    ######################################################################
```

In [28]:

```
#training and saving the models with best hyperparameter n_estimator=150
best_n = 150
train_pred,test_pred,d2_y = custom_ensemble(train,y_train,test,best_n,LogisticRegression(class_weight="bal
                                            LogisticRegression(class_weight="balanced"))
```

In [ ]:

Hence We have saved all the objects and models that are necessary for further use/deployment.

We have selected the best performing model as of now.

In [ ]: