Programming Assignment: Numerical Optimization for Logistic Regression.

Name: Chris Flanagan

0. You will do the following:

1. Read the lecture note: click here

2. Read, complete, and run my code.

3. **Implement mini-batch SGD** and evaluate the performance.

4. Convert the .IPYNB file to .HTML file.

    - The HTML file must contain **the code** and **the output after execution**.

    - Missing **the output after execution** will not be graded.

1. Upload this .HTML file to your Google Drive, Dropbox, or your Github repo. (If you submit the file to Google Drive or Dropbox, you must make the file "open-access". The delay caused by "deny of access" may result in late penalty.)

2. On Canvas, submit the Google Drive/Dropbox/Github link to the HTML file.

Grading criteria:

1. When computing the `gradient` and `objective function value` using a batch of samples, use **matrix-vector multiplication** rather than a FOR LOOP of **vector-vector multiplications**.

2. Plot `objective function value` against `epochs`. In the plot, compare GD, SGD, and MB-SGD (with $b = 8$ and $b = 64$). The plot must look reasonable.

In [1]:

```
import math
```

1. Data processing

- Download the Diabete dataset from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/diabetes
- Load the data using sklearn.
- Preprocess the data.

1.1. Load the data

In [2]:
```
from sklearn import datasets
import numpy

x_sparse, y = datasets.load_svmlight_file('diabetes')
x = x_sparse.todense()

print('Shape of x: ' + str(x.shape))
print('Shape of y: ' + str(y.shape))
```

```
Shape of x: (768, 8)
Shape of y: (768,)
```

1.2. Partition to training and test sets

In [3]:
```
# partition the data to training and test sets
n = x.shape[0]
n_train = 640
n_test = n - n_train

rand_indices = numpy.random.permutation(n)
train_indices = rand_indices[0:n_train]
test_indices = rand_indices[n_train:n]

x_train = x[train_indices, :]
x_test = x[test_indices, :]
y_train = y[train_indices].reshape(n_train, 1)
y_test = y[test_indices].reshape(n_test, 1)

print('Shape of x_train: ' + str(x_train.shape))
print('Shape of x_test: ' + str(x_test.shape))
print('Shape of y_train: ' + str(y_train.shape))
print('Shape of y_test: ' + str(y_test.shape))
```

```
Shape of x_train: (640, 8)
Shape of x_test: (128, 8)
Shape of y_train: (640, 1)
Shape of y_test: (128, 1)
```

1.3. Feature scaling

Use the standardization to trainsform both training and test features

In [4]:
```
# Standardization
import numpy

# calculate mu and sig using the training set
d = x_train.shape[1]
mu = numpy.mean(x_train, axis=0).reshape(1, d)
sig = numpy.std(x_train, axis=0).reshape(1, d)

# transform the training features
x_train = (x_train - mu) / (sig + 1E-6)

# transform the test features
```

```
x_test = (x_test - mu) / (sig + 1E-6)

print('test mean = ')
print(numpy.mean(x_test, axis=0))

print('test std = ')
print(numpy.std(x_test, axis=0))
```

```
test mean =
[[ 0.00226527 -0.0298446  -0.09078006 -0.22306472 -0.0411703  -0.09631385
  -0.00306507  0.03497572]]
test std =
[[0.848527   0.89118662 1.05376247 0.97332575 1.0162102  1.14688175
  0.94875956 0.95505409]]
```

1.4. Add a dimension of all ones

In [5]:

```
n_train, d = x_train.shape
x_train = numpy.concatenate((x_train, numpy.ones((n_train, 1))), axis=1)

n_test, d = x_test.shape
x_test = numpy.concatenate((x_test, numpy.ones((n_test, 1))), axis=1)

print('Shape of x_train: ' + str(x_train.shape))
print('Shape of x_test: ' + str(x_test.shape))
```

```
Shape of x_train: (640, 9)
Shape of x_test: (128, 9)
```

2. Logistic regression model

The objective function is $Q(w; X, y) = \frac{1}{n} \sum_{i=1}^{n} \log \Big(1 + \exp \big(- y_i x_i^T w \big) \Big) + \frac{\lambda}{2} \| w \|_2^2$.

In [6]:

```
# Calculate the objective function value
# Inputs:
#     w: d-by-1 matrix
#     x: n-by-d matrix
#     y: n-by-1 matrix
#     lam: scalar, the regularization parameter
# Return:
#     objective function value (scalar)
def objective(w, x, y, lam):
    n, d = x.shape
    yx = numpy.multiply(y, x) # n-by-d matrix
    yxw = numpy.dot(yx, w) # n-by-1 matrix
    vec1 = numpy.exp(-yxw) # n-by-1 matrix
    vec2 = numpy.log(1 + vec1) # n-by-1 matrix
    loss = numpy.mean(vec2) # scalar
    reg = lam / 2 * numpy.sum(w * w) # scalar
    return loss + reg
```

In [7]:

```
# initialize w
d = x_train.shape[1]
```

```
w = numpy.zeros((d, 1))

# evaluate the objective function value at w
lam = 1E-6
objval0 = objective(w, x_train, y_train, lam)
print('Initial objective function value = ' + str(objval0))
```

```
Initial objective function value = 0.6931471805599453
```

3. Numerical optimization

3.1. Gradient descent

The gradient at w is $g = - \frac{1}{n} \sum_{i=1}^{n} \frac{y_i x_i}{1 + \exp(y_i x_i^T w)} + \lambda w$

In [8]:

```
# Calculate the gradient
# Inputs:
#     w: d-by-1 matrix
#     x: n-by-d matrix
#     y: n-by-1 matrix
#     lam: scalar, the regularization parameter
# Return:
#     g: g: d-by-1 matrix, full gradient
def gradient(w, x, y, lam):
    n, d = x.shape
    yx = numpy.multiply(y, x) # n-by-d matrix
    yxw = numpy.dot(yx, w) # n-by-1 matrix
    vec1 = numpy.exp(yxw) # n-by-1 matrix
    vec2 = numpy.divide(yx, 1+vec1) # n-by-d matrix
    vec3 = -numpy.mean(vec2, axis=0).reshape(d, 1) # d-by-1 matrix
    g = vec3 + lam * w
    return g
```

In [9]:

```
# Gradient descent for solving logistic regression
# Inputs:
#     x: n-by-d matrix
#     y: n-by-1 matrix
#     lam: scalar, the regularization parameter
#     stepsize: scalar
#     max_iter: integer, the maximal iterations
#     w: d-by-1 matrix, initialization of w
# Return:
#     w: d-by-1 matrix, the solution
#     objvals: a record of each iteration's objective value
def grad_descent(x, y, lam, stepsize, max_iter=100, w=None):
    n, d = x.shape
    objvals = numpy.zeros(max_iter) # store the objective values
    if w is None:
        w = numpy.zeros((d, 1)) # zero initialization

    for t in range(max_iter):
        objval = objective(w, x, y, lam)
        objvals[t] = objval
        print('Objective value at t=' + str(t) + ' is ' + str(objval))
        g = gradient(w, x, y, lam)
```

```
        w -= stepsize * g

    return w, objvals
```

Run gradient descent.

In [10]:

```
lam = 1E-6
stepsize = 1.0
w, objvals_gd = grad_descent(x_train, y_train, lam, stepsize)
```

```
Objective value at t=0 is 0.6931471805599453
Objective value at t=1 is 0.5901995438510925
Objective value at t=2 is 0.5487159266278951
Objective value at t=3 is 0.5267345295206912
Objective value at t=4 is 0.5131798231812591
Objective value at t=5 is 0.504087608696939
Objective value at t=6 is 0.4976668063884439
Objective value at t=7 is 0.49297419483933314
Objective value at t=8 is 0.48945908517622005
Objective value at t=9 is 0.48677620799316473
Objective value at t=10 is 0.4846978076981804
Objective value at t=11 is 0.4830679005738614
Objective value at t=12 is 0.4817765324300812
Objective value at t=13 is 0.48074439311283895
Objective value at t=14 is 0.47991318309318004
Objective value at t=15 is 0.47923935707988063
Objective value at t=16 is 0.4786899396934321
Objective value at t=17 is 0.478239658332373
Objective value at t=18 is 0.4778689379370303
Objective value at t=19 is 0.4775624735077126
Objective value at t=20 is 0.4773081979669816
Objective value at t=21 is 0.4770965254433125
Objective value at t=22 is 0.4769197895049115
Objective value at t=23 is 0.4767718213685542
Objective value at t=24 is 0.4766476299216803
Objective value at t=25 is 0.47654315668235886
Objective value at t=26 is 0.47645508651937446
Objective value at t=27 is 0.4763807002808601
Objective value at t=28 is 0.4763177592141917
Objective value at t=29 is 0.47626441371014583
Objective value at t=30 is 0.47621913080673095
Objective value at t=31 is 0.4761806362681937
Objective value at t=32 is 0.4761478680658543
Objective value at t=33 is 0.4761199388351899
Objective value at t=34 is 0.4760961054414269
Objective value at t=35 is 0.47607574420549414
Objective value at t=36 is 0.47605833066025216
Objective value at t=37 is 0.4760434229497674
Objective value at t=38 is 0.47603064817112706
Objective value at t=39 is 0.4760196911027711
Objective value at t=40 is 0.4760102848758184
Objective value at t=41 is 0.47600220323294323
Objective value at t=42 is 0.475995254088716
Objective value at t=43 is 0.4759892741602023
Objective value at t=44 is 0.4759841244802463
Objective value at t=45 is 0.47597968664072815
Objective value at t=46 is 0.47597585964104167
Objective value at t=47 is 0.4759725572395563
Objective value at t=48 is 0.4759697057240417
```

```
Objective value at t=49 is 0.475967242031796
Objective value at t=50 is 0.47596511216225357
Objective value at t=51 is 0.47596326983466164
Objective value at t=52 is 0.47596167535146866
Objective value at t=53 is 0.47596029463466744
Objective value at t=54 is 0.4759590984077875
Objective value at t=55 is 0.47595806150071773
Objective value at t=56 is 0.4759571622582669
Objective value at t=57 is 0.47595638203645124
Objective value at t=58 is 0.47595570477306504
Objective value at t=59 is 0.4759551166212265
Objective value at t=60 is 0.47595460563637443
Objective value at t=61 is 0.4759541615086815
Objective value at t=62 is 0.4759537753340971
Objective value at t=63 is 0.47595343941828544
Objective value at t=64 is 0.4759531471085934
Objective value at t=65 is 0.4759528926499355
Objective value at t=66 is 0.4759526710610973
Objective value at t=67 is 0.47595247802848784
Objective value at t=68 is 0.4759523098148177
Objective value at t=69 is 0.4759521631805511
Objective value at t=70 is 0.47595203531630315
Objective value at t=71 is 0.4759519237846175
Objective value at t=72 is 0.47595182646979384
Objective value at t=73 is 0.47595174153462566
Objective value at t=74 is 0.4759516673830742
Objective value at t=75 is 0.47595160262804664
Objective value at t=76 is 0.4759515460635641
Objective value at t=77 is 0.4759514966407093
Objective value at t=78 is 0.4759514534468292
Objective value at t=79 is 0.4759514156875431
Objective value at t=80 is 0.47595138267116976
Objective value at t=81 is 0.47595135379524184
Objective value at t=82 is 0.4759513285348224
Objective value at t=83 is 0.4759513064323781
Objective value at t=84 is 0.475951287088997
Objective value at t=85 is 0.4759512701567705
Objective value at t=86 is 0.4759512553321803
Objective value at t=87 is 0.4759512423503572
Objective value at t=88 is 0.4759512309800938
Objective value at t=89 is 0.4759512210195106
Objective value at t=90 is 0.475951212292288
Objective value at t=91 is 0.47595120464439067
Objective value at t=92 is 0.4759511979412158
Objective value at t=93 is 0.47595119206511377
Objective value at t=94 is 0.4759511869132267
Objective value at t=95 is 0.4759511823956092
Objective value at t=96 is 0.4759511784335885
Objective value at t=97 is 0.47595117495833705
Objective value at t=98 is 0.47595117190962744
Objective value at t=99 is 0.47595116923474684
```

3.2. Stochastic gradient descent (SGD)

Define $Q_i(w) = \log \Big(1 + \exp \big(- y_i x_i^T w \big) \Big) + \frac{\lambda}{2} \| w \|_2^2$.

The stochastic gradient at w is $g_i = \frac{\partial Q_i}{\partial w} = - \frac{y_i x_i}{1 + \exp (y_i x_i^T w)} + \lambda w$.

In [11]:

```
# Calculate the objective Q_i and the gradient of Q_i
# Inputs:
#     w: d-by-1 matrix
#     xi: 1-by-d matrix
#     yi: scalar
#     lam: scalar, the regularization parameter
# Return:
#     obj: scalar, the objective Q_i
#     g: d-by-1 matrix, gradient of Q_i
def stochastic_objective_gradient(w, xi, yi, lam):


    yx = yi * xi # 1-by-d matrix
    yxw = float(numpy.dot(yx, w)) # scalar

    # calculate objective function Q_i
    loss = numpy.log(1 + numpy.exp(-yxw)) # scalar
    reg = lam / 2 * numpy.sum(w * w) # scalar
    obj = loss + reg

    # calculate stochastic gradient
    g_loss = -yx.T / (1 + numpy.exp(yxw)) # d-by-1 matrix
    g = g_loss + lam * w # d-by-1 matrix

    return obj, g
```

In [12]:

```
# SGD for solving logistic regression
# Inputs:
#     x: n-by-d matrix
#     y: n-by-1 matrix
#     lam: scalar, the regularization parameter
#     stepsize: scalar
#     max_epoch: integer, the maximal epochs
#     w: d-by-1 matrix, initialization of w
# Return:
#     w: the solution
#     objvals: record of each iteration's objective value
def sgd(x, y, lam, stepsize, max_epoch=100, w=None):
    n, d = x.shape
    objvals = numpy.zeros(max_epoch) # store the objective values
    if w is None:
        w = numpy.zeros((d, 1)) # zero initialization

    for t in range(max_epoch):
        # randomly shuffle the samples
        rand_indices = numpy.random.permutation(n)
        x_rand = x[rand_indices, :]
        y_rand = y[rand_indices, :]

        objval = 0 # accumulate the objective values
        for i in range(n):
            xi = x_rand[i, :] # 1-by-d matrix
            yi = float(y_rand[i, :]) # scalar
            obj, g = stochastic_objective_gradient(w, xi, yi, lam)
            objval += obj
            w -= stepsize * g

        stepsize *= 0.9 # decrease step size
```

```
        objval /= n
        objvals[t] = objval
        print('Objective value at epoch t=' + str(t) + ' is ' + str(objval))

    return w, objvals
```

Run SGD.

In [13]:

```
lam = 1E-6
stepsize = 0.1
w, objvals_sgd = sgd(x_train, y_train, lam, stepsize)
```

```
Objective value at epoch t=0 is 0.5367586920566291
Objective value at epoch t=1 is 0.5282720059268533
Objective value at epoch t=2 is 0.5108619957498952
Objective value at epoch t=3 is 0.5147192924741535
Objective value at epoch t=4 is 0.502725362549941
Objective value at epoch t=5 is 0.5171185200631129
Objective value at epoch t=6 is 0.5009505539738497
Objective value at epoch t=7 is 0.5019502914863259
Objective value at epoch t=8 is 0.5009684532276398
Objective value at epoch t=9 is 0.49817821333316037
Objective value at epoch t=10 is 0.4968997698418214
Objective value at epoch t=11 is 0.4954254649303499
Objective value at epoch t=12 is 0.4925278811976062
Objective value at epoch t=13 is 0.48859795519825644
Objective value at epoch t=14 is 0.4884044334028241
Objective value at epoch t=15 is 0.4903213774338635
Objective value at epoch t=16 is 0.4883896078574307
Objective value at epoch t=17 is 0.48636212154363834
Objective value at epoch t=18 is 0.4861439854174191
Objective value at epoch t=19 is 0.48421611815416876
Objective value at epoch t=20 is 0.4841156650057872
Objective value at epoch t=21 is 0.48330363672030313
Objective value at epoch t=22 is 0.4821699769810275
Objective value at epoch t=23 is 0.48189630762565494
Objective value at epoch t=24 is 0.4808856740634022
Objective value at epoch t=25 is 0.48107448898750427
Objective value at epoch t=26 is 0.48027303174240926
Objective value at epoch t=27 is 0.47912408199278633
Objective value at epoch t=28 is 0.480004938852377
Objective value at epoch t=29 is 0.4793976315226346
Objective value at epoch t=30 is 0.4789943110122361
Objective value at epoch t=31 is 0.4786504594389306
Objective value at epoch t=32 is 0.47830681071444936
Objective value at epoch t=33 is 0.47823098376864104
Objective value at epoch t=34 is 0.4780506693527899
Objective value at epoch t=35 is 0.47777795669469747
Objective value at epoch t=36 is 0.4776168651888435
Objective value at epoch t=37 is 0.477452954942765
Objective value at epoch t=38 is 0.4773023293069339
Objective value at epoch t=39 is 0.4771749755527961
Objective value at epoch t=40 is 0.4770506290011577
Objective value at epoch t=41 is 0.47693117180269484
Objective value at epoch t=42 is 0.47685157503578807
Objective value at epoch t=43 is 0.4767589630737147
Objective value at epoch t=44 is 0.4766803240022205
Objective value at epoch t=45 is 0.4766018609305786
Objective value at epoch t=46 is 0.4765427948042002
```

```
Objective value at epoch t=47 is 0.47648380080157454
Objective value at epoch t=48 is 0.4764294807997362
Objective value at epoch t=49 is 0.47637946775735696
Objective value at epoch t=50 is 0.4763401940689892
Objective value at epoch t=51 is 0.4762988021023439
Objective value at epoch t=52 is 0.4762663432105403
Objective value at epoch t=53 is 0.47623555422142594
Objective value at epoch t=54 is 0.4762064353703949
Objective value at epoch t=55 is 0.4761817799816354
Objective value at epoch t=56 is 0.47615781785091704
Objective value at epoch t=57 is 0.47613615562372075
Objective value at epoch t=58 is 0.4761193423527154
Objective value at epoch t=59 is 0.4761018778516428
Objective value at epoch t=60 is 0.4760875481519025
Objective value at epoch t=61 is 0.4760738050587455
Objective value at epoch t=62 is 0.4760615086525636
Objective value at epoch t=63 is 0.47605065113819967
Objective value at epoch t=64 is 0.47604044672703516
Objective value at epoch t=65 is 0.4760316689543571
Objective value at epoch t=66 is 0.4760236146784935
Objective value at epoch t=67 is 0.476016425596646 7
Objective value at epoch t=68 is 0.47600985763308834
Objective value at epoch t=69 is 0.47600404001723656
Objective value at epoch t=70 is 0.4759987643806709
Objective value at epoch t=71 is 0.47599402090207094
Objective value at epoch t=72 is 0.47598972431903663
Objective value at epoch t=73 is 0.4759858638794068
Objective value at epoch t=74 is 0.4759824112539313
Objective value at epoch t=75 is 0.47597929070464245
Objective value at epoch t=76 is 0.47597647579957447
Objective value at epoch t=77 is 0.4759739589375719
Objective value at epoch t=78 is 0.4759716774353245
Objective value at epoch t=79 is 0.4759696369682362
Objective value at epoch t=80 is 0.4759677905636829
Objective value at epoch t=81 is 0.47596611700606317
Objective value at epoch t=82 is 0.47596462973163867
Objective value at epoch t=83 is 0.47596329274918
Objective value at epoch t=84 is 0.47596208444020327
Objective value at epoch t=85 is 0.4759609955613904
Objective value at epoch t=86 is 0.4759600119327635
Objective value at epoch t=87 is 0.47595913251190514
Objective value at epoch t=88 is 0.47595833864529835
Objective value at epoch t=89 is 0.4759576241439631
Objective value at epoch t=90 is 0.47595698108101947
Objective value at epoch t=91 is 0.47595640244458604
Objective value at epoch t=92 is 0.4759558817303076
Objective value at epoch t=93 is 0.4759554126442456
Objective value at epoch t=94 is 0.47595499078415954
Objective value at epoch t=95 is 0.4759546110703553
Objective value at epoch t=96 is 0.4759542691429021
Objective value at epoch t=97 is 0.4759539614488606
Objective value at epoch t=98 is 0.4759536845955652
Objective value at epoch t=99 is 0.4759534354269036
```

4. Compare GD with SGD

Plot objective function values against epochs.

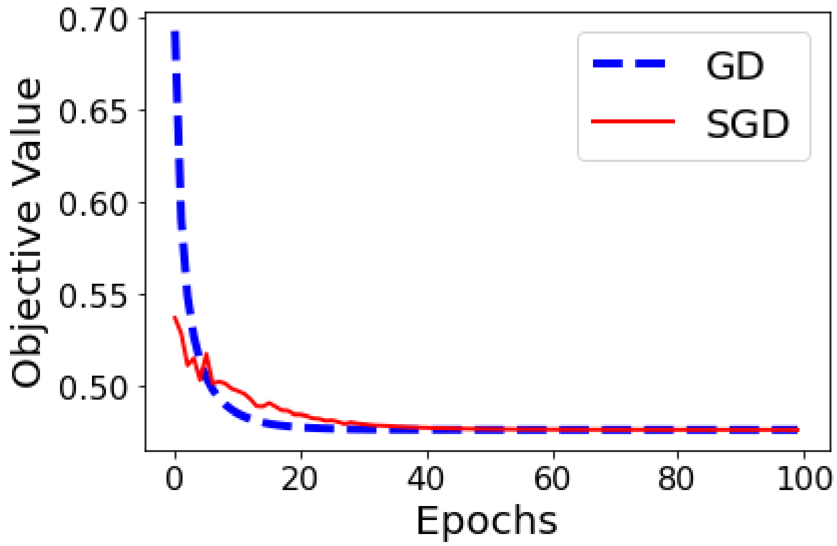
In [14]:
```
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(6, 4))

epochs_gd = range(len(objvals_gd))
epochs_sgd = range(len(objvals_sgd))

line0, = plt.plot(epochs_gd, objvals_gd, '--b', LineWidth=4)
line1, = plt.plot(epochs_sgd, objvals_sgd, '-r', LineWidth=2)
plt.xlabel('Epochs', FontSize=20)
plt.ylabel('Objective Value', FontSize=20)
plt.xticks(FontSize=16)
plt.yticks(FontSize=16)
plt.legend([line0, line1], ['GD', 'SGD'], fontsize=20)
plt.tight_layout()
plt.show()
fig.savefig('compare_gd_sgd.pdf', format='pdf', dpi=1200)
```

```
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/535934915.py:9:
MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in 3.3
and support will be removed two minor releases later
  line0, = plt.plot(epochs_gd, objvals_gd, '--b', LineWidth=4)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/535934915.py:1
0: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  line1, = plt.plot(epochs_sgd, objvals_sgd, '-r', LineWidth=2)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/535934915.py:1
1: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.xlabel('Epochs', FontSize=20)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/535934915.py:1
2: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.ylabel('Objective Value', FontSize=20)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/535934915.py:1
3: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.xticks(FontSize=16)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/535934915.py:1
4: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.yticks(FontSize=16)
```



5. Prediction

In [15]:

```
# Predict class label
# Inputs:
#     w: d-by-1 matrix
#     X: m-by-d matrix
# Return:
#     f: m-by-1 matrix, the predictions
def predict(w, X):
    xw = numpy.dot(X, w)
    f = numpy.sign(xw)
    return f
```

In [16]:

```
# evaluate training error
f_train = predict(w, x_train)
diff = numpy.abs(f_train - y_train) / 2
error_train = numpy.mean(diff)
print('Training classification error is ' + str(error_train))
```

```
Training classification error is 0.221875
```

In [17]:

```
# evaluate test error
f_test = predict(w, x_test)
diff = numpy.abs(f_test - y_test) / 2
error_test = numpy.mean(diff)
print('Test classification error is ' + str(error_test))
```

```
Test classification error is 0.2109375
```

6. Mini-batch SGD (fill the code)

6.1. Compute the objective Q_I and its gradient using a batch of samples

Define $Q_I(w) = \frac{1}{b}\sum_{i\in I}\log\Big(1+\exp\big(-y_i x_i^T w\big)\Big) + \frac{\lambda}{2}\|w\|_2^2$, where I is a set containing b indices randomly drawn from $\{1,\cdots,n\}$ without replacement.

The stochastic gradient at w is $g_I = \frac{\partial Q_I}{\partial w} = \frac{1}{b}\sum_{i\in I}\frac{-y_i x_i}{1+\exp(y_i x_i^T w)} + \lambda w$.

In [25]:

```
# Calculate the objective Q_I and the gradient of Q_I
# Inputs:
#     w: d-by-1 matrix
#     xi: b-by-d matrix
#     yi: b-by-1 matrix
#     lam: scalar, the regularization parameter
#     b: integer, the batch size
# Return:
#     obj: scalar, the objective Q_i
#     g: d-by-1 matrix, gradient of Q_i
def mb_stochastic_objective_gradient(w, xi, yi, lam, b):
    # Fill the function
    # Follow the implementation of stochastic_objective_gradient
    # Use matrix-vector multiplication; do not use FOR LOOP of vector-vector mul
    ...
    # Fill the function
    # Follow the implementation of stochastic_objective_gradient
    # Use matrix-vector multiplication; do not use FOR LOOP of vector-vector mul
    d = xi.shape[1]
    yx = numpy.multiply(yi, xi) # b-by-d matrix
    yxw = numpy.dot(yx, w) # b-by-1 matrix

    # Get objective function
    loss = numpy.mean(numpy.log(1 + numpy.exp(-yxw))) # scalar
    reg = lam / 2 * numpy.sum(w * w) # scalar
    obj = loss + reg

    # Get stochastic gradient
    vec1 = numpy.exp(yxw)
    vec2 = numpy.divide(yx, 1 + vec1)
    vec3 = -numpy.mean(vec2, axis=0).reshape(d, 1)
    g = vec3 + lam * w


    return obj, g
```

6.2. Implement mini-batch SGD

Hints:

1. In every epoch, randomly permute the n samples (just like SGD).
2. Each epoch has $\frac{n}{b}$ iterations. In every iteration, use b samples, and compute the gradient and objective using the `mb_stochastic_objective_gradient` function. In the next iteration, use the next b samples, and so on.

In [26]:

```
# Mini-Batch SGD for solving logistic regression
# Inputs:
#     x: n-by-d matrix
```

```
#     y: n-by-1 matrix
#     lam: scalar, the regularization parameter
#     b: integer, the batch size
#     stepsize: scalar
#     max_epoch: integer, the maximal epochs
#     w: d-by-1 matrix, initialization of w
# Return:
#     w: the solution
#     objvals: record of each iteration's objective value
def mb_sgd(x, y, lam, b, stepsize, max_epoch=100, w=None):
    # Fill the function
    # Follow the implementation of sgd
    # Record one objective value per epoch (not per iteration!)
    ...
    # Fill the function
    # Follow the implementation of sgd
    # Record one objective value per epoch (not per iteration!)
    n, d = x.shape
    objvals = numpy.zeros(max_epoch)
    batch_nums = math.floor(n / b)
    if w is None:
        w = numpy.zeros((d, 1))


    for t in range(max_epoch):
        # randomly shuffle the samples
        rand_indices = numpy.random.permutation(n)
        x_rand = x[rand_indices, :]
        y_rand = y[rand_indices, :]
        objval = 0 # accumulate the objective values

        # Select minibatches of b columns
        for i in range(0, n, b):
            # Select batches to feed in
            xi = x_rand[i:i+b,:] #b-by-d matrix
            yi = y_rand[i:i+b,:] #n-by-1 matrix
            obj, g = mb_stochastic_objective_gradient(w, xi, yi, lam, b)
            objval += obj
            w -= stepsize * g
        stepsize *= 0.9
        objval = objval/batch_nums
        objvals[t] = objval
        print('Objective value at epoch t=' + str(t) + ' is ' + str(objval))

    return w, objvals
```

6.3. Run MB-SGD

In [27]:

```
# MB-SGD with batch size b=8
lam = 1E-6 # do not change
b = 8 # do not change
stepsize = 0.1 # you must tune this parameter

w, objvals_mbsgd8 = mb_sgd(x_train, y_train, lam, b, stepsize)
```

```
Objective value at epoch t=0 is 0.5451212223115369
Objective value at epoch t=1 is 0.49236836469193407
```

```
Objective value at epoch t=2 is 0.48491072867490725
Objective value at epoch t=3 is 0.4835156587698647
Objective value at epoch t=4 is 0.4818469915238649
Objective value at epoch t=5 is 0.4803152905587373
Objective value at epoch t=6 is 0.4804091672013218
Objective value at epoch t=7 is 0.4798175053207291
Objective value at epoch t=8 is 0.47982286514431616
Objective value at epoch t=9 is 0.4794191460619003
Objective value at epoch t=10 is 0.4789683553501488
Objective value at epoch t=11 is 0.4786237919627959
Objective value at epoch t=12 is 0.47853944787227876
Objective value at epoch t=13 is 0.4780209066890623
Objective value at epoch t=14 is 0.4781097404977882
Objective value at epoch t=15 is 0.4778473561996459
Objective value at epoch t=16 is 0.47760926521046354
Objective value at epoch t=17 is 0.47764328344663964
Objective value at epoch t=18 is 0.47735232571017167
Objective value at epoch t=19 is 0.47721623604643915
Objective value at epoch t=20 is 0.47711999304578756
Objective value at epoch t=21 is 0.47695231851510467
Objective value at epoch t=22 is 0.4769725928113469
Objective value at epoch t=23 is 0.476784490526985
Objective value at epoch t=24 is 0.47672714031794855
Objective value at epoch t=25 is 0.4766351468128315
Objective value at epoch t=26 is 0.4765072613027265
Objective value at epoch t=27 is 0.4764799311734129
Objective value at epoch t=28 is 0.47640321018864845
Objective value at epoch t=29 is 0.4763459921694838
Objective value at epoch t=30 is 0.476312126904176
Objective value at epoch t=31 is 0.47633114678905686
Objective value at epoch t=32 is 0.4762578864459278
Objective value at epoch t=33 is 0.47620791055816963
Objective value at epoch t=34 is 0.47621542489450325
Objective value at epoch t=35 is 0.47620583454768967
Objective value at epoch t=36 is 0.4761661250589608
Objective value at epoch t=37 is 0.47612547619633705
Objective value at epoch t=38 is 0.4761273846527659
Objective value at epoch t=39 is 0.47609378598205243
Objective value at epoch t=40 is 0.4760908501603369
Objective value at epoch t=41 is 0.47606418213523183
Objective value at epoch t=42 is 0.47605783407497676
Objective value at epoch t=43 is 0.4760513011081374
Objective value at epoch t=44 is 0.47603878931716476
Objective value at epoch t=45 is 0.4760287954241367
Objective value at epoch t=46 is 0.47602334467536506
Objective value at epoch t=47 is 0.4760218669227213
Objective value at epoch t=48 is 0.4760160953910157
Objective value at epoch t=49 is 0.47600730970974636
Objective value at epoch t=50 is 0.47599985720704147
Objective value at epoch t=51 is 0.4759954645043291
Objective value at epoch t=52 is 0.4759940712205153
Objective value at epoch t=53 is 0.47598605908055325
Objective value at epoch t=54 is 0.47598230014537213
Objective value at epoch t=55 is 0.47597891168632706
Objective value at epoch t=56 is 0.4759773600583773
Objective value at epoch t=57 is 0.47597269116801116
Objective value at epoch t=58 is 0.4759730926249149
Objective value at epoch t=59 is 0.47597145580799366
Objective value at epoch t=60 is 0.4759697627309919
Objective value at epoch t=61 is 0.4759695972527503
```

```
Objective value at epoch t=62 is 0.47596583731538067
Objective value at epoch t=63 is 0.47596391963327844
Objective value at epoch t=64 is 0.47596231674207656
Objective value at epoch t=65 is 0.47596118236204
Objective value at epoch t=66 is 0.47596023231380713
Objective value at epoch t=67 is 0.47596012292474155
Objective value at epoch t=68 is 0.4759593973849003
Objective value at epoch t=69 is 0.475957885645003
Objective value at epoch t=70 is 0.47595736620003504
Objective value at epoch t=71 is 0.47595663267519434
Objective value at epoch t=72 is 0.47595625372618333
Objective value at epoch t=73 is 0.47595546399204036
Objective value at epoch t=74 is 0.4759553587757015
Objective value at epoch t=75 is 0.47595511822989306
Objective value at epoch t=76 is 0.4759545829034583
Objective value at epoch t=77 is 0.4759543465944983
Objective value at epoch t=78 is 0.4759540677119146
Objective value at epoch t=79 is 0.4759537827190806
Objective value at epoch t=80 is 0.4759537132247439
Objective value at epoch t=81 is 0.47595335640573716
Objective value at epoch t=82 is 0.4759531496638686
Objective value at epoch t=83 is 0.47595290488153735
Objective value at epoch t=84 is 0.4759526811044973
Objective value at epoch t=85 is 0.47595258660059836
Objective value at epoch t=86 is 0.47595241667682425
Objective value at epoch t=87 is 0.47595246778690414
Objective value at epoch t=88 is 0.47595237191329803
Objective value at epoch t=89 is 0.4759522411964422
Objective value at epoch t=90 is 0.4759521129964469
Objective value at epoch t=91 is 0.47595206472977986
Objective value at epoch t=92 is 0.475952015093659
Objective value at epoch t=93 is 0.47595193791091317
Objective value at epoch t=94 is 0.4759518887565986
Objective value at epoch t=95 is 0.4759518437593086
Objective value at epoch t=96 is 0.47595183445728334
Objective value at epoch t=97 is 0.4759517950953208
Objective value at epoch t=98 is 0.47595177329317984
Objective value at epoch t=99 is 0.4759517241521361
```

