Suhail Basalama - Machine Learning - D. Lu Zhang – HW2

Implementation of SVM via Gradient Descent

1- Equation for $\nabla_b J(w,b)$ is as follows:

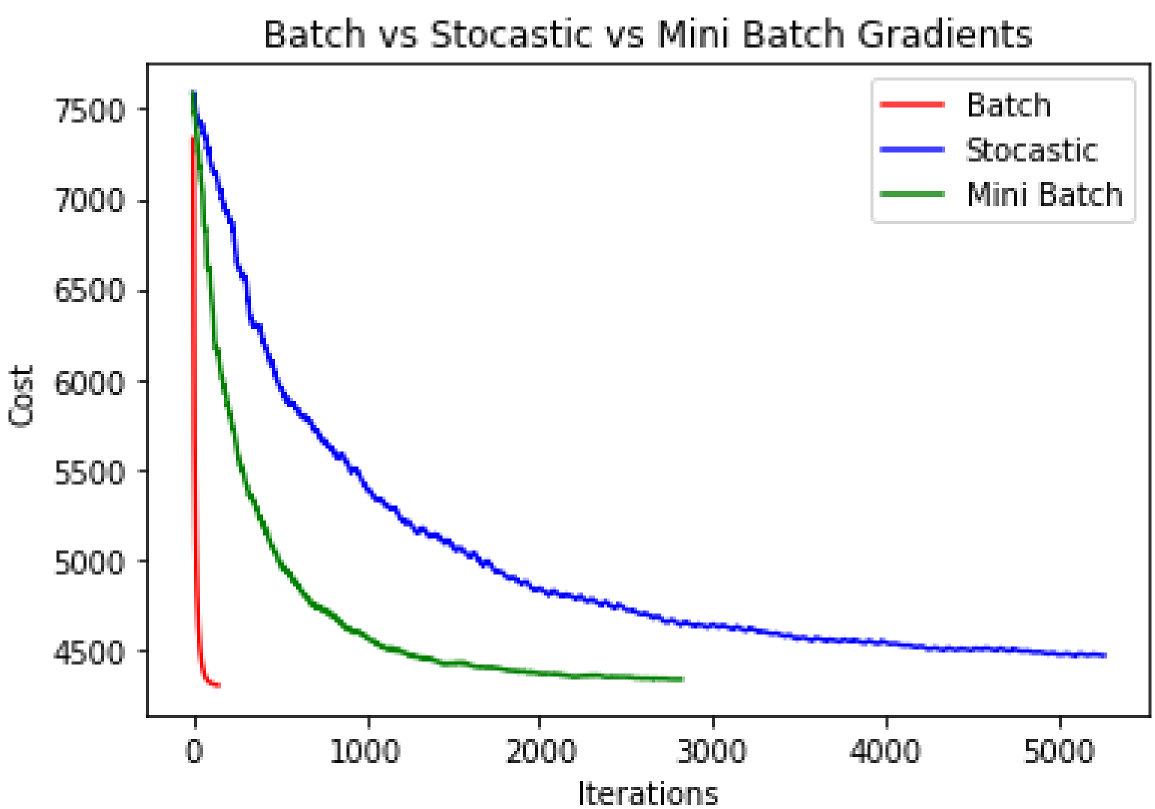
$$\frac{\partial J(w,b)}{\partial b} = b + C\sum_{i=1}^{m}\frac{\partial L\left(x^{(i)},y^{(i)}\right)}{\partial b}$$

$$\frac{\partial L\left(x^{(i)},y^{(i)}\right)}{\partial b} = \begin{cases} 0, & if\ y^{(i)}\left(x^{(i)}w+b\right) \geq 1 \\ -y^{(i)}, & otherwise \end{cases}$$

2- Plot of cost vs iterations for the different algorithms:

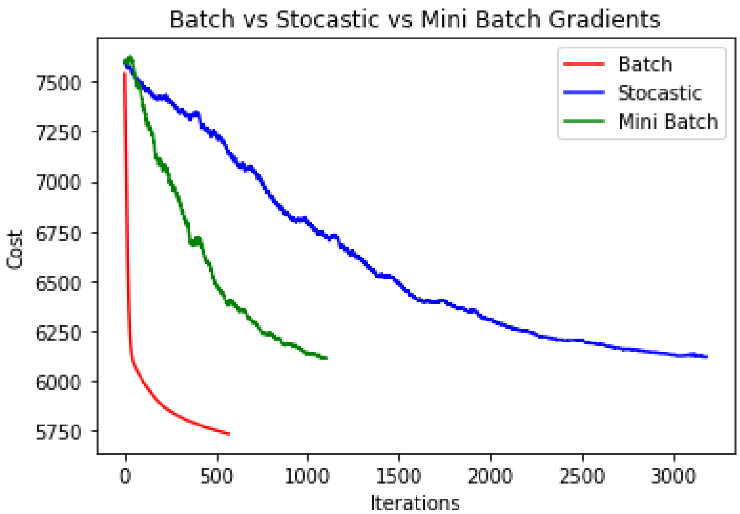
a- With regularized data.

```
Initial Cost                                 7600.0
Final   Cost using Batch_SVM                 4303.45042125608
Final   Cost using Stocastic_SVM             4468.824789794449
Final   Cost using Mini_Batch_SVM            4336.301431078447
```



b- With raw data:

```
Initial Cost                        7600.0
Final   Cost using Batch_SVM        5735.542158321961     time to converge      2.4876219995348947  ms
Final   Cost using Stocastic_SVM    6121.716901668114     time to converge      0.08602900015830528  ms
Final   Cost using Mini_Batch_SVM   6114.434279521937     time to converge      0.27650300216919277  ms
```



3- Convergence times:

```
Initial Cost                       7600.0
Final Cost using Batch_SVM         5735.5 time to converge 2.4876 ms
Final Cost using Stocastic_SVM     6075.4 time to converge 0.0860 ms
Final Cost using Mini_Batch_SVM    6084.3 time to converge 0.2765 ms
```

4- Source Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import timeit



my_data = pd.read_csv('data.txt',
sep='\t')#,names=["f1","f2","f3","f4","f5","f6","f7","f8","f9","f10","f11","f
12","f13","f14","f15","l"]) #read the data


#prepare X matrix
X = my_data.iloc[:,0:8].values
#ones = np.ones([X.shape[0],1])
#X = np.concatenate((ones,X),axis=1) #X = pd.DataFrame.from_records(X)
##prepare y matrix
```

```
y = my_data.iloc[:,8:9].values #.values converts it from
pandas.core.frame.DataFrame to numpy.ndarray
##prepare w matrix
w = np.zeros([1,8])
b = 0

C = 10


#cost/loss function
def Cost(X,y,w,b,C):

    sum1 = X @ w.T + b
    sum1 = y * sum1
    sum1 = 1-sum1
    sum1 = sum1.clip(min=0)
    sum2 = np.power(w,2)
    return (np.sum(sum2)/2) + C*np.sum(sum1)

print("Initial\tCost\t\t\t\t",Cost(X,y,w,b,10))

def L_w(X,y,w,b,j):
    z = X @ w.T + b
    z = y*z
    v = np.zeros([760,1])
    Xj = X[:,j]
    Xj = Xj.reshape(len(X),1)
    z = np.where(z>=1,v,-1*y*Xj)
    return z

def L_b(y,w,b):
    z = X @ w.T + b
    z = y*z
    v = np.zeros([760,1])
    z = np.where(z>=1,v,-1*y)
    return z

def Batch_Gradient_w(X,y,w,b,j):
    return w[0,j]+C*np.sum(L_w(X,y,w,b,j))

def Batch_Gradient_b(y,w,b):
    return b + C*np.sum(L_b(y,w,b))


def Stocastic_Gradient_w(X,y,w,b,j,i):
    Lw = L_w(X,y,w,b,j)
    return w[0,j] + C*Lw[i,0]

def Stocastic_Gradient_b(y,w,b,i):
    Lb = L_b(y,w,b)
    return b + C*Lb[i,0]

def Mini_Batch_Gradient_w(X,y,w,b,j,batch_size,l):
   i = int(l*batch_size+1)
   n = int(min(len(X),((l+1)*batch_size)))
   Lw = L_w(X,y,w,b,j)
   batch_sum = np.sum(Lw[i:n])
```

```
    return w[0,j]+C*batch_sum

def Mini_Batch_Gradient_b(y,w,b,batch_size,l):
    i = int(l*batch_size+1)
    n = int(min(len(X),((l+1)*batch_size)))
    Lb = L_b(y,w,b)
    batch_sum = np.sum(Lb[i:n])
    return b + C*batch_sum

#helper function for the batch gradient and used in other svms too
def dCost(cost,k):
    if(k != 0):
        return (abs(cost[k-1]-cost[k])*100)/cost[k-1]
    else: return 1

def Batch_SVM(X,y,w,b,C,alpha,epsilon):

    k=0
    cost = []
    tempCost = 10
    while(tempCost>epsilon):

