Problem 4, Parts F-H: Stochastic Gradient Descent with a Larger Dataset

Use this notebook to write your code for problem 4 parts F-H by filling in the sections marked `# TODO` and running all cells.

In [3]:
```
# Setup.
import csv
import numpy as np
import matplotlib.pyplot as plt
import math
import random

%matplotlib inline
```

Problem 4F: Perform SGD with the new dataset

For the functions below, you may re-use your code from parts 4C-E. Note that you can now modify your SGD function to return the final weight vector instead of the weights after every epoch.

In [4]:
```
def loss(X, Y, w):
    '''
    Calculate the squared loss function.

    Inputs:
        X: A (N, D) shaped numpy array containing the data points.
        Y: A (N, ) shaped numpy array containing the (float) labels of the data po
ints.
        w: A (D, ) shaped numpy array containing the weight vector.

    Outputs:
        The loss evaluated with respect to X, Y, and w.
    '''
    predict = []
    for x in X:
        predict.append(np.inner(w, x))
    predict = np.asarray(predict)

    loss = 0
    for i in range(len(predict)):
        loss += (predict[i] - Y[i]) ** 2

    return loss

def gradient(x, y, w):
    '''
    Calculate the gradient of the loss function with  respect to
    a single point (x, y), and using weight vector w.

    Inputs:
        x: A (D, ) shaped numpy array containing a single data point.
        y: The float label for the data point.
        w: A (D, ) shaped numpy array containing the weight vector.

    Output:
        The gradient of the loss with respect to x, y, and w.
    '''

    #=============================================
    # TODO: Implement the gradient of the loss function.
    #=============================================
    grad = -2 * (y - np.inner(w, x)) * x
    return grad
```

In [ ]:
```
def SGD(X, Y, w_start, eta, N_epochs):
    '''
    Perform SGD using dataset (X, Y), initial weight vector w_start,
    learning rate eta, and N_epochs epochs.


    Outputs:
        w: A (D, ) shaped array containing the final weight vector.
        losses: A (N_epochs, ) shaped array containing the losses from all iterati
ons.
    '''

    #=============================================
    # TODO: Implement the SGD algorithm.
    #=============================================

    totalLoss = []
    weights = w_start

    #start loss func (for some reason it won't work when i call the function?)
    #Python TypeError: 'list' object is not callable.
    #and
    #TypeError: 'numpy.float64' object is not callable
    predict = []
    for x in X:
        predict.append(np.inner(weights, x))
    predict = np.asarray(predict)

    loss = 0
    assert(len(predict) == len(Y))
    for i in range(len(predict)):
        loss += (predict[i] - Y[i]) ** 2
    #end loss func

    totalLoss.append(loss)

    for n in range(N_epochs):
        assert(len(X) == 1000)

        for i in range(len(X)):
            g = gradient(X[i],Y[i],weights)
            assert(len(g) == len(weights))
            weights -= eta * g

        #start loss func
        predict = []
        for x in X:
            predict.append(np.inner(weights, x))
        predict = np.asarray(predict)

        loss = 0
        assert(len(predict) == len(Y))
        for i in range(len(predict)):
            loss += (predict[i] - Y[i]) ** 2

        #currLoss = loss(X, Y, weights)
        totalLoss.append(loss)

    return np.asarray(weights), np.asarray(totalLoss)
```

Next, we need to load the dataset. In doing so, the following function may be helpful:

In [5]:
```
def load_data(filename):
    """
    Function loads data stored in the file filename and returns it as a numpy ndar
ray.

    Inputs:
        filename: GeneratorExitiven as a string.

    Outputs:
        Data contained in the file, returned as a numpy ndarray
    """
    return np.loadtxt(filename, skiprows=1, delimiter=',')

def getData(data):
    '''
    This function takes the raw data and returns two arrays for inputs(x)
    and outputs(y)
    '''
    x = []
    y = []
    arr = np.asarray([1.0])
    for i in data:
        x.append(np.concatenate((arr, i[:-1]), axis = 0))
        y.append(i[(len(i) - 1)])
    return np.asarray(x), np.asarray(y)
```

Now, load the dataset in `sgd_data.csv` and run SGD using the given parameters; print out the final weights.