In [21]:

```
# MB-SGD with batch size b=64
lam = 1E-6 # do not change
b = 64 # do not change
stepsize = 0.1 # you must tune this parameter

w, objvals_mbsgd64 = mb_sgd(x_train, y_train, lam, b, stepsize)
```

```
Objective value at epoch t=0 is 0.6479289967733071
Objective value at epoch t=1 is 0.5834682372147804
Objective value at epoch t=2 is 0.552380502675546
Objective value at epoch t=3 is 0.5347166594695928
Objective value at epoch t=4 is 0.523199694850516
Objective value at epoch t=5 is 0.5156004366107709
Objective value at epoch t=6 is 0.5097877404734804
Objective value at epoch t=7 is 0.5057565010486716
Objective value at epoch t=8 is 0.5024293447467897
Objective value at epoch t=9 is 0.49982761641576656
Objective value at epoch t=10 is 0.49796461307504786
Objective value at epoch t=11 is 0.4962329426162496
Objective value at epoch t=12 is 0.49488123171133225
```

```
Objective value at epoch t=13 is 0.4936494472269489
Objective value at epoch t=14 is 0.49279712823255845
Objective value at epoch t=15 is 0.49197426225493085
Objective value at epoch t=16 is 0.49123555025312554
Objective value at epoch t=17 is 0.4906473244735179
Objective value at epoch t=18 is 0.49017893392607564
Objective value at epoch t=19 is 0.4897351843250847
Objective value at epoch t=20 is 0.4893236304697289
Objective value at epoch t=21 is 0.48900055017988536
Objective value at epoch t=22 is 0.48868271889852843
Objective value at epoch t=23 is 0.48844019917595816
Objective value at epoch t=24 is 0.4882358234034946
Objective value at epoch t=25 is 0.48802111748545307
Objective value at epoch t=26 is 0.4878398247096477
Objective value at epoch t=27 is 0.48768789877108115
Objective value at epoch t=28 is 0.4875486896994309
Objective value at epoch t=29 is 0.4874151485471042
Objective value at epoch t=30 is 0.4873161107034357
Objective value at epoch t=31 is 0.48720461291409894
Objective value at epoch t=32 is 0.4871132098453878
Objective value at epoch t=33 is 0.4870455406103713
Objective value at epoch t=34 is 0.48696590380813626
Objective value at epoch t=35 is 0.48690164909067785
Objective value at epoch t=36 is 0.4868432152495827
Objective value at epoch t=37 is 0.48679696543113415
Objective value at epoch t=38 is 0.48674853967131104
Objective value at epoch t=39 is 0.4867096292779716
Objective value at epoch t=40 is 0.4866772218422451
Objective value at epoch t=41 is 0.48664399164323574
Objective value at epoch t=42 is 0.48661499771512623
Objective value at epoch t=43 is 0.4865862155709998
Objective value at epoch t=44 is 0.4865640117603823
Objective value at epoch t=45 is 0.4865423826041183
Objective value at epoch t=46 is 0.4865271951004456
Objective value at epoch t=47 is 0.486506933164556
Objective value at epoch t=48 is 0.48649178122711734
Objective value at epoch t=49 is 0.4864778183126722
Objective value at epoch t=50 is 0.4864645419392618
Objective value at epoch t=51 is 0.48645304798496924
Objective value at epoch t=52 is 0.48644475826322575
Objective value at epoch t=53 is 0.4864342635383134
Objective value at epoch t=54 is 0.4864255715803164
Objective value at epoch t=55 is 0.4864203744302015
Objective value at epoch t=56 is 0.486412126218168
Objective value at epoch t=57 is 0.4864069389506673
Objective value at epoch t=58 is 0.4864019418849386
Objective value at epoch t=59 is 0.4863966353105086
Objective value at epoch t=60 is 0.4863926243290281
Objective value at epoch t=61 is 0.48638920464136914
Objective value at epoch t=62 is 0.4863845542721438
Objective value at epoch t=63 is 0.48638187536301275
Objective value at epoch t=64 is 0.4863797338678951
Objective value at epoch t=65 is 0.4863765340673229
Objective value at epoch t=66 is 0.48637471151531636
Objective value at epoch t=67 is 0.48637225620531677
Objective value at epoch t=68 is 0.4863706417270426
Objective value at epoch t=69 is 0.486368952815584
Objective value at epoch t=70 is 0.48636754310342767
Objective value at epoch t=71 is 0.48636591348601765
Objective value at epoch t=72 is 0.4863647428174268
```

```
Objective value at epoch t=73 is 0.4863638513102738
Objective value at epoch t=74 is 0.48636284766455534
Objective value at epoch t=75 is 0.48636190081295494
Objective value at epoch t=76 is 0.48636100439132707
Objective value at epoch t=77 is 0.48636032592244166
Objective value at epoch t=78 is 0.48635965010129595
Objective value at epoch t=79 is 0.48635906568855525
Objective value at epoch t=80 is 0.48635854399685885
Objective value at epoch t=81 is 0.4863580660443757
Objective value at epoch t=82 is 0.4863577301459704
Objective value at epoch t=83 is 0.486357299149744
Objective value at epoch t=84 is 0.48635697581016435
Objective value at epoch t=85 is 0.4863566606951114
Objective value at epoch t=86 is 0.4863564019813573
Objective value at epoch t=87 is 0.48635614080190226
Objective value at epoch t=88 is 0.4863559214735827
Objective value at epoch t=89 is 0.4863557216727389
Objective value at epoch t=90 is 0.48635556809830616
Objective value at epoch t=91 is 0.4863553851940671
Objective value at epoch t=92 is 0.4863552140920352
Objective value at epoch t=93 is 0.4863550932304677
Objective value at epoch t=94 is 0.4863549662058942
Objective value at epoch t=95 is 0.48635486907440856
Objective value at epoch t=96 is 0.4863547839531809
Objective value at epoch t=97 is 0.4863546891605083
Objective value at epoch t=98 is 0.48635459921335233
Objective value at epoch t=99 is 0.48635453725564026
```

