PRML HOMEWORK 1

TADIPATRI UDAY KIRAN REDDY
EE19BTECH11038

January 30, 2022

HW-1

Apply Newtons method to steepest-descent algorithm to the optimal step size η, and check how many iterations are required for convergence.

$$\overline{w}^{new} = \overline{w}^{old} + \eta \mathbf{X}^T(\mathbf{t} - \mathbf{X}\overline{w})|_{\overline{w}=\overline{w}^{old}}$$

We find that optimal step size η should be the inverse of Hessian matrix, $\left(\frac{\partial^2 J(\overline{w})}{\partial \overline{w} \partial \overline{w}^T}\right)^{-1} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}$.
Using substituting the optimal η in update equation we get,
#Iteration = 1

$$\overline{w}^{new} = \overline{w}^{old} + \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T(\mathbf{t} - \mathbf{X}\overline{w})|_{\overline{w}=\overline{w}^{old}}$$
$$\implies \overline{w}^{new} = \left(\mathbf{I} - \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{X}\right)\overline{w}_{old} + \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t}$$
$$\implies \overline{w}_{new} = w^*$$

So theoritically only one iteration is sufficient with optimal eta to make the solution to converge to optimal weights.
Now implementing the steepest-descent algorithm in python to verify number of iteration taken to converge.

```python
import numpy as np
import matplotlib.pyplot as plt

# Random linear data
def gen_data(N):
  X = np.linspace(-4, 4, N)
  X = X.reshape(N, 1)
  t = 3.879*X - 65
  t = t + np.random.rand(N, 1)
  X = X + np.random.rand(N, 1)
  return t, X

# J(w)
def cost(t, X, w):
  e = t - X@w
  return 0.5*e.T@e

N = 100

t, X_data = gen_data(N)

ones = np.ones((N, 1))
X = np.hstack((X_data, ones))

w = np.random.rand(2, 1)
```

```
w_init = w


# GDA
def steepest_decent(t, X, w, eta):
  # print(w.shape, (X.T@(t-X@w)).shape)
  if type(eta) is np.ndarray:
    return w + eta@(X.T@(t-X@w))
  else:
    return w + eta*(X.T@(t-X@w))

tolerance = 1e-3
def converged(w_old, w_new, tolerance):
  x = np.linalg.norm(w_old-w_new)/np.linalg.norm(w_old)
  return x <= tolerance

itr = 0

eta = np.linalg.inv(X.T@X)

err_data = []
err_data.append(np.array(([w[0, 0], w[1, 0], cost(t, X, w)[0, 0]])))

w_star = np.linalg.inv(X.T@X)@(X.T)@t



while not converged(w, w_star, tolerance):
  w = steepest_decent(t, X, w, eta)
  itr += 1
  err_data.append(np.array(([w[0, 0], w[1, 0], cost(t, X, w)[0, 0]])))

err_data = np.array(err_data).T

print("Number of iteration it took to converge is ", itr)
print("Initial weights: {}".format(w_init))
print("Final weights: {}".format(w))
print("Optimal weights: {}".format(w_star))

w_1 = np.linspace(-10, 10, 100)
w_0 = np.linspace(-100, 0, 100)

XX, YY = np.meshgrid(w_1, w_0)

ZZ = np.empty_like(XX)

for i in range(XX.shape[0]):
  for j in range(XX.shape[1]):
    w_ = np.array([XX[i, j], YY[i, j]]).reshape(2, 1)
    e_ = t - X@w_
    ZZ[i, j] = 0.5*e_.T@e_

plt.figure()
plt.title("Data points")
plt.plot(X_data[:, 0], t[:, 0], ".")
plt.plot(np.linspace(-4, 4, 100), 3.879*np.linspace(-4, 4, 100) - 65, "--", label="Actual")
plt.plot(np.linspace(-4, 4, 100), w[0, 0]*np.linspace(-4, 4, 100) + w[1, 0], "--", label="
    Predicted")
plt.grid()
plt.legend()
plt.xlabel("x")
plt.ylabel("t")

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(XX, YY, ZZ, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
ax.plot(err_data[0, :], err_data[1, :], err_data[2, :])
ax.scatter(w_star[0, 0], w_star[1, 0], cost(t, X, w_star)[0, 0], label=r'$w^*$')
```

```
ax.scatter(w_init[0, 0], w_init[1, 0], cost(t, X, w_init)[0, 0], label=r'$w_{init}$')
ax.set_xlabel('w_1')
ax.set_ylabel('w_0')
ax.set_zlabel('J(w)');
ax.legend()
plt.show()
```

HW-2

Suppose you are experimenting with L1 and L2 regularization. Further imagine that you are running gradient descent and at some iteration your weight vector is $\overline{w} = [1, \epsilon] \in R^2$ where $\epsilon > 0$ is very small. With the help of this example explain why L2 norm does not encourage sparsity i.e., it will not try to drive ϵ to 0 to produce a sparse weight vector. Give mathematical explanation.

HW-3

Till now we have been considering a scalar target t from a vector of input observations $\overline{x}$. How do you extend this approach for regressing a vector of targets $\bar{t} = (t1, t2, ..., tp)$. Derive the close form solutions and write sequential update equations using SGD.