```python
from mnist import MNIST
import sklearn.metrics as metrics
import numpy as np
import scipy
import pdb
import time
from numpy.linalg import inv
from numpy.linalg import solve
import matplotlib.pyplot as plt

NUM_CLASSES = 10
d = 1000 # the raisen dimension
G_transpose = np.random.normal(scale = 0.1, size = (d, 784)) #the transpose of G, dim matched
b = np.random.random((d,1))*6.2832
def load_dataset():
    mndata = MNIST('./data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0
    return (X_train, labels_train), (X_test, labels_test)


def train(X_train, y_train, reg=0):
    ''' Build a model from X_train -> y_train ''' #dim of X_train is 5000,600000
    a = np.dot(np.matrix.transpose(X_train), X_train) + reg*np.identity(d)
    y = np.dot(np.transpose(X_train), y_train)
    w = solve(a,y)
    return w

def train_gd(X_train, y_train, alpha=0.1, reg=0, num_iter=10000):
    ''' Build a model from X_train -> y_train using batch gradient descent '''
    #initalize a W
    alpha = alpha/X_train.shape[0]
    W = np.zeros((d,10))
    help1 = np.dot(np.transpose(X_train), X_train)
    help2 = np.dot(np.transpose(X_train), y_train)
    Wlist = []
    for i in range(num_iter):
        # if (i%100 == 0):
        #     pdb.set_trace()
        l = reg*W + np.dot(help1, W) - help2
        W = W - l*alpha
        if ((i+1) % 100 == 0):
            Wlist.append(W)
    return Wlist
    return W
def train_sgd(X_train, y_train, alpha=0.1, reg=0, num_iter=10000):
    ''' Build a  from X_train -> y_train using stochastic gradient descent '''
    W = np.zeros((d,10))
    Wlist = []#for plotting data
    for i in range(num_iter):
        index = np.random.randint(low = 0, high = 60000-1)
        vector = X_train[index].T
        yvector = y_train[index]
        derivative = reg*W + np.dot(vector[:, None], (np.dot(vector, W) - yvector)[None,:])
        W = W - derivative*alpha*(1-i/(num_iter)) #linear
        if ((i+1) % 100 == 0):
```

```
        if ((i+1) % 100 == 0):
            Wlist.append(W)
    return Wlist
    return W

def one_hot(labels_train):
    '''Convert categorical labels 0,1,2,....9 to standard basis vectors in R^{10} ''' #return 60000*784
    return np.array([[1 if i == labels_train[k] else 0 for i in range(10)] for k in range(len(labels_train))])

def predict(model, X):
    ''' From model and data points, output prediction vectors '''
    result = np.dot(np.matrix.transpose(model), np.transpose(X)) #get a vector
    return [np.argmax(i) for i in np.matrix.transpose(result)]

def phi(X):
    ''' Featurize the inputs using random Fourier features '''
    X = np.cos(np.dot(G_transpose, np.transpose(X)) + b) #dim of X is 5000,60000
    return np.transpose(X) #60000,5000



if __name__ == "__main__":
    print("The data has been raisen to dimension {0}".format(d))
    (X_train, labels_train), (X_test, labels_test) = load_dataset()
    y_train = one_hot(labels_train)
    y_test = one_hot(labels_test)
    X_train, X_test = phi(X_train), phi(X_test)
    start_time = time.time()
    model = train(X_train, y_train, reg=0.1)
    print("Training through closed form solution takes :{0}".format(time.time() - start_time))
    pred_labels_train = predict(model, X_train)
    pred_labels_test = predict(model, X_test)
    print("Closed form solution")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_train, pred_labels_train)))
    print("Test accuracy: {0}".format(metrics.accuracy_score(labels_test, pred_labels_test)))

    start_time = time.time()
    model = train_gd(X_train, y_train, alpha=1e-3, reg=0.1, num_iter=20000)[-1]
    print("Training though batch gradient descent takes :{0}".format(time.time() - start_time))
    pred_labels_train = predict(model, X_train)
    pred_labels_test = predict(model, X_test)
    print("Batch gradient descent")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_train, pred_labels_train)))
    print("Test accuracy: {0}".format(metrics.accuracy_score(labels_test, pred_labels_test)))



    model = train_sgd(X_train, y_train, alpha=1e-3, reg=0.1, num_iter=500000)[-1]
    print("Training though stochastic gradient descent takes :{0}".format(time.time() - start_time))
    pred_labels_train = predict(model, X_train)
    pred_labels_test = predict(model, X_test)
    print("Stochastic gradient descent")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_train, pred_labels_train)))
    print("Test accuracy: {0}".format(metrics.accuracy_score(labels_test, pred_labels_test)))

```

```
#plot the error against the num_iter
WgdList = train_gd(X_train, y_train, alpha=1e-3, reg=0, num_iter=40000)
WsdList = train_sgd(X_train, y_train, alpha=1e-3, reg=0, num_iter=40000)
WgdErrorList = [1-metrics.accuracy_score(labels_train, predict(model, X_train)) for model in WgdList]
WsdErrorList = [1-metrics.accuracy_score(labels_train, predict(model, X_train)) for model in WsdList]
plt.figure()
x = np.arange(100,40000 + 1, 100)
print(x)
print(WgdErrorList)
print(WsdErrorList)

plt.plot(x, WgdErrorList, "r")
plt.plot(x, WsdErrorList, "b")
plt.axis([100,40000,0,0.3])
plt.title("red: gd error against Num_iter  blue: sgd error against Num_iter")
plt.show()
```