Project: Stochastic Gradient Descent

Agnibh Dey, Vinayak Sonandkar

1) Introduction

Goals of the project:

Implement and test SGD algorithm for logistic regression in two different scenarios namely, Hypercube and unit Ball. The project starts with defining domain and parameter set for both the scenarios. Then we define the ρ - Lipschitz and M bond over the parameter set. To calculate SGD model, we provide input training samples, generated randomly according to a gaussian distribution. Then, the model is tested on fresh samples to determine the logistic loss and binary classification error. Finally, we plot the graph of the expected excess risk and expected classification error with respect to number of input training samples. Furthermore, we show the standard deviation of these estimates on the plots.

Outline of algorithm for stochastic gradient descent:

Inputs:

Hypercube Scenario:

- Domain Set
- Parameter Set
- ρ - Lipschitz bond
- M bond

Unit Ball Scenario:

- Domain Set
- Parameter Set
- ρ - Lipschitz Bond
- M bond

Training Samples:

Gaussian vectors u, each composed of 4 i.i.d Gaussian components generated from distribution

(a) $u \sim N(\mu_0, \sigma^2 I_{d-1})$, having label y = -1, with probability = 0.5,

$\mu_0 = (1/4, 1/4, 1/4, 1/4)$, $\sigma = \{0.05, 0.3\}$

(b) $u \sim N(\mu_1, \sigma^2 I_{d-1})$, having label y = 1, with probability = 0.5,

$\mu_1 = (-1/4, -1/4, -1/4, -1/4)$, $\sigma = \{0.05, 0.3\}$

Number of samples = 50, 100, 500, 1000

Number of iterations = Number of samples

A loss function: $\ell_{\text{logist}}(w, (x, y)) = \ln(1 + \exp(-y\langle w, \tilde{x}\rangle))$

where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{d-1}$, $\tilde{\mathbf{x}} \triangleq (\mathbf{x}, 1)$, $y \in \{-1, +1\}$, and $\mathbf{w} \in \mathcal{C} \subset \mathbb{R}^d$

Binary classification error: $1(\text{sign}(y\langle w, \tilde{x}\rangle) \neq y)$

Learning Rate: $\alpha = M/(\rho\sqrt{T})$, where T is the number of iterations

Steps –

1. Generate the test samples, N = 400 (200 with label y =1 and 200 with label y = -1) as Gaussian vectors u, each composed of 4 i.i.d Gaussian components generated from distribution

   -- $u \sim N(\mu_0, \sigma^2 I_{d-1})$, having label y = -1, with probability = 0.5,
   $\mu_0 = (1/4, 1/4, 1/4, 1/4)$, $\sigma = \{0.05, 0.3\}$

   -- $u \sim N(\mu_1, \sigma^2 I_{d-1})$, having label y = 1, with probability = 0.5,
   $\mu_1 = (-1/4, -1/4, -1/4, -1/4)$, $\sigma = \{0.05, 0.3\}$

2. Perform the Euclidean Projection onto χ (in case it lies outside), according to the domain set such that $x = \Pi_\chi(u)$.

3. Generate fresh training examples n = 50 , and perform step (2) to get the Euclidean projections

4. Perform SGD algorithm to get 'w' (parameter vector) with input training samples as follows:
   FOR T = 1……. n
   - Initialize $w_1 = 0$ (vector of all zeros)
   - Calculate ∇w
   - $w_{t+1} = w_t - \alpha \nabla w$
   - Modify w_t according to the parameter set C (if it lies outside set C)
   - Return w_t

   END

5. Return $w = \frac{1}{T} \sum w_t$

6. Calculate the average logistic loss and average the binary classification loss with respect to the 'w' in step (5) over the test samples generated in the step (1)

7. Iterate the steps (3) to step (6), 30 times

8. Calculate the mean of all the 30 logistic losses which is basically, $\mathbb{E}[L(\hat{w}; D)]$

9. Find the min L (w; D) in the 30 logistic losses

10. Calculate the expected excess risk by, $\mathbb{E}[L(\hat{w}; D)] - \min L(w; D)$

11. Calculate the mean of binary classification losses which is basically, $\mathbb{E}[err(\hat{w}; D)]$

12. Calculate the standard deviation for estimates in step (8) and step (11)

13. Iterate the steps (3) to step (11), by changing the number of fresh training samples n = 100, 500, 1000 respectively

14. Plot the graph of the estimates versus Number of samples

Project: Stochastic Gradient Descent

Agnibh Dey, Vinayak Sonandkar

2) Experiments:

1) Generation of test data

Generated the test samples, N = 400 (200 with label y =1 and 200 with label y = -1) as,
Gaussian vectors u, each composed of 4 i.i.d Gaussian components generated from distribution
(a) $u \sim N(\mu_0, \sigma^2 I_{d-1})$, having label y = -1, with probability = 0.5,
$\mu_0 = (1/4, 1/4, 1/4, 1/4)$, $\sigma = \{0.05, 0.3\}$
(b) $u \sim N(\mu_1, \sigma^2 I_{d-1})$, having label y = 1, with probability = 0.5,
$\mu_1 = (-1/4, -1/4, -1/4, -1/4)$, $\sigma = \{0.05, 0.3\}$

The function used in the MATLAB to generate test sample is `R = mvnrnd(mu,sigma,n)`
As the gaussian vector is d – 1 dimension another row of all ones is added to make it d dimension extended vector. This vector is given as input to functions Xcheck_ball and Xcheck_cube(X) to perform the Euclidean Projection onto χ (in case it lies outside), according to the domain set such that $x = \Pi_\chi (u)$.

