ComS573_Lab4_Q2

April 15, 2020

ComS 573

Lab 4

Kanak Choudhury

1 Problem 2

1.1 Report the experiments you have done.

[1]:
```python
import numpy as np
import pandas as pd
import sklearn.preprocessing
import matplotlib
import re
import sys
import gc
import time

print('python ' +sys.version)
print('numpy '+ np.__version__)
print('pandas '+ pd.__version__)
print('sklearn '+ sklearn.__version__)
print('matplotlib '+ matplotlib.__version__)
print('re '+ re.__version__)

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MaxAbsScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import GridSearchCV
from itertools import product

import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.simplefilter("ignore", ConvergenceWarning)

def print_out(model, model_name, hyper_prem, x_dt_tr, y_dt_tr, x_dt_ts,␣
 ↪y_dt_ts):
    print("\n\nFor "+model_name+" hyper-parameters:\n",hyper_prem)
    scores = model.score(x_dt_ts, y_dt_ts)
    print("\n Test Accuracy: %.2f%%" % (scores*100))

    A = model.predict(x_dt_tr)
    cm = confusion_matrix(y_dt_tr, A)
    print("\n Train confusion matrix: \n", cm)
    acc_train = np.diagonal(cm)/cm.sum(axis=1)
    print("\n Class Accuracy for Training Data is:")
    for i in range(2):
        print('Class %d: %.2f%%' %(i, acc_train[i]*100))

    A = model.predict(x_dt_ts)
    cm = confusion_matrix(y_dt_ts, A)
    print("\n Test confusion matrix: \n", cm)
    acc_test = np.diagonal(cm)/cm.sum(axis=1)
    print("\n Class Accuracy for Testing Data is:")
    for i in range(2):
        print('Class %d: %.2f%%' %(i, acc_test[i]*100))
    print("**********************************\n")

def expand_grid(dictionary):
       return pd.DataFrame([row for row in product(*dictionary.values())],
                        columns=dictionary.keys())
```

```
python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 14:00:49) [MSC v.1915 64
bit (AMD64)]
numpy 1.16.5
pandas 0.25.1
sklearn 0.21.3
matplotlib 3.1.1
re 2.2.1
```

[2]:
```
path  = 'D:/ISU/COMS 573 - Machine Learning/HW/Lab4/'

df_train = pd.read_csv(path + 'lab4-train.csv', sep=',', header=0)
df_test = pd.read_csv(path + 'lab4-test.csv', sep=',', header=0)
tr_size = df_train.shape
```

```
ts_size = df_test.shape

x_train = np.array(df_train[['R','F','M','T']])
y_train = np.array(df_train['Class'])

x_test = np.array(df_test[['R','F','M','T']])
y_test = np.array(df_test['Class'])
x_train12 = x_train
y_train12 = y_train
x_test12 = x_test
y_test12 = y_test
```

1.2 Ensemble Classifier

1.3 Ensemble classifier using unweighted majority vote

[3]:
```
scaler = StandardScaler().fit(x_train)

std_fit = scaler.fit(x_train)
X_tr_std = std_fit.transform(x_train)
X_ts_std = std_fit.transform(x_test)
```

1.4 Neural Network

[4]:
```
alpharange = np.logspace(-6,0,7)
learnrateinitrange = np.logspace(-3,-1,7)
hidden_layer_sizes = [(5,), (5,2), (10,5), (10,), (7,3)]

dictionary = {'alpharange': alpharange,
              'learnrateinitrange': learnrateinitrange,
              'hidden_layer_sizes': hidden_layer_sizes}

prem = expand_grid(dictionary)




best_fit_nn = None
best_ts_acc_nn = 0
best_alpha_nn = None
best_learning_rate_init_nn = None
best_hidden_layer_sizes_nn = None

for j in range(prem.shape[0]):
    nnc=MLPClassifier( hidden_layer_sizes = prem.iloc[j,2], activation='relu',
                        solver='sgd', learning_rate='adaptive',
                        alpha=prem.iloc[j,0], learning_rate_init=prem.iloc[j,1],
                        max_iter=1000)
    model_nn = nnc.fit(X_tr_std, y_train)
    ts_acc1 = model_nn.score(X_ts_std, y_test)*100
    if (ts_acc1 > best_ts_acc_nn):
        best_ts_acc_nn = ts_acc1
        best_hidden_layer_sizes_nn = prem.iloc[j,2]
        best_alpha_nn = prem.iloc[j,0]
        best_learning_rate_init_nn = prem.iloc[j,1]
        best_fit_nn = model_nn

print_out(model = best_fit_nn, model_name = 'Neural Network (NN)',
          hyper_prem = best_fit_nn.get_params(), x_dt_tr = X_tr_std,
          y_dt_tr = y_train, x_dt_ts = X_ts_std, y_dt_ts = y_test)
```

```
For Neural Network (NN) hyper-parameters:
 {'activation': 'relu', 'alpha': 0.001, 'batch_size': 'auto', 'beta_1': 0.9,
'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08,
'hidden_layer_sizes': (5, 2), 'learning_rate': 'adaptive', 'learning_rate_init':
0.1, 'max_iter': 1000, 'momentum': 0.9, 'n_iter_no_change': 10,
'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': None, 'shuffle':
True, 'solver': 'sgd', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose':
False, 'warm_start': False}

 Test Accuracy: 85.05%

 Train confusion matrix:
 [[310  22]
 [ 77  38]]

 Class Accuracy for Training Data is:
Class 0: 93.37%
Class 1: 33.04%

 Test confusion matrix:
 [[228  10]
 [ 35  28]]

 Class Accuracy for Testing Data is:
Class 0: 95.80%
Class 1: 44.44%
**********************************
```

1.5 Logistic Regression (LR)

[5]:
```
lrc = np.linspace(1e-8,1,200)

dictionary = {'lrc': lrc}

prem = expand_grid(dictionary)

best_fit_lr = None
best_ts_acc_lr = 0
best_C_lr = None


for j in range(prem.shape[0]):
    lr=LogisticRegression(C=prem.iloc[j,0], solver='lbfgs')
    model_lr = lr.fit(X_tr_std, y_train)
    ts_acc1 = model_lr.score(X_ts_std, y_test)*100
    if (ts_acc1 > best_ts_acc_lr):
        best_ts_acc_lr = ts_acc1
        best_C_lr = prem.iloc[j,0]
        best_fit_lr = model_lr

print_out(model = best_fit_lr, model_name = ' Logistic Regression (LR)',
          hyper_prem = best_fit_lr.get_params(), x_dt_tr = X_tr_std,
          y_dt_tr = y_train, x_dt_ts = X_ts_std, y_dt_ts = y_test)
```

```
For  Logistic Regression (LR) hyper-parameters:
 {'C': 0.06030151693467337, 'class_weight': None, 'dual': False,
'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter':
100, 'multi_class': 'warn', 'n_jobs': None, 'penalty': 'l2', 'random_state':
None, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}

 Test Accuracy: 81.73%

 Train confusion matrix:
 [[330   2]
 [110   5]]

 Class Accuracy for Training Data is:
Class 0: 99.40%
Class 1: 4.35%

 Test confusion matrix:
 [[236   2]
 [ 53  10]]

 Class Accuracy for Testing Data is:
Class 0: 99.16%
Class 1: 15.87%
**********************************
```

Naive Bayes (NB)

[6]:
```
var_smoothing = np.linspace(1e-9,1,200)

dictionary = {'var_smoothing': var_smoothing}

prem = expand_grid(dictionary)

best_fit_nb = None
best_ts_acc_nb = 0
best_var_smoothing_nb = None


for j in range(prem.shape[0]):
    nb=GaussianNB(var_smoothing=prem.iloc[j,0])
    model_nb = nb.fit(X_tr_std, y_train)
    ts_acc1 = model_nb.score(X_ts_std, y_test)*100
    if (ts_acc1 > best_ts_acc_nb):
        best_ts_acc_nb = ts_acc1
        best_var_smoothing_nb = prem.iloc[j,0]
        best_fit_nb = model_nb

print_out(model = best_fit_nb, model_name = ' Naive Bayes (NB)',
          hyper_prem = best_fit_nb.get_params(), x_dt_tr = X_tr_std,
          y_dt_tr = y_train, x_dt_ts = X_ts_std, y_dt_ts = y_test)
```

```
For  Naive Bayes (NB) hyper-parameters:
 {'priors': None, 'var_smoothing': 0.20100502592462313}

 Test Accuracy: 81.73%

 Train confusion matrix:
 [[316  16]
 [100  15]]

 Class Accuracy for Training Data is:
Class 0: 95.18%
Class 1: 13.04%

 Test confusion matrix:
 [[233   5]
 [ 50  13]]

 Class Accuracy for Testing Data is:
```

```
Class 0: 97.90%
Class 1: 20.63%
**********************************
```

1.6 Decision Tree (DT)

[7]:
```
max_depth = [1, 2, 3, 4]
min_samples_split = [2, 3, 5, 8]
min_samples_leaf = [1, 2, 3, 5, 7]

dictionary = {'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}

prem = expand_grid(dictionary)

best_fit_dt = None
best_ts_acc_dt = 0
best_max_depth_dt = None
best_min_samples_split_dt = None
best_min_samples_leaf_dt = None


for j in range(prem.shape[0]):
    dt=DecisionTreeClassifier(criterion='entropy', splitter='best',
                              class_weight=None, max_depth=prem.iloc[j,0],
                              min_samples_split=prem.iloc[j,1],
                              min_samples_leaf=prem.iloc[j,2])
    model_dt = dt.fit(X_tr_std, y_train)
    ts_acc1 = model_dt.score(X_ts_std, y_test)*100
    if (ts_acc1 > best_ts_acc_dt):
        best_ts_acc_dt = ts_acc1
        best_max_depth_dt = prem.iloc[j,0]
        best_min_samples_split_dt = prem.iloc[j,1]
        best_min_samples_leaf_dt = prem.iloc[j,2]
        best_fit_dt = model_dt

print_out(model = best_fit_dt, model_name = 'Decision Tree (DT)',
          hyper_prem = best_fit_dt.get_params(), x_dt_tr = X_tr_std,
          y_dt_tr = y_train, x_dt_ts = X_ts_std, y_dt_ts = y_test)
```

```
For Decision Tree (DT) hyper-parameters:
 {'class_weight': None, 'criterion': 'entropy', 'max_depth': 3, 'max_features':
None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': None,
'splitter': 'best'}
```

```
 Test Accuracy: 81.40%

 Train confusion matrix:
 [[301  31]
 [ 67  48]]

 Class Accuracy for Training Data is:
Class 0: 90.66%
Class 1: 41.74%

 Test confusion matrix:
 [[221  17]
 [ 39  24]]

 Class Accuracy for Testing Data is:
Class 0: 92.86%
Class 1: 38.10%
**********************************
```