In [6]:
```
#============================================
# TODO:
# (1) load the dataset
# (2) run SGD using the given parameters
# (3) print out the final weights.
#============================================
weights = []
loss = []
numEpochs = 800
epochs = [800, 900, 1000, 1100, 1200, 1300]
steps = [math.exp(-10), math.exp(-11), math.exp(-12), math.exp(-13), math.exp(-1
4), math.exp(-15)]
stp = math.exp(-15)
allData = load_data("data/sgd_data.csv")
x, y = getData(allData)

initial = [0.001, 0.001, 0.001, 0.001, 0.001]
# X, Y, w_start, eta, N_epochs
fw, _ = SGD(x, y, initial, stp, numEpochs)
print(fw)

for e in epochs:
    for step in steps:
        initial = [0.001, 0.001, 0.001, 0.001, 0.001]
        fw, totalLoss = SGD(x, y, initial, step, numEpochs)
        weights.append(fw)
        loss.append(totalLoss)
```

```
[ -0.22720591  -5.94229011   3.94369494 -11.72402388   8.78549375]
```

Problem 4G: Convergence of SGD

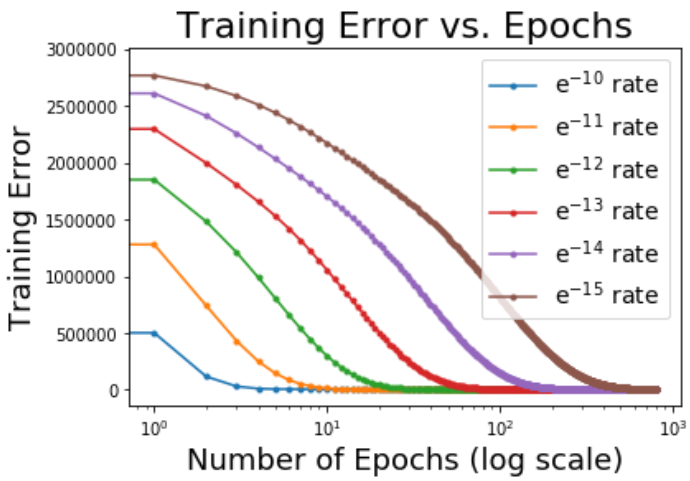
This problem examines the convergence of SGD for different learning rates. Please implement your code in the cell below:

In [10]:
```
#=============================================
# TODO: create a plot showing the convergence
# of SGD for the different learning rates.
#=============================================
print(loss[0])


fig = plt.figure()
x = epochs
plt.title('Training Error vs. Epochs', fontsize = 22)
plt.plot(loss[0], marker = '.')
plt.plot(loss[1], marker = '.')
plt.plot(loss[2], marker = '.')
plt.plot(loss[3], marker = '.')
plt.plot(loss[4], marker = '.')
plt.plot(loss[5], marker = '.')
plt.legend(('e$^{-10}$ rate', 'e$^{-11}$ rate', 'e$^{-12}$ rate', 'e$^{-13}$ rate
', 'e$^{-14}$ rate', 'e$^{-15}$ rate'), loc = 'upper right', fontsize = 14)
plt.xlabel('Number of Epochs (log scale)', fontsize = 18)
plt.xscale('log')
plt.ylabel('Training Error', fontsize = 18)
```

```
[2874307.79361655  500820.74432231  116012.3436524    29495.32628487
    9933.17467149    5483.21974217    4463.39592301    4227.13315194
    4171.40613985    4157.83834506    4154.34642156    4153.36282484
    4153.04781185    4152.93004034    4152.87821425    4152.85132614
    4152.83483885    4152.82299375    4152.81332182    4152.80471177
    4152.79664943    4152.78889313    4152.78132811    4152.77389936
    4152.76658012    4152.75935713    4152.75222348    4152.74517535
    4152.73821038    4152.73132692    4152.72452366    4152.71779947
    4152.71115329    4152.70458412    4152.698091      4152.69167298
    4152.68532912    4152.6790585     4152.67286023    4152.6667334
    4152.66067713    4152.65469055    4152.64877281    4152.64292305
    4152.63714043    4152.63142413    4152.62577334    4152.62018724
    4152.61466504    4152.60920596    4152.60380922    4152.59847405
    4152.59319971    4152.58798544    4152.58283051    4152.57773419
    4152.57269577    4152.56771453    4152.56278978    4152.55792082
    4152.55310699    4152.54834759    4152.54364198    4152.53898949
    4152.53438948    4152.5298413     4152.52534434    4152.52089795
    4152.51650154    4152.51215449    4152.50785621    4152.5036061
    4152.49940357    4152.49524806    4152.491139      4152.48707581
    4152.48305795    4152.47908487    4152.47515603    4152.47127089
    4152.46742893    4152.46362962    4152.45987246    4152.45615693
    4152.45248254    4152.44884879    4152.4452552     4152.44170127
    4152.43818653    4152.43471053    4152.43127278    4152.42787283
    4152.42451024    4152.42118454    4152.41789531    4152.41464211
    4152.41142451    4152.40824207    4152.4050944     4152.40198106
    4152.39890165    4152.39585578    4152.39284303    4152.38986302
    4152.38691535    4152.38399965    4152.38111554    4152.37826264
    4152.37544058    4152.372649      4152.36988754    4152.36715584
    4152.36445355    4152.36178033    4152.35913583    4152.35651971
    4152.35393165    4152.3513713     4152.34883835    4152.34633248
    4152.34385336    4152.34140069    4152.33897415    4152.33657343
    4152.33419825    4152.33184829    4152.32952326    4152.32722288
    4152.32494686    4152.3226949     4152.32046674    4152.31826209
    4152.31608069    4152.31392226    4152.31178653    4152.30967325
    4152.30758215    4152.30551298    4152.30346548    4152.30143941
    4152.29943451    4152.29745054    4152.29548727    4152.29354444
    4152.29162184    4152.28971922    4152.28783635    4152.28597301
    4152.28412898    4152.28230403    4152.28049795    4152.27871052
    4152.27694152    4152.27519075    4152.273458      4152.27174307
    4152.27004574    4152.26836582    4152.26670312    4152.26505743
    4152.26342857    4152.26181634    4152.26022055    4152.25864103
    4152.25707758    4152.25553002    4152.25399818    4152.25248187
    4152.25098093    4152.24949517    4152.24802444    4152.24656855
    4152.24512735    4152.24370067    4152.24228834    4152.24089021
    4152.23950611    4152.2381359     4152.2367794     4152.23543649
    4152.23410699    4152.23279076    4152.23148766    4152.23019754
    4152.22892025    4152.22765565    4152.2264036     4152.22516397
    4152.22393661    4152.22272139    4152.22151818    4152.22032684
    4152.21914724    4152.21797926    4152.21682277    4152.21567763
    4152.21454373    4152.21342094    4152.21230914    4152.21120821
    4152.21011804    4152.20903849    4152.20796947    4152.20691085
    4152.20586252    4152.20482437    4152.20379629    4152.20277817
    4152.2017699     4152.20077137    4152.19978248    4152.19880313
    4152.19783321    4152.19687262    4152.19592126    4152.19497903
    4152.19404583    4152.19312157    4152.19220615    4152.19129947
    4152.19040144    4152.18951197    4152.18863097    4152.18775835
    4152.18689402    4152.18603788    4152.18518986    4152.18434986
    4152.1835178     4152.18269361    4152.18187718    4152.18106845
    4152.18026732    4152.17947373    4152.17868758    4152.17790881
    4152.17713733    4152.17637307    4152.17561595    4152.1748659
    4152.17412284    4152.1733867     4152.1726574     4152.17193488
    4152.17121907    4152.17050989    4152.16980728    4152.16911117
    4152.16842149    4152.16773817    4152.16706115    4152.16639036
    4152.16572574    4152.16506722    4152.16441474    4152.16376825
    4152.16312767    4152.16249294    4152.16186402    4152.16124083
```

Out[10]: Text(0, 0.5, 'Training Error')



Problem 4H

Provide your code for computing the least-squares analytical solution below.

In [9]:
```
#=============================================
# TODO: implement the least-squares
# analytical solution.
#=============================================

x, y = getData(allData)
x = np.matrix(x)
y = np.matrix(y)
ret = np.linalg.inv(x.transpose() * x) * x.transpose() * y.transpose()

print(list(ret))
```

```
[matrix([[-0.31644251]]), matrix([[-5.99157048]]), matrix([[4.01509955]]), matri
x([[-11.93325972]]), matrix([[8.99061096]])]
```

In [ ]:
```

```