2) Scenario 1:

Euclidian projection for the hypercube

- Domain set $\chi = [-1, 1]^{d-1}$
  For this a function named Xcheck_cube(X) is written which performs the following function:
  If any of the i.i.d component of the test or training sample have absolute value of each coordinate greater than 1 then it is conditioned to limit to +1 or -1 depending upon if the coordinate is positive or negative respectively.
  If the coordinate's absolute value is less than or equal to 1 then it is kept as it is.

- Parameter set $C = [-1, 1]^d$
  For this a function named Wcheck_cube(W) is written which performs the following function:
  If any component of parameter vector 'w', generated by SGD algorithm have absolute value of each coordinate greater than 1 then it is conditioned to limit to +1 or -1 depending upon if the coordinate is positive or negative respectively.
  If the coordinate's absolute value is less than or equal to 1 then it is kept as it is.

3) Scenario 2:

Euclidian projection for the unit ball

- Domain Set $\chi = \{x \in \mathbb{R}^{d-1} : ||x|| <= 1\}$
  For this a function named Xcheck_ball(X) is written which performs the following function:
  If the feature vector of the test or training sample has norm greater than 1 then it is conditioned to limit within the unit sphere by dividing the vector by its norm
  If the feature vector of the test or training sample has norm less than or equal to 1then it is kept as it is.

- Parameter set $C = \{w \in \mathbb{R}^{d} : ||w|| <= 1\}$
  For this a function named Wcheck_ball(W) is written which performs the following function:
  If the parameter vector 'w', generated by SGD algorithm has norm greater than 1 then it is conditioned to limit within the unit sphere by dividing the vector by its norm
  If the parameter vector has norm less than or equal to 1 then it is kept as it is.

4) Generation of training data:

Fresh training samples are generated in the similar way as the test samples, but they are generated 30 times for every different value of 'n'. We start with generating n= 50 (25 with label y =1 and 25 with label y = -1) training samples, such as

Gaussian vectors u, each composed of 4 i.i.d Gaussian components generated from distribution
(a) $u \sim N(\mu_0, \sigma^2 I_{d-1})$, having label y = -1, with probability = 0.5,
$\mu_0 = (1/4, 1/4, 1/4, 1/4)$, $\sigma = \{0.05, 0.3\}$

(b) $u \sim N(\mu_1, \sigma^2 I_{d-1})$, having label y = 1, with probability = 0.5,
$\mu_1 = (-1/4, -1/4, -1/4, -1/4)$, $\sigma = \{0.05, 0.3\}$

Then, the samples are given as input to the Euclidian projection function of either hypercube or unit ball for conditioning. Next we run SGD algorithm for 50 times (number of samples), to produce final parameter vector 'w'. Then, calculate the logistic and binary classification loss over fresh samples. Finally, we estimate the expected excess risk and classification error, along with their variance.

Now the whole process is repeated by changing the number of training samples to 100, 500, 1000 one by one. We observe that as the size of training samples increases the excess risk and the classification error decreases

5) SGD algorithm:

Once we get the feature vector after Euclidian projection, we perform the SGD as follows

- Initialize the parameter vector as all zeros $w_1 = 0$
- Calculate the gradient as the derivative of the loss function as
  $\nabla w = [1/(1 + \exp(-y\langle w, \tilde{x}\rangle))] * \exp(-y\langle w, \tilde{x}\rangle * -y$
- Calculate the new parameter vector (w_{t+1}) using the learning rate (α) and w_t as:
  $w_{t+1} = w_t - \alpha \nabla w$
- Modify w_{t+1} according to the parameter set C (if it lies outside set C)
- Return final parameter vector and store it in matrix of 'w'
- Repeat the loop for as many times as there are samples
- Return the matrix of 'w', containing 'w_t' from each iteration

Finally, Evaluate the mean of the matrix 'w'

6) Calculation of loss and error:

Once we get the evaluated parameter vector, we test it over the fresh samples, which are basically our test samples (N = 400). For each test samples we get certain, we get certain logistic loss

$$\ell_{logist}(w, (x, y)) = \ln(1 + \exp(-y\langle w, \tilde{x}\rangle))$$

and certain binary classification error: $err(w; D) = 1(sign(y\langle w, \tilde{x}\rangle) \neq y)$

Next we calculate the mean of logistic loss and binary classification error of all the 400 test samples and store the result in a matrix. We iterate the process for 30 times for a particular value of n (n = 50, 100, 500, 1000) . So, the resultant error matrix contains 30 columns with each column having losses values from every iteration.

Finally, we evaluate the mean of 30 logistic losses, which is basically, $\mathbb{E}[L(\hat{w}; D)]$. Find the min L (w; D) in the 30 logistic losses and then, calculate the expected excess risk by, $\mathbb{E}[L(\hat{w}; D)] - \min L(w; D)$.

Similarly, Calculate the mean of binary classification losses which is basically, $\mathbb{E}[err(\hat{w}; D)]$. At the end, we calculate standard deviation for estimates.

7) Plotting of graph

We find the losses and error by varying the training samples n = 50, 100, 500, 1000 and plot the graph for: expected excess risk versus the number of training samples, expected classification error versus the number of training samples.

Project: Stochastic Gradient Descent

Agnibh Deb, Vinayak Sonandkar

3) Analysis of ρ- Lipschitz Properties

- **Hypercube**

Domain set χ = [-1, 1]$^{d-1}$, 4- Dimensional hypercube with edge length 2 and centered around the origin

Parameter set C = [-1, 1]^{d} , 5- Dimensional hypercube with edge length 2 and centered around the origin. Therefore, the largest distance for the convex set will be the diametrically opposite point which are (1,1,1,1,1) and (-1, -1, -1, -1, -1).

M- Bond = $2\sqrt{5}$

For logistic loss, gradient is bounded by $\sqrt{(||x||^2_{max} + 1^2)}$. Since, $||x||_{max}\ is\ at\ x = (1,1,1,1), so -$

$\boldsymbol{\rho}$- Lipschitz bond = $\sqrt{1^2 + 1^2+1^2+1^2 + 1^2}$ = $\sqrt{5}$

Also,

α (Learning rate) = M/ $\rho\sqrt{T}$, where T is the number of iterations in SGD algorithm which are equivalent to the number of training samples

- **Unit Centered ball**

Domain Set χ = {x ϵ $\mathbb{R}^{d-1}$: ||x|| <= 1}, 4- Dimensional unit ball centered around the origin

Parameter set C = {w ϵ $\mathbb{R}^{d}$: ||w|| <= 1}, 5- Dimensional unit ball centered around the origin Therefore, the largest distance for the convex set will be the diametrically opposite points are (1,0,0,0,0) and (-1,0,0,0,0)

M- Bond = 2

For logistic loss, gradient is bounded by $\sqrt{(||x||^2_{max} + 1^2)}$.Since, $||x||_{max} = 1\ for\ all\ x, so -$

$\boldsymbol{\rho}$- Lipschitz bond = $\sqrt{1^2 + 0^2+0^2+0^2 + 1}$ = $\sqrt{2}$

Also,

α (Learning rate) = M/ $\rho\sqrt{T}$, where T is the number of iterations in SGD algorithm

which are equivalent to the number of training samples

Project: Stochastic Gradient Descent

Agnibh Dey, Vinayak Sonandkar

4) Results:

					Logistic Loss				Classification Error		Figures
Scenario	S	n	N	# trials	Mean	Std. Dev	Min	Excess Risk	Mean	Std. Dev	
1	0.05	50	400	30	0.51483133	0.01016734	0.49952099	0.015310333	0.02866667	0.01020761	
1	0.05	100	400	30	0.47064128	0.00510565	0.4589186	0.011722674	0.02208333	0.00612666	Figures
1	0.05	500	400	30	0.39255787	0.001465	0.3902774	0.00228047	0.01891667	0.00233815	1 & 2
1	0.05	1000	400	30	0.37526939	0.0009111	0.37369556	0.001573839	0.01891667	0.00126002	
1	0.3	50	400	30	0.55369736	0.01918834	0.51744922	0.036248134	0.19158333	0.0150213	
1	0.3	100	400	30	0.53219776	0.01380571	0.4978396	0.034358166	0.1905	0.01402584	Figures
1	0.3	500	400	30	0.46416396	0.00525581	0.45361946	0.010544499	0.18108333	0.01018542	3 & 4
1	0.3	1000	400	30	0.44847831	0.00297421	0.44366188	0.004816436	0.178	0.00573856	
2	0.05	50	400	30	0.55272692	0.00561224	0.54306588	0.009661045	0.0105	0.00749713	
2	0.05	100	400	30	0.53124748	0.00276859	0.52629593	0.00495155	0.008	0.00606715	Figures
2	0.05	500	400	30	0.50540467	0.0010342	0.50349849	0.001906183	0.00683333	0.00112444	5 & 6
2	0.05	1000	400	30	0.49936614	0.00043995	0.49871756	0.000648583	0.00716667	0.00086436	
2	0.3	50	400	30	0.61029499	0.00988408	0.59215195	0.018143042	0.21466667	0.01272612	
2	0.3	100	400	30	0.59628658	0.00872184	0.58136139	0.014925189	0.21341667	0.01467292	Figures
2	0.3	500	400	30	0.57030454	0.00211788	0.56642671	0.003877838	0.20408333	0.00864739	7 & 8
2	0.3	1000	400	30	0.56526807	0.00138124	0.56263469	0.002633389	0.20308333	0.00864739	

