STATE FARM: CLASSIFICATION EXERCISE 40 - PYTHON CODE

BY FRANCK OLILO

Hello,

Thank you for checking my code submission of the take home assessment,

I really enjoy working on it even if the time was short.

In this first page you will see the libraries I used/ and needed to run my code, follows by the outline

```
import pandas as pd
import numpy as np
import os
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
import pickle
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFECV
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, \
    recall_scoreimport sys
!{sys.executable} -m pip install xgboost
```

OUTLINE

 # TRAINING DATA

#Before starting building a ML model, the first step is to import the data set.

#We will name our training data as “td”

```
#Import Training Data
td=pd.read_csv(r"C:\Users\olilo\Downloads\exercise_40_train.csv")
```

#EXPLORATORY DATA ANALYSIS

#Let have a look of our data by printing the first and last few rows

```
td.head() # first few row
td.tail() # last few row

td.info() # general info of the data
```

We can observe that our training data has 101 variables to to X100/columns and 40000 rows

12 columns in our data set are object, so we may have categorical variables and numerical columns

we may now start our data preparation since we don’t have any more information about the data

let see how balanced our data /variable y is

```
y_td.value_counts().plot(kind='bar')
```

our data set is really imbalanced , we come back on it while building our model

#DATA PREPARATION

Let start by checking 12 categorical columns and see if we need to clean up

```
category_td =td.loc[:,td.dtypes == object]
category_td.describe(include=[object])
```

#We can do more about seeing our unique each of them are :

```
td.unique()
```

x3 column seems to be some days but we have more than 7 different days , let check it closely

td['x3'].unique()

some of the days have different entries

for example "Wed" and "Wednesday"

let mask them, meaning have each date with the same attribute

```
day_mask1 = category_td['x3'] == 'Mon'
day_mask2 = category_td['x3'] == 'Tue'
day_mask3 = category_td['x3'] == 'Wed'
day_mask4 = category_td['x3']  == 'Thur'
day_mask5 = category_td['x3']  == 'Fri'
day_mask6 = category_td['x3']  == 'Sat'
day_mask7 = category_td['x3']  == 'Sun'
```

```
category_td['x3'][day_mask1] = 'Monday'
category_td['x3'][day_mask2] = 'Tuesday'
category_td['x3'][day_mask3] = 'wednesday'
category_td['x3'][day_mask4] = 'Thurday'
category_td['x3'][day_mask5] = 'Friday'
category_td['x3'][day_mask6] = 'Saturday'
category_td['x3'][day_mask7] = 'Sunday'
```

the next categorical column that need some edit are x7 and x19

#the percentage and dollar sign to be replaced respectively

```
category_td['x7'] = category_td['x7'].str.replace('%','').astype(float)
category_td['x19']category_td['x19'].str.replace('$','').astype(float)
```

#The last categorical columns that need some edit is x39 , which have the miles,

#however 5-10 miles is on all rows

so we can either delete the column or replace the value with 1

for future data : 0-5 mile : 0 , 5-10 miles : 1 , 10-15 miles : 2 etc...

```
category_td['x39'] = category_td['x39'].str.replace('5-10
miles','1').astype(float)
```

#Let set up a numerical variables category and move x7 x19 and x39 as they are not more object

```
#Seperate numerical variables
num_td = td.loc[:,td.dtypes == np.float64]
#add converted vars to numeric df
num_td['x7'] = category_td['x7']
num_td['x19'] = category_td['x19']
num_td['x39'] = category_td['x39']
category_td.drop(['x7','x19','x39'],axis = 1, inplace = True)
```

#The next part of our data preparation is to work on missing value

#We will use different technique for missing value imputation for categorical and numerical value

Let start with categorical variables

```
#Missing value in categorical columns
pd.options.display.max_rows=None
category_td.isnull().sum()
```

#We have more 30000 missing value across our categorical

variable x99 have missing value which I suspect to be NO , so I will replace the n/a by no

```
td['x99'] = td['x99'].fillna('no')
```

for the others categorical value I will replace the n/a by the most frequent one

```
for col in category_td:
    max_freq = category_td[col].value_counts().index[0]
    category_td[col][pd.isna(category_td[col])] = max_freq
```

