Statistical Machine Learning: Code Listing Assignment 1

Joris van Vugt, s4279859
Luc Nies, s4136748

September 27, 2016

1 exercise1.py

```
"""
Code for assignment 1, exercise 1 of Statistical Machine Learning

Author: Joris van Vugt & Luc Nies
"""
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return 1 + np.sin(6 * (x - 2))

def noisy_f(x):
    noise = np.random.normal(0, 0.3)
    return noise + f(x)

def polynomial(X, w):
    """Evaluate a polynomial with weights w at point X"""
    return np.polyval(list(reversed(w)), X)

def RMSE(observed, target):
    """Calculate the root mean squared error"""
    error = 0.5 * np.sum((observed - target)**2)
    return np.sqrt(2*error / len(observed))

def pol_cur_fit(D, M):
    """Fit weights for an Mth order polynomial with data D"""
    x = D[0, :]
    t = D[1, :]
    A = np.zeros((M, M))
    for i in range(M):
        for j in range(M):
            A[i, j] = np.sum(x ** (i+j))
    T = np.zeros(M)
    for i in range(M):
        T[i] = np.sum(t * x**i)
    w = np.linalg.solve(A, T)
```

```
    return w

def pol_cur_fit_reg(D, M, l=0.01):
    """Fit regularized weights for an Mth order polynomial with data D"""
    x = D[0, :]
    t = D[1, :]
    A = np.zeros((M, M))
    for i in range(M):
        for j in range(M):
            A[i, j] = np.sum(x ** (i+j))
    A += l * np.identity(M)
    T = np.zeros(M)
    for i in range(M):
        T[i] = np.sum(t * x**i)
    w = np.linalg.solve(A, T)
    return w

def generate_data(n):
    """Generate n data points"""
    return [noisy_f(x) for x in np.linspace(0, 1, n)]


def plot_data():
    """Plot data for exercise 1.1"""
    D = generate_data(10)
    T = generate_data(100)
    plt.scatter(np.linspace(0, 1, 10), D)
    X = np.linspace(0, 1, 100)
    y = [f(x) for x in X]
    plt.plot(X, y)
    plt.savefig('sin_d.png')
    plt.show()

def plot_curvefitting():
    """Plot fitted curve for different values of M"""
    N = 10
    N_test = 10
    T = generate_data(N_test)
    D = generate_data(N)
    training_data = np.vstack((np.linspace(0, 1, N), D))
    test_data = np.vstack((np.linspace(0, 1, N_test), T))

    X = np.linspace(0, 1, 100)
    y = [f(x) for x in X]
    # Calculate and plot the errors of different m values (for exercise 1.3)
    for m in range(10):
        w = pol_cur_fit(training_data, m)
        fitted_curve = polynomial(X, w)

        rms_train = RMSE(polynomial(training_data[0, :], w), training_data[1, :])
        rms_test = RMSE(polynomial(test_data[0, :], w), test_data[1, :])
```

```
        plt.plot(X, fitted_curve, 'b', label='Fitted Curve')
        plt.plot(X, y, 'r', label='True function')
        plt.scatter(training_data[0, :], training_data[1, :], c='g', label='Noisy observati
        plt.title('M=%d: train RMSE=%.2f, test RMSE=%.2f' % (m, rms_train, rms_test))
        plt.xlabel(r'$x$')
        plt.ylabel(r'$y$')
        plt.legend()
        plt.savefig('images/fit_m%d_n_%d.png' % (m, training_data.shape[1]))
        plt.show()

def reg_test():
    """Plot the effect of regularization"""
    N = 10
    N_test = 100
    X = np.linspace(0, 1, N_test)
    y = [f(x) for x in X]
    T = generate_data(N_test)
    D = generate_data(N)
    training_data = np.vstack((np.linspace(0, 1, N), D))
    test_data = np.vstack((np.linspace(0, 1, N_test), T))
    l = 0.1
    for m in range(10):
        w = pol_cur_fit_reg(training_data, m, l=l)
        w_noreg = pol_cur_fit(training_data, m)
        print('regularized w:', w)
        print('nonreg w:', w_noreg)
        fitted_curve = polynomial(X, w)

        rms_train = RMSE(polynomial(training_data[0, :], w), training_data[1, :])
        rms_test = RMSE(polynomial(test_data[0, :], w), test_data[1, :])

        plt.plot(X, fitted_curve, 'b', label='Fitted Curve')
        plt.plot(X, y, 'r', label='True function')
        plt.scatter(training_data[0, :], training_data[1, :], c='g', label='Noisy observati
        plt.title(r'M=%d: train RMSE=%.2f, test RMSE=%.2f, $\lambda=%.1g$' % (m, rms_train
        plt.xlabel(r'$x$')
        plt.ylabel(r'$y$')
        plt.legend()
        plt.savefig('images/fit_m%d_n_%d_reg.png' % (m, training_data.shape[1]))
        plt.show()


def train_vs_test_reg():
    """Tests and plots the difference between the train and testset (with regularization)"
    ls = np.arange(-40, -20)
    exp_l = np.exp(ls)
    errors_train = []
    errors_test = []

    N = 10
```

```
    N_test = 10
    T = generate_data(N_test)
    D = generate_data(N)
    training_data = np.vstack((np.linspace(0, 1, N), D))
    test_data = np.vstack((np.linspace(0, 1, N_test), T))

    for l in exp_l:
        w = pol_cur_fit_reg(training_data, 9, l=l)
        rms_train = RMSE(polynomial(training_data[0, :], w), training_data[1, :])
        rms_test = RMSE(polynomial(test_data[0, :], w), test_data[1, :])
        errors_train.append(rms_train)
        errors_test.append(rms_test)

    plt.plot(ls, errors_test, 'r', label='Test')
    plt.plot(ls, errors_train, 'b', label='Training')
    plt.ylim([0, 0.4])
    plt.legend()
    plt.xlabel(r'$\ln {\lambda}$')
    plt.ylabel(r'$E_{RMS}}$')
    plt.show()

if __name__ == '__main__':
    train_vs_test_reg()
```

