HM1: Logistic Regression.

Name: [Bhargavi Katta]

For this assignment, you will build 6 models. You need to train Logistic Regression/Regularized Logistic Regression each with Batch Gradient Descent, Stochastic Gradient Descent and Mini Batch Gradient Descent. Also, you should plot their objective values versus epochs and compare their training and testing accuracy. You will need to tune the parameters a little bit to obtain reasonable results.

You do not have to follow the following procedure. You may implement your own functions and methods, but you need to show your results and plots.

In [1]:
```
# Load Packages
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

1. Data processing

- Download the Breast Cancer dataset from canvas or from https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)
- Load the data.
- Preprocess the data.

1.1. Load the data

Your implementations should include:

• Load data, clean data and partition them into training and testing data.

• Build logistic regression and L2-regularized logistic regression models.

• Implement three gradient descent algorithms for each model: Batch Gradient Descent (GD), Mini-Batch Gradient Descent (MB-GD) and Stochastic Gradient Descent (SGD).

- Compare the loss curve of three gradient descent algorithms (GD/MB-GD/SGD).
- Compare logistic regression and regularized version in terms of training and testing error.
- Try to tune different parameters (regularization parameter, learning rate, etc.) to see their effects.

You may find more detailed procedure in the IPython notebook file. You could use sklearn or any other packages to load and process the data, but you can not directly use the package to train the model.

In [2]:
```
data=pd.read_csv("/Users/gopalrao000/Desktop/Spring-2023/DL/HW1/data.csv")
```

1.2 Examine and clean data

In [3]:
```
# Some columns may not be useful for the model (For example, the first column contains ID number which may be i
# You need to get rid of the ID number feature.
# Also you should transform target labels in the second column from 'B' and 'M' to 1 and -1.
data.head(2)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	842302	M	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.30
1	842517	M	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.08

2 rows × 33 columns

In [4]:
```
data.columns
```

Out[4]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
```

```
Unnamed: 32                569
dtype: int64
```

In [6]:

```python
data.drop(columns=['id','Unnamed: 32'],inplace=True)
```

In [7]:

```python
data.head(2)
```

Out[7]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	c points
0	M	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	
1	M	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	

2 rows × 31 columns

In [8]:

```python
data['diagnosis']=data['diagnosis'].map({'B':1,'M':-1})
```

In [9]:

```python
data.head(5)
```

Out[9]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	c points
0	-1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	-1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	-1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	-1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	-1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 31 columns

In [10]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
```

```
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    int64  
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

1.3. Partition to training and testing sets

In [11]:
```
data['diagnosis']=data['diagnosis'].astype('category')
```

In [12]:

```
#data.info()
```

In [13]:

```
# You can partition using 80% training data and 20% testing data. It is a commonly used ratio in machine learni

x = data.drop(columns=['diagnosis'])
y = data['diagnosis']
```

In [14]:

```
x = np.asmatrix(x.to_numpy())
y = y.to_numpy()
```

In [15]:

```
x.shape,y.shape
```

Out[15]:

```
((569, 30), (569,))
```

In [16]:

```
# You can partition using 80% training data and 20% testing data. It is a commonly used ratio in machinel learn
n = x.shape[0]
n_train = 455
n_test = n - n_train

rand_indices = np.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: (455, 30)
Shape of x_test: (114, 30)
Shape of y_train: (455, 1)
Shape of y_test: (114, 1)
```

In [17]:

```
type(x_train),type(x_test),x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[17]:

```
(numpy.matrix, numpy.matrix, (455, 30), (114, 30), (455, 1), (114, 1))
```

1.4. Feature scaling

Use the standardization to transform both training and test features

In [18]:

```
# 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 =
[[ 0.10732032  0.07832683  0.09911903  0.08809503 -0.1344621  -0.04875326
   0.01521245  0.05135706 -0.06632541 -0.20669228  0.00129425 -0.03981079
   0.04235322  0.00731292 -0.15579762 -0.01906228  0.00795078 -0.00184491
  -0.21995674 -0.05092802  0.09328927  0.12523489  0.10662202  0.07629375
  -0.12129203  0.00371521  0.05423122  0.04704891 -0.04019425 -0.04444425]]
test std =
[[0.95557525 1.05776046 0.94910397 0.95630025 0.96656671 0.90581894
  0.91038545 0.96953792 0.83366609 0.86779119 0.96990242 1.00052985
  0.96484205 0.8320579  1.09012648 1.12360022 0.84451331 0.99674623
  0.94362849 0.81327384 0.99060093 1.08389202 1.00229211 1.0338781
  0.91209775 0.96076317 0.9493357  0.94839003 1.05775114 0.90086356]]
```

2. Logistic Regression Model

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

When $\lambda = 0$, the model is a regular logistic regression and when $\lambda > 0$, it essentially becomes a regularized logistic regression.

In [19]:

```
# Calculate the objective function value, or loss
# Inputs:
#     w: weight: d-by-1 matrix
#     x: data: n-by-d matrix
#     y: label: n-by-1 matrix
#     lam: regularization parameter: scalar
# Return:
#     objective function value, or loss (scalar)
def objective(w, x, y, lam, regularized):  # regularized is added to make it applocable for L1/L2 reg model
    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
    if regularized == True:
        reg = lam / 2 * numpy.sum(w * w)
        obj = loss + reg
    else:
        obj = loss
    return obj
```

3. Numerical optimization

3.1. Gradient descent

The gradient at w for regularized logistic regression 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 [20]:

```
# Calculate the gradient
# Inputs:
#     w: weight: d-by-1 matrix
```

```
#     x: data: n-by-d matrix
#     y: label: n-by-1 matrix
#     lam: regularization parameter: scalar
# Return:
#     g: gradient: d-by-1 matrix

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

In [21]:

```
# Gradient descent for solving logistic regression
# You will need to do iterative processes (loops) to obtain optimal weights in this function

# Inputs:
#     x: data: n-by-d matrix
#     y: label: n-by-1 matrix
#     lam: scalar, the regularization parameter
#     learning_rate: scalar
#     w: weights: d-by-1 matrix, initialization of w
#     max_epoch: integer, the maximal epochs
# Return:
#     w: weights: d-by-1 matrix, the solution
#     objvals: a record of each epoch's objective value

def gradient_descent(x, y, lam, learning_rate, w,regularized, max_epoch=100):
    n, d = x.shape
    objvals = numpy.zeros(max_epoch) # store the objective values

    for t in range(max_epoch):
        objval = objective(w, x, y, lam, regularized)
        objvals[t] = objval
        print('Loss at epoch=' + str(t) + ' is ' + str(objval))
        g = gradient(w, x, y, lam)
        w -= learning_rate * g

    return w, objvals
```

Use gradient_descent function to obtain your optimal weights and a list of objective values over each epoch.

In [22]:
```
# Train logistic regression
# You should get the optimal weights and a list of objective values by using gradient_descent function.
# Train logistic regression
learning_rate = 1.0
w = numpy.zeros((d, 1))
lam = 0

log_gradient_descent_w, log_objvals_gd = gradient_descent(x_train, y_train, lam, learning_rate, w,regularized=F
```

```
Loss at epoch=0 is 0.6931471805599453
Loss at epoch=1 is 0.17397487653150231
Loss at epoch=2 is 0.1401894804553756
Loss at epoch=3 is 0.12375915856641086
Loss at epoch=4 is 0.1159843544802855
Loss at epoch=5 is 0.11101095256006202
Loss at epoch=6 is 0.10712292254007011
Loss at epoch=7 is 0.10387413108032116
Loss at epoch=8 is 0.10107812193081776
Loss at epoch=9 is 0.09862940085343075
Loss at epoch=10 is 0.09645807250654821
Loss at epoch=11 is 0.09451383961705848
Loss at epoch=12 is 0.09275877180130707
Loss at epoch=13 is 0.09116340281590454
Loss at epoch=14 is 0.089704357207381
Loss at epoch=15 is 0.08836279024866601
Loss at epoch=16 is 0.08712330842751773
Loss at epoch=17 is 0.0859731955412631
Loss at epoch=18 is 0.08490184336031188
Loss at epoch=19 is 0.08390032434162208
Loss at epoch=20 is 0.08296106571329322
Loss at epoch=21 is 0.08207759745926146
Loss at epoch=22 is 0.08124435511959924
Loss at epoch=23 is 0.08045652384860183
Loss at epoch=24 is 0.07970991392160352
Loss at epoch=25 is 0.07900086048361049
Loss at epoch=26 is 0.07832614217368424
Loss at epoch=27 is 0.07768291458234429
Loss at epoch=28 is 0.07706865546387304
Loss at epoch=29 is 0.07648111933721242
Loss at epoch=30 is 0.0759182996402077
Loss at epoch=31 is 0.07537839700214984
Loss at epoch=32 is 0.07485979250392628
```

```
Loss at epoch=33 is 0.07436102502853142
Loss at epoch=34 is 0.07388077198515698
Loss at epoch=35 is 0.07341783283063491
Loss at epoch=36 is 0.07297111492223175
Loss at epoch=37 is 0.07253962132280464
Loss at epoch=38 is 0.07212244024844239
Loss at epoch=39 is 0.07171873590393178
Loss at epoch=40 is 0.07132774049575448
Loss at epoch=41 is 0.07094874724814992
Loss at epoch=42 is 0.07058110427686998
Loss at epoch=43 is 0.07022420919897587
Loss at epoch=44 is 0.06987750437647425
Loss at epoch=45 is 0.06954047270759162
Loss at epoch=46 is 0.06921263389271631
Loss at epoch=47 is 0.06889354111301664
Loss at epoch=48 is 0.0685827780688904
Loss at epoch=49 is 0.06827995633305224
Loss at epoch=50 is 0.06798471297948554
Loss at epoch=51 is 0.06769670845489299
Loss at epoch=52 is 0.06741562466385073
Loss at epoch=53 is 0.06714116324274329
Loss at epoch=54 is 0.06687304400085195
Loss at epoch=55 is 0.06661100350977835
Loss at epoch=56 is 0.06635479382478861
Loss at epoch=57 is 0.06610418132372427
Loss at epoch=58 is 0.06585894565090004
Loss at epoch=59 is 0.06561887875493681
Loss at epoch=60 is 0.0653837840107998
Loss at epoch=61 is 0.06515347541745786
Loss at epoch=62 is 0.06492777686357394
Loss at epoch=63 is 0.06470652145450374
Loss at epoch=64 is 0.06448955089463461
Loss at epoch=65 is 0.06427671491975848
Loss at epoch=66 is 0.0640678707747509
Loss at epoch=67 is 0.06386288273233745
Loss at epoch=68 is 0.06366162164917556
Loss at epoch=69 is 0.06346396455587475
Loss at epoch=70 is 0.06326979427792569
Loss at epoch=71 is 0.06307899908481719
Loss at epoch=72 is 0.06289147236489304
Loss at epoch=73 is 0.06270711232374256
Loss at epoch=74 is 0.06252582170413509
Loss at epoch=75 is 0.062347507525699446
Loss at epoch=76 is 0.06217208084272147
Loss at epoch=77 is 0.061999456518585384
```

```
Loss at epoch=78 is 0.06182955301552145
Loss at epoch=79 is 0.06166229219844506
Loss at epoch=80 is 0.061497599151782635
Loss at epoch=81 is 0.061335402008278056
Loss at epoch=82 is 0.0611756317888631
Loss at epoch=83 is 0.061018222252754356
Loss at epoch=84 is 0.06086310975701247
Loss at epoch=85 is 0.060710233124864205
Loss at epoch=86 is 0.060559533522146755
Loss at epoch=87 is 0.06041095434128796
Loss at epoch=88 is 0.060264441092283245
Loss at epoch=89 is 0.06011994130017542
Loss at epoch=90 is 0.05997740440858225
Loss at epoch=91 is 0.05983678168885395
Loss at epoch=92 is 0.05969802615447504
Loss at epoch=93 is 0.059561092480355515
Loss at epoch=94 is 0.05942593692668411
Loss at epoch=95 is 0.05929251726704041
Loss at epoch=96 is 0.05916079272048695
Loss at epoch=97 is 0.059030723887382154
Loss at epoch=98 is 0.05890227268867499
Loss at epoch=99 is 0.0587754023084596
```

In [23]:

```
# Train regularized logistic regression
# You should get the optimal weights and a list of objective values by using gradient_descent function.
learning_rate = 1.0
w = numpy.zeros((d, 1))
lam = 1E-6  #0.000001 very small value
print(lam)
reg_log_gradient_descent_w, reg_log_objvals_gd = gradient_descent(x_train, y_train, lam, learning_rate, w,True,
```

```
1e-06
Loss at epoch=0 is 0.6931471805599453
Loss at epoch=1 is 0.17397586953942917
Loss at epoch=2 is 0.14019055469466132
Loss at epoch=3 is 0.12376038394231742
Loss at epoch=4 is 0.1159857338940121
Loss at epoch=5 is 0.11101246817750737
Loss at epoch=6 is 0.10712456291732136
Loss at epoch=7 is 0.10387588791566714
Loss at epoch=8 is 0.10107998865281233
Loss at epoch=9 is 0.09863137201529035
Loss at epoch=10 is 0.09646014345978293
Loss at epoch=11 is 0.0945160063137094
Loss at epoch=12 is 0.09276103066461422
```

```
Loss at epoch=13 is 0.09116575065059444
Loss at epoch=14 is 0.08970679113440155
Loss at epoch=15 is 0.08836530765584678
Loss at epoch=16 is 0.08712590693110703
Loss at epoch=17 is 0.08597587295520373
Loss at epoch=18 is 0.08490459767121687
Loss at epoch=19 is 0.0839031536880726
Loss at epoch=20 is 0.08296396836849279
Loss at epoch=21 is 0.082080571816349
Loss at epoch=22 is 0.08124739967910254
Loss at epoch=23 is 0.08045963720763848
Loss at epoch=24 is 0.07971309476452346
Loss at epoch=25 is 0.07900410757384138
Loss at epoch=26 is 0.07832945434658702
Loss at epoch=27 is 0.0776862907389242
Loss at epoch=28 is 0.07707209456521995
Loss at epoch=29 is 0.07648462039956594
Loss at epoch=30 is 0.07592186173056034
Loss at epoch=31 is 0.07538201923431652
Loss at epoch=32 is 0.07486347403501853
Loss at epoch=33 is 0.07436476505578533
Loss at epoch=34 is 0.0738845697430709
Loss at epoch=35 is 0.07342168758837948
Loss at epoch=36 is 0.07297502598130023
Loss at epoch=37 is 0.07254358801487594
Loss at epoch=38 is 0.07212646193343278
Loss at epoch=39 is 0.07172281196821471
Loss at epoch=40 is 0.0713318703505302
Loss at epoch=41 is 0.07095293032794965
Loss at epoch=42 is 0.07058534003818016
Loss at epoch=43 is 0.07022849711897106
Loss at epoch=44 is 0.06988184395184747
Loss at epoch=45 is 0.06954486345347287
Loss at epoch=46 is 0.06921707534167121
Loss at epoch=47 is 0.06889803281411729
Loss at epoch=48 is 0.06858731958685237
Loss at epoch=49 is 0.06828454724743142
Loss at epoch=50 is 0.0679893528839298
Loss at epoch=51 is 0.06770139695644405
Loss at epoch=52 is 0.06742036138229175
Loss at epoch=53 is 0.06714594780998862
Loss at epoch=54 is 0.0668778760603756
Loss at epoch=55 is 0.06661588271607792
Loss at epoch=56 is 0.06635971984288229
Loss at epoch=57 is 0.06610915382867776
```

```
Loss at epoch=58 is 0.06586396432738188
Loss at epoch=59 is 0.06562394329679928
Loss at epoch=60 is 0.06538889412068416
Loss at epoch=61 is 0.06515863080642163
Loss at epoch=62 is 0.06493297725073914
Loss at epoch=63 is 0.06471176656672425
Loss at epoch=64 is 0.06449484046618158
Loss at epoch=65 is 0.06428204869202253
Loss at epoch=66 is 0.06407324849595987
Loss at epoch=67 is 0.06386830415728881
Loss at epoch=68 is 0.06366708653898266
Loss at epoch=69 is 0.06346947267772562
Loss at epoch=70 is 0.06327534540485406
Loss at epoch=71 is 0.06308459299548465
Loss at epoch=72 is 0.0628971088433818
Loss at epoch=73 is 0.06271279115935835
Loss at epoch=74 is 0.06253154269121908
Loss at epoch=75 is 0.06235327046344926
Loss at epoch=76 is 0.06217788553502042
Loss at epoch=77 is 0.06200530277383949
Loss at epoch=78 is 0.06183544064650384
Loss at epoch=79 is 0.061668221022147446
Loss at epoch=80 is 0.06150356898927321
Loss at epoch=81 is 0.0613414126845659
Loss at epoch=82 is 0.06118168313276814
Loss at epoch=83 is 0.06102431409678314
Loss at epoch=84 is 0.06086924193723922
Loss at epoch=85 is 0.06071640548081676
Loss at epoch=86 is 0.06056574589669746
Loss at epoch=87 is 0.060417206580548846
Loss at epoch=88 is 0.06027073304550565
Loss at epoch=89 is 0.060126272819653596
Loss at epoch=90 is 0.059983775349560994
Loss at epoch=91 is 0.05984319190943967
Loss at epoch=92 is 0.05970447551555051
Loss at epoch=93 is 0.05956758084549798
Loss at epoch=94 is 0.0594324641620864
Loss at epoch=95 is 0.05929908324143519
Loss at epoch=96 is 0.059167397305073845
Loss at epoch=97 is 0.059037366955757606
Loss at epoch=98 is 0.05890895411676477
Loss at epoch=99 is 0.05878212197445379
```

3.2. Stochastic gradient descent (SGD)

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

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

You may need to implement a new function to calculate the new objective function and gradients.

In [24]:

```
# Calculate the objective Q_i and the gradient of Q_i
# Inputs:
#     w: weights: d-by-1 matrix
#     xi: data: 1-by-d matrix
#     yi: label: 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, regularized):
    d = xi.shape[0]
    yx = yi * xi
    yxw = float(numpy.dot(yx, w))

    #print(yxw)
    loss= numpy.log(1+ numpy.exp(-yxw))
    if regularized == True:
        reg = lam / 2 * numpy.sum(w**2, axis=0)
        obj = loss + reg
    else:
        obj = loss

    # stochastic_gradient calculation
    g_loss = -yx.T / (1 + numpy.exp(yxw))
    g = g_loss + lam *w
    return obj,g
```

Hints:

1. In every epoch, randomly permute the n samples.
2. Each epoch has n iterations. In every iteration, use 1 sample, and compute the gradient and objective using the `stochastic_objective_gradient` function. In the next iteration, use the next sample, and so on.

In [25]:
```
# SGD for solving logistic regression
# You will need to do iterative process (loops) to obtain optimal weights in this function

# Inputs:
#     x: data: n-by-d matrix
#     y: label: n-by-1 matrix
#     lam: scalar, the regularization parameter
#     learning_rate: scalar
#     w: weights: d-by-1 matrix, initialization of w
#     max_epoch: integer, the maximal epochs
# Return:
#
#     w: weights: d-by-1 matrix, the solution
#     objvals: a record of each epoch's objective value
#     Record one objective value per epoch (not per iteration)

def sgd(x, y, lam, learning_rate, w, max_epoch=100,regularized=False):
    n, d = x.shape
    objvals = numpy.zeros(max_epoch)

    for t in range(max_epoch):
        # Shuffle Data using Permutation method
        rand_indices = numpy.random.permutation(n)
        x_rand = numpy.asmatrix(x[rand_indices, :])
        y_rand = y[rand_indices, :]
        objval = 0
        for i in range(n):
            xi = x_rand[i, :]
            yi = float(y_rand[i, :])
            obj, g = stochastic_objective_gradient(w, xi, yi, lam, regularized)
            objval += obj
            w -= learning_rate * g

        learning_rate *= 0.9
        objval /= n
        objvals[t] = objval
        print('Loss at epoch=' + str(t) + ' is ' + str(objval))

    return w, objvals
```

Use sgd function to obtain your optimal weights and a list of objective values over each epoch.

In [26]:
```
# Train logistic regression
# You should get the optimal weights and a list of objective values by using gradient_descent function.

learning_rate = 0.1
w = numpy.zeros((d, 1))
lam = 1E-6
log_stoch_gradient_descent_w, log_objvals_sgd = sgd(x_train, y_train, lam, learning_rate, w, 100)
```

```
Loss at epoch=0 is 0.10894978889893073
Loss at epoch=1 is 0.07380276652003888
Loss at epoch=2 is 0.06368376880663983
Loss at epoch=3 is 0.058582908470312764
Loss at epoch=4 is 0.05630918448458589
Loss at epoch=5 is 0.05473570692177975
Loss at epoch=6 is 0.052866739151216584
Loss at epoch=7 is 0.051324369403885885
Loss at epoch=8 is 0.049967607413236456
Loss at epoch=9 is 0.049155308726044664
Loss at epoch=10 is 0.04815852314491584
Loss at epoch=11 is 0.04737764951734254
Loss at epoch=12 is 0.04688851023394784
Loss at epoch=13 is 0.04640763768407451
Loss at epoch=14 is 0.045808247509085
Loss at epoch=15 is 0.04548304299979288
Loss at epoch=16 is 0.04505398478956685
Loss at epoch=17 is 0.044718633605378644
Loss at epoch=18 is 0.04447460823761133
Loss at epoch=19 is 0.0442192483388033
Loss at epoch=20 is 0.04400799662576368
Loss at epoch=21 is 0.04376856077884093
Loss at epoch=22 is 0.04363291956657056
Loss at epoch=23 is 0.04345763916156599
Loss at epoch=24 is 0.04331726985371995
Loss at epoch=25 is 0.04319204810565873
Loss at epoch=26 is 0.04308330950426218
Loss at epoch=27 is 0.04298171088848944
Loss at epoch=28 is 0.04289157757047945
Loss at epoch=29 is 0.04280694791231791
Loss at epoch=30 is 0.0427336816237222
Loss at epoch=31 is 0.042668215427135864
Loss at epoch=32 is 0.04260939226555159
Loss at epoch=33 is 0.04255772516388436
Loss at epoch=34 is 0.0425068990548807
Loss at epoch=35 is 0.04246583376688097
```

```
Loss at epoch=36 is 0.04242843245045669
Loss at epoch=37 is 0.042393632039261525
Loss at epoch=38 is 0.042361847399606176
Loss at epoch=39 is 0.04233475022165459
Loss at epoch=40 is 0.04230920542534165
Loss at epoch=41 is 0.042287307821253776
Loss at epoch=42 is 0.04226692269154898
Loss at epoch=43 is 0.042248939909447135
Loss at epoch=44 is 0.04223230732260851
Loss at epoch=45 is 0.04221789156744292
Loss at epoch=46 is 0.04220456420589163
Loss at epoch=47 is 0.042192673260727905
Loss at epoch=48 is 0.04218208975510883
Loss at epoch=49 is 0.04217244767411776
Loss at epoch=50 is 0.04216377322408993
Loss at epoch=51 is 0.042155917281395186
Loss at epoch=52 is 0.042149002086297405
Loss at epoch=53 is 0.04214268254659933
Loss at epoch=54 is 0.042137004142124984
Loss at epoch=55 is 0.042131896433065176
Loss at epoch=56 is 0.0421273099409105
Loss at epoch=57 is 0.042123178153179626
Loss at epoch=58 is 0.04211945026678195
Loss at epoch=59 is 0.04211612192162175
Loss at epoch=60 is 0.0421131096089684
Loss at epoch=61 is 0.042110398743792986
Loss at epoch=62 is 0.04210796255215192
Loss at epoch=63 is 0.04210576785401824
Loss at epoch=64 is 0.0421037931675433
Loss at epoch=65 is 0.04210201629895581
Loss at epoch=66 is 0.042100416744911696
Loss at epoch=67 is 0.042098976518203976
Loss at epoch=68 is 0.04209768297395485
Loss at epoch=69 is 0.04209651740702001
Loss at epoch=70 is 0.04209546851511762
Loss at epoch=71 is 0.04209452462369686
Loss at epoch=72 is 0.0420936742561814
Loss at epoch=73 is 0.04209291013455318
Loss at epoch=74 is 0.042092222138996986
Loss at epoch=75 is 0.04209160217993313
Loss at epoch=76 is 0.0420910456523914
Loss at epoch=77 is 0.042090543775000336
Loss at epoch=78 is 0.04209009275820086
Loss at epoch=79 is 0.042089686345584854
Loss at epoch=80 is 0.04208932084057923
```

```
Loss at epoch=81 is 0.04208899179278535
Loss at epoch=82 is 0.042088695627575445
Loss at epoch=83 is 0.04208842913665628
Loss at epoch=84 is 0.042088189287689
Loss at epoch=85 is 0.042087973383463234
Loss at epoch=86 is 0.04208777907212799
Loss at epoch=87 is 0.042087604228488756
Loss at epoch=88 is 0.04208744685225124
Loss at epoch=89 is 0.04208730518816295
Loss at epoch=90 is 0.04208717776968512
Loss at epoch=91 is 0.042087063057920135
Loss at epoch=92 is 0.042086959800146095
Loss at epoch=93 is 0.04208686686495965
Loss at epoch=94 is 0.04208678325220291
Loss at epoch=95 is 0.042086707975451054
Loss at epoch=96 is 0.04208664021754291
Loss at epoch=97 is 0.04208657927211323
Loss at epoch=98 is 0.042086524401628155
Loss at epoch=99 is 0.04208647500967623
```

In [27]:

```
# Train regularized logistic regression
# You should get the optimal weights and a list of objective values by using gradient_descent function.

lam = 1E-6
stepsize = 0.1
w = numpy.zeros((d, 1))
reg_log_stoch_gradient_descent_w, reg_log_objvals_sgd = sgd(x_train, y_train, lam, stepsize, w, 100, True)
```

```
Loss at epoch=0 is [0.11632974]
Loss at epoch=1 is [0.07166799]
Loss at epoch=2 is [0.06410202]
Loss at epoch=3 is [0.06046332]
Loss at epoch=4 is [0.05759188]
Loss at epoch=5 is [0.0545317]
Loss at epoch=6 is [0.0529521]
Loss at epoch=7 is [0.05205668]
Loss at epoch=8 is [0.05050454]
Loss at epoch=9 is [0.04941367]
Loss at epoch=10 is [0.04858536]
Loss at epoch=11 is [0.04788664]
Loss at epoch=12 is [0.04727181]
Loss at epoch=13 is [0.04662983]
Loss at epoch=14 is [0.04617167]
Loss at epoch=15 is [0.04550323]
Loss at epoch=16 is [0.04537741]
```

```
Loss at epoch=17 is [0.04506104]
Loss at epoch=18 is [0.04471916]
Loss at epoch=19 is [0.04447279]
Loss at epoch=20 is [0.04425382]
Loss at epoch=21 is [0.04405721]
Loss at epoch=22 is [0.04385658]
Loss at epoch=23 is [0.04367466]
Loss at epoch=24 is [0.04355995]
Loss at epoch=25 is [0.04343082]
Loss at epoch=26 is [0.04331221]
Loss at epoch=27 is [0.04320959]
Loss at epoch=28 is [0.04311543]
Loss at epoch=29 is [0.04303089]
Loss at epoch=30 is [0.0429534]
Loss at epoch=31 is [0.04288496]
Loss at epoch=32 is [0.04282904]
Loss at epoch=33 is [0.04277451]
Loss at epoch=34 is [0.04272579]
Loss at epoch=35 is [0.04268333]
Loss at epoch=36 is [0.04264336]
Loss at epoch=37 is [0.04260822]
Loss at epoch=38 is [0.04257722]
Loss at epoch=39 is [0.04254872]
Loss at epoch=40 is [0.04252343]
Loss at epoch=41 is [0.04250056]
Loss at epoch=42 is [0.04247986]
Loss at epoch=43 is [0.04246125]
Loss at epoch=44 is [0.04244468]
Loss at epoch=45 is [0.04242979]
Loss at epoch=46 is [0.04241625]
Loss at epoch=47 is [0.04240412]
Loss at epoch=48 is [0.04239326]
Loss at epoch=49 is [0.04238349]
Loss at epoch=50 is [0.04237463]
Loss at epoch=51 is [0.04236667]
Loss at epoch=52 is [0.04235953]
Loss at epoch=53 is [0.04235316]
Loss at epoch=54 is [0.04234739]
Loss at epoch=55 is [0.0423422]
Loss at epoch=56 is [0.04233752]
Loss at epoch=57 is [0.04233332]
Loss at epoch=58 is [0.04232952]
Loss at epoch=59 is [0.04232612]
Loss at epoch=60 is [0.04232306]
Loss at epoch=61 is [0.0423203]
```

```
Loss at epoch=62 is [0.04231782]
Loss at epoch=63 is [0.04231558]
Loss at epoch=64 is [0.04231358]
Loss at epoch=65 is [0.04231177]
Loss at epoch=66 is [0.04231014]
Loss at epoch=67 is [0.04230868]
Loss at epoch=68 is [0.04230736]
Loss at epoch=69 is [0.04230617]
Loss at epoch=70 is [0.04230511]
Loss at epoch=71 is [0.04230415]
Loss at epoch=72 is [0.04230328]
Loss at epoch=73 is [0.0423025]
Loss at epoch=74 is [0.0423018]
Loss at epoch=75 is [0.04230117]
Loss at epoch=76 is [0.04230061]
Loss at epoch=77 is [0.04230009]
Loss at epoch=78 is [0.04229963]
Loss at epoch=79 is [0.04229922]
Loss at epoch=80 is [0.04229885]
Loss at epoch=81 is [0.04229851]
Loss at epoch=82 is [0.04229821]
Loss at epoch=83 is [0.04229794]
Loss at epoch=84 is [0.0422977]
Loss at epoch=85 is [0.04229748]
Loss at epoch=86 is [0.04229728]
Loss at epoch=87 is [0.0422971]
Loss at epoch=88 is [0.04229694]
Loss at epoch=89 is [0.0422968]
Loss at epoch=90 is [0.04229667]
Loss at epoch=91 is [0.04229655]
Loss at epoch=92 is [0.04229645]
Loss at epoch=93 is [0.04229635]
Loss at epoch=94 is [0.04229627]
Loss at epoch=95 is [0.04229619]
Loss at epoch=96 is [0.04229612]
Loss at epoch=97 is [0.04229606]
Loss at epoch=98 is [0.042296]
Loss at epoch=99 is [0.04229595]
```

3.3 Mini-Batch Gradient Descent (MBGD)

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

You may need to implement a new function to calculate the new objective function and gradients.

In [28]:

```
# Calculate the objective Q_I and the gradient of Q_I
# Inputs:
#     w: weights: d-by-b matrix
#     xi: data: b-by-d matrix
#     yi: label: scalar
#     lam: scalar, the regularization parameter
# Return:
#     obj: scalar, the objective Q_i
#     g: d-by-1 matrix, gradient of Q_i

def mb_objective_gradient(w, xi, yi, lam,regularized):
    d = xi.shape[1]
    yx = numpy.multiply(yi, xi)  # b-by-d matrix
    yxw = numpy.dot(yx, w)       # b-by-1 matrix

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

    # calculate stochastic gradient
    g_loss = -yx/ (1+ numpy.exp(yxw))
    g_loss2 = numpy.mean(g_loss, axis=0).reshape(d,1)
    g=(g_loss2 + lam*w)

    return obj, g
```

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_objective_gradient` function. In the next iteration, use the next b samples, and so on.

In [31]:

```
# MBGD for solving logistic regression
# You will need to do iterative process (loops) to obtain optimal weights in this function

# Inputs:
#     x: data: n-by-d matrix
#     y: label: n-by-1 matrix
#     lam: scalar, the regularization parameter
#     learning_rate: scalar
#     w: weights: d-by-1 matrix, initialization of w
#     max_epoch: integer, the maximal epochs
# Return:
#     w: weights: d-by-1 matrix, the solution
#     objvals: a record of each epoch's objective value
#     Record one objective value per epoch (not per iteration)


def mbgd(x, y, lam, learning_rate, w, max_epoch=100, b=8, regularized=False): ## added b and reg term
    n,d = x.shape
    objvals=numpy.zeros(max_epoch)
    if w is None:
        w = numpy.zeros((d,1))

    for t in range(max_epoch):
        #rand_indices = numpy.random.permutation(n).reshape((n // b, b)) # left few samples
        rand_indices = numpy.random.permutation(n)
        x_rand = x[rand_indices, :]
        y_rand = y[rand_indices, :]

        #create batches
        batches = []
        data = numpy.hstack((x_rand,y_rand))
        num_batches =data.shape[0] // b
        for i in range(num_batches):  # even 5 entries missed
            #print(i)
            batch=data[i*b:(i+1)*b,:]
            xb=batch[:,:-1]
            yb=batch[:,-1]
            batches.append((xb,yb))

        objval = 0
        for batch in batches:
            xi,yi = batch
            obj, g = mb_objective_gradient(w, xi, yi, lam, regularized)
            objval += obj
```

```
            w -= learning_rate * g

        learning_rate *= 0.9
        objval /= (n/b)
        objvals[t] = objval
        print('Loss at epoch='+str(t)+' is '+str(objval))


    return w, objvals
```

Use mbgd function to obtain your optimal weights and a list of objective values over each epoch.

In [32]:

```
# Train logistic regression
# You should get the optimal weights and a list of objective values by using gradient_descent function.

lam = 1E-6
b = 8
stepsize = 0.1
w = numpy.zeros((d, 1))

log_mini_batch_gradient_descent_w, log_objvals_mbsgd8 = mbgd(x_train, y_train, lam, stepsize, w, 100, b,False)
```

```
Loss at epoch=0 is 0.21253605724880206
Loss at epoch=1 is 0.11376346082211844
Loss at epoch=2 is 0.09447592133244687
Loss at epoch=3 is 0.09089787931068619
Loss at epoch=4 is 0.08551455242390152
Loss at epoch=5 is 0.08246524380315806
Loss at epoch=6 is 0.07969960980918057
Loss at epoch=7 is 0.07543308426314368
Loss at epoch=8 is 0.07599911403177244
Loss at epoch=9 is 0.07463461698931814
Loss at epoch=10 is 0.06510447648411068
Loss at epoch=11 is 0.07002147456438937
Loss at epoch=12 is 0.07137039662029626
Loss at epoch=13 is 0.07163160031235098
Loss at epoch=14 is 0.0711044660154244
Loss at epoch=15 is 0.0699173337802074
Loss at epoch=16 is 0.06678556816473832
Loss at epoch=17 is 0.06823104714680953
Loss at epoch=18 is 0.06947861666298952
Loss at epoch=19 is 0.06764860761607475
Loss at epoch=20 is 0.06871956674547515
Loss at epoch=21 is 0.06813515743942408
```