Let now work on the numerical columns

```
#Missing value in numerical columns
pd.options.display.max_rows=None
num_td.isnull().sum()
```

#We have more than 50000 missing value in the numerical part

for the numerical part , I will use the inductive imputation to replace our missing value

```
#inductive imputation
imp = IterativeImputer()
imp_num_td = pd.DataFrame(imp.fit_transform(num_td))
imp_num_td.index = num_td.index
imp_num_td.columns = num_td.columns
```

Let import pickle and save our imputation

```
import pickle
with open('imp', 'wb') as i:
    pickle.dump(imp, i, pickle.HIGHEST_PROTOCOL)
```

The last part of our data preparation is to convert the categories into nominal interest

#Convert categories into nominal integers

```
# cree a dictionair who will receive the new dummies and concate it later to our
data set

le_dict = {}
label_td = pd.DataFrame()
for col in category_td:
    le = LabelEncoder()
    label_td[col] = le.fit_transform(category_td[col])
    le_dict[col] = le
```

Let save the label

```
with open('le_dict', 'wb') as l:
    pickle.dump(le_dict, l, pickle.HIGHEST_PROTOCOL)

#create df of binary cols representing instance of each category across
multipile columns
enc = OneHotEncoder()
onehot_td = pd.DataFrame(enc.fit_transform(label_td).toarray())
#feature_names = enc.get_feature_names()
#onehot_df.columns = feature_names
```

#Save one-hot-enc

```
with open('enc', 'wb') as e:
    pickle.dump(enc, e, pickle.HIGHEST_PROTOCOL)


scalar = StandardScaler()
scaled_num_td = pd.DataFrame(scalar.fit_transform(imp_num_td))


#create df of x vars from imputed df and encoded df
x_td = pd.concat([scaled_num_td, onehot_td], axis = 1)


with open('x_td', 'wb') as x:
    pickle.dump(x_td, x, pickle.HIGHEST_PROTOCOL)
```

#Seperate dependent 'y' binary variable

```
y_td = td['y'].astype('int')
```

#Save y var

```
with open('y_td', 'wb') as y:
    pickle.dump(y_td, y, pickle.HIGHEST_PROTOCOL)
```

 # TESTING DATA

#We will mostly follow the same concept as in the training data set so less comment will be made here

we will name our testing data as " tesda'

```
# Importing training data
tesda=pd.read_csv(r"C:\Users\olilo\Downloads\exercise_40_test.csv")
# let have an overview of the data by printing it
tesda

tesda.head()
tesda.tail()
```

#we have 10000 rows and 100 columns

we will use the save pickle function used in training data to convert into nominal our categorical

```
#Seperate numerical variables
num_tesda = tesda.loc[:,tesda.dtypes == np.float64]
#Seperate numerical variables
category_tesda =tesda.loc[:,tesda.dtypes == np.object]
category_tesda.describe(include=[np.object])
```

```
#Open models
with open('imp', 'rb') as i:
    imp = pickle.load(i)

with open('le_dict', 'rb') as l:
    le_dict = pickle.load(l)

with open('enc', 'rb') as e:
    enc = pickle.load(e)

with open('scalar', 'rb') as s:
    scalar = pickle.load(s)
```

```
category_tesda['x7'] = category_tesda['x7'].str.replace('%','').astype(float)
category_tesda['x19'] = category_tesda['x19'].str.replace('$','').astype(float)
category_tesda['x39'] = category_tesda['x39'].str.replace('5-10
miles','1').astype(float)
```

```
num_tesda['x7'] = category_tesda['x7']
num_tesda['x19'] = category_tesda['x19']
num_tesda['x39'] = category_tesda['x39']
category_tesda.drop(['x7','x19','x39'],axis = 1, inplace = True)
```

```
day_mask1 = category_tesda['x3'] == 'Mon'
day_mask2 = category_tesda['x3'] == 'Tue'
```

```
day_mask3 = category_tesda['x3'] == 'Wed'
day_mask4 = category_tesda['x3’] == 'Thur'
day_mask5 = category_tesda['x3']  == 'Fri'
day_mask6 = category_tesda['x3']  == 'Sat'
day_mask7 = category_tesda['x3']  == 'Sun'
```

```
category_tesda['x3'][day_mask1] = 'Monday'
category_tesda['x3'][day_mask2] = 'Tuesday'
category_tesda['x3'][day_mask3] = 'wednesday'
category_tesda['x3'][day_mask4] = 'Thurday'
category_tesda['x3'][day_mask5] = 'Friday'
category_tesda['x3'][day_mask6] = 'Saturday'
category_tesda['x3'][day_mask7] = 'Sunday'
```

```
#inductive imputation
imp = IterativeImputer()
imp_num_tesda = pd.DataFrame(imp.fit_transform(num_tesda))
imp_num_tesda.index = num_tesda.index
imp_num_tesda.columns = num_tesda.columns
```

#impute missing categorical vars as most frequent in category except x99

```
tesda['x99'] = tesda['x99'].fillna('no')
for col in category_tesda:
    max_freq = category_tesda[col].value_counts().index[0]
    category_tesda[col][pd.isna(category_tesda[col])] = max_freq
```

#convert categories into nominal integers

```
label_tesda = pd.DataFrame()
for col in category_tesda:
    le = le_dict[col]
    label_tesda[col] = le.transform(category_tesda[col])
```

#create df of binary cols representing instance of each category across multipile columns

```
onehot_tesda = pd.DataFrame(enc.transform(label_tesda).toarray())
```

```
scaled_num_tesda = pd.DataFrame(scalar.transform(imp_num_tesda))
x_tesda = pd.concat([scaled_num_tesda, onehot_tesda], axis = 1)
#save x vars
with open('x_tesda_test', 'wb') as x:
    pickle.dump(x_tesda, x, pickle.HIGHEST_PROTOCOL)
```

#MODELS DEVELOPMENT

#We are planning to build two models, one logistic regression and gradient boost machine xgboosr

#We saw in data preparation that our data was unbalanced , so I decided to build our two models using the #unbalanced at first and balanced as second , which will help me decide later on which on the model is a good choice

#Logistic regression under Unbalanced data (with and without feature selection)

```
#open our save data x's and y with pickle

#open x abd y df's
with open('x_td', 'rb') as x:
    x_td = pickle.load(x)

with open('y_td', 'rb') as y:
    y_td = pickle.load(y)

convert y as array
x_td=np.array(x_td)
```

#split x and y data into training 80% and test 20@ set

```
X_train, X_test, y_train, y_test = train_test_split(x_td,
                                    y_td,
                                    test_size=0.2,
                                    random_state=1)
```

#Since we need to use Auc as metric evaluation, however I will add to some model accuracy, precision and recall

```
score = 'roc_auc'


from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, \
    recall_score
```

#logistic regression without feature selection

```
classifier = LogisticRegression(random_state = 0,max_iter=10000)
classifier = LogisticRegression(random_state = 0,max_iter=10000)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

Score

```
print('AUC: ', roc_auc_score(y_test, y_pred))

print ("Accuracy : ", accuracy_score(y_test, y_pred))
print('Precision: ', precision_score(y_test, y_pred))
print('Recall: ', recall_score(y_test, y_pred))
```

```
AUC:  0.5575721177443657
Accuracy:  0.85975
```

#Setup recursive feature reduction w/ cross validation

```
clf2 = RFECV(LogisticRegression(max_iter=10000),
      scoring = score,
      n_jobs = -1,
      cv = 3,
      step = 5)

clf2.fit(X_train, y_train)
```

#Generate predicted probabilites

```
clf2_probs = clf2.predict_proba(X_test)
print('AUC: ', roc_auc_score(y_test, clf2_probs[:,1]))
print('Accuracy: ', clf2.score(X_test, y_test))
```

```
AUC:  0.7742262661979552
Accuracy:  0.8595
```

#XGBOOST UNDER UNBALANCED

#import xgboost and set up parameters

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

param = {
'objective': 'multi:softprob',
'eta': 0.4,
'max_depth': 3,
'gamma': 1.0,
'num_class': 3 }
```

#set up xgboost

```
model = xgb.train(param, dtrain, 20)
y_pred = model.predict(dtest)
predicts = np.asarray([np.argmax(val) for val in y_pred])
```

score

```
acc = accuracy_score(y_test, predicts)
print('Accuracy = {}'.format(acc))
print('AUC: ', roc_auc_score(y_test, predicts))
```

```
Accuracy = 0.86225
AUC:  0.5447476615189291
```

#Now that we built both model with the unbalanced data

#let build the same model with balanced data

We are going to balanced the data using the function smote , which we oversample our data

```
sm = SMOTE(random_state=42)
x_td=np.array(x_td)
X_res, y_res = sm.fit_resample(x_td, y_td)

#double check

y_res.value_counts().plot(kind='bar')
```

#save resample y var

```
with open('y_res', 'wb') as y:
    pickle.dump(y_res, y, pickle.HIGHEST_PROTOCOL)
```

#Save resample x var

```
with open('X_res', 'wb') as x:
    pickle.dump(X_res, x, pickle.HIGHEST_PROTOCOL)
```

#Load both x_res and y_res into new notbook

```
with open('X_res', 'rb') as x:
    X_res = pickle.load(x)
```

```
with open('y_res', 'rb') as y:
    y_res = pickle.load(y)
```

#slipt resample data as training and testing

```
X_train, X_test, y_train, y_test = train_test_split(X_res,
                                    y_res,
                                    test_size=0.2,
                                    random_state=1)
```

#logistic regression under balanced data without feature selection

```
classifier = LogisticRegression(random_state = 0,max_iter=10000)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

#score

```
print('AUC: ', roc_auc_score(y_test, y_pred))
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

```
AUC:  0.72239926445074
Accuracy :  0.7221288105855691
```

#Setup recursive feature reduction w/ cross validation

```
clf2 = RFECV(LogisticRegression(max_iter=10000),
      scoring = score,
      n_jobs = -1,
      cv = 3,
      step = 5)

clf2.fit(X_train, y_train)
```

#Generate predicted probabilities and score

```
clf2_probs = clf2.predict_proba(X_test)
print('AUC: ', roc_auc_score(y_test, clf2_probs[:,1]))
print('Accuracy: ', clf2.score(X_test, y_test))
```

#XGBOOST MODEL UNDER BALANCED DATA

#set up parameters

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
param = {
'objective': 'multi:softprob',
'eta': 0.4,
'max_depth': 3,
'gamma': 1.0,
'num_class': 3 }
```

Set up model

```
model = xgb.train(param, dtrain, 100)
y_pred = model.predict(dtest)
predicts = np.asarray([np.argmax(val) for val in y_pred])
```

#Score

```
acc = accuracy_score(y_test, predicts)
print('Accuracy = {}'.format(acc))
print('AUC: ', roc_auc_score(y_test, predicts))
```

```
AUC:  0.9175363632301481
```

MODEL VALIDATION

#Open both models and the test data

```
with open('model', 'rb') as c:
    model = pickle.load(c)

with open('clf2', 'rb') as c2:
    clf2 = pickle.load(c2)

with open('x_tesda_test', 'rb') as x:
    x_tesda = pickle.load(x)

x_tesda=np.array(x_tesda)

C#convert numpy into Dmatrix
x_tesd = xgb.DMatrix(x_tesda)
```

#Save the 10000 predictions of both model

#10000 prediction of glmresults

```
clf2_probs = clf2.predict_proba(x_tesda)
np.savetxt("glmresults.csv", clf2_probs[:,1], delimiter=",") # logistic regression
prediction
```

#10000 prediction of nonglmresults

```
clf_probs = model.predict(x_tesd)
np.savetxt("nonglmresults.csv", clf_probs[:,1], delimiter=",") # xgboost prediction
```