        #update weights and bias with batch gradient descent
        for j in range(len(w[0])):
          w[0][j] = w[0][j] - alpha*Batch_Gradient_w(X,y,w,b,j)
        b = b - alpha*Batch_Gradient_b(y,w,b)

        #calculate the cost criteria
        cost.append(Cost(X,y,w,b,C))
        tempCost = dCost(cost,k)

        #increment loop parameters
        k += 1
    return w,cost,k,b

def Stocastic_SVM(X,y,w,b,C,alpha,epsilon):

    k=0
    i=0
    cost = []
    costk = [10]
    tempCost = 10.0
    m = len(X)
    while(costk[k]>epsilon):

        #update weights and bias with stocastic gradient descent
        for j in range(len(w[0])):
          w[0][j] = w[0][j] - alpha*Stocastic_Gradient_w(X,y,w,b,j,i)
        b = b - alpha*Stocastic_Gradient_b(y,w,b,i)

        #calculate the cost criteria
        cost.append(Cost(X,y,w,b,C))
        tempCost = dCost(cost,k)
        costk.append(0.5*costk[k-1]+0.5*tempCost)

        #increment loop parameters
        i = (i+1)%m
```

```
        k += 1
    return w,cost,k,b

def Mini_Batch_SVM(X,y,w,b,C,alpha,epsilon):

    l = 0
    k = 0
    batch_size = 4
    cost = []
    costk = [10]

    while(costk[k]>epsilon):

        #update weights and bias with mini batch gradient descent
        for j in range (len(w[0])):
            w[0][j] = w[0][j] -
alpha*Mini_Batch_Gradient_w(X,y,w,b,j,batch_size,l)
        b = b - alpha*Mini_Batch_Gradient_b(y,w,b,batch_size,l)

        #calculate the cost criteria
        cost.append(Cost(X,y,w,b,C))
        tempCost = dCost(cost,k)
        costk.append(0.5*costk[k-1]+0.5*tempCost)

        #increment loop parameters
        l = (l+1) % ((len(X)+batch_size-1)/batch_size)
        k = k+1

    return w,cost,k,b

w = np.zeros([1,8])
b = 0
alpha = 0.000000001
epsilon = 0.004
start = timeit.timeit()
w_,cost1,count1,b_ = Batch_SVM(X,y,w,b,C,alpha,epsilon)
end = timeit.timeit()
print("Final\tCost using Batch_SVM\t\t",Cost(X,y,w_,b_,C),"\t time to
converge\t",1000*(start - end)," ms")

my_data = pd.read_csv('data.txt',
sep='\t')#,names=["f1","f2","f3","f4","f5","f6","f7","f8","f9","f10","f11","f
12","f13","f14","f15","l"]) #read the data
my_data = my_data.sample(frac=1)
X = my_data.iloc[:,0:8].values
y = my_data.iloc[:,8:9].values

w = np.zeros([1,8])
b = 0
alpha = 0.00000001
epsilon = 0.0003
start = timeit.timeit()
w_,cost2,count2,b_ = Stocastic_SVM(X,y,w,b,C,alpha,epsilon)
print("Final\tCost using Stocastic_SVM\t",Cost(X,y,w_,b_,C),"\t time to
converge\t",1000*(start - end)," ms")
end = timeit.timeit()
```

```
my_data = pd.read_csv('data.txt',
sep='\t')#,names=["f1","f2","f3","f4","f5","f6","f7","f8","f9","f10","f11","f
12","f13","f14","f15","l"]) #read the data
my_data = my_data.sample(frac=1)
X = my_data.iloc[:,0:8].values
y = my_data.iloc[:,8:9].values
w = np.zeros([1,8])
b = 0
alpha = 0.00000001
epsilon = 0.004
start = timeit.timeit()
w_,cost3,count3,b_ = Mini_Batch_SVM(X,y,w,b,C,alpha,epsilon)
print("Final\tCost using Mini_Batch_SVM\t", Cost(X,y,w_,b_,C),"\t time to
converge\t",1000*(start - end)," ms")
end = timeit.timeit()



#plot the costs
fig, ax = plt.subplots()
ax.plot(np.arange(count1), cost1, color = 'r', label = 'Batch')
ax.plot(np.arange(count2), cost2, color = 'b', label = 'Stocastic')
ax.plot(np.arange(count3), cost3, color = 'g', label = 'Mini Batch')

ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Batch vs Stocastic vs Mini Batch Gradients')
plt.legend(loc='best')
```