```
Loss at epoch=22 is 0.06799704975204722
Loss at epoch=23 is 0.06727230858382963
Loss at epoch=24 is 0.06777017407265253
Loss at epoch=25 is 0.06806384775115715
Loss at epoch=26 is 0.06638091246862388
Loss at epoch=27 is 0.06742234795236246
Loss at epoch=28 is 0.06721450696547279
Loss at epoch=29 is 0.06749244246348185
Loss at epoch=30 is 0.06750332731003235
Loss at epoch=31 is 0.06717930827839856
Loss at epoch=32 is 0.06656705182295253
Loss at epoch=33 is 0.06723355478165483
Loss at epoch=34 is 0.06711204139031676
Loss at epoch=35 is 0.06696829044556116
Loss at epoch=36 is 0.06714780492820098
Loss at epoch=37 is 0.066792033874887
Loss at epoch=38 is 0.06678883037480679
Loss at epoch=39 is 0.06720296938491742
Loss at epoch=40 is 0.066829599927815
Loss at epoch=41 is 0.06533717489322957
Loss at epoch=42 is 0.0668985586510543
Loss at epoch=43 is 0.0667252726430094
Loss at epoch=44 is 0.0660826490038976
Loss at epoch=45 is 0.06687380097025633
Loss at epoch=46 is 0.06686363691407385
Loss at epoch=47 is 0.06669370420797921
Loss at epoch=48 is 0.06677059947651726
Loss at epoch=49 is 0.0670758622873998
Loss at epoch=50 is 0.06581052199225655
Loss at epoch=51 is 0.06276496302773672
Loss at epoch=52 is 0.0668737288146295
Loss at epoch=53 is 0.06265020440258787
Loss at epoch=54 is 0.06636437859982755
Loss at epoch=55 is 0.06703289950524195
Loss at epoch=56 is 0.06640096252039807
Loss at epoch=57 is 0.06680771194496328
Loss at epoch=58 is 0.06656575767377582
Loss at epoch=59 is 0.06677764912753685
Loss at epoch=60 is 0.0667845932239578
Loss at epoch=61 is 0.06695905225496475
Loss at epoch=62 is 0.06638671337759519
Loss at epoch=63 is 0.0667528094432481
Loss at epoch=64 is 0.06464915169452344
Loss at epoch=65 is 0.06402248582978431
Loss at epoch=66 is 0.0642206971639221
```

```
Loss at epoch=67 is 0.06577702849922155
Loss at epoch=68 is 0.06072953627394712
Loss at epoch=69 is 0.06589418721793423
Loss at epoch=70 is 0.06685530700404126
Loss at epoch=71 is 0.06493428452503881
Loss at epoch=72 is 0.06687097676791924
Loss at epoch=73 is 0.0664920292591044
Loss at epoch=74 is 0.06657293794562438
Loss at epoch=75 is 0.06663842454840486
Loss at epoch=76 is 0.06472337237241782
Loss at epoch=77 is 0.06576682478893152
Loss at epoch=78 is 0.06589845551878958
Loss at epoch=79 is 0.06518648696466345
Loss at epoch=80 is 0.06682954262300998
Loss at epoch=81 is 0.06689393050489452
Loss at epoch=82 is 0.06545573970695433
Loss at epoch=83 is 0.06675771486956202
Loss at epoch=84 is 0.06618528240541795
Loss at epoch=85 is 0.06659840569926798
Loss at epoch=86 is 0.06672901755461279
Loss at epoch=87 is 0.06614197773150429
Loss at epoch=88 is 0.06568545475497253
Loss at epoch=89 is 0.0668590627017882
Loss at epoch=90 is 0.0666990791135619
Loss at epoch=91 is 0.06562133411133224
Loss at epoch=92 is 0.06648308174446549
Loss at epoch=93 is 0.0666236599348444
Loss at epoch=94 is 0.066808165717877
Loss at epoch=95 is 0.06623661634189523
Loss at epoch=96 is 0.0665120019027845
Loss at epoch=97 is 0.06642518293308453
Loss at epoch=98 is 0.06624056328791712
Loss at epoch=99 is 0.06588367803627831
```

In [33]:

```
# Train regularized logistic regression
# You should get the optimal weights and a list of objective values by using gradient_descent function.

lam = 1E-6
b = 8
stepsize = 0.1
w = numpy.zeros((d, 1))

reg_log_mini_batch_gradient_descent_w, reg_log_objvals_mbsgd8 = mbgd(x_train, y_train, lam, stepsize, w, 100, b
```

```
Loss at epoch=0 is 0.2091117304565086
Loss at epoch=1 is 0.11443676869822697
Loss at epoch=2 is 0.09634181564519334
Loss at epoch=3 is 0.09043646082670598
Loss at epoch=4 is 0.0852465680963402
Loss at epoch=5 is 0.08169071092515771
Loss at epoch=6 is 0.07859455644716984
Loss at epoch=7 is 0.07738952853030964
Loss at epoch=8 is 0.0739320173130774
Loss at epoch=9 is 0.071139050936365
Loss at epoch=10 is 0.07324444056404038
Loss at epoch=11 is 0.07272808785909904
Loss at epoch=12 is 0.0692478380579584
Loss at epoch=13 is 0.0694452258567772
Loss at epoch=14 is 0.07027129237292008
Loss at epoch=15 is 0.06927455516865062
Loss at epoch=16 is 0.06999867242940025
Loss at epoch=17 is 0.06950890214244458
Loss at epoch=18 is 0.06573137967633699
Loss at epoch=19 is 0.06823519254109348
Loss at epoch=20 is 0.06840783742004558
Loss at epoch=21 is 0.06558640101439742
Loss at epoch=22 is 0.06837100738803918
Loss at epoch=23 is 0.0682361664437106
Loss at epoch=24 is 0.06684858643312443
Loss at epoch=25 is 0.06797544208675421
Loss at epoch=26 is 0.0675082635567911
Loss at epoch=27 is 0.06732427136522445
Loss at epoch=28 is 0.06758748403086663
Loss at epoch=29 is 0.06737948704127554
Loss at epoch=30 is 0.0672810371662463
Loss at epoch=31 is 0.06709730708937414
Loss at epoch=32 is 0.06652897623344128
Loss at epoch=33 is 0.06711491481467899
Loss at epoch=34 is 0.06688876738412264
Loss at epoch=35 is 0.06674760681245238
Loss at epoch=36 is 0.06508076472674587
Loss at epoch=37 is 0.06699683589231968
Loss at epoch=38 is 0.0657871442036773
Loss at epoch=39 is 0.0667693385352642
Loss at epoch=40 is 0.06688824262692501
Loss at epoch=41 is 0.06687791551895432
Loss at epoch=42 is 0.06675843837796112
Loss at epoch=43 is 0.06687441148463316
Loss at epoch=44 is 0.06671911508652587
```

```
Loss at epoch=45 is 0.06642309099451085
Loss at epoch=46 is 0.06594899869318177
Loss at epoch=47 is 0.06645947586375915
Loss at epoch=48 is 0.06684361569862954
Loss at epoch=49 is 0.06657890897572749
Loss at epoch=50 is 0.06686940312281729
Loss at epoch=51 is 0.0665535350266495
Loss at epoch=52 is 0.06581643271213881
Loss at epoch=53 is 0.06650959817763478
Loss at epoch=54 is 0.06685280483800013
Loss at epoch=55 is 0.06683085778298085
Loss at epoch=56 is 0.06516078381438858
Loss at epoch=57 is 0.06659766309603875
Loss at epoch=58 is 0.06460864005934411
Loss at epoch=59 is 0.0667066518009516
Loss at epoch=60 is 0.06571170842022504
Loss at epoch=61 is 0.0667541066686791
Loss at epoch=62 is 0.06638637408836898
Loss at epoch=63 is 0.06561584973889524
Loss at epoch=64 is 0.06655272572305988
Loss at epoch=65 is 0.06538586108004511
Loss at epoch=66 is 0.0667643726981288
Loss at epoch=67 is 0.06649416087079651
Loss at epoch=68 is 0.06673597433379053
Loss at epoch=69 is 0.06511904311930748
Loss at epoch=70 is 0.06682458962668185
Loss at epoch=71 is 0.06484890907547224
Loss at epoch=72 is 0.06669513887942129
Loss at epoch=73 is 0.0655580984615466
Loss at epoch=74 is 0.06585506998067549
Loss at epoch=75 is 0.06563867575429237
Loss at epoch=76 is 0.06679390459368222
Loss at epoch=77 is 0.06432910376085191
Loss at epoch=78 is 0.06653418288535558
Loss at epoch=79 is 0.066353142952012
Loss at epoch=80 is 0.057263725954195245
Loss at epoch=81 is 0.06677470529222919
Loss at epoch=82 is 0.06674336347770672
Loss at epoch=83 is 0.06658379899447968
Loss at epoch=84 is 0.06652219565808233
Loss at epoch=85 is 0.06634811534407425
Loss at epoch=86 is 0.06638334624765266
Loss at epoch=87 is 0.06668229457767992
Loss at epoch=88 is 0.06581449694159344
Loss at epoch=89 is 0.06673215894808913
```