7. Plot and compare GD, SGD, and MB-SGD

You are required to compare the following algorithms:

- Gradient descent (GD)
- SGD
- MB-SGD with b=8
- MB-SGD with b=64

Follow the code in Section 4 to plot `objective function value` against `epochs`. There should be four curves in the plot; each curve corresponds to one algorithm.

Hint: Logistic regression with ℓ_2-norm regularization is a strongly convex optimization problem. All the algorithms will converge to the same solution. **In the end, the `objective function value` of the 4 algorithms will be the same. If not the same, your implementation must be wrong. Do NOT submit wrong code and wrong result!**

In [29]:

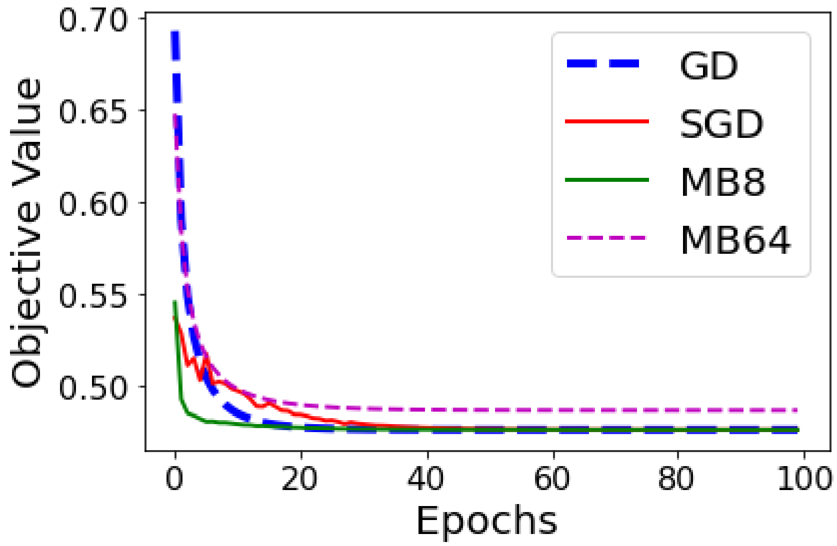
```
# plot the 4 curves:
fig = plt.figure(figsize=(6, 4))

epochs_gd = range(len(objvals_gd))
epochs_sgd = range(len(objvals_sgd))
epochs_mbsgd8 = range(len(objvals_mbsgd8))
epochs_mbsgd64 = range(len(objvals_mbsgd64))
```

```
line0, = plt.plot(epochs_gd, objvals_gd, '--b', LineWidth=4)
line1, = plt.plot(epochs_sgd, objvals_sgd, '-r', LineWidth=2)
line2, = plt.plot(epochs_mbsgd8, objvals_mbsgd8, '-g', LineWidth=2, label='MB-SG
line3, = plt.plot(epochs_mbsgd64, objvals_mbsgd64, '--m', LineWidth=2, label='MB

plt.xlabel('Epochs', FontSize=20)
plt.ylabel('Objective Value', FontSize=20)
plt.xticks(FontSize=16)
plt.yticks(FontSize=16)
plt.legend([line0, line1,line2, line3], ['GD', 'SGD','MB8', 'MB64'], fontsize=20
plt.tight_layout()
plt.show()
#fig.savefig('compare_gd_sgd_mbsgd8/64.pdf', format='pdf', dpi=1200)
```

```
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:
9: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  line0, = plt.plot(epochs_gd, objvals_gd, '--b', LineWidth=4)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
0: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  line1, = plt.plot(epochs_sgd, objvals_sgd, '-r', LineWidth=2)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
1: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  line2, = plt.plot(epochs_mbsgd8, objvals_mbsgd8, '-g', LineWidth=2, label='MB-
SGD w/ b=8')
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
2: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  line3, = plt.plot(epochs_mbsgd64, objvals_mbsgd64, '--m', LineWidth=2, label
='MB-SGD w/ b=64')
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
4: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.xlabel('Epochs', FontSize=20)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
5: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.ylabel('Objective Value', FontSize=20)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
6: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.xticks(FontSize=16)
/var/folders/cl/nfydl9vx5cxfn055kr1bmk400000gn/T/ipykernel_13029/1809375040.py:1
7: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in
3.3 and support will be removed two minor releases later
  plt.yticks(FontSize=16)
```


0.70
0.65
0.60
0.55
0.50
Objective Value
0
20
40
60
80
100
Epochs
GD
SGD
MB8
MB64