```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr  3 14:41:43 2019

@author: Amirhossein Forouzani
"""
from sklearn.preprocessing import Imputer
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import sklearn
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
import random
import math
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
import csv
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
from IPython.core.interactiveshell import InteractiveShell
from sklearn.svm import SVC
from sklearn.base import BaseEstimator, TransformerMixin
import seaborn as sns
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE
#-----------------------------------------------
#MSE_binary classifier using linear regression
class MSE_binary ( LinearRegression ) :
    def __init__ ( self ) :
        print ( " Calling newly created MSE binary function . . . " )
        super (MSE_binary , self ). __init__ ( )
    def predict ( self , X) :
        thr = 0.5 # may vary depending on how you defineb in Xw = b
        y = self._decision_function(X)
        y_binary = (np.zeros(y.shape)).astype(int)
        y_binary [y>thr] = 1
        return y_binary


#-----------------------------------------------------------
# costum encoder
def number_encode_features(df):
    result = df.copy()
    encoders = {}
    for column in result.columns:
```

```
        if result.dtypes[column] == np.object:
            result[column].str.rstrip()
            result[column].str.lstrip()
            encoders[column] = preprocessing.LabelEncoder()
            result[column] = encoders[column].fit_transform(result[column

    return result, encoders
#--------------------------------------------------
#Costum Imputer
class ImputeCategorical(BaseEstimator, TransformerMixin):
    """
    Encodes a specified list of columns or all columns if None.
    """
    def __init__(self, columns=None):
        self.columns = columns
        self.imputer = None
    def fit(self, data, target=None):
        """
        Expects a data frame with named columns to impute.
        """
        # Encode all columns if columns is None
        if self.columns is None:
            self.columns = data.columns
        # Fit an imputer for each column in the data frame
        self.imputer = Imputer(missing_values=0, strategy='most_frequent'
        self.imputer.fit(data[self.columns])
        return self
    def transform(self, data):
        """
        Uses the encoders to transform a data frame.
        """
        output = data.copy()
        output[self.columns] = self.imputer.transform(output[self.columns
        return output
#--------------------------------------------------
def resamplingdata_downsample(x_train, y_train):
    X = pd.concat([x_train, y_train], axis=1)

# separate minority and majority classes
    not_fraud = X[X.label==0]
    fraud = X[X.label==1]

# upsample minority
    fraud_upsampled = resample(fraud,
                               replace=True, # sample with replacement
                               n_samples=len(not_fraud), # match number in maj
                               random_state=27) # reproducible results

# combine majority and upsampled minority
    upsampled = pd.concat([not_fraud, fraud_upsampled])
```

```
# check new class counts

    print(upsampled.label.value_counts())
    xtrain  = X.drop(['label'], axis  = 1)
    y_train  = X['label']
    return x_train, y_train
#-----------------------------------------------
def resample_smote(X_train,y_train):
    sm = SMOTE(random_state=27, ratio=1.0)
    X_train, y_train = sm.fit_sample(X_train, y_train)
    return X_train, y_train

#-----------------------------------------------
#File Importer/Imputer/Encoder
#inputs: File Name
#outputs:
def import_file (data_name):
    dataset_name = data_name
    df_train = pd.read_csv(dataset_name + ".train_SMALLER.csv")
    df_test = pd.read_csv(dataset_name + "_test.csv")
    df_train.columns = ["Age", "Workclass", "fnlwgt", "Education", "Educa
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Cap
        "Hours per week", "Country", "label"]
    df_test.columns = ["Age", "Workclass", "fnlwgt", "Education", "Educat
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Cap
        "Hours per week", "Country", "label"]
    encoded_train,encoders_train = number_encode_features(df_train)
    imputer = ImputeCategorical(['Workclass', 'Country', 'Occupation'])
    encoded_train = imputer.fit_transform(encoded_train)

    encoded_test, encoders_test = number_encode_features(df_test)
    imputer = ImputeCategorical(['Workclass', 'Country', 'Occupation'])
    encoded_test = imputer.fit_transform(encoded_test)

    y_train = encoded_train['label']
    x_train = encoded_train.drop(['label'], axis = 1)
    y_test = encoded_test['label']
    x_test = encoded_test.drop(['label'], axis = 1)
    return x_train, y_train, x_test, y_test, encoded_train, encoded_test,
#scaler = StandardScaler().fit(x_train)
#-----------------------------------------------------------
def distribution_finder (data_name):
    og_data = pd.read_csv(data_name + ".train_SMALLER.csv")
    og_data.columns = ["Age", "Workclass", "fnlwgt", "Education", "Educat
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Cap
        "Hours per week", "Country", "label"]
    encoded_train,encoders_train = number_encode_features(og_data)
    imputer = ImputeCategorical(['Workclass', 'Country', 'Occupation'])
    encoded_train = imputer.fit_transform(encoded_train)
    print (encoded_train)
```

```
    fig = plt.figure(figsize=(20,15))
    cols = 5
    rows = math.ceil(float(encoded_train.shape[1]) / cols)
    for i, column in enumerate(encoded_train.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if  encoded_train.dtypes[column] == np.object:
            encoded_train[column].value_counts().plot(kind="bar", axes=ax
        else:
            encoded_train[column].hist(axes=ax)
            plt.xticks(rotation="vertical")
        plt.subplots_adjust(hspace=0.7, wspace=0.2)

#-----------------------------------------------------------
def frequency_finder (data_name,frame_name):
    og_data = pd.read_csv(data_name + ".train_SMALLER.csv")
    og_data.columns = ["Age", "Workclass", "fnlwgt", "Education", "Educat
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Cap
        "Hours per week", "Country", "label"]
    og_data.head()
    f, axes = plt.subplots(1, 1, figsize=(7, 7), sharex=True)
    sns.countplot(y = frame_name, hue='label', data=og_data,)
#-----------------------------------------------------------
def corellation_ploter(data):
    a,b,c,d,e,f = import_file(data)
    sns.heatmap(e.corr(), square=True)
    plt.show()
#-----------------------------------------------------------
def per_rec_acc(tn, fp, fn, tp):
    recall = tp/(tp+fp)
    persicion = tp/(tp+fn)
    total = tn+fp+fn+tp
    acc = (tp+tn)/total
    return recall,persicion,acc
```