```
Loss at epoch=90 is 0.06651723096172162
Loss at epoch=91 is 0.06623601210107724
Loss at epoch=92 is 0.0668147485884125
Loss at epoch=93 is 0.06659714500586542
Loss at epoch=94 is 0.06556091252478603
Loss at epoch=95 is 0.06541837461662864
Loss at epoch=96 is 0.06654173499113442
Loss at epoch=97 is 0.06667863878590864
Loss at epoch=98 is 0.06631049111961096
Loss at epoch=99 is 0.06652861564862055
```

4. Compare GD, SGD, MBGD

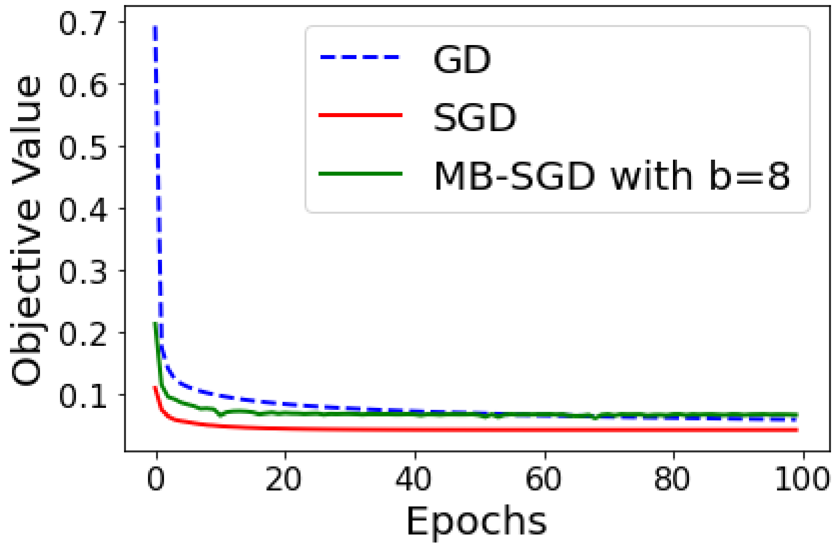
Plot objective function values against epochs.

In [34]:

```
import matplotlib.pyplot as plt
%matplotlib inline
fig = plt.figure(figsize=(6, 4))

epochs_gd = range(len(log_objvals_gd))
epochs_sgd = range(len(log_objvals_sgd))
epochs_mbsgd8 = range(len(log_objvals_mbsgd8))

line0, = plt.plot(epochs_gd, log_objvals_gd, '--b', linewidth=2)
line1, = plt.plot(epochs_sgd, log_objvals_sgd, '-r', linewidth=2)
line2, = plt.plot(epochs_mbsgd8, log_objvals_mbsgd8,'-g', 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, line2], ['GD', 'SGD','MB-SGD with b=8'], fontsize=20)
plt.tight_layout()
plt.show()
fig.savefig('compare_gd_sgd_mbsgd8_mbgd64.pdf', format='pdf', dpi=1200)
```



5. Prediction

Compare the training and testing accuracy for logistic regression and regularized logistic regression.

In [35]:

```
# Predict class label
# Inputs:
#     w: weights: d-by-1 matrix
#     X: data: 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 [36]:

```
# evaluate training error of logistic regression and regularized version
weightsList = {
    'log_gradient': log_gradient_descent_w,
    'reg_log_gradient': reg_log_gradient_descent_w,
    'log_stoch_gradient': log_stoch_gradient_descent_w,
    'reg_log_stoch_gradient': reg_log_stoch_gradient_descent_w,
    'log_mini_batch_gradient': log_mini_batch_gradient_descent_w,
```

```
    'reg_log_mini_batch_gradient': reg_log_mini_batch_gradient_descent_w
}

training_errors = {}

for key in weightsList:
    f_train = predict(weightsList[key], x_train)
    diff = numpy.abs(f_train - y_train) / 2
    error_train = numpy.mean(diff)
    training_errors[key] = error_train
training_errors
```

Out[36]:

```
{'log_gradient': 0.017582417582417582,
 'reg_log_gradient': 0.017582417582417582,
 'log_stoch_gradient': 0.01098901098901099,
 'reg_log_stoch_gradient': 0.01098901098901099,
 'log_mini_batch_gradient': 0.01978021978021978,
 'reg_log_mini_batch_gradient': 0.017582417582417582}
```

In [37]:

```
# evaluate testing error of logistic regression and regularized version
testing_errors = {}

for key in weightsList:
    f_test = predict(weightsList[key], x_test)
    diff = numpy.abs(f_test - y_test) / 2
    error_test = numpy.mean(diff)
    testing_errors[key] = error_test
testing_errors
```

Out[37]:

```
{'log_gradient': 0.008771929824561403,
 'reg_log_gradient': 0.008771929824561403,
 'log_stoch_gradient': 0.043859649122807015,
 'reg_log_stoch_gradient': 0.03508771929824561,
 'log_mini_batch_gradient': 0.008771929824561403,
 'reg_log_mini_batch_gradient': 0.008771929824561403}
```

6. Parameters tuning

In this section, you may try different combinations of parameters (regularization value, learning rate, etc) to see their effects on the model. (Open ended question)

In [39]:
```
# logestic reg with diff learning rate : 01
learning_rate = 0.1
w = numpy.zeros((d, 1))
lam = 0

tune_log_gradient_descent_w, tune_log_objvals_gd = gradient_descent(x_train, y_train, lam, learning_rate, w,reg
```

```
Loss at epoch=0 is 0.6931471805599453
Loss at epoch=1 is 0.5254593148536312
Loss at epoch=2 is 0.43955101701585386
Loss at epoch=3 is 0.38664970293447576
Loss at epoch=4 is 0.350099973714949
Loss at epoch=5 is 0.32297035924512524
Loss at epoch=6 is 0.30182466816032316
Loss at epoch=7 is 0.28474842440438886
Loss at epoch=8 is 0.2705831661751768
Loss at epoch=9 is 0.25858324533457877
Loss at epoch=10 is 0.24824504813068335
Loss at epoch=11 is 0.23921500288739833
Loss at epoch=12 is 0.23123683430104827
Loss at epoch=13 is 0.22411975000446213
Loss at epoch=14 is 0.21771842986732814
Loss at epoch=15 is 0.2119199855641099
Loss at epoch=16 is 0.20663520105938923
Loss at epoch=17 is 0.2017924922746107
Loss at epoch=18 is 0.19733364515111582
Loss at epoch=19 is 0.1932107469828583
Loss at epoch=20 is 0.18938393672125983
Loss at epoch=21 is 0.185819728778289
Loss at epoch=22 is 0.18248974572222826
Loss at epoch=23 is 0.17936974726756322
Loss at epoch=24 is 0.17643887714273768
Loss at epoch=25 is 0.17367907233305438
Loss at epoch=26 is 0.171074594832335
Loss at epoch=27 is 0.1686116568817222
Loss at epoch=28 is 0.16627811830736638
Loss at epoch=29 is 0.16406324001451594
Loss at epoch=30 is 0.16195748162884524
Loss at epoch=31 is 0.15995233414915258
Loss at epoch=32 is 0.15804018059665537
Loss at epoch=33 is 0.15621417922727562
Loss at epoch=34 is 0.15446816506284125
Loss at epoch=35 is 0.15279656639979883
Loss at epoch=36 is 0.15119433364468038
```

```
Loss at epoch=37 is 0.14965687835815988
Loss at epoch=38 is 0.1481800208033525
Loss at epoch=39 is 0.14675994461788908
Loss at epoch=40 is 0.14539315748455514
Loss at epoch=41 is 0.14407645687783285
Loss at epoch=42 is 0.14280690012544162
Loss at epoch=43 is 0.1415817781539753
Loss at epoch=44 is 0.14039859239283747
Loss at epoch=45 is 0.13925503439614087
Loss at epoch=46 is 0.13814896781211738
Loss at epoch=47 is 0.13707841238702126
Loss at epoch=48 is 0.13604152973795447
Loss at epoch=49 is 0.13503661066842387
Loss at epoch=50 is 0.13406206383327338
Loss at epoch=51 is 0.1331164055871284
Loss at epoch=52 is 0.13219825087360484
Loss at epoch=53 is 0.13130630503204713
Loss at epoch=54 is 0.1304393564150855
Loss at epoch=55 is 0.12959626972435534
Loss at epoch=56 is 0.1287759799837037
Loss at epoch=57 is 0.12797748707946283
Loss at epoch=58 is 0.12719985080616877
Loss at epoch=59 is 0.12644218636367674
Loss at epoch=60 is 0.12570366025816024
Loss at epoch=61 is 0.12498348656513598
Loss at epoch=62 is 0.12428092351756093
Loss at epoch=63 is 0.12359527038631334
Loss at epoch=64 is 0.12292586462408583
Loss at epoch=65 is 0.12227207924696509
Loss at epoch=66 is 0.1216333204308139
Loss at epoch=67 is 0.12100902530206344
Loss at epoch=68 is 0.12039865990471293
Loss at epoch=69 is 0.11980171732726339
Loss at epoch=70 is 0.11921771597501148
Loss at epoch=71 is 0.11864619797463313
Loss at epoch=72 is 0.11808672769931751
Loss at epoch=73 is 0.11753889040389064
Loss at epoch=74 is 0.11700229096041588
Loss at epoch=75 is 0.11647655268569142
Loss at epoch=76 is 0.11596131625289348
Loss at epoch=77 is 0.11545623868035633
Loss at epoch=78 is 0.11496099239114105
Loss at epoch=79 is 0.11447526433763776
Loss at epoch=80 is 0.11399875518597667
Loss at epoch=81 is 0.11353117855549925
```

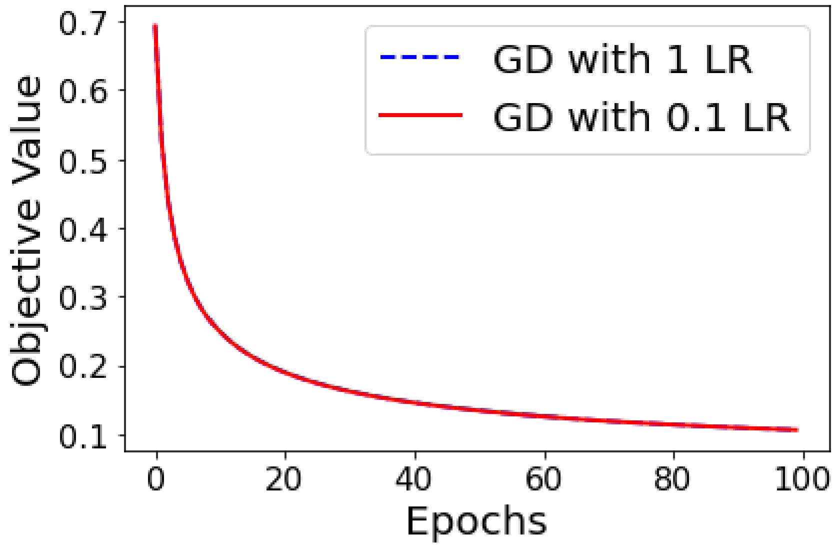
```
Loss at epoch=82 is 0.11307226030896872
Loss at epoch=83 is 0.11262173788958407
Loss at epoch=84 is 0.11217935970120811
Loss at epoch=85 is 0.11174488452853254
Loss at epoch=86 is 0.11131808099418589
Loss at epoch=87 is 0.11089872705004408
Loss at epoch=88 is 0.11048660950023567
Loss at epoch=89 is 0.11008152355354142
Loss at epoch=90 is 0.10968327240307882
Loss at epoch=91 is 0.10929166683133322
Loss at epoch=92 is 0.10890652483875529
Loss at epoch=93 is 0.10852767129428444
Loss at epoch=94 is 0.10815493760629061
Loss at epoch=95 is 0.10778816141254224
Loss at epoch=96 is 0.10742718628791777
Loss at epoch=97 is 0.10707186146867588
Loss at epoch=98 is 0.10672204159219001
Loss at epoch=99 is 0.10637758645113482
```

In [42]:

```
import matplotlib.pyplot as plt
%matplotlib inline
fig = plt.figure(figsize=(6, 4))

epochs_gd = range(len(log_objvals_gd))
tune_epochs_gd = range(len(tune_log_objvals_gd))

line0, = plt.plot(epochs_gd, log_objvals_gd, '--b', linewidth=2)
line1, = plt.plot(tune_epochs_gd, tune_log_objvals_gd, '-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, line2], ['GD with 1 LR', 'GD with 0.1 LR',], fontsize=20)
plt.tight_layout()
plt.show()
fig.savefig('GD with differnt learning rates.pdf', format='pdf', dpi=1200)
```



In [45]:

```
# MB SGB with diff batch size

lam = 1E-6
b = 15
stepsize = 0.1
w = numpy.zeros((d, 1))

tune_log_mini_batch_gradient_descent_w, tune_log_objvals_mbsgd8 = mbgd(x_train, y_train, lam, stepsize, w, 100,
```

```
Loss at epoch=0 is 0.2632273734599003
Loss at epoch=1 is 0.14263278332894322
Loss at epoch=2 is 0.11455116059980051
Loss at epoch=3 is 0.10677659338817001
Loss at epoch=4 is 0.10178468060253038
Loss at epoch=5 is 0.09581624336154942
Loss at epoch=6 is 0.09306477060986641
Loss at epoch=7 is 0.0915313227287497
Loss at epoch=8 is 0.08921932797967617
Loss at epoch=9 is 0.08769993339173311
Loss at epoch=10 is 0.08486550314685001
Loss at epoch=11 is 0.08488586742380548
Loss at epoch=12 is 0.084216379035246
Loss at epoch=13 is 0.08340594789616965
Loss at epoch=14 is 0.08265848422964214
Loss at epoch=15 is 0.08175318515816045
Loss at epoch=16 is 0.08114458460594652
```

```
Loss at epoch=17 is 0.08059114208974841
Loss at epoch=18 is 0.08030234116874188
Loss at epoch=19 is 0.07946199607290512
Loss at epoch=20 is 0.07969126718002469
Loss at epoch=21 is 0.07930472271178624
Loss at epoch=22 is 0.07945726402177379
Loss at epoch=23 is 0.07777349518385213
Loss at epoch=24 is 0.0789762913544886
Loss at epoch=25 is 0.07789470490245518
Loss at epoch=26 is 0.07868706514233663
Loss at epoch=27 is 0.07710214095898066
Loss at epoch=28 is 0.07737690807811753
Loss at epoch=29 is 0.0776823027935473
Loss at epoch=30 is 0.07835941869948909
Loss at epoch=31 is 0.0770884853965492
Loss at epoch=32 is 0.07646570531765051
Loss at epoch=33 is 0.07666075843835252
Loss at epoch=34 is 0.07814058290656156
Loss at epoch=35 is 0.07668099047821651
Loss at epoch=36 is 0.07760325105781313
Loss at epoch=37 is 0.07794553217109579
Loss at epoch=38 is 0.07791029361154339
Loss at epoch=39 is 0.07754499188576265
Loss at epoch=40 is 0.0762631015318568
Loss at epoch=41 is 0.0772773475680035
Loss at epoch=42 is 0.06801012278074202
Loss at epoch=43 is 0.07762070259811399
Loss at epoch=44 is 0.07671592674049602
Loss at epoch=45 is 0.07587829408621126
Loss at epoch=46 is 0.07737769084937107
Loss at epoch=47 is 0.07769763933342375
Loss at epoch=48 is 0.07738208284135631
Loss at epoch=49 is 0.07767798104959742
Loss at epoch=50 is 0.07428160598076483
Loss at epoch=51 is 0.07710085129168137
Loss at epoch=52 is 0.07609973123246763
Loss at epoch=53 is 0.07764263024759481
Loss at epoch=54 is 0.07698600843489957
Loss at epoch=55 is 0.07692398344109214
Loss at epoch=56 is 0.077472706923711
Loss at epoch=57 is 0.07760639164795287
Loss at epoch=58 is 0.07736885740155715
Loss at epoch=59 is 0.0768835847468131
Loss at epoch=60 is 0.07567513736845483
Loss at epoch=61 is 0.07741851343791968
```

```
Loss at epoch=62 is 0.07754455097338926
Loss at epoch=63 is 0.0768323577410191
Loss at epoch=64 is 0.07697669756999856
Loss at epoch=65 is 0.07689653033743012
Loss at epoch=66 is 0.07730017470838198
Loss at epoch=67 is 0.0769003288836291
Loss at epoch=68 is 0.07758406728447277
Loss at epoch=69 is 0.07761863480783997
Loss at epoch=70 is 0.07641031195504337
Loss at epoch=71 is 0.07711444131891063
Loss at epoch=72 is 0.077399514289918
Loss at epoch=73 is 0.07732270267452047
Loss at epoch=74 is 0.07759811686047864
Loss at epoch=75 is 0.07722881741612873
Loss at epoch=76 is 0.07607980751397139
Loss at epoch=77 is 0.0771191419026475
Loss at epoch=78 is 0.07603465162079545
Loss at epoch=79 is 0.07756290994133716
Loss at epoch=80 is 0.07755203005779589
Loss at epoch=81 is 0.07747541756359494
Loss at epoch=82 is 0.0769257854721415
Loss at epoch=83 is 0.07638065756732262
Loss at epoch=84 is 0.07578591618490946
Loss at epoch=85 is 0.07326170034210469
Loss at epoch=86 is 0.07723315769681684
Loss at epoch=87 is 0.07749875958567246
Loss at epoch=88 is 0.07748761846126018
Loss at epoch=89 is 0.07756848752871343
Loss at epoch=90 is 0.07740824606276422
Loss at epoch=91 is 0.07743377014283219
Loss at epoch=92 is 0.07756216385072691
Loss at epoch=93 is 0.0755367206121165
Loss at epoch=94 is 0.07685083639689409
Loss at epoch=95 is 0.07643648662759825
Loss at epoch=96 is 0.0774436069700923
Loss at epoch=97 is 0.07723221550329988
Loss at epoch=98 is 0.07708591282682137
Loss at epoch=99 is 0.07548886090048208
```

