```
import numpy as np
import math
import time
from matplotlib import pyplot as plt
from data.data_utils import load_dataset
from tqdm import tqdm

datasets = ['mauna_loa', 'rosenbrock', 'pumadyn32nm', 'iris', 'mnist_small']

regression_datasets = ['mauna_loa', 'rosenbrock', 'pumadyn32nm']
classification_datasets = ['iris', 'mnist_small']

def loadData(dataset):
'''
To load the dataset. Include one exception for the dataset 'rosenbrock'
'''
if dataset not in datasets:
return 0,0,0,0,0,0
elif dataset == 'rosenbrock':
xtrain, xvalid, xtest, ytrain, yvalid, ytest = load_dataset('rosenbrock', n_train=1000, d=2)
else:
xtrain, xvalid, xtest, ytrain, yvalid, ytest = load_dataset(str(dataset))
return xtrain, xvalid, xtest, ytrain, yvalid, ytest

def RMSE(predict, target):
return np.sqrt(((predict - target)**2).mean()) #root mean square error

def svd_regression(dataset):
'''
find the regression model by minimizing the least
square loss function using SVD
'''
x_train, x_valid, x_test, y_train, y_valid, y_test = loadData(dataset)
start = time.time()
```

```
x_train_valid = np.vstack([x_train, x_valid])
y_train_valid = np.vstack([y_train, y_valid])

X = np.ones((len(x_train_valid), len(x_train_valid[0])+1))
X[:, 1:] = x_train_valid

# compute the matrices taht will be used for SVD
U, s, V = np.linalg.svd(X, full_matrices=True)
sigma = np.diag(s)
zero = np.zeros([len(x_train_valid)-len(sigma), len(sigma)]) #matrix filled with zeros
S = np.vstack([sigma, zero])
w = np.dot(V.T, np.dot(np.linalg.pinv(S), np.dot(U.T, y_train_valid))) # equation obtain in lecture

X_test = np.ones((len(x_test), len(x_test[0]) + 1))
X_test[:, 1:] = x_test
y_pred = np.dot(X_test, w) # prediction is simply y = Wx

error = RMSE(y_test, y_pred)
runtime = time.time() - start

if dataset == "mauna_loa":
    # only plot for mauna loa
    plt.figure(1)
    plt.plot(x_test, y_test, '-g', label='Actual')
    plt.plot(x_test, y_pred, '-r', label='Prediction')
    plt.title('SVD predictions for mauna loa dataset')
    plt.xlabel('x test')
    plt.ylabel('y')
    plt.legend(loc='upper right')
    plt.savefig('mauna_loa_svd.png')

return (runtime, error)
```

```
def svd_classification(dataset):
'''
Make classification predictions by minizing the least square error using SVD
'''
x_train, x_valid, x_test, y_train, y_valid, y_test = loadData(dataset)
start = time.time()
```

```
x_train_valid = np.vstack([x_train, x_valid])
y_train_valid = np.vstack([y_train, y_valid])

X = np.ones((len(x_train_valid), len(x_train_valid[0])+1))
X[:, 1:] = x_train_valid

# compute the matrices taht will be used for SVD
U, s, V = np.linalg.svd(X, full_matrices=True)
sigma = np.diag(s)
zero = np.zeros([len(x_train_valid)-len(sigma), len(sigma)]) #matrix filled with zeros
S = np.vstack([sigma, zero])
w = np.dot(V.T, np.dot(np.linalg.pinv(S), np.dot(U.T, y_train_valid)))


X_test = np.ones((len(x_test), len(x_test[0]) + 1))
X_test[:, 1:] = x_test
########################CODE ABOVE IS SIMILAR TO REGRESSION##############################
#find the maximum values of the predictions
y_predict = np.argmax(np.dot(X_test, w), axis=1)
y_test = np.argmax(y_test, axis=1)

#counting number
ratio = (y_predict==y_test).sum()/len(y_test)
runtime = time.time() - start
return runtime, ratio
```

```
##########REGRESSION##########
for dataset in regression_datasets:
runtime, error = svd_regression(dataset)
print("############################")
print('Dataset: ' + str(dataset))
print('Runtime: ' + str(runtime))
print('RMSE: ' + str(error))
##########CLASSFICATION##########
for dataset in classification_datasets:
runtime, ratio = svd_classification(dataset)
print("############################")
print('Dataset: ' + str(dataset))
print('Runtime: ' + str(runtime))
print('Accuracy: ' + str(ratio))
```