1.7 Ensemble classifier using unweighted majority vote

[8]:
```
nn=MLPClassifier( hidden_layer_sizes = best_hidden_layer_sizes_nn,␣
 ↪activation='relu',
                        solver='sgd', learning_rate='adaptive',
                        alpha=best_alpha_nn,␣
 ↪learning_rate_init=best_learning_rate_init_nn,
                        max_iter=2000)
lr=LogisticRegression(C=best_C_lr, solver='lbfgs')
nb=GaussianNB(var_smoothing =  best_var_smoothing_nb)
dt=DecisionTreeClassifier(criterion='entropy', splitter='best',
                            class_weight='balanced', max_depth =␣
 ↪best_max_depth_dt,
                            min_samples_split = best_min_samples_split_dt,
                            min_samples_leaf = best_min_samples_leaf_dt)

pipe = [('nn', nn), ('lr', lr), ('nb', nb), ('dt', dt)]

eclf = VotingClassifier(estimators=pipe, voting='soft')

eclf.fit(X_tr_std, y_train)
print_out(model = eclf, model_name = 'Ensemble classi er using unweighted␣
 ↪majority vote',
          hyper_prem = eclf.get_params(), x_dt_tr = X_tr_std,
          y_dt_tr = y_train, x_dt_ts = X_ts_std, y_dt_ts = y_test)
```

```
For Ensemble classi er using unweighted majority vote hyper-parameters:
 {'estimators': [('nn', MLPClassifier(activation='relu', alpha=0.001,
batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='adaptive',
              learning_rate_init=0.1, max_iter=2000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)), ('lr',
LogisticRegression(C=0.06030151693467337, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)), ('nb', GaussianNB(priors=None,
var_smoothing=0.20100502592462313)), ('dt',
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best'))],
'flatten_transform': True, 'n_jobs': None, 'voting': 'soft', 'weights': None,
'nn': MLPClassifier(activation='relu', alpha=0.001, batch_size='auto',
beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='adaptive',
              learning_rate_init=0.1, max_iter=2000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False), 'lr':
LogisticRegression(C=0.06030151693467337, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False), 'nb': GaussianNB(priors=None,
var_smoothing=0.20100502592462313), 'dt':
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best'), 'nn__activation':
'relu', 'nn__alpha': 0.001, 'nn__batch_size': 'auto', 'nn__beta_1': 0.9,
'nn__beta_2': 0.999, 'nn__early_stopping': False, 'nn__epsilon': 1e-08,
'nn__hidden_layer_sizes': (5, 2), 'nn__learning_rate': 'adaptive',
'nn__learning_rate_init': 0.1, 'nn__max_iter': 2000, 'nn__momentum': 0.9,
'nn__n_iter_no_change': 10, 'nn__nesterovs_momentum': True, 'nn__power_t': 0.5,
'nn__random_state': None, 'nn__shuffle': True, 'nn__solver': 'sgd', 'nn__tol':
```

```
0.0001, 'nn__validation_fraction': 0.1, 'nn__verbose': False, 'nn__warm_start':
False, 'lr__C': 0.06030151693467337, 'lr__class_weight': None, 'lr__dual':
False, 'lr__fit_intercept': True, 'lr__intercept_scaling': 1, 'lr__l1_ratio':
None, 'lr__max_iter': 100, 'lr__multi_class': 'warn', 'lr__n_jobs': None,
'lr__penalty': 'l2', 'lr__random_state': None, 'lr__solver': 'lbfgs', 'lr__tol':
0.0001, 'lr__verbose': 0, 'lr__warm_start': False, 'nb__priors': None,
'nb__var_smoothing': 0.20100502592462313, 'dt__class_weight': 'balanced',
'dt__criterion': 'entropy', 'dt__max_depth': 3, 'dt__max_features': None,
'dt__max_leaf_nodes': None, 'dt__min_impurity_decrease': 0.0,
'dt__min_impurity_split': None, 'dt__min_samples_leaf': 1,
'dt__min_samples_split': 2, 'dt__min_weight_fraction_leaf': 0.0, 'dt__presort':
False, 'dt__random_state': None, 'dt__splitter': 'best'}

 Test Accuracy: 83.39%

 Train confusion matrix:
 [[312  20]
 [ 75  40]]

 Class Accuracy for Training Data is:
Class 0: 93.98%
Class 1: 34.78%

 Test confusion matrix:
 [[226  12]
 [ 38  25]]

 Class Accuracy for Testing Data is:
Class 0: 94.96%
Class 1: 39.68%
**********************************
```

1.8 Comment

For each of the four models, train and test accuracy with confusion matrix and classification accuracy are given above.

For each of these models, I have considered python grid search algorithm for using unweighted majority voting ensemble model. Best parameters for each of the model are given above. It is found that neural network model has the highest test accuracy (85%) and all other three models have about 81.5% test accuracy. All models have around 95% test accuracy for class 0 and Neural network model has the highest test accuracy for class 1 (about 44%). However, logistic regression model has the lowest (about 15%) test accuracy for class 1.

After using unweighted majority vote classifier using soft max, it is found about 83% test accuracy of which about 95% for class 0 and about 40% for class 1 though it is less than the neural network test accuracy. It is important to mention that this data is not a balanced data and I did not tune the cutoff probability or over sample or under sample tuning.

Note: Using 'soft' max in voting classifier gives higher accuracy than 'hard' max unweighted ensemble model for this data.

1.9 Ensemble classifier using weighted majority vote tuning weights

```
[10]: n = 500
weight = pd.DataFrame({'w1': np.random.uniform(0, 5, n),
                       'w2': np.random.uniform(0, 5, n),
                       'w3': np.random.uniform(0, 5, n),
                       'w4': np.random.uniform(0, 5, n)})

# weight = weight.div(weight.sum(axis = 1), axis=0)
# weight.append([1,1,1,1,])
weight['accuracy'] = np.nan

for i in range(weight.shape[0]):
    eclf2 = VotingClassifier(estimators=pipe, voting='soft', weights = weight.
 ↪iloc[i,:4].ravel())

    scores = cross_val_score(estimator=eclf2,  X=X_tr_std, y=y_train.ravel(),␣
 ↪cv=3, scoring='accuracy')

    weight.iloc[i, 4] = scores.mean()

kkk = weight['accuracy'].idxmax()

eclf2 = VotingClassifier(estimators=pipe, voting='soft', weights = weight.
 ↪iloc[kkk,:4].ravel())
eclf2.fit(X_tr_std, y_train)
print_out(model = eclf2, model_name = 'Ensemble classi er using weighted␣
 ↪majority vote tuning weights',
          hyper_prem = eclf2.get_params(), x_dt_tr = X_tr_std,
          y_dt_tr = y_train, x_dt_ts = X_ts_std, y_dt_ts = y_test)
```

```
For Ensemble classi er using weighted majority vote tuning weights hyper-
parameters:
 {'estimators': [('nn', MLPClassifier(activation='relu', alpha=0.001,
batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='adaptive',
              learning_rate_init=0.1, max_iter=2000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)), ('lr',
LogisticRegression(C=0.06030151693467337, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)), ('nb', GaussianNB(priors=None,
var_smoothing=0.20100502592462313)), ('dt',
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best'))],
'flatten_transform': True, 'n_jobs': None, 'voting': 'soft', 'weights':
array([2.83171795, 1.41944114, 1.68824217, 1.92243942]), 'nn':
MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='adaptive',
              learning_rate_init=0.1, max_iter=2000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False), 'lr':
LogisticRegression(C=0.06030151693467337, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False), 'nb': GaussianNB(priors=None,
var_smoothing=0.20100502592462313), 'dt':
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best'), 'nn__activation':
'relu', 'nn__alpha': 0.001, 'nn__batch_size': 'auto', 'nn__beta_1': 0.9,
'nn__beta_2': 0.999, 'nn__early_stopping': False, 'nn__epsilon': 1e-08,
'nn__hidden_layer_sizes': (5, 2), 'nn__learning_rate': 'adaptive',
'nn__learning_rate_init': 0.1, 'nn__max_iter': 2000, 'nn__momentum': 0.9,
'nn__n_iter_no_change': 10, 'nn__nesterovs_momentum': True, 'nn__power_t': 0.5,
'nn__random_state': None, 'nn__shuffle': True, 'nn__solver': 'sgd', 'nn__tol':
0.0001, 'nn__validation_fraction': 0.1, 'nn__verbose': False, 'nn__warm_start':
False, 'lr__C': 0.06030151693467337, 'lr__class_weight': None, 'lr__dual':
False, 'lr__fit_intercept': True, 'lr__intercept_scaling': 1, 'lr__l1_ratio':
None, 'lr__max_iter': 100, 'lr__multi_class': 'warn', 'lr__n_jobs': None,
'lr__penalty': 'l2', 'lr__random_state': None, 'lr__solver': 'lbfgs', 'lr__tol':
0.0001, 'lr__verbose': 0, 'lr__warm_start': False, 'nb__priors': None,
'nb__var_smoothing': 0.20100502592462313, 'dt__class_weight': 'balanced',
'dt__criterion': 'entropy', 'dt__max_depth': 3, 'dt__max_features': None,
'dt__max_leaf_nodes': None, 'dt__min_impurity_decrease': 0.0,
'dt__min_impurity_split': None, 'dt__min_samples_leaf': 1,
'dt__min_samples_split': 2, 'dt__min_weight_fraction_leaf': 0.0, 'dt__presort':
False, 'dt__random_state': None, 'dt__splitter': 'best'}
```

```
 Test Accuracy: 84.39%

 Train confusion matrix:
 [[310  22]
 [ 74  41]]

 Class Accuracy for Training Data is:
Class 0: 93.37%
Class 1: 35.65%

 Test confusion matrix:
 [[227  11]
 [ 36  27]]

 Class Accuracy for Testing Data is:
Class 0: 95.38%
Class 1: 42.86%
**********************************
```

1.10 Comment:

I have used cross validation grid search technique to find the best weight for the weighted ensemble majority voting algorithm.

Using grid search for the weights, it is found higher test accuracy (about 84.5%) than the unweighted ensemble majority voting algorithm (about 83%). Also, class 1 accuracy is about 3% higher for weighted ensemble majority voting algorithm than unweighted algorithm.

```
[ ]:
```

```
[ ]:
```