In [49]:

```
lam = 1E-6
b = 70
stepsize = 0.1
w = numpy.zeros((d, 1))

tune1_log_mini_batch_gradient_descent_w, tune1_log_objvals_mbsgd8 = mbgd(x_train, y_train, lam, stepsize, w, 10
```

```
Loss at epoch=0 is 0.41455244246282247
Loss at epoch=1 is 0.24817734755701795
Loss at epoch=2 is 0.20410322160789807
Loss at epoch=3 is 0.18340172934841265
Loss at epoch=4 is 0.16708694822887551
Loss at epoch=5 is 0.15657965538664279
Loss at epoch=6 is 0.15142292808022625
Loss at epoch=7 is 0.1435999600052896
Loss at epoch=8 is 0.13727727391772804
Loss at epoch=9 is 0.13567617308061714
Loss at epoch=10 is 0.13809955953516304
Loss at epoch=11 is 0.1326315136649539
Loss at epoch=12 is 0.1314474677793897
Loss at epoch=13 is 0.12987384928924423
Loss at epoch=14 is 0.12654964927575799
Loss at epoch=15 is 0.1263845747130342
Loss at epoch=16 is 0.12301670319672198
Loss at epoch=17 is 0.12502812418134857
Loss at epoch=18 is 0.12186490849514443
Loss at epoch=19 is 0.11961636196957239
Loss at epoch=20 is 0.11869419748119439
Loss at epoch=21 is 0.12522766106416705
Loss at epoch=22 is 0.11837304201598767
Loss at epoch=23 is 0.11938243348622367
Loss at epoch=24 is 0.12086275826940031
Loss at epoch=25 is 0.12318333767379842
Loss at epoch=26 is 0.1127068135089621
Loss at epoch=27 is 0.11861157097295472
Loss at epoch=28 is 0.1155046763537335
Loss at epoch=29 is 0.1217665937216365
Loss at epoch=30 is 0.11756088677644802
Loss at epoch=31 is 0.12117463662918762
Loss at epoch=32 is 0.11891555524678041
Loss at epoch=33 is 0.12281066306652924
Loss at epoch=34 is 0.12252737200330678
Loss at epoch=35 is 0.12014685202122966
Loss at epoch=36 is 0.11668817723181593
Loss at epoch=37 is 0.11412509138394421
Loss at epoch=38 is 0.12192877678394298
Loss at epoch=39 is 0.1163636616304877
Loss at epoch=40 is 0.12042352526500166
Loss at epoch=41 is 0.1188370247084226
Loss at epoch=42 is 0.11813593625272602
Loss at epoch=43 is 0.11573608213074833
```

```
Loss at epoch=44 is 0.11893264263356312
Loss at epoch=45 is 0.11554830424874887
Loss at epoch=46 is 0.12023001771502002
Loss at epoch=47 is 0.1204271396321046
Loss at epoch=48 is 0.11371073076415761
Loss at epoch=49 is 0.11718763909714247
Loss at epoch=50 is 0.1150424061749059
Loss at epoch=51 is 0.11552458281645563
Loss at epoch=52 is 0.11351541925600607
Loss at epoch=53 is 0.11907336079043193
Loss at epoch=54 is 0.11456042195364521
Loss at epoch=55 is 0.12031998474799721
Loss at epoch=56 is 0.11659340775566848
Loss at epoch=57 is 0.11457242094921548
Loss at epoch=58 is 0.11595459020774743
Loss at epoch=59 is 0.11889501223332455
Loss at epoch=60 is 0.11746272112006613
Loss at epoch=61 is 0.11783174746459281
Loss at epoch=62 is 0.11830570756496342
Loss at epoch=63 is 0.11187521830192981
Loss at epoch=64 is 0.11404225297722753
Loss at epoch=65 is 0.1178768167047465
Loss at epoch=66 is 0.11643755359041402
Loss at epoch=67 is 0.11476659040161391
Loss at epoch=68 is 0.11900447828053763
Loss at epoch=69 is 0.11868169130910577
Loss at epoch=70 is 0.1134307037284012
Loss at epoch=71 is 0.11921828016583838
Loss at epoch=72 is 0.11243944087753094
Loss at epoch=73 is 0.11393508631433541
Loss at epoch=74 is 0.11783784218245033
Loss at epoch=75 is 0.11305688755240893
Loss at epoch=76 is 0.11599130870893012
Loss at epoch=77 is 0.11493867291031602
Loss at epoch=78 is 0.1159380874542413
Loss at epoch=79 is 0.11629265544132984
Loss at epoch=80 is 0.11784028998289336
Loss at epoch=81 is 0.1167685183277592
Loss at epoch=82 is 0.1184681019730865
Loss at epoch=83 is 0.11828146091897453
Loss at epoch=84 is 0.11670242638784348
Loss at epoch=85 is 0.11620032203451477
Loss at epoch=86 is 0.11268266242583241
Loss at epoch=87 is 0.12006827359245954
Loss at epoch=88 is 0.11579174747698179
```

```
Loss at epoch=89 is 0.11504979816723758
Loss at epoch=90 is 0.11840570716234225
Loss at epoch=91 is 0.1217538074649452
Loss at epoch=92 is 0.11782551590658909
Loss at epoch=93 is 0.11975107612423923
Loss at epoch=94 is 0.11881953105258215
Loss at epoch=95 is 0.11248483998904034
Loss at epoch=96 is 0.11853857230142487
Loss at epoch=97 is 0.11874453645199876
Loss at epoch=98 is 0.12125313010342675
Loss at epoch=99 is 0.11694345941442884
```

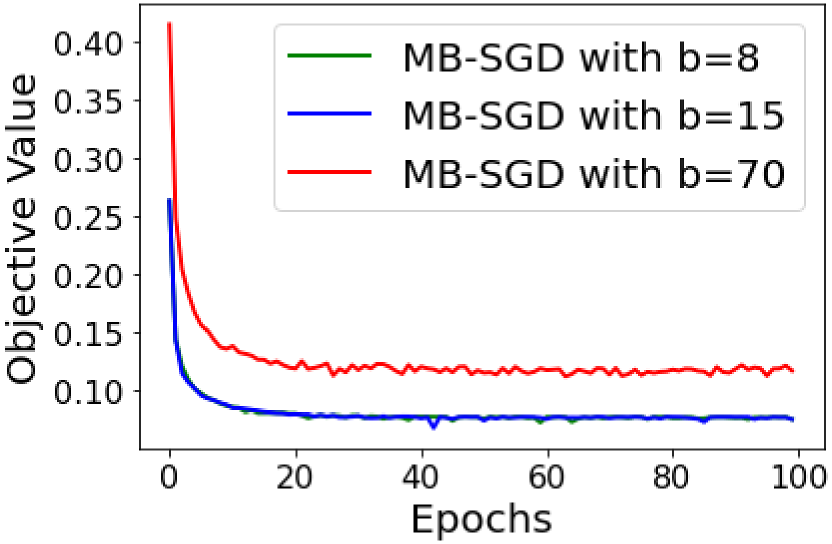
In [51]:

```
import matplotlib.pyplot as plt
%matplotlib inline
fig = plt.figure(figsize=(6, 4))


epochs_mbsgd8 = range(len(log_objvals_mbsgd8))
epochs_mbsgd15 = range(len(tune_log_objvals_mbsgd8))
epochs_mbsgd70 = range(len(tune1_log_objvals_mbsgd8))



line0, = plt.plot(epochs_mbsgd8, log_objvals_mbsgd8,'-g', linewidth=2)
line1, = plt.plot(epochs_mbsgd15, tune_log_objvals_mbsgd8,'-b', linewidth=2)
line2, = plt.plot(epochs_mbsgd70, tune1_log_objvals_mbsgd8,'-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, line2], ['MB-SGD with b=8','MB-SGD with b=15','MB-SGD with b=70'], fontsize=20)
plt.tight_layout()
plt.show()
fig.savefig('Comaparison: MB of 8 and 15.pdf', format='pdf', dpi=1200)
```



In [48]:

```
# evaluate training error of logistic regression and regularized version
weightsList = {
    'log_gradient': log_gradient_descent_w,
    'tune_log_gradient': tune_log_gradient_descent_w,
    'log_mini_batch_gradient_descent_w': log_mini_batch_gradient_descent_w,
    'tune_log_mini_batch_gradient_descent_w': tune_log_mini_batch_gradient_descent_w
}

training_errors = {}

for key in weightsList:
    f_train = predict(weightsList[key], x_train)
    diff = numpy.abs(f_train - y_train) / 2
    error_train = numpy.mean(diff)
    training_errors[key] = error_train
training_errors
```

Out[48]:

```
{'log_gradient': 0.026373626373626374,
 'tune_log_gradient': 0.026373626373626374,
 'log_mini_batch_gradient_descent_w': 0.01978021978021978,
 'tune_log_mini_batch_gradient_descent_w': 0.01978021978021978}
```

In [ ]: