Source Codes

Notebook.py

```python
# %%
import numpy as np
import matplotlib.pyplot as plt
import random
import csv
from utils.data_load import load
import codes
# Load matplotlib images inline
%matplotlib inline
# These are important for reloading any code you write in external .py files.
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

# %% [markdown]
# # Problem 4: Linear Regression
# Please follow our instructions in the same order to solve the linear regresssion problem.
#
# Please print out the entire results and codes when completed.

# %%
def get_data():
    """
    Load the dataset from disk and perform preprocessing to prepare it for the linear regression problem.
    """
    X_train, y_train = load('./data/regression/regression_train.csv')
    X_test, y_test = load('./data/regression/regression_test.csv')
    X_valid, y_valid = load('./data/regression/regression_valid.csv')
    return X_train, y_train, X_test, y_test, X_valid, y_valid

X_train, y_train, X_test, y_test, X_valid, y_valid= get_data()


print('Train data shape: ', X_train.shape)
print('Train target shape: ', y_train.shape)
print('Test data shape: ',X_test.shape)
print('Test target shape: ',y_test.shape)
print('Valid data shape: ',X_valid.shape)
print('Valid target shape: ',y_valid.shape)

# %%
## PART (a):
## Plot the training and test data ##

plt.plot(X_train, y_train,'o', color='black')
plt.plot(X_test, y_test,'o', color='blue')
plt.xlabel('Input')
plt.ylabel('Target')
plt.show()
```

```
50
51  # %% [markdown]
52  # ## Training Linear Regression
53  # In the following cells, you will build a linear regression. You will implement
    its loss function, then subsequently train it with gradient descent. You will
    choose the learning rate of gradient descent to optimize its classification
    performance. Finally, you will get the opimal solution using closed form
    expression.
54
55  # %%
56  from codes.Regression import Regression
57
58  # %%
59  ## PART (c):
60  ## Complete loss_and_grad function in Regression.py file and test your results.
61  regression = Regression(m=1, reg_param=0)
62  loss, grad = regression.loss_and_grad(X_train,y_train)
63  print('Loss value',loss)
64  print('Gradient value',grad)
65
66  ##
67
68  # %%
69  ## PART (d):
70  ## Complete train_LR function in Regression.py file
71  loss_history, theta = regression.train_LR(X_train,y_train, alpha=1e-2, B=30,
    num_iters=10000)
72  plt.plot(loss_history)
73  plt.xlabel('iterations')
74  plt.ylabel('Loss function')
75  plt.show()
76  print(theta)
77  print('Final loss:',loss_history[-1])
78
79  # %%
80  ## PART (d) (Different Learning Rates):
81  from numpy.linalg import norm
82  alphas = [1e-1, 1e-2, 1e-3, 1e-4]
83  losses = np.zeros((len(alphas),10000))
84  # ================================================================ #
85  # YOUR CODE HERE:
86  # Train the Linear regression for different learning rates
87  # ================================================================ #
88
89  for i in range(0, len(alphas)):
90      loss_history, theta = regression.train_LR(X_train,y_train, alpha=alphas[i], B=
    30, num_iters=10000)
91      losses[i] = loss_history
92
93  # ================================================================ #
94  # END YOUR CODE HERE
95  # ================================================================ #
96  fig = plt.figure()
97  for i, loss in enumerate(losses):
98      plt.plot(range(10000), loss, label='alpha='+str(alphas[i]))
```

```
plt.xlabel('Iterations')
plt.ylabel('Training loss')
plt.legend()
plt.show()

# %%
## PART (d) (Different Batch Sizes):
from numpy.linalg import norm
Bs = [1, 10, 20, 30]
losses = np.zeros((len(Bs),10000))
# ================================================================ #
# YOUR CODE HERE:
# Train the Linear regression for different learning rates
# ================================================================ #

for i in range(0, len(Bs)):
    loss_history, theta = regression.train_LR(X_train,y_train, alpha=1e-2, B=Bs[i], num_iters=10000)
    losses[i] = loss_history

# ================================================================ #
# END YOUR CODE HERE
# ================================================================ #
fig = plt.figure()
for i, loss in enumerate(losses):
    plt.plot(range(10000), loss, label='B='+str(Bs[i]))
plt.xlabel('Iterations')
plt.ylabel('Training loss')
plt.legend()
plt.show()
fig.savefig('./LR_Batch_test.pdf')

# %%
## PART (e):
## Complete closed_form function in Regression.py file
loss_2, theta_2 = regression.closed_form(X_train, y_train)
print('Optimal solution loss',loss_2)
print('Optimal solution theta',theta_2)

# %%
## PART (f):
train_loss=np.zeros((10,1))
valid_loss=np.zeros((10,1))
test_loss=np.zeros((10,1))
# ================================================================ #
# YOUR CODE HERE:
# complete the following code to plot both the training, validation
# and test loss in the same plot for m range from 1 to 10
# ================================================================ #

for m in range(1, 11):
    regression = Regression(m = m)
    train_loss[m - 1] = regression.closed_form(X_train, y_train)[0]
    test_loss[m - 1] = regression.loss_and_grad(X_test, y_test)[0]
```

```
    valid_loss[m - 1] = regression.loss_and_grad(X_valid, y_valid)[0]


# ============================================================ #
# END YOUR CODE HERE
# ============================================================ #
plt.plot(train_loss, label='train')
plt.plot(valid_loss, color='purple', label='valid')
plt.plot(test_loss, color='black', label='test')
plt.legend()
plt.show()

# %%
#PART (g):
train_loss=np.zeros((10,1))
train_reg_loss=np.zeros((10,1))
valid_loss=np.zeros((10,1))
test_loss=np.zeros((10,1))
# ============================================================ #
# YOUR CODE HERE:
# complete the following code to plot the training, validation
# and test loss in the same plot for m range from 1 to 10
# ============================================================ #
lambdas = [0, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e0]
regression.m = 10
X_poly = regression.get_poly_features(X_train) # Assuming you have a method to get polynomial features

# ... [unchanged code before the loop]

for idx, reg in enumerate(lambdas):

    regression=Regression(10,reg)
    train_reg_loss[idx] = regression.closed_form(X_train,y_train)[0]
    train_loss[idx] = regression.loss_and_grad(X_train,y_train)[0]
    test_loss[idx] = regression.loss_and_grad(X_test,y_test)[0]
    valid_loss[idx] = regression.loss_and_grad(X_valid,y_valid)[0]

# ============================================================ #
# END YOUR CODE HERE
# ============================================================ #
print(test_loss)
plt.plot(np.arange(1, 11), train_loss, label='train')
plt.plot(np.arange(1, 11), valid_loss, color='purple', label='valid')
plt.plot(np.arange(1, 11), test_loss, color='black', label='test')
plt.plot(np.arange(1, 11), train_reg_loss, color = 'orange', linestyle="dashed", label='train_reg')
plt.legend()
plt.show()


```