2 exercise2.py

```
"""
Code for assignment1, exercise 2 of Statistical Machine Learning

Author: Joris van Vugt & Luc Nies
"""
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def h(x, y):
    return 100 * (y - x**2)**2 + (1 - x)**2

def surface_plot():
    """Surface plot of h"""
    X = np.linspace(-2, 2, 100)
    Y = np.linspace(-1, 3, 100)
    [x, y] = np.meshgrid(X, Y)
    z = h(x, y)

    plt.style.use('classic')
    fig = plt.figure()
    ax = fig.gca(projection='3d')

    ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=plt.cm.viridis, linewidth=0, antial
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    ax.set_zlabel('$h(x, y)$')
    plt.show()

def gradient_descent(eta=0.005, start_x=0, start_y=0, max_iter=500):
    """
    Do gradient descent on h. Stops when h stops decreasing.
    Note that the minimum of h is at (1, 1)

    parameters:
    eta is the learning rate
    start_x and start_y represent the starting point
    max_iter is the maximum number of iterations before stopping

    returns:
    List of all intermediary x, y and h(x, y)
    """
    x_list, y_list, h_list  = [start_x], [start_y], [h(start_x, start_y)]
    x = start_x
    y = start_y
    for i in range(max_iter):
        x_temp = x - eta * (400 * x * (x**2-y) - 2 + 2 * x)
        y = y - eta * (200 * (y - x**2))
        x = x_temp
```

```
        h_new = h(x,y)
        if h_new >= h_list[-1]:
            # Stop when h hasn't decreased in the last step
            break
        x_list.append(x)
        y_list.append(y)
        h_list.append(h(x,y))

    return x_list, y_list, h_list

def contour_trajectory_plot():
    """
    Plot the contour of h and the trajectory
    taken with gradient descent for different values of eta
    """
    X = np.linspace(-2, 2, 100)
    Y = np.linspace(-1, 3, 100)
    [x, y] = np.meshgrid(X, Y)
    z = h(x, y)
    etas = [0.0001, 0.001, 0.005, 0.01]
    levels = np.linspace(z.min(), z.max(), 100)

    for eta in etas:
        x_list, y_list, h_list = gradient_descent(start_x=0, start_y=0, eta=eta, max_iter=
        plt.plot(x_list, y_list, c='r', label=str(eta), linewidth=2.0)
        plt.contourf(x,y,z, cmap=plt.cm.viridis, levels=levels)
        plt.title('$\eta = {}$ n_steps = {}'.format(eta, len(h_list)))
        plt.xlabel('$x$')
        plt.ylabel('$y$')
        plt.colorbar(label='$h$')
        plt.savefig('eta_{}.png'.format(str(eta).replace('.', '_')))
        plt.show()


if __name__ == '__main__':
    contour_trajectory_plot()
```