PROJECT CODE:
 # Author: Akshata Bhat
 # USCID: 9560895350

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn import linear_model
from sklearn.feature_selection import VarianceThreshold
import matplotlib.pyplot as plt
from imblearn.combine import SMOTEENN
from itertools import *


def load_dataset_from_file():
  input_file = '/home/ak/Downloads/bank-additional.csv'
  dataset = pd.read_csv(input_file, delimiter=',')
  return dataset


# Preprocessing data - Replacing all the categorical strings with numerical values
def preprocessing_data(data):
  data.job.replace(('unknown', 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management',
          'retired', 'self-employed', 'services', 'student', 'technician',
          'unemployed'),(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11), inplace=True)  # 11
  data.marital.replace(('unknown', 'divorced', 'married', 'single'), (0, 1, 2, 3), inplace=True) # 3
  data.education.replace(('unknown', 'illiterate', 'basic.4y', 'basic.6y', 'basic.9y', 'high.school',
                 'university.degree', 'professional.course'), (0, 1, 2, 3, 4, 5, 6, 7), inplace=True)  # 7
  data.housing.replace(('unknown', 'yes', 'no'), (0, 1, 2),inplace=True) # 2
  data.loan.replace(('unknown', 'yes', 'no'), (0, 1, 2),inplace=True) # 2
```

```
  data.contact.replace(('cellular', 'telephone'), (1, 0),inplace=True) # 2
  data.month.replace(('jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec'), (0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11), inplace=True) # 12
  data.day_of_week.replace(('mon', 'tue', 'wed', 'thu', 'fri'), (0, 1, 2, 3, 4), inplace=True) # 5
  data.poutcome.replace(('nonexistent', 'success', 'failure'), (0, 1, 2), inplace=True) #3
  return data


# Replacing the string labels with numerical values
def preprocessing_labels(labels):
  labels.replace(('no', 'yes'), (0, 1), inplace=True)
  return labels


# Use most frequent method to impute the unknown values in the dataset
def imputing_data(data):
  impute_func = Imputer(missing_values=0, strategy='most_frequent', axis=0)
  data.loc[:, 'job'] = impute_func.fit_transform(data['job'].values.reshape(-1, 1))
  data.loc[:, 'marital'] = impute_func.fit_transform(data['marital'].values.reshape(-1, 1))
  data.loc[:, 'education'] = impute_func.fit_transform(data['education'].values.reshape(-1, 1))
  data.loc[:, 'housing'] = impute_func.fit_transform(data['housing'].values.reshape(-1, 1))
  data.loc[:, 'loan'] = impute_func.fit_transform(data['loan'].values.reshape(-1, 1))
  return data


# one hot encoding
def one_hot_encoding(data):
  onehot_v1 = pd.get_dummies(data.loc[:, 'contact'], prefix='contact')
  data = data.join(onehot_v1)
  data = data.drop('contact', axis=1)

  onehot_v1 = pd.get_dummies(data.loc[:, 'poutcome'], prefix='poutcome')
  data = data.join(onehot_v1)
  data = data.drop('poutcome', axis=1)

  onehot_v2 = pd.get_dummies(data.loc[:, 'job'],  prefix='job')
  data = data.join(onehot_v2)
  data = data.drop('job', axis=1)

  onehot_v3 = pd.get_dummies(data.loc[:, 'marital'],  prefix='marital')
  data = data.join(onehot_v3)
  data = data.drop('marital', axis=1)
```

```
    onehot_v4 = pd.get_dummies(data.loc[:, 'housing'],  prefix='housing')
    data = data.join(onehot_v4)
    data = data.drop('housing', axis=1)

    onehot_v6 = pd.get_dummies(data.loc[:, 'education'],  prefix='education')
    data = data.join(onehot_v6)
    data = data.drop('education', axis=1)

    onehot_v5 = pd.get_dummies(data.loc[:, 'loan'],  prefix='loan')
    data = data.join(onehot_v5)
    data = data.drop('loan', axis=1)

    onehot_v7 = pd.get_dummies(data.loc[:, 'month'],  prefix='month')
    data = data.join(onehot_v7)
    data = data.drop('month', axis=1)

    onehot_v8 = pd.get_dummies(data.loc[:, 'day_of_week'],  prefix='day_of_week')
    data = data.join(onehot_v8)
    data = data.drop('day_of_week', axis=1)

    data = data.drop('default', axis=1)
    data = data.drop('pdays', axis=1)
    return data


# Calculating the mean and variance parameters and then standardizing the data
def standardizing_data(X_train, X_test):
    # standardization
    scaler = MinMaxScaler()
    scaler.fit(X_train)  # Obtain the mean and variance parameter values from the training data
    X_train = scaler.transform(X_train)  # Standardize the training data using the values obtained 
earlier
    X_test = scaler.transform(X_test)  # Standardize the testing data using the values obtained 
earlier
    return X_train, X_test


# Performing classification using cross validation on different data
def run_classifier(cf_name, cf_model, parameters, X_train, y_train, X_test, y_test):
    print("Performing classification using ", cf_name, " ...")
    label_names = ['yes', 'no']

    #Performing Cross Validation
```

```
    CV_type = GridSearchCV(estimator=cf_model, param_grid=parameters, cv=5)
    CV_type.fit(X_train, y_train)
    print(np.shape(X_train))
    print(np.shape(y_train))

    y_pred = CV_type.predict(X_train)
    print(np.shape(y_pred))
    print(' ')
    print('---------' + cf_name + ' Classifier--------')
    print('Training data- Actual Label size: ', np.shape(y_train))
    print('Training data- Predicted Label size: ', np.shape(y_pred))
    print(cf_name + " Train Accuracy: {0:.3f}".format(float((y_pred == y_train).sum()) /
float(len(y_train))))

    y_pred = CV_type.predict(X_test)
    AUC_score = roc_auc_score(y_test, y_pred)
    print(cf_name + " Test Accuracy: {0:.3f}".format(float((y_pred == y_test).sum()) /
float(len(y_test))))
    print(cf_name + " F1 Score: {0:.3f}".format(f1_score(y_test, y_pred, average='weighted')))
    print(cf_name + " AUC Score: {0:.3f}".format(AUC_score))
    conf = confusion_matrix(y_test, y_pred).ravel()
    print("Classification Report scores: ")
    print(classification_report(y_test, y_pred, target_names=label_names))
    return conf, y_pred, y_test, AUC_score


def calculate_plot_roc_curve(y_test, y_pred):
    fpr, tpr, threshold = roc_curve(y_test, y_pred)
    plt.figure()
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label='AUC = %0.3f' % roc_auc_score(y_test, y_pred))
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'g--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
    return


def obtain_plots(conf_matrix, cf_name, y_pred, y_test, AUC_score):
    plt.figure()
```

```
    plt.title(cf_name + ' :Confusion Matrix')
    conf = np.reshape(conf_matrix, (2, 2))
    plt.imshow(conf, cmap=plt.cm.Blues, interpolation='nearest')
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, ['Predicted No', 'Predicted Yes'])
    plt.yticks(tick_marks, ['Actual No', 'Actual Yes'], rotation='vertical')
    thresh = conf.max() / 2.
    for i, j in product(range(conf.shape[0]), range(conf.shape[1])):
        plt.text(j, i, conf[i, j], horizontalalignment="center", color="white" if conf[i, j] > thresh else
"black")
    plt.show()
    calculate_plot_roc_curve(y_test, y_pred)
    return
# -----------------------------------------------------------------------------------------------------------------------
# -----------------------------------------------------------------------------------------------------------------------

# Reading the file into a data frame
dataset = load_dataset_from_file()

# Dropping the row with education as illiterate,since it leads to the data being divided unevenly
dataset = dataset[dataset.education.str.contains("illiterate") == False]
# Separating the labels and data from the dataset
y = dataset['y']
X = dataset.drop('y', axis=1)

# splitting data into train and test - train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
print("Training data shape:")
print("X_train: ", np.shape(X_train))
print("y_train: ", np.shape(y_train))
print("Test data shape: ")
print("X_test: ", np.shape(X_test))
print("y_test: ", np.shape(y_test))


