```
# ----------------------------
# SETScholars: DSR-008.py ------
# ----------------------------

# -*- coding: utf-8 -*-

#########################################################################
# This is an end-2-end Applied Machine Learning Script with RDBMS
#
# Title:-->
# Multiclass Classification using multiple Support Vector Machine
# in Python: Applied Machine Learning Recipe 010

# DataSet: IRIS Flower Data
# Source: UCI Machine Learning Repository
# Knowledge required: Basic Python, Scikit-Learn and MySQL
# System requirements:
#   a) Python (3.X) distribution from Anaconda (Anaconda 3)
#   b) MySQL 5.7 with an user: root and password:
#########################################################################

"""
@author:
    Dr. Nilimesh Halder (Principle Data Scientist @ Crown)
    BSc in Computer Science and Engineering,
        @ Khulna University, Bangladesh.
    PhD in Artificial Intelligence and Applied Machine Learning,
        @ The University of Western Australia, Australia.

"""

# ------------------------------------------------------------------
# Steps in Applied Machine Learning:
#
# 1. Load Library
# 2. Load Dataset to which Machine Learning Algorithm to be applied
#    Either a) load from a CSV file or b) load from a Database
# 3. Summarisation of Data to understand dataset (Descriptive Statistics
# 4. Visualisation of Data to understand dataset (Plots, Graphs etc.)
# 5. Data pre-processing & Data transformation
#    (split into train-test datasets)
# 6. Application of a Machine Learning Algorithm to training dataset
#   a) setup a ML algorithm and parameter settings
#   b) cross validation setup with training dataset
#   c) training & fitting Algorithm with training Dataset
#   d) evaluation of trained Algorithm (or Model) and result
#   e) saving the trained model for future prediction
```

```
# 7. Finalise the trained modela and make prediction
# ------------------------------------------------------------------------

import warnings
warnings.filterwarnings("ignore")

# ------------------------------------------------------------------------
# 1. Load necessary libraries
# ------------------------------------------------------------------------
import time
import sqlalchemy as sa
import pandas as pd
import pickle as pk
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
from sklearn.svm import NuSVC
from sklearn.svm import LinearSVC
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score

# ------------------------------------------------------------------------
# Load DataSet from CSV file
# ------------------------------------------------------------------------
def loadFrCSVFile(filename):
    print(filename)
    col_names = ['sepal_length', 'sepal_width', 'petal_length',
                 'petal_width', 'class']
    dataset = pd.read_csv(filename, names=col_names)
    return dataset

# ------------------------------------------------------------------------
# Import DataSet to a MySQL Database
# ------------------------------------------------------------------------
def import2MySQL(dataset):
    engine_str = (
      'mysql+pymysql://{user}:{password}@{server}/{database}'.format(
       user      =  'root',
       password  =  'root888',
       server    =  'localhost',
       database  =  'DataScienceRecipes'))
    engine = sa.create_engine(engine_str)
    conn = engine.connect()
```

```
    # check whether connection is Successful or not
    if (conn):
        print("MySQL Connection is Successful ... ... ...")
    else:
        print("MySQL Connection is not Successful ... ... ...")

    dataset.to_sql(name='irisdata', con=engine,
                   schema='datasciencerecipes',
                   if_exists = 'replace',
                   chunksize = 1000, index=False)
    conn.close()

# ----------------------------------------------------------------------
# Load DataSet from MySQL Database to Pandas a DataFrame
# ----------------------------------------------------------------------
def loadDataSetFrMySQLTable():
    engine_str = (
      'mysql+pymysql://{user}:{password}@{server}/{database}'.format(
       user      =  'root',
       password  =  'root888',
       server    =  'localhost',
       database  =  'datasciencerecipes'))
    engine = sa.create_engine(engine_str)
    conn = engine.connect()

    # check whether connection is Successful or not
    if (conn):
        print("MySQL Connection is Successful ... ... ...")
    else:
        print("MySQL Connection is not Successful ... ... ...")

# ----------------------------------------------------------------------
    # MySQL Query with few generated Attributes/Features
# ----------------------------------------------------------------------
    query = '''
    SELECT  sepal_length,
            sepal_width,
            petal_length,
            petal_width,
            round(sepal_length/sepal_width,2) as ratio1,
            round(sepal_width/petal_length,2) as ratio2,
            round(petal_length/petal_width,2) as ratio3,
            round(petal_width/sepal_length,2) as ratio4,
            round(sepal_width/sepal_length,2) as ratio5,
            round(petal_length/sepal_width,2) as ratio6,
            round(petal_width/petal_length,2) as ratio7,
```

```
            round(sepal_length/petal_width,2) as ratio8,
            class
    FROM irisdata;
    '''

    query_result = conn.execute(query)
    dataset =  pd.DataFrame(query_result.fetchall(),
                            columns =  query_result.keys())
    print('DataFrame Size',dataset.shape);
    print('ROW',dataset.shape[0]);print('COLUMN',dataset.shape[1]);
    conn.close()
    return dataset

# -----------------------------------------------------------------------
# Data Summarisation to understand Dataset using Descriptive Statistics
# -----------------------------------------------------------------------
def summariseDataset(dataset):
    cols1 = ['sepal_length','sepal_width','petal_length','petal_width']
    cols2 = ['ratio1','ratio2','ratio3','ratio4']
    cols3 = ['ratio5','ratio6','ratio7','ratio8']

    # shape
    print(dataset[cols1].shape)
    print(dataset[cols2].shape)
    print(dataset[cols3].shape)

    # head
    print(dataset[cols1].head(5))
    print(dataset[cols2].head(5))
    print(dataset[cols3].head(5))

    # descriptions
    print(dataset[cols1].describe())
    print(dataset[cols2].describe())
    print(dataset[cols3].describe())

    # class distribution
    print(dataset.groupby('class').size())

# -----------------------------------------------------------------------
# Data Visualisation to understand Dataset
# -----------------------------------------------------------------------
def visualiseDataset(dataset):
    cols1 = ['sepal_length','sepal_width','petal_length','petal_width']
    cols2 = ['ratio1','ratio2','ratio3','ratio4']
    cols3 = ['ratio5','ratio6','ratio7','ratio8']
# -----------------------------------------------------------------------
```

```
    # box and whisker plots
# ------------------------------------------------------------------------
    dataset[cols1].plot(kind='box', subplots=True, layout=(2,2),
                        sharex=False, sharey=False)
    pyplot.show()

    dataset[cols2].plot(kind='box', subplots=True, layout=(2,2),
                        sharex=False, sharey=False)
    pyplot.show()

    dataset[cols3].plot(kind='box', subplots=True, layout=(2,2),
                        sharex=False, sharey=False)
    pyplot.show()

# ------------------------------------------------------------------------
    # histograms
# ------------------------------------------------------------------------
    dataset[cols1].hist()
    pyplot.show()

    dataset[cols2].hist()
    pyplot.show()

    dataset[cols3].hist()
    pyplot.show()

# ------------------------------------------------------------------------
    # scatter plot matrix
# ------------------------------------------------------------------------
    scatter_matrix(dataset[cols1])
    pyplot.show()

    scatter_matrix(dataset[cols2])
    pyplot.show()

    scatter_matrix(dataset[cols3])
    pyplot.show()

# ------------------------------------------------------------------------
# Data Pre-Processing
# ------------------------------------------------------------------------
def preProcessingData(dataset):

    # 1. Data Cleaning
      # There is no missing value.
      # We could "Outlier treatment" but nothing was done here.
```

```
    # 2. Feature Selection
    cols_X = ['sepal_length','sepal_width','petal_length','petal_width',
              'ratio1','ratio2','ratio3','ratio4',
              'ratio5','ratio6','ratio7','ratio8']
    cols_Y = 'class'

    # 3. Data Transform - Split out train : test datasets
    train_X, test_X, train_Y, test_Y = train_test_split(
                                        dataset.loc[:, cols_X],
                                        dataset.loc[:, cols_Y],
                                        test_size=0.33)
    return train_X, test_X, train_Y, test_Y

# ------------------------------------------------------------------------
# Applied Machine Learning Algorithm ... ... ...
# ------------------------------------------------------------------------
def evaluateAlgorithm(train_X, test_X, train_Y, test_Y):
    # ------------------------------------------------------------------------
    # Machine Lreaning Algorithm, Parameter settings
    # ------------------------------------------------------------------------
    model_List = []

    # ------------------------------------------------------------------------
    # Algorithms are applied with different parameter settings
    # - manual parameter tuning.
    # [Note: Grid Search and Random Search for
    # Parameters tuning will be introduced later]
    # ------------------------------------------------------------------------
    # Support Vactor Machine (SVM) with manual parameter settings

    # 1. Support Vector Machine - SVC : SVC()
    # kernel = ['poly', 'rbf', 'sigmoid']

    # 1.1 SVC : SVC();
    SVC_1 = SVC(C=1.0, kernel='rbf',
                degree=3, gamma='auto', coef0=0.0, shrinking=True,
                probability=False, tol=0.001, cache_size=200,
                class_weight=None, verbose=False, max_iter=-1,
                decision_function_shape='ovr', random_state=None)
    model_List.append(('SVC-1',
                'Support Vector Machine: SVC - PS-1', SVC_1))

    # 1.2 SVC : SVC();
    SVC_2 = SVC(C=1.0, kernel='poly',
                degree=3, gamma='auto', coef0=0.0, shrinking=True,
                probability=False, tol=0.001, cache_size=200,
                class_weight=None, verbose=False, max_iter=-1,
```

```
            decision_function_shape='ovr', random_state=None)
model_List.append(('SVC-2',
            'Support Vector Machine: SVC - PS-2', SVC_2))

# 1.3 SVC : SVC();
SVC_3 = SVC(C=1.0, kernel='sigmoid', degree=3, gamma='auto',
            coef0=0.0, shrinking=True, probability=False,
            tol=0.001, cache_size=200, class_weight=None,
            verbose=False, max_iter=-1,
            decision_function_shape='ovr', random_state=None)
model_List.append(('SVC-3',
            'Support Vector Machine: SVC - PS-3', SVC_3))

# 2. Support Vector Machine - NuSVC : NuSVC();
# kernel = ['poly', 'rbf', 'sigmoid']

# 2.1 NuSVC : NuSVC();
NuSVC_1 = NuSVC(nu=0.5, kernel='rbf', degree=3, gamma='auto',
                coef0=0.0, shrinking=True, probability=False,
                tol=0.001, cache_size=200,
                class_weight=None, verbose=False,
                max_iter=-1, decision_function_shape='ovr',
                random_state=None)
model_List.append(('NuSVC-1',
      'Support Vector Machine: NuSVC - PS-1', NuSVC_1))

# 2.2 NuSVC : NuSVC();
NuSVC_2 = NuSVC(nu=0.5, kernel='poly', degree=3, gamma='auto',
                coef0=0.0,
                shrinking=True, probability=False,
                tol=0.001, cache_size=200,
                class_weight=None, verbose=False,
                max_iter=-1,
                decision_function_shape='ovr',
                random_state=None)
model_List.append(('NuSVC-2',
      'Support Vector Machine: NuSVC - PS-2', NuSVC_2))

# 2.3 NuSVC : NuSVC();
NuSVC_3 = NuSVC(nu=0.5, kernel='sigmoid', degree=3, gamma='auto',
                coef0=0.0,
                shrinking=True, probability=False, tol=0.001,
                cache_size=200,
                class_weight=None, verbose=False, max_iter=-1,
                decision_function_shape='ovr',
                random_state=None)
model_List.append(('NuSVC-3',
```

```
      'Support Vector Machine: NuSVC - PS-3', NuSVC_3))

# 3. Support Vector Machine - LinearSVC : LinearSVC();
# loss = ['squared_hinge', 'hinge'] ; penalty = ['l2', 'l1']

# 3.1 LinearSVC : LinearSVC();
LSVC_1 = LinearSVC(penalty='l2', loss='squared_hinge',
                   dual=True, tol=0.0001,
                   C=1.0, multi_class='ovr',
                   fit_intercept=True,
                   intercept_scaling=1,
                   class_weight=None, verbose=0,
                   random_state=None, max_iter=1000)
model_List.append(('LSVC-1',
      'Support Vector Machine: LinearSVC - PS-1', LSVC_1))

# 3.2 LinearSVC : LinearSVC();
LSVC_2 = LinearSVC(penalty='l2', loss='hinge',
                   dual=True, tol=0.0001,
                   C=1.0, multi_class='ovr',
                   fit_intercept=True,
                   intercept_scaling=1,
                   class_weight=None, verbose=0,
                   random_state=None, max_iter=1000)
model_List.append(('LSVC-2',
      'Support Vector Machine: LinearSVC - PS-2', LSVC_2))

# ------------------------------------------------------------------
# Cross Validation -------------------------------------------------
# ------------------------------------------------------------------
print("Cross Validation Results ")
outcomes = []
description = []
shortDescription = []
for shtDes, des, model in model_List:
    cv_results = cross_val_score(model, train_X, train_Y,
                                 cv = 5, scoring='accuracy',
                                 n_jobs = 4, verbose = 0)
    outcomes.append(cv_results)
    description.append(des)
    shortDescription.append(shtDes)
    prt_string = "\n %s:\n \tMean Accuracy: %f (Std: %f)" % (des,
                                          cv_results.mean(),
                                          cv_results.std())
    print(prt_string)

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

```
    # Visualise the outcomes / results from Cross Validation
    # ---------------------------------------------------------------
    fig = pyplot.figure()
    fig.suptitle('Cross Validation Results (Algorithm Comparison)')
    ax = fig.add_subplot(111)
    #pyplot.boxplot(outcomes)
    #ax.set_xticklabels(shortDescription)
    pyplot.boxplot(outcomes, vert = False)
    ax.set_yticklabels(shortDescription)

    pyplot.show()

    # ---------------------------------------------------------------
    # Training & Fitting of each Algorithm with training Dataset
    # ---------------------------------------------------------------
    print('\nEvaluate Algorithms ... ... ... ')
    for shtDes, des, model in model_List:
        trained_Model = model.fit(train_X, train_Y)

    # ---------------------------------------------------------------
    # Evaluation of trained Algorithm (or Model) and result
    # ---------------------------------------------------------------
        pred_Class    = trained_Model.predict(test_X)
        acc           = accuracy_score(test_Y, pred_Class)
        classReport   = classification_report(test_Y, pred_Class)
        confMatrix    = confusion_matrix(test_Y, pred_Class)
        kappa_score   = cohen_kappa_score(test_Y, pred_Class)

        print("\n%s: " % (des))
        print('The accuracy: {}'.format(round(acc,2)))
        print('The kappa score: {}'.format(round(kappa_score,2)))
        print('The Classification Report:\n {}'.format(classReport))
        print('The Confusion Matrix:\n {}'.format(confMatrix))

    # ---------------------------------------------------------------
    # Save the trained Model
    # ---------------------------------------------------------------
        with open('model_'+shtDes+'.pickle', 'wb') as f:
                pk.dump(trained_Model, f)

# ---------------------------------------------------------------
# Load a (new or existing ) dataset to make prediction
# ---------------------------------------------------------------
def loadPredictionDataset():
    engine_str = (
     'mysql+pymysql://{user}:{password}@{server}/{database}'.format(
      user      =  'root',
```

```
      password  =  'root888',
      server    =  'localhost',
      database  =  'datasciencerecipes'))

    engine = sa.create_engine(engine_str)
    conn = engine.connect()

    #check whether connection is Successful or not
    #if (conn): print("MySQL Connection is Successful ... ... ...")
    #else:      print("MySQL Connection is not Successful ... ... ...")

    # MySQL Query - New Query is required for Prediction DataSet
    query = '''
    SELECT  sepal_length,
            sepal_width,
            petal_length,
            petal_width,
            round(sepal_length/sepal_width,2) as ratio1,
            round(sepal_width/petal_length,2) as ratio2,
            round(petal_length/petal_width,2) as ratio3,
            round(petal_width/sepal_length,2) as ratio4,
            round(sepal_width/sepal_length,2) as ratio5,
            round(petal_length/sepal_width,2) as ratio6,
            round(petal_width/petal_length,2) as ratio7,
            round(sepal_length/petal_width,2) as ratio8
    FROM irisdata;
    '''

    query_result = conn.execute(query)
    dataset =  pd.DataFrame(query_result.fetchall(),
                            columns =  query_result.keys())
    conn.close()
    return dataset

# ------------------------------------------------------------------------
# Load the trained model and make prediction
# ------------------------------------------------------------------------
def loadTrainedModelForPrediction(pred_dataset):
    # trained models are:
    # model_LR, model_SGD, model_PA
    f = open('model_LSVC-2.pickle', 'rb')
    model = pk.load(f); f.close();
    pred_Class = model.predict(pred_dataset)
    pred_dataset.loc[:, 'classResult'] = pred_Class
    return pred_dataset

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

```
# Finalise the results and update the audiance
# ----------------------------------------------------------------
def finaliseResult(result):
    #Save Result in a CSV file
    result.to_csv('finalResult.csv', index = False)
    print("\n\nSave Result in a CSV file ... ... Done ...")

    #Save Result in a MySQl Table
    engine_str = (
      'mysql+pymysql://{user}:{password}@{server}/{database}'.format(
       user      =  'root',
       password  =  'root888',
       server    =  'localhost',
       database  =  'datasciencerecipes'))

    engine = sa.create_engine(engine_str)
    conn = engine.connect()

    #check whether connection is Successful or not
    #if (conn): print("MySQL Connection is Successful ... ... ...")
    #else:      print("MySQL Connection is not Successful ... ... ...")

    result.to_sql(name='irisresult', con=engine,
                  schema='datasciencerecipes',
                  if_exists = 'replace', chunksize = 1000,
                  index=False)
    print("Save Result in a MySQl Table ... ... Done ...")
    conn.close()

# ----------------------------------------------------------------
# End-to-End Applied Machine Learning Recipes for Developers
# ----------------------------------------------------------------
if __name__ == '__main__':
    start_time = time.time()
# ----------------------------------------------------------------
# 2. Load Dataset to which Machine Learning Algorithm to be applied
# ----------------------------------------------------------------
    filename = 'iris.data.csv'
    dataset = loadFrCSVFile(filename)
    import2MySQL(dataset)
    dataset = loadDataSetFrMySQLTable()

# ----------------------------------------------------------------
# 3. Summarisation of Data to understand dataset:Descriptive Statistics
# ----------------------------------------------------------------
    summariseDataset(dataset)
```

```
# ----------------------------------------------------------------------
# 4. Visualisation of Data to understand dataset (Plots, Graphs etc.)
# ----------------------------------------------------------------------
    visualiseDataset(dataset)

# ----------------------------------------------------------------------
# 5. Data pre-processing and Data transformation
#   (split into train-test datasets)
# ----------------------------------------------------------------------
    train_X, test_X, train_Y, test_Y = preProcessingData(dataset)

# ----------------------------------------------------------------------
# 6. Application of a Machine Learning Algorithm to training dataset
# ----------------------------------------------------------------------
    evaluateAlgorithm(train_X, test_X, train_Y, test_Y)

# ----------------------------------------------------------------------
# 7. Load the saved model and apply it to new dataset for prediction
# ----------------------------------------------------------------------
    pred_Dataset = loadPredictionDataset()
    result = loadTrainedModelForPrediction(pred_Dataset)
    finaliseResult(result)

# ----------------------------------------------------------------------
    print('\nEnd-to-End Applied Machine Learning Recipes\n')
    print("Execution Time %s seconds: " % (time.time()-start_time))
# ----------------------------------------------------------------------

'''
---------------
Disclaimer ----
---------------
The information and recipe presented within this recipe
is only for educational and coaching purposes
for beginners and app-developers.

Anyone can practice and apply the recipe presented here,
but the reader is taking full responsibility for his/her actions.

The author of this recipe (code / program) has made every effort
to ensure the accuracy of the information was correct
at time of publication.

The author does not assume and hereby disclaims any liability
to any party for any loss, damage, or disruption caused
by errors or omissions, whether such errors or omissions
result from accident, negligence, or any other cause.
```

```
Some of the information presented here could be also found
in public knowledge domains.

----------------------
Acknowledgement ------
----------------------
The author of this eArticle thanks to all readers of
SETScholars: Applied Machine Learning and Data Science Recipes.

The author also thanks to the publishers and contributors
of open source datasets at
UCI Machine Learning Repository and Kaggle
as well as acknowledges their kind efforts.
'''
```