Programming Assignment: Numerical Optimization for Logistic Regression.

Name: [Soorya Suresh]

0. You will do the following:

1. Read the lecture note: click here (https://github.com/wangshusen/DeepLearning/blob/master/LectureNotes/Logistic/paper/logistic.
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.
5. 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.)
6. 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.

1. Data processing

- Download the Diabete dataset from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/diabetes (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 [1]:
```
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 [2]:
```
# 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 [3]:
```
# 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 =
[[-0.07086932  0.10513049  0.0512098   0.11588343 -0.0180108   0.05946569
  -0.08438162  0.09829334]]
test std =
[[1.065022   1.0255224  1.11841399 1.07613639 0.97045244 1.1300538
  0.86300657 1.0941474 ]]
```

1.4. Add a dimension of all ones

In [4]:
```
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 $Q(w; X, y) = \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i x_i^T w\right)\right) + \frac{\lambda}{2}\|w\|_2^2$.

In [5]:
```
# 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 [6]:
```
# 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 [7]:

```python
# 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 [8]:

```python
# 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 [9]:

```
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 0.5957411105417552
Objective value at t=2 is 0.5556691726850684
Objective value at t=3 is 0.5341302795143784
Objective value at t=4 is 0.5207074377314754
Objective value at t=5 is 0.5116275277965681
Objective value at t=6 is 0.505169717970275
Objective value at t=7 is 0.5004199183716509
Objective value at t=8 is 0.49684035936851434
Objective value at t=9 is 0.49409177208703875
Objective value at t=10 is 0.4919492902102697
Objective value at t=11 is 0.4902583508269297
Objective value at t=12 is 0.4889096878556349
Objective value at t=13 is 0.4878242893558486
Objective value at t=14 is 0.48694393238579814
Objective value at t=15 is 0.4862250151439117
Objective value at t=16 is 0.4856344232007914
Objective value at t=17 is 0.4851466936936782
Objective value at t=18 is 0.48474203084323253
Objective value at t=19 is 0.4844048928426671
Objective value at t=20 is 0.4841229698932907
Objective value at t=21 is 0.4838864347140234
Objective value at t=22 is 0.4836873858442059
Objective value at t=23 is 0.4835194293089502
Objective value at t=24 is 0.4833773608832603
Objective value at t=25 is 0.48325692238098766
Objective value at t=26 is 0.4831546130232238
Objective value at t=27 is 0.4830675422147897
Objective value at t=28 is 0.48299331375129306
Objective value at t=29 is 0.4829299340976812
Objective value at t=30 is 0.4828757392564761
Objective value at t=31 is 0.4828293361041332
Objective value at t=32 is 0.4827895550695025
Objective value at t=33 is 0.48275541176391495
Objective value at t=34 is 0.4827260757207428
Objective value at t=35 is 0.4827008448145035
Objective value at t=36 is 0.48267912424197623
Objective value at t=37 is 0.48266040918634195
Objective value at t=38 is 0.48264427046881964
Objective value at t=39 is 0.48263034263431875
Objective value at t=40 is 0.4826183140283418
Objective value at t=41 is 0.4826079185091532
Objective value at t=42 is 0.4825989285076859
Objective value at t=43 is 0.4825911492019145
Objective value at t=44 is 0.48258441361567284
Objective value at t=45 is 0.48257857848652175
Objective value at t=46 is 0.4825735207751348
Objective value at t=47 is 0.48256913471117674
Objective value at t=48 is 0.48256532928891116
Objective value at t=49 is 0.4825620261406438
Objective value at t=50 is 0.4825591577282644
Objective value at t=51 is 0.4825566658031174
Objective value at t=52 is 0.4825545000926342
Objective value at t=53 is 0.48255261717893216
Objective value at t=54 is 0.48255097954018455
Objective value at t=55 is 0.48254955473021893
Objective value at t=56 is 0.4825483146756717
```

```
Objective value at t=57 is 0.48254723507325015
Objective value at t=58 is 0.4825462948723566
Objective value at t=59 is 0.48254547583058816
Objective value at t=60 is 0.4825447621315179
Objective value at t=61 is 0.4825441400557732
Objective value at t=62 is 0.4825435976977537
Objective value at t=63 is 0.4825431247214853
Objective value at t=64 is 0.4825427121500529
Objective value at t=65 is 0.4825423521838746
Objective value at t=66 is 0.4825420380437682
Objective value at t=67 is 0.482541763835345
Objective value at t=68 is 0.4825415244317651
Objective value at t=69 is 0.48254131537231365
Objective value at t=70 is 0.4825411327746133
Objective value at t=71 is 0.48254097325860473
Objective value at t=72 is 0.482540833880684
Objective value at t=73 is 0.48254071207661237
Objective value at t=74 is 0.4825406056120104
Objective value at t=75 is 0.48254051253940783
Objective value at t=76 is 0.48254043116096723
Objective value at t=77 is 0.4825403599961186
Objective value at t=78 is 0.48254029775344776
Objective value at t=79 is 0.4825402433062683
Objective value at t=80 is 0.48254019567139084
Objective value at t=81 is 0.4825401539906607
Objective value at t=82 is 0.4825401175149001
Objective value at t=83 is 0.48254008558993705
Objective value at t=84 is 0.4825400576444447
Objective value at t=85 is 0.48254003317935545
Objective value at t=86 is 0.48254001175864153
Objective value at t=87 is 0.4825399930012839
Objective value at t=88 is 0.48253997657427516
Objective value at t=89 is 0.4825399621865211
Objective value at t=90 is 0.48253994958352364
Objective value at t=91 is 0.4825399385427451
Objective value at t=92 is 0.482539928869565
Objective value at t=93 is 0.4825399203937525
Objective value at t=94 is 0.48253991296638843
Objective value at t=95 is 0.4825399064571791
Objective value at t=96 is 0.48253990075211156
Objective value at t=97 is 0.4825398957514058
Objective value at t=98 is 0.4825398913677269
Objective value at t=99 is 0.48253988752462335
```

3.2. Stochastic gradient descent (SGD)

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

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 [10]:

```python
# 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 [11]:

```
# 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(objva

    return w, objvals
```

Run SGD.

In [12]:

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

```
Objective value at epoch t=0 is 0.5330283706031178
Objective value at epoch t=1 is 0.5377912639277465
Objective value at epoch t=2 is 0.5280745145900925
Objective value at epoch t=3 is 0.5234109910796725
Objective value at epoch t=4 is 0.5197614707783966
Objective value at epoch t=5 is 0.5216092396919081
Objective value at epoch t=6 is 0.5129866714892085
Objective value at epoch t=7 is 0.5072846943979503
Objective value at epoch t=8 is 0.5052092459443591
Objective value at epoch t=9 is 0.5041085538999666
Objective value at epoch t=10 is 0.5045223498677792
Objective value at epoch t=11 is 0.5011118658407197
Objective value at epoch t=12 is 0.4980178073933126
Objective value at epoch t=13 is 0.4986153238575802
Objective value at epoch t=14 is 0.49501565253506624
Objective value at epoch t=15 is 0.4943645238918319
Objective value at epoch t=16 is 0.49611958051543886
Objective value at epoch t=17 is 0.4929037823732402
Objective value at epoch t=18 is 0.4925070355037355
Objective value at epoch t=19 is 0.4924799556238944
Objective value at epoch t=20 is 0.49106242429522196
Objective value at epoch t=21 is 0.4900794248844103
Objective value at epoch t=22 is 0.48956805272399845
Objective value at epoch t=23 is 0.48850104646593034
Objective value at epoch t=24 is 0.4880195282281039
Objective value at epoch t=25 is 0.48777752490854465
Objective value at epoch t=26 is 0.48722628380684735
Objective value at epoch t=27 is 0.4868550067121243
Objective value at epoch t=28 is 0.4859939435615742
Objective value at epoch t=29 is 0.4860709421182845
Objective value at epoch t=30 is 0.4854536491205743
Objective value at epoch t=31 is 0.4853718009270044
Objective value at epoch t=32 is 0.4850710939336934
Objective value at epoch t=33 is 0.4848235764577719
Objective value at epoch t=34 is 0.48454674798973646
Objective value at epoch t=35 is 0.4843165586087901
Objective value at epoch t=36 is 0.484219701138277
Objective value at epoch t=37 is 0.48402747158707704
Objective value at epoch t=38 is 0.4839085846548471
Objective value at epoch t=39 is 0.483776694282635
Objective value at epoch t=40 is 0.48365778950126703
Objective value at epoch t=41 is 0.4835599366906491
Objective value at epoch t=42 is 0.4834628513040629
Objective value at epoch t=43 is 0.48334195290070536
Objective value at epoch t=44 is 0.4832811309691308
Objective value at epoch t=45 is 0.4832086685152423
Objective value at epoch t=46 is 0.4831409010999407
Objective value at epoch t=47 is 0.48308104388884515
Objective value at epoch t=48 is 0.48303098955988855
Objective value at epoch t=49 is 0.48297760715428906
Objective value at epoch t=50 is 0.4829343378191247
Objective value at epoch t=51 is 0.48289680525583245
Objective value at epoch t=52 is 0.48286274828667974
Objective value at epoch t=53 is 0.4828287090426383
Objective value at epoch t=54 is 0.4828006741445875
Objective value at epoch t=55 is 0.48277466957031007
Objective value at epoch t=56 is 0.48275182076849876
```

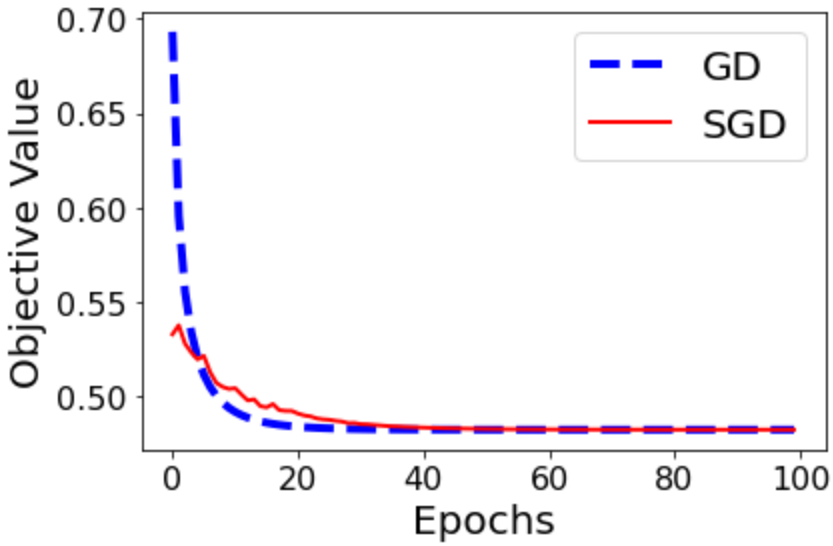
```
Objective value at epoch t=57 is 0.4827299226863251
Objective value at epoch t=58 is 0.4827116186928122
Objective value at epoch t=59 is 0.4826942294512017
Objective value at epoch t=60 is 0.4826793312834513
Objective value at epoch t=61 is 0.48266490115916233
Objective value at epoch t=62 is 0.48265274994797097
Objective value at epoch t=63 is 0.4826415019714876
Objective value at epoch t=64 is 0.4826314757137422
Objective value at epoch t=65 is 0.48262237674816494
Objective value at epoch t=66 is 0.4826141306866708
Objective value at epoch t=67 is 0.4826066423516129
Objective value at epoch t=68 is 0.48260010531586073
Objective value at epoch t=69 is 0.48259411328242613
Objective value at epoch t=70 is 0.48258868816693673
Objective value at epoch t=71 is 0.48258384274526467
Objective value at epoch t=72 is 0.48257945457256557
Objective value at epoch t=73 is 0.4825755458545812
Objective value at epoch t=74 is 0.48257201312873077
Objective value at epoch t=75 is 0.4825687907170953
Objective value at epoch t=76 is 0.4825659513263985
Objective value at epoch t=77 is 0.4825633805845079
Objective value at epoch t=78 is 0.4825610535546329
Objective value at epoch t=79 is 0.4825589621691827
Objective value at epoch t=80 is 0.482557076937428
Objective value at epoch t=81 is 0.4825553897533223
Objective value at epoch t=82 is 0.48255386272235795
Objective value at epoch t=83 is 0.4825524932078694
Objective value at epoch t=84 is 0.48255126050188374
Objective value at epoch t=85 is 0.48255014820348885
Objective value at epoch t=86 is 0.48254914911329483
Objective value at epoch t=87 is 0.4825482491588532
Objective value at epoch t=88 is 0.48254743698181224
Objective value at epoch t=89 is 0.48254670914979786
Objective value at epoch t=90 is 0.4825460530754807
Objective value at epoch t=91 is 0.4825454619036349
Objective value at epoch t=92 is 0.4825449305208771
Objective value at epoch t=93 is 0.48254445193256545
Objective value at epoch t=94 is 0.48254402161137583
Objective value at epoch t=95 is 0.48254363397127553
Objective value at epoch t=96 is 0.48254328512196343
Objective value at epoch t=97 is 0.48254297116825545
Objective value at epoch t=98 is 0.48254268860750926
Objective value at epoch t=99 is 0.48254243427857785
```

4. Compare GD with SGD

Plot objective function values against epochs.

In [13]:

```
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)
```



5. Prediction

In [14]:

```
# 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 [15]:
```
# 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 0.2265625
```

In [16]:
```
# 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 0.1875
```

6. Mini-batch SGD (fill the code)

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

Define $Q_I(w) = \frac{1}{b}\sum_{i\in I} \log\left(1+\exp\left(-y_i x_i^T w\right)\right) + \frac{\lambda}{2}\|w\|_2^2$, 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 [23]:

```
# 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):

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

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

    # calculate stochastic gradient
    g_loss = numpy.mean(-yx / (1 + numpy.exp(yxw)),axis=0).reshape(d,1) #
    g = g_loss + lam * w # d-by-1 matrix
    # Fill the function
    # Follow the implementation of stochastic_objective_gradient
    # Use matrix-vector multiplication; do not use FOR LOOP of vector-vec
    ...

    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 [24]:

```
# 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!)

    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(0,n,b):
            xi = x_rand[i:i + b] # 1-by-d matrix
            yi = (y_rand[i:i+b]) # scalar
            obj, g = mb_stochastic_objective_gradient(w, xi, yi, lam,b)
            objval += obj
            w -= stepsize * g

        stepsize *= 0.9 # decrease step size
        objval /= n/b
        objvals[t] = objval
        print('Objective value at epoch t=' + str(t) + ' is ' + str(objva

    return w, objvals
```

6.3. Run MB-SGD

In [25]:

```
# 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 0.5556642446298549
Objective value at epoch t=1 is 0.4985714953268937
Objective value at epoch t=2 is 0.4918533059698581
Objective value at epoch t=3 is 0.4894409333414284
Objective value at epoch t=4 is 0.4874694241339667
Objective value at epoch t=5 is 0.4877448134351364
Objective value at epoch t=6 is 0.4875112961040885
Objective value at epoch t=7 is 0.4863747925445446
Objective value at epoch t=8 is 0.48662053847891656
Objective value at epoch t=9 is 0.48592570263392565
Objective value at epoch t=10 is 0.4856174626805035
Objective value at epoch t=11 is 0.48537513536488097
Objective value at epoch t=12 is 0.48527695033693935
Objective value at epoch t=13 is 0.48485149175685394
Objective value at epoch t=14 is 0.48443704164464363
Objective value at epoch t=15 is 0.4843781051381223
Objective value at epoch t=16 is 0.48427583607568225
Objective value at epoch t=17 is 0.48412546527157707
Objective value at epoch t=18 is 0.4839034763711732
Objective value at epoch t=19 is 0.4838178221852621
Objective value at epoch t=20 is 0.4837329239731553
Objective value at epoch t=21 is 0.48364204806806
Objective value at epoch t=22 is 0.4834152898033353
Objective value at epoch t=23 is 0.4834341622669708
Objective value at epoch t=24 is 0.4832858848277066
Objective value at epoch t=25 is 0.4831707505900932
Objective value at epoch t=26 is 0.48314870867743515
Objective value at epoch t=27 is 0.483130275342161
Objective value at epoch t=28 is 0.4830882133316604
Objective value at epoch t=29 is 0.4830021579486325
Objective value at epoch t=30 is 0.4829420545410471
Objective value at epoch t=31 is 0.4828648479945903
Objective value at epoch t=32 is 0.4828781610451049
Objective value at epoch t=33 is 0.4828450116608912
Objective value at epoch t=34 is 0.4828041645048824
Objective value at epoch t=35 is 0.482751281009815
Objective value at epoch t=36 is 0.4827719741014704
Objective value at epoch t=37 is 0.48271345542735516
Objective value at epoch t=38 is 0.482703694935991
Objective value at epoch t=39 is 0.4827124660963027
Objective value at epoch t=40 is 0.48268180950638023
Objective value at epoch t=41 is 0.48268414691950295
Objective value at epoch t=42 is 0.4826574170942065
Objective value at epoch t=43 is 0.48264283422787235
Objective value at epoch t=44 is 0.48263277389439063
Objective value at epoch t=45 is 0.4826244162920936
Objective value at epoch t=46 is 0.4826181823611659
Objective value at epoch t=47 is 0.4826062544012125
Objective value at epoch t=48 is 0.4825974690929809
Objective value at epoch t=49 is 0.4825980977775318
Objective value at epoch t=50 is 0.48258782746273543
Objective value at epoch t=51 is 0.4825886159343901
Objective value at epoch t=52 is 0.4825789928688713
Objective value at epoch t=53 is 0.48258341069866156
Objective value at epoch t=54 is 0.4825732617633012
Objective value at epoch t=55 is 0.48257071248889893
Objective value at epoch t=56 is 0.4825649161399956
```

```
Objective value at epoch t=57 is 0.48256693094216896
Objective value at epoch t=58 is 0.48256236916218515
Objective value at epoch t=59 is 0.4825587365532778
Objective value at epoch t=60 is 0.48255832632733353
Objective value at epoch t=61 is 0.48255565586416693
Objective value at epoch t=62 is 0.48255377331329835
Objective value at epoch t=63 is 0.4825534758448873
Objective value at epoch t=64 is 0.48255227425308445
Objective value at epoch t=65 is 0.4825514851356857
Objective value at epoch t=66 is 0.4825496802036689
Objective value at epoch t=67 is 0.48254909764328185
Objective value at epoch t=68 is 0.48254820618421973
Objective value at epoch t=69 is 0.4825473152613168
Objective value at epoch t=70 is 0.48254741546088836
Objective value at epoch t=71 is 0.48254578214140453
Objective value at epoch t=72 is 0.4825457849772855
Objective value at epoch t=73 is 0.4825453188386481
Objective value at epoch t=74 is 0.4825450572727622
Objective value at epoch t=75 is 0.4825446444420482
Objective value at epoch t=76 is 0.4825438766749069
Objective value at epoch t=77 is 0.4825437806115177
Objective value at epoch t=78 is 0.4825435938821118
Objective value at epoch t=79 is 0.4825433826407628
Objective value at epoch t=80 is 0.48254306629497146
Objective value at epoch t=81 is 0.4825427491707431
Objective value at epoch t=82 is 0.4825425768246136
Objective value at epoch t=83 is 0.48254241452362645
Objective value at epoch t=84 is 0.4825422278572488
Objective value at epoch t=85 is 0.4825422189257497
Objective value at epoch t=86 is 0.4825420539387065
Objective value at epoch t=87 is 0.4825419628733341
Objective value at epoch t=88 is 0.48254179263657127
Objective value at epoch t=89 is 0.48254178329378783
Objective value at epoch t=90 is 0.48254162467281636
Objective value at epoch t=91 is 0.4825415731657715
Objective value at epoch t=92 is 0.4825415153568744
Objective value at epoch t=93 is 0.4825414013951169
Objective value at epoch t=94 is 0.4825413933408935
Objective value at epoch t=95 is 0.48254134866719306
Objective value at epoch t=96 is 0.48254130830151576
Objective value at epoch t=97 is 0.4825412477894252
Objective value at epoch t=98 is 0.48254122591976867
Objective value at epoch t=99 is 0.482541201942231
```

In [27]:

```
# MB-SGD with batch size b=64
lam = 1E-6 # do not change
b = 64 # do not change
stepsize = 0.9 # you must tune this parameter

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

```
Objective value at epoch t=0 is 0.5530678719588462
Objective value at epoch t=1 is 0.4978766512983623
Objective value at epoch t=2 is 0.4939217171154371
Objective value at epoch t=3 is 0.4898052976489796
Objective value at epoch t=4 is 0.48978263464961874
Objective value at epoch t=5 is 0.48745831620452995
Objective value at epoch t=6 is 0.48834620643274745
Objective value at epoch t=7 is 0.4872692867173612
Objective value at epoch t=8 is 0.4887136303531455
Objective value at epoch t=9 is 0.4853567878826808
Objective value at epoch t=10 is 0.4856058556374835
Objective value at epoch t=11 is 0.4850665706373073
Objective value at epoch t=12 is 0.4855934251564655
Objective value at epoch t=13 is 0.4846417210571019
Objective value at epoch t=14 is 0.48446484289572445
Objective value at epoch t=15 is 0.48437091729572146
Objective value at epoch t=16 is 0.48377284406330734
Objective value at epoch t=17 is 0.484345777173948
Objective value at epoch t=18 is 0.4837552744665996
Objective value at epoch t=19 is 0.4833503042681591
Objective value at epoch t=20 is 0.4834659525941866
Objective value at epoch t=21 is 0.48384234856617425
Objective value at epoch t=22 is 0.4835015502305879
Objective value at epoch t=23 is 0.483389238884123
Objective value at epoch t=24 is 0.48353675925465184
Objective value at epoch t=25 is 0.48320645456471406
Objective value at epoch t=26 is 0.48333948213372163
Objective value at epoch t=27 is 0.4832077393849013
Objective value at epoch t=28 is 0.48293734929136106
Objective value at epoch t=29 is 0.48305028305757747
Objective value at epoch t=30 is 0.482946897138998
Objective value at epoch t=31 is 0.4829505345512784
Objective value at epoch t=32 is 0.48281330801827294
Objective value at epoch t=33 is 0.48282748859181124
Objective value at epoch t=34 is 0.4828140956303556
Objective value at epoch t=35 is 0.48277391715273443
Objective value at epoch t=36 is 0.48279262280868995
Objective value at epoch t=37 is 0.4827777209725781
Objective value at epoch t=38 is 0.48271313448454267
Objective value at epoch t=39 is 0.4827243073482702
Objective value at epoch t=40 is 0.48272436575157124
Objective value at epoch t=41 is 0.48272272036199304
Objective value at epoch t=42 is 0.4826595350512039
Objective value at epoch t=43 is 0.48267165614536917
Objective value at epoch t=44 is 0.4826485029595644
Objective value at epoch t=45 is 0.4826278050905063
Objective value at epoch t=46 is 0.48260914046710346
Objective value at epoch t=47 is 0.4826200751074758
Objective value at epoch t=48 is 0.48260313506144054
Objective value at epoch t=49 is 0.4825921704466364
Objective value at epoch t=50 is 0.48259219177980056
Objective value at epoch t=51 is 0.48258814973054437
Objective value at epoch t=52 is 0.48259212694811887
Objective value at epoch t=53 is 0.48258521843432334
Objective value at epoch t=54 is 0.4825738793443599
Objective value at epoch t=55 is 0.4825587897418842
Objective value at epoch t=56 is 0.48256499078825454
```

