```
# -*- 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 majority class
                          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", "Education-Num",
"Martial Status",
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
        "Hours per week", "Country", "label"]
    df_test.columns = ["Age", "Workclass", "fnlwgt", "Education", "Education-Num",
"Martial Status",
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
        "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, encoders_train,
encoders_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", "Education-Num",
"Martial Status",
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
        "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", "Education-Num",
"Martial Status",
        "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
        "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



# -*- coding: utf-8 -*-
"""
Created on Wed Apr  3 14:03:16 2019

@author: Amirhossein Forouzani
"""

import sklearn.metrics as metrics
import sklearn as skl
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
import random
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 sklearn import linear_model
from IPython.core.interactiveshell import InteractiveShell
import sklearn.preprocessing as preprocessing
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import KernelPCA
from sklearn.neural_network import MLPClassifier
#from imblearn.over_sampling import SMOTE
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.ensemble import (RandomTreesEmbedding, RandomForestClassifier,
                              GradientBoostingClassifier)
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
#------------------------------------------------------------------------
#the import function would encode and impute the data attributes
x_train, y_train, x_test, y_test, encoded_train,
encoded_test ,encoders_train,encoders_test= import_file("adult")
og_x_train = x_train
og_x_test = x_test
og_y_train = y_train
og_y_test = y_test
distribution_finder("adult")
x_train, y_train = resamplingdata_downsample(x_train,y_train)
#x_train, y_train = resample_smote(x_train,y_train)
#x_test, y_test = resample_smote(x_test,y_test)
'''
scalar =preprocessing.StandardScaler(with_std=True)
x_train  = scalar.fit_transform(x_train)
x_test = scalar.fit_transform(x_test)
'''
frequency_finder("adult", "Occupation")
#impute and encode

#Scaling the data
'''
scalar =preprocessing.StandardScaler()
x_train  = scalar.fit_transform(x_train)
x_test = scalar.fit_transform(x_test)

'''

# using dummy variables to encode the data

binary_data_train = pd.get_dummies(x_train)
binary_data2_test = pd.get_dummies(x_test)


scalar =preprocessing.StandardScaler(with_std=True)
x_train  = pd.DataFrame(scalar.fit_transform(x_train))
x_test = pd.DataFrame(scalar.fit_transform(x_test))

#fdeature dimesnasio reduction for future use
```

```
# running the linear regression model on the data set(F1 score: 0.536377)

cls = linear_model.LogisticRegression(solver='lbfgs', max_iter=1000)

cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)


fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy for LogisticRegression: %f"% roc_auc)
print ("F1 score for Logistic Regression: %f" % skl.metrics.f1_score(y_test, y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='LR , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")
#print(cross_val_score(cls, x_train, y_train, cv=8))

coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
coefs.sort_values()
ax = plt.subplot(2,1,2)
coefs.plot(kind="bar")

plt.show()
#-----------------------------------------------------------------


X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.5)
```

```
# It is important to train the ensemble of trees on a different subset
# of the training data than the linear regression model to avoid
# overfitting, in particular if the total number of leaves is
# similar to the number of training samples
X_train, X_train_lr, Y_train, Y_train_lr = train_test_split(
   X_train, Y_train, test_size=0.5)

rt = RandomTreesEmbedding(max_depth=3, n_estimators=10,
                          random_state=0)
rt_lm = LogisticRegression(solver='lbfgs', max_iter=1000)
pipeline = make_pipeline(rt, rt_lm)
pipeline.fit(X_train, Y_train)
y_pred_rt = pipeline.predict_proba(X_test)[:, 1]

y_pred = pipeline.predict(X_test)
print ("F1 score for Logistic embedded trees: %f" % skl.metrics.f1_score(Y_test,
y_pred))
trainerror = accuracy_score ( Y_test ,y_pred )
print ("Accuracy Logistic embedded trees: ",trainerror )
cm = metrics.confusion_matrix(Y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(Y_test, y_pred).ravel()
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")
plt.show()

fpr_rt_lm, tpr_rt_lm, _ = roc_curve(Y_test, y_pred_rt)
roc_auc = auc(fpr_rt_lm, tpr_rt_lm)
#print ("F1 score for Linear Regression: %f" % skl.metrics.f1_score(Y_test, y_pred_rt))
#ax = plt.subplot(2,1,1)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rt_lm, tpr_rt_lm, label='RT + LR(area = %0.2f)'% roc_auc)
#plt.plot(fpr_rf_lm, tpr_rf_lm, label='RF + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()


#-----------------------------
```

```
#
#Now we can trey to classify using perceptron
cls = OneVsRestClassifier(Perceptron(tol=1e-3, random_state=0))

cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy for Perceptron: %f"% roc_auc)
print ("F1 score for Perceptron using One vurses rest classifier: %f" %
skl.metrics.f1_score(y_test, y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("train error is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='Perceptron , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
coefs.sort_values()
ax = plt.subplot(2,1,2)
coefs.plot(kind="bar")
plt.show()
#now we can run the MSE_binary Using One Vs. Rest Classifier
binary_model = MSE_binary ( )
mc_model = OneVsRestClassifier (binary_model)
mc_model.fit(x_train, y_train)
y_pred = mc_model.predict(x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy for MSE_binary LR: %f"% roc_auc)
print ("F1 score for MSE_binary with linear Regression is : %f" %
skl.metrics.f1_score(y_test, y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(tpr_rf_lm, tpr_rf_lm, label='MSE Binary , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
coefs.sort_values()
ax = plt.subplot(2,1,2)
coefs.plot(kind="bar")
plt.show()



# Now we can classify iusing support vector machines(F1 Score: 0.228)
cls = SVC(kernel ='rbf', C = 50, gamma = 5)
cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy forr SVC RBF is: %f"% roc_auc)
print ("F1 score for SVM with RBF Kernel: %f" % skl.metrics.f1_score(y_test, y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='RBF SVC , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

#coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
#coefs.sort_values()
#ax = plt.subplot(2,1,2)
#coefs.plot(kind="bar")
plt.show()




#SVM with Sigmoid Kernel(F1 Score: 36%)
cls = SVC(kernel ='sigmoid', C = 50, gamma = 5)
cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy for Sigmoid is: %f"% roc_auc)
print ("F1 score for SVM with sigmoid Kernel: %f" % skl.metrics.f1_score(y_test,
y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, perision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",perision)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='SVC Sigmoid , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

#coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
#coefs.sort_values()
#ax = plt.subplot(2,1,2)
#coefs.plot(kind="bar")
plt.show()


cls  = MultinomialNB()
cls.fit(og_x_train, og_y_train)
y_pred = cls.predict(og_x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy for NB %f"% roc_auc)
print ("F1 score for Naive Bayes: %f" % skl.metrics.f1_score(og_y_test, y_pred))
trainerror = accuracy_score ( og_y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='MultinomialNB , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

#coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
#coefs.sort_values()
#ax = plt.subplot(2,1,2)
```

```
#coefs.plot(kind="bar")

#X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.5)

# It is important to train the ensemble of trees on a different subset
# of the training data than the linear regression model to avoid
# overfitting, in particular if the total number of leaves is
# similar to the number of training samples
#X_train, X_train_lr, Y_train, Y_train_lr = train_test_split(
#   X_train, Y_train, test_size=0.5)

#rt = RandomTreesEmbedding(max_depth=3, n_estimators=10,
#                          random_state=0)
#rt_lm = LogisticRegression(solver='lbfgs', max_iter=1000)
#pipeline = make_pipeline(rt, rt_lm)
#pipeline.fit(X_train, Y_train)
#y_pred_rt = pipeline.predict_proba(X_test)[:, 1]


plt.show()


# Now we can Use k nearest neighbours classifier Select From Model Feature
Reduction Technique(60%)

lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(x_train, y_train)
model = SelectFromModel(lsvc, prefit=True)
xtrain_new = model.transform(x_train)
#x_test_new = x_test[xtrain_new.columns]

'''
lsvc1 = LinearSVC(C=0.01, penalty="l1", dual=False).fit(x_test, y_test)
model1 = SelectFromModel(lsvc1, prefit=True)
xtest_new = model1.transform(x_test)
'''
cls  = KNeighborsClassifier(n_neighbors=3, algorithm = 'ball_tree')
#print (x_train)
cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print("AUC Accuracy for KNN is : %f"% roc_auc)
print ("F1 score for K nearest Neighbours: %f" % skl.metrics.f1_score(y_test, y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
```

```
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='KNN , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

#coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
#coefs.sort_values()
#ax = plt.subplot(2,1,2)
#coefs.plot(kind="bar")
plt.show()


#MUlti layer nueral netweok classification with back propagation(64%)

cls  = MLPClassifier(solver='lbfgs', alpha=1e-7,hidden_layer_sizes=(15, 5),
random_state=1)
#print (x_train)
cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
print ("F1 score for ANN: %f" % skl.metrics.f1_score(y_test, y_pred))
trainerror = accuracy_score ( y_test ,y_pred )
print ("Accuracy is: ",trainerror )
print("AUC Accuracy for ANN is : %f"% roc_auc)
cm = metrics.confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='MPL(ANN) , (area = %0.2f)'% roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")

#coefs = pd.Series(cls.coef_[0], index=encoded_train.drop(['label'],axis = 1).columns)
#coefs.sort_values()
#ax = plt.subplot(2,1,2)
#coefs.plot(kind="bar")
plt.show()
X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.5)

# It is important to train the ensemble of trees on a different subset
# of the training data than the linear regression model to avoid
# overfitting, in particular if the total number of leaves is
# similar to the number of training samples
X_train, X_train_lr, Y_train, Y_train_lr = train_test_split(
   X_train, Y_train, test_size=0.5)

rt = RandomTreesEmbedding(max_depth=3, n_estimators=10,
                          random_state=0)
rt_lm = MLPClassifier(solver='lbfgs', alpha=1e-7,hidden_layer_sizes=(15, 5),
random_state=1)
pipeline = make_pipeline(rt, rt_lm)
pipeline.fit(X_train, Y_train)
y_pred_rt = pipeline.predict_proba(X_test)[:, 1]
y_pred = pipeline.predict(X_test)
print ("F1 score for MLP embedded trees: %f" % skl.metrics.f1_score(Y_test, y_pred))
trainerror = accuracy_score ( Y_test ,y_pred )
print ("Accuracy mlp embedded trees: ",trainerror )
cm = metrics.confusion_matrix(Y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(Y_test, y_pred).ravel()
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")
plt.show()
```

```
fpr_rt_lm, tpr_rt_lm, _ = roc_curve(Y_test, y_pred_rt)
roc_auc = auc(fpr_rt_lm, tpr_rt_lm)

plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rt_lm, tpr_rt_lm, label='RT + MPL(area = %0.2f)'% roc_auc)
#plt.plot(fpr_rf_lm, tpr_rf_lm, label='RF + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')

plt.show()

 #random forest With Logistic Regeression


X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size=0.5)

# It is important to train the ensemble of trees on a different subset
# of the training data than the linear regression model to avoid
# overfitting, in particular if the total number of leaves is
# similar to the number of training samples
X_train, X_train_lr, Y_train, Y_train_lr = train_test_split(
   X_train, Y_train, test_size=0.5)


cls = RandomForestClassifier(n_estimators=10, max_depth=3,
                             random_state=0)
rf_enc = OneHotEncoder(categories='auto')
rf_lm = LogisticRegression(solver='lbfgs', max_iter=1000)
cls.fit(X_train, Y_train)
rf_enc.fit(cls.apply(X_train))
rf_lm.fit(rf_enc.transform(cls.apply(X_train_lr)), Y_train_lr)

y_pred_rf_lm = rf_lm.predict_proba(rf_enc.transform(cls.apply(X_test)))[:, 1]
y_pred = rf_lm.predict(rf_enc.transform(cls.apply(X_test)))
print ("F1 score for random forest logistic reg: %f" % skl.metrics.f1_score(Y_test,
y_pred))
trainerror = accuracy_score ( Y_test ,y_pred )
print ("Accuracy is: ",trainerror )
cm = metrics.confusion_matrix(Y_test, y_pred)
tn, fp, fn, tp = metrics.confusion_matrix(Y_test, y_pred).ravel()
recall, percision,acc = per_rec_acc(tn, fp, fn, tp)
print ("recall is:",recall, "persicion is:",percision)
print ("tn:",tn,"fp:", fp,"fn: " ,fn,"tp:", tp)
```

```
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d", xticklabels=encoders_train["label"].classes_,
yticklabels=encoders_train["label"].classes_)
plt.ylabel("Real value")
plt.xlabel("Predicted value")
plt.show()
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(Y_test, y_pred_rf_lm)
roc_auc = auc(fpr_rf_lm, tpr_rf_lm)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='RF + LR (area = %0.2f)'% roc_auc)

plt.title('ROC curve')

plt.legend(loc='best')


plt.show()
```