# Preprocessing the Training data
X_train = preprocessing_data(X_train)
y_train = preprocessing_labels(y_train)
X_train = imputing_data(X_train)
X_train = one_hot_encoding(X_train)

# Preprocessing the Testing data
```

```
X_test = preprocessing_data(X_test)
y_test = preprocessing_labels(y_test)
X_test = imputing_data(X_test)
X_test = one_hot_encoding(X_test)

print("Training data shape after preprocessing:")
print("X_train: ", np.shape(X_train))
print("y_train: ", np.shape(y_train))
print("Test data shape after preprocessing: ")
print("X_test: ", np.shape(X_test))
print("y_test: ", np.shape(y_test))

#standardization
X_train, X_test = standardizing_data(X_train, X_test)

# Performing oversampling using SMOTE and the cleaning up the extra noise using ENN
smote_and_enn=SMOTEENN(random_state=0)
X_oversampled,y_oversampled=smote_and_enn.fit_sample(X_train,y_train)
X_train = X_oversampled
y_train = y_oversampled
print(np.shape(X_train))
print(np.shape(y_train))

# try different classifiers
classifier_names=["Random Forest", "SVM", "Naive Bayes", "Stochastic Gradient Descent",  "K
Nearest Neighbors", "Perceptron"]

# Random Forest Classifier
clf = RandomForestClassifier()
parameter_grid = {'n_estimators': range(1, 30), 'max_features': ['auto', 'sqrt', 'log2']}
confusion_mat, y_prediction, y_test, AUC_score = run_classifier(classifier_names[0], clf,
parameter_grid, X_train, y_train, X_test, y_test)
obtain_plots(confusion_mat, classifier_names[0], y_prediction, y_test, AUC_score)

# Support Vector Classifier
clf = SVC()
parameter_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100]}, {'kernel': ['linear'], 'C':
[1, 10, 100]}]
confusion_mat, y_prediction, y_test, AUC_score = run_classifier(classifier_names[1], clf,
parameter_grid, X_train, y_train, X_test, y_test)
obtain_plots(confusion_mat, classifier_names[1], y_prediction, y_test, AUC_score)
```

```
# Naive Bayes Classifier
clf = GaussianNB()
parameter_grid = {}
confusion_mat, y_prediction, y_test, AUC_score =
run_classifier(classifier_names[2],clf,parameter_grid,X_train,y_train,X_test,y_test)
obtain_plots(confusion_mat, classifier_names[2], y_prediction, y_test, AUC_score)


#Stochastic Gradient Descent Classifier
clf = SGDClassifier(loss="hinge", penalty="l2")
parameter_grid = {}
confusion_mat, y_prediction, y_test, AUC_score =
run_classifier(classifier_names[3],clf,parameter_grid,X_train,y_train,X_test,y_test)
obtain_plots(confusion_mat, classifier_names[3], y_prediction, y_test, AUC_score)


# K- Nearest Neighbor Classifier
clf = KNeighborsClassifier()
parameter_grid = {'n_neighbors': range(1, 11)}
confusion_mat, y_prediction, y_test, AUC_score =
run_classifier(classifier_names[4],clf,parameter_grid,X_train,y_train,X_test,y_test)
obtain_plots(confusion_mat, classifier_names[4], y_prediction, y_test, AUC_score)

# Perceptron classifier
clf = linear_model.Perceptron()
parameter_grid = {'eta0' : np.logspace(-5, 0, 5), 'max_iter' : range(1,100)}
confusion_mat, y_prediction, y_test, AUC_score =
run_classifier(classifier_names[5],clf,parameter_grid,X_train,y_train,X_test,y_test)
obtain_plots(confusion_mat, classifier_names[5], y_prediction, y_test, AUC_score)
```

SAMPLE OUTPUT:

```
/home/ak/PycharmProjects/PatternHW1/venv/bin/python
/home/ak/PycharmProjects/PatternHW1/bhat_akshata_project_code.py
Training data shape:
X_train:  (3088, 19)
y_train:  (3088,)
Test data shape:
X_test:  (1030, 19)
y_test:  (1030,)

Training data shape after preprocessing:
X_train:  (3088, 52)
```

```
y_train:  (3088,)
Test data shape after preprocessing:
X_test:  (1030, 52)
y_test:  (1030,)
(4137, 52)
(4137,)
(4137, 52)
Performing classification using  Random Forest  ...
(4137, 52)
(4137,)
(4137,)

---------Random Forest Classifier--------
Training data- Actual Label size:  (4137,)
Training data- Predicted Label size:  (4137,)
Random Forest Train Accuracy: 1.000
Random Forest Test Accuracy: 0.846
Random Forest F1 Score: 0.856
Random Forest AUC Score: 0.721
Classification Report scores:
             precision    recall  f1-score   support

        yes       0.93      0.89      0.91       904
         no       0.40      0.56      0.47       126

avg / total       0.87      0.85      0.86      1030

Performing classification using  SVM  ...
(4137, 52)
(4137,)
(4137,)

---------SVM Classifier--------
Training data- Actual Label size:  (4137,)
Training data- Predicted Label size:  (4137,)
SVM Train Accuracy: 0.806
SVM Test Accuracy: 0.640
SVM F1 Score: 0.702
SVM AUC Score: 0.686
Classification Report scores:
             precision    recall  f1-score   support

        yes       0.95      0.62      0.75       904
```

no	0.22	0.75	0.34	126
avg / total	0.86	0.64	0.70	1030

Performing classification using Naive Bayes ...
(4137, 52)
(4137,)
(4137,)

---------Naive Bayes Classifier--------
Training data- Actual Label size: (4137,)
Training data- Predicted Label size: (4137,)
Naive Bayes Train Accuracy: 0.723
Naive Bayes Test Accuracy: 0.817
Naive Bayes F1 Score: 0.835
Naive Bayes AUC Score: 0.708
Classification Report scores:

	precision	recall	f1-score	support
yes	0.93	0.85	0.89	904
no	0.35	0.56	0.43	126
avg / total	0.86	0.82	0.83	1030

Performing classification using Stochastic Gradient Descent ...

(4137, 52)
(4137,)
(4137,)

---------Stochastic Gradient Descent Classifier--------
Training data- Actual Label size: (4137,)
Training data- Predicted Label size: (4137,)
Stochastic Gradient Descent Train Accuracy: 0.718
Stochastic Gradient Descent Test Accuracy: 0.830
Stochastic Gradient Descent F1 Score: 0.843
Stochastic Gradient Descent AUC Score: 0.705
Classification Report scores:

	precision	recall	f1-score	support
yes	0.93	0.87	0.90	904
no	0.37	0.54	0.44	126

avg / total 0.86 0.83 0.84 1030

Performing classification using K Nearest Neighbors ...
(4137, 52)
(4137,)
(4137,)

---------K Nearest Neighbors Classifier--------
Training data- Actual Label size: (4137,)
Training data- Predicted Label size: (4137,)
K Nearest Neighbors Train Accuracy: 1.000
K Nearest Neighbors Test Accuracy: 0.641
K Nearest Neighbors F1 Score: 0.702
K Nearest Neighbors AUC Score: 0.611
Classification Report scores:

	precision	recall	f1-score	support
yes	0.92	0.65	0.76	904
no	0.19	0.57	0.28	126
avg / total	0.83	0.64	0.70	1030

Performing classification using Perceptron ...
(4137, 52)
(4137,)
(4137,)

---------Perceptron Classifier--------
Training data- Actual Label size: (4137,)
Training data- Predicted Label size: (4137,)
Perceptron Train Accuracy: 0.658
Perceptron Test Accuracy: 0.207
Perceptron F1 Score: 0.193
Perceptron AUC Score: 0.521
Classification Report scores:

	precision	recall	f1-score	support
yes	0.92	0.11	0.19	904
no	0.13	0.94	0.22	126
avg / total	0.83	0.21	0.19	1030