```
Objective value at epoch t=57 is 0.482568106030671
Objective value at epoch t=58 is 0.48256635242309454
Objective value at epoch t=59 is 0.4825607622506028
Objective value at epoch t=60 is 0.48255919831795724
Objective value at epoch t=61 is 0.48255849360531594
Objective value at epoch t=62 is 0.4825581517691626
Objective value at epoch t=63 is 0.4825507289984494
Objective value at epoch t=64 is 0.4825523646805344
Objective value at epoch t=65 is 0.48254960156737037
Objective value at epoch t=66 is 0.4825471025817145
Objective value at epoch t=67 is 0.4825492588105746
Objective value at epoch t=68 is 0.4825497401822371
Objective value at epoch t=69 is 0.4825459540681181
Objective value at epoch t=70 is 0.48254619412907324
Objective value at epoch t=71 is 0.48254644797127594
Objective value at epoch t=72 is 0.4825455157006308
Objective value at epoch t=73 is 0.48254486483814424
Objective value at epoch t=74 is 0.4825443966809687
Objective value at epoch t=75 is 0.4825436078543711
Objective value at epoch t=76 is 0.4825444505880694
Objective value at epoch t=77 is 0.4825438829961902
Objective value at epoch t=78 is 0.48254338419410925
Objective value at epoch t=79 is 0.48254328670297414
Objective value at epoch t=80 is 0.48254198616626864
Objective value at epoch t=81 is 0.48254250258944076
Objective value at epoch t=82 is 0.482542092646419
Objective value at epoch t=83 is 0.48254151955517177
Objective value at epoch t=84 is 0.48254241380006746
Objective value at epoch t=85 is 0.48254157353540245
Objective value at epoch t=86 is 0.482541560460478
Objective value at epoch t=87 is 0.4825412423547674
Objective value at epoch t=88 is 0.4825413966945513
Objective value at epoch t=89 is 0.4825412740994738
Objective value at epoch t=90 is 0.48254113494666095
Objective value at epoch t=91 is 0.48254099491677493
Objective value at epoch t=92 is 0.48254096452772766
Objective value at epoch t=93 is 0.48254069160487517
Objective value at epoch t=94 is 0.48254073179604173
Objective value at epoch t=95 is 0.4825407678998245
Objective value at epoch t=96 is 0.4825406711987707
Objective value at epoch t=97 is 0.48254067696049513
Objective value at epoch t=98 is 0.4825406947757921
Objective value at epoch t=99 is 0.4825405871584289
```

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

You are required to compare the following algorithms:

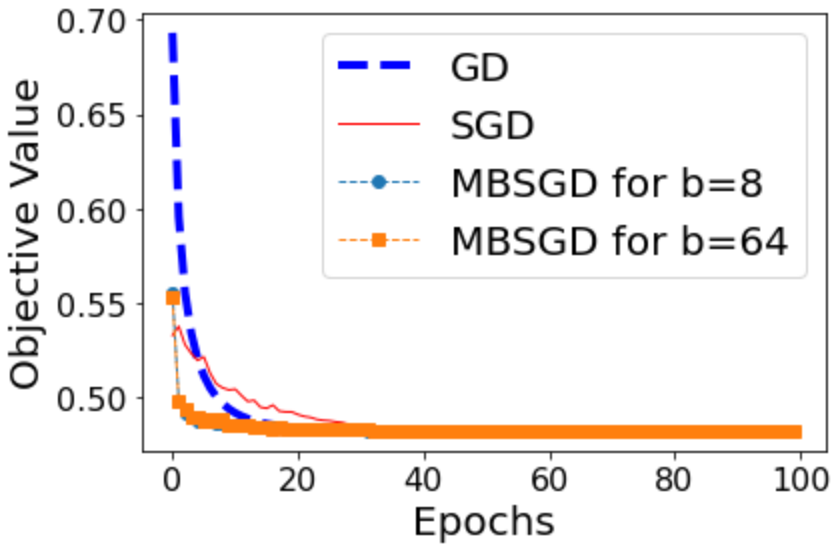
- Gradient descent (GD)
- SGD
- MB-SGD with b=8
- MB-SGD with b=64

Follow the code in Section 4 to plot `objective function value` against `epochs`. There should

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 [28]:

```
# plot the 4 curves:
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))
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=1)
line2,=plt.plot(epochs_mbsgd8,objvals_mbsgd8,'--o',LineWidth=1)
line3,=plt.plot(epochs_mbsgd64,objvals_mbsgd64,'--s',LineWidth=1)
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','MBSGD for b=8','MBSGD f
plt.tight_layout()
plt.show()


```



Type *Markdown* and LaTeX: α^2

Evaluating the Mini Batch Performance:Notice that the SGD performance had a low prediction error. Therefore, MBSGD resulting in the similar answers show that this method is accurate as well. The benefit of MBSGD compared to GD or SGD is that since it takes mini batches it requires less work and produces results faster.