Excess risk for Hypercube and $\sigma = 0.05$

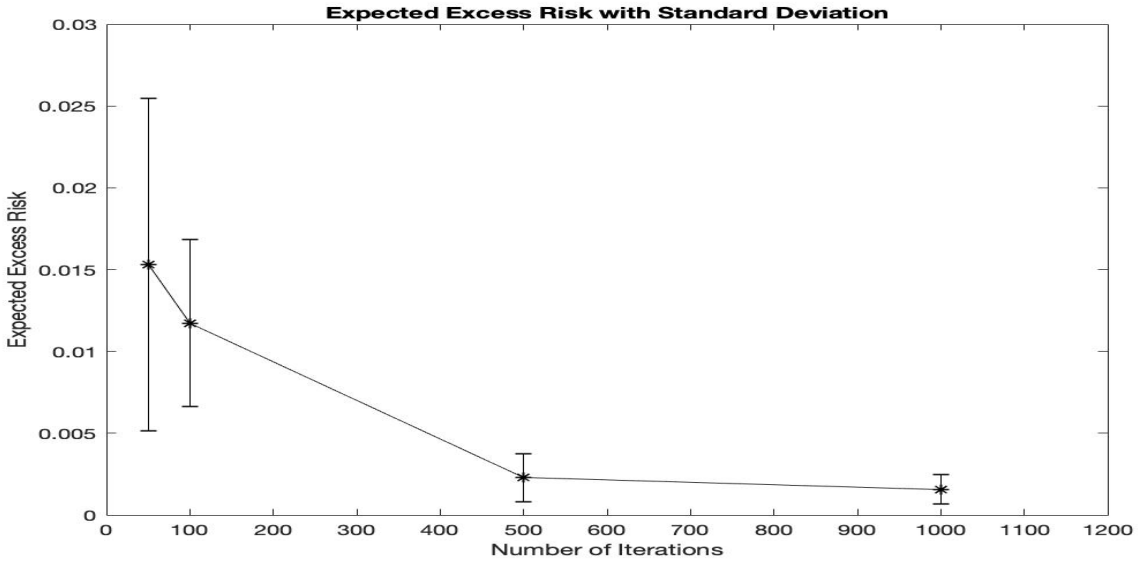

Figure 1

Classification Error for Hypercube and $\sigma = 0.05$

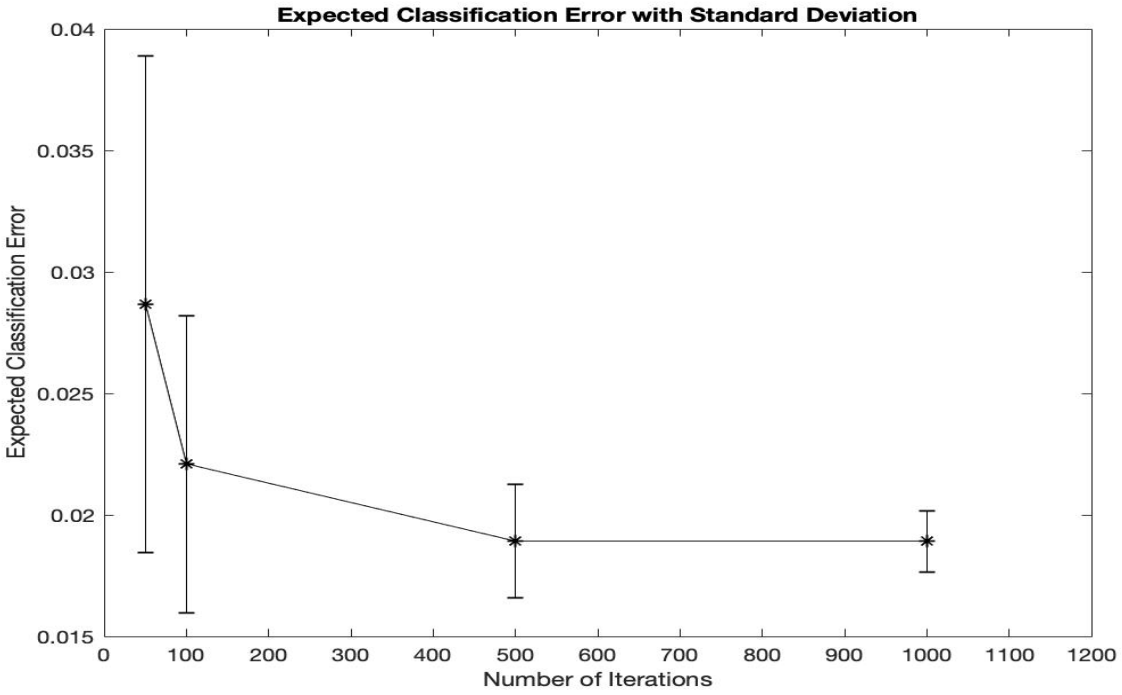

Figure 2

Excess risk for Hypercube and $\sigma = 0.3$

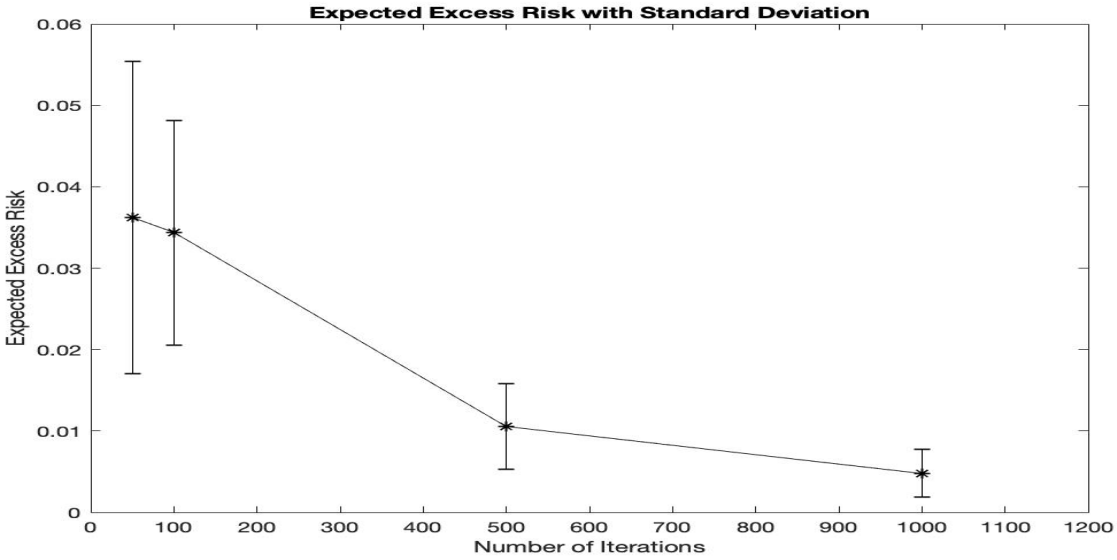

Figure 3

Classification Error for Hypercube and $\sigma = 0.3$

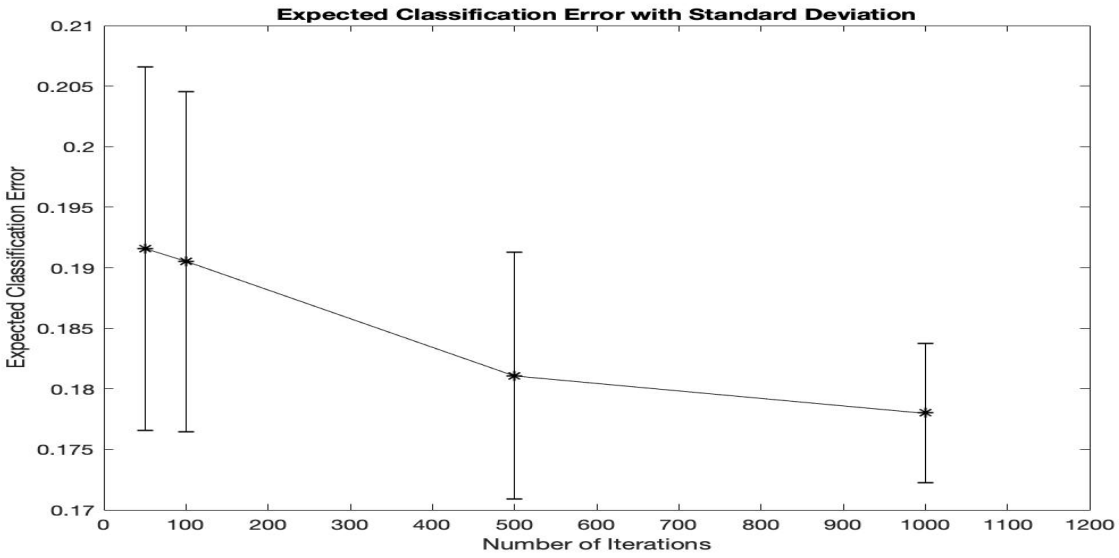

Figure 4

Excess risk for Hypersphere and $\sigma = 0.05$

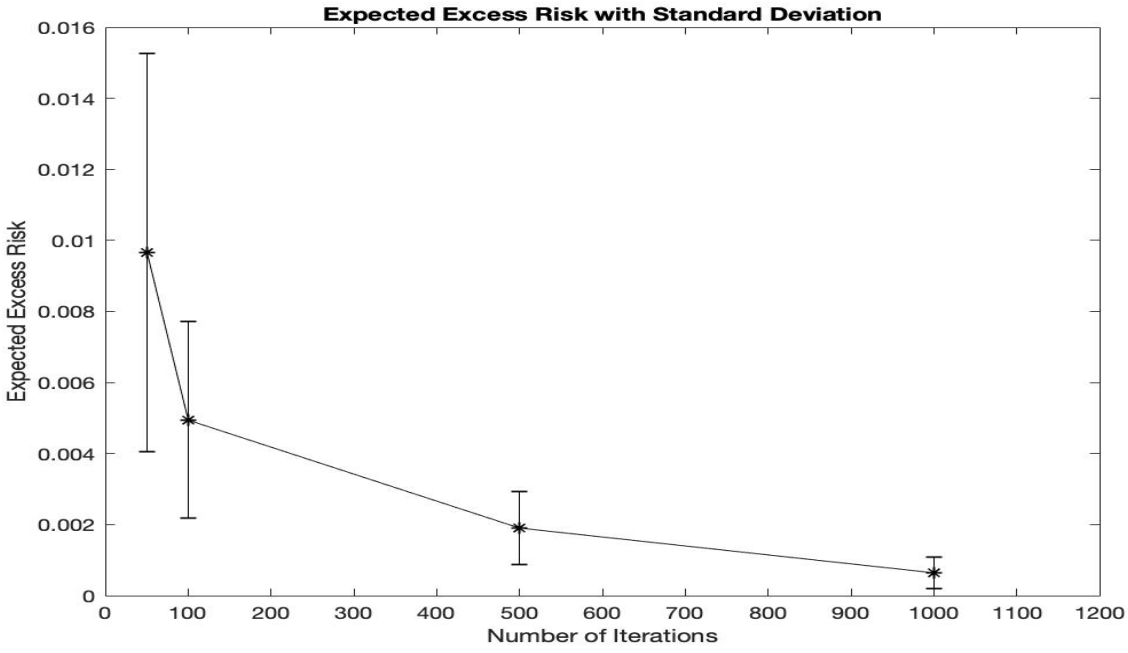

Figure 5

Classification Error for Hypersphere and $\sigma = 0.05$

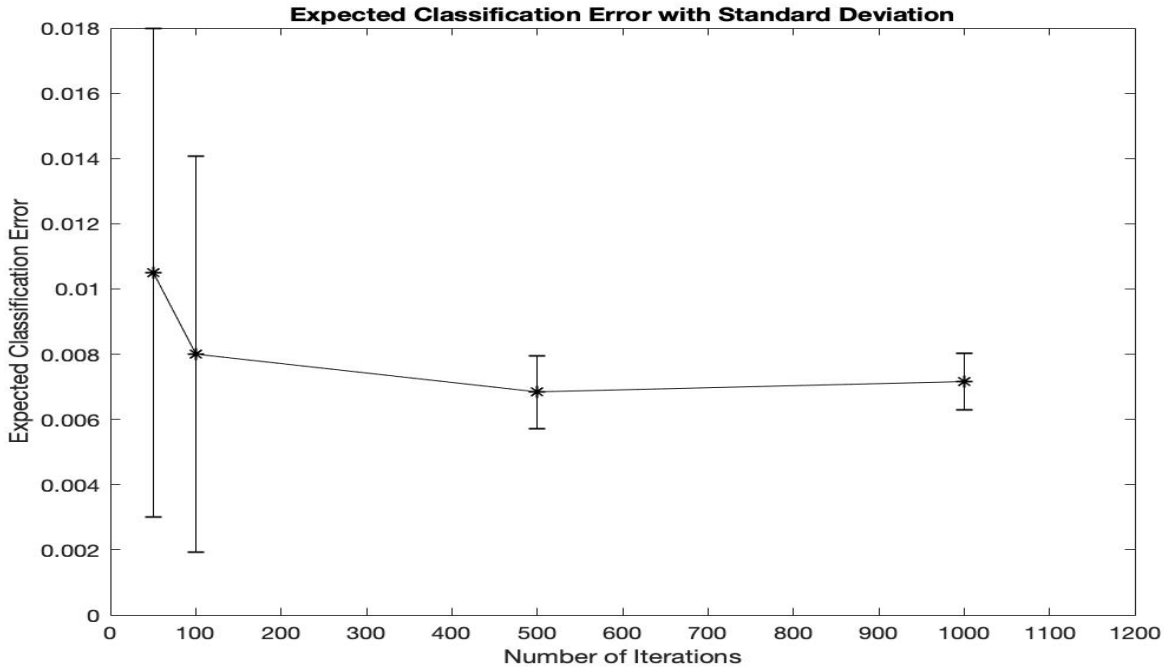

Figure 6

Excess risk for Hypersphere and $\sigma = 0.3$

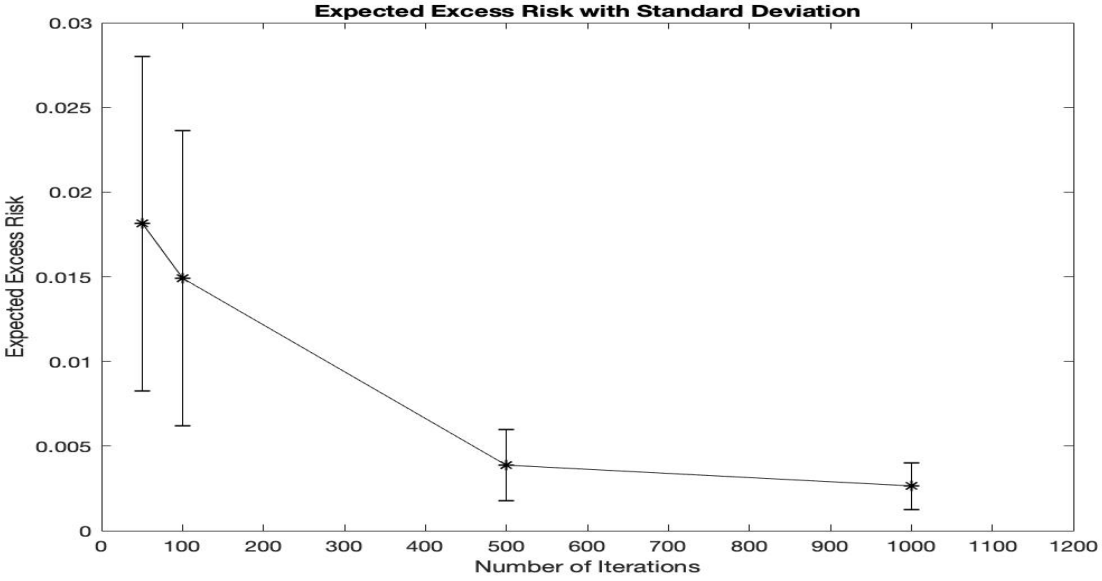

Figure 7

Classification Error for Hypersphere and $\sigma = 0.3$

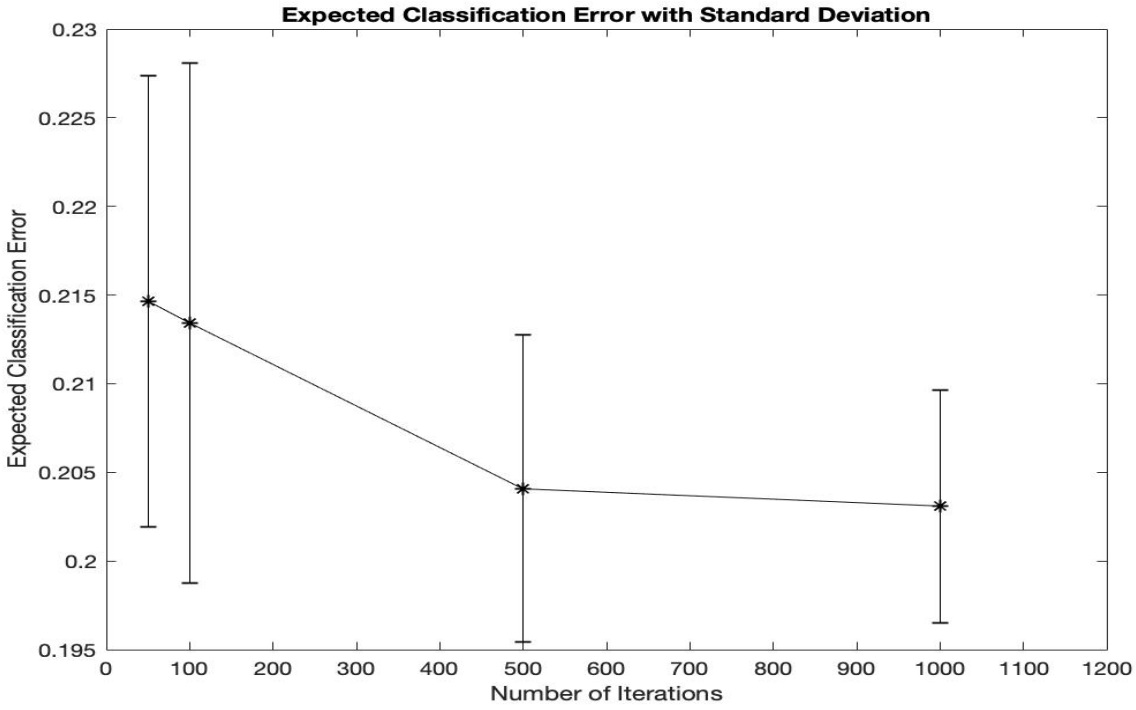

Figure 8

Project: Stochastic Gradient Descent

Agnibh Dey, Vinayak Sonandkar

5) Conclusion

(a) Comparison between theoretical values and experimental results-

According to theory, excess risk of SGD learner is bounded according to the equation-

$$\mathbb{E}_{S\sim D^T}[L(\widehat{w_s}; D)] - \min_{w\in C} L(w; D) \leq \frac{M\rho}{\sqrt{T}}$$

Let's verify the results considering two cases

-- In the case for **Hypercube** -

M = $2\sqrt{5}$

$\rho = \sqrt{5}$

- For T=50 and $\sigma = 0.05 -$

  $\Rightarrow \mathbb{E}_{S\sim D^T}[L(\widehat{w_s}; D)] - \min_{w\in C} L(w; D) \leq 1.414$ (according to the formula above)

  According to the experiments, excess risk $-\mathbb{E}_{S\sim D^T}[L(\widehat{w_s}; D)] - \min_{w\in C} L(w; D) = 0.01533$

  which is much lesser than the theoretical values

-- Similarly, in case of **Hypersphere** -

M = 2

$\rho = \sqrt{2}$

- For T=1000 and $\sigma = 0.3 -$

  $\Rightarrow \mathbb{E}_{S\sim D^T}[L(\widehat{w_s}; D)] - \min_{w\in C} L(w; D) \leq 0.089$ (according to the formula above)

  According to the experiments, excess risk $-\mathbb{E}_{S\sim D^T}[L(\widehat{w_s}; D)] - \min_{w\in C} L(w; D) = 0.0026$

  which is much lesser than the theoretical values

(b) Comparison between results of Hypercube and Hypersphere-

The accuracy in the case of Hypersphere is better as every coordinate of input data is unique and not repeated due to the constraint added. Hence the training samples is better represented.

However, in the case of Hypercube, the coordinates of the points lying outside are same if they are present in the same hyperoctant of the multidimensional plane. Thus, the training samples may be repeated.

(c) Comparison between the standard deviations and accuracy obtained-

Accuracy is better when the standard deviation is less as the training samples are contained within a small space and less spread out, hence they are easier to be classified and less likely to mix for different labels. When the standard deviation is more, data is spread out and more difficult to classify.

Project: Stochastic Gradient Descent

Agnibh Dey, Vinayak Sonandkar

Appendix:

A) Symbol Listing

- Symbols used in analysis

  N = Number of test samples
  n = Number of training Samples
  w = parameter vector
  ∇w = gradient
  w_t = parameter vector in t^{th} iteration
  α = learning rate
  M = Maximum distance in a convex set
  ρ = Lipschitz bond
  T = Number of iterations in SGD algorithm
  σ = Standard deviation
  μ = mean of gaussian distribution

B) Library Routines

1) Y = sign(x) returns an array Y the same size as x, where each element of Y is:

- 1 if the corresponding element of x is greater than 0.
- 0 if the corresponding element of x equals 0.
- -1 if the corresponding element of x is less than 0.

2) n = norm(X) returns the 2-norm or maximum singular value of matrix X

3) Y = abs(X) returns the absolute value of each element in array X.

4) R = mvnrnd(mu,sigma,n) returns a matrix R of n random vectors chosen from the same multivariate normal distribution, with mean vector mu and covariance matrix sigma.

5) p = randperm(n) returns a row vector containing a random permutation of the integers from 1 to n without repeating elements.

6) I = eye(n) returns an n-by-n identity matrix with ones on the main diagonal and zeros elsewhere.

7) sz = size(A) returns a row vector whose elements are the lengths of the corresponding dimensions of A

8) B = sqrt(X) returns the square root of each element of the array X

9) X = zeros(sz1,...,szN) returns an sz1-by-...-by-szN array of zeros where sz1,...,szN indicate the size of each dimension. For example, zeros(2,3) returns a 2-by-3 matrix.

10) X = ones(sz1,...,szN) returns an sz1-by-...-by-szN array of ones where sz1,...,szN indicates the size of each dimension. For example, ones(2,3) returns a 2-by-3 array of ones.

11) M = mean(A,dim) returns the mean along dimension dim. For example, if A is a matrix, then mean(A,2) is a column vector containing the mean of each row.

12) S = std(A) returns the standard deviation of the elements of A. If A is a vector of observations, then the standard deviation is a scalar.

13) errorbar(y,err) creates a line plot of the data in y and draws a vertical error bar at each data point. The values in err determine the lengths of each error bar above and below the data points, so the total error bar lengths are double the err values.

C) Code

Function to keep X within unit cube of 4D –

```
function X_crc = Xcheck_cube(X)  % function to keep X within unit cube of 4D

X_ac = X(1:4,:);             % First 4 rows has coordinates
col = size(X_ac,2);          % No. of samples
X_crc = zeros(4,col);        % Restricted X
for a = 1:col                % Loop through the samples
    if (abs(X_ac(1,a))<=1)
        X_crc(1,a) = X_ac(1,a);       % Keep same X if absolute value <=1
    else
        X_crc(1,a) = sign(X_ac(1,a)); % Limit in unit cube if absolute value
                                        >1
    end

    if (abs(X_ac(2,a))<=1)
        X_crc(2,a) = X_ac(2,a);       % Keep same X if absolute value <=1
    else
        X_crc(2,a) = sign(X_ac(2,a)); % Limit in unit cube if absolute value
                                        >1
    end

    if (abs(X_ac(3,a))<=1)
        X_crc(3,a) = X_ac(3,a);       % Keep same X if absolute value <=1
    else
        X_crc(3,a) = sign(X_ac(3,a)); % Limit in unit cube if absolute value
                                        >1
    end

    if (abs(X_ac(4,a))<=1)
        X_crc(4,a) = X_ac(4,a);       % Keep same X if absolute value <=1
    else
        X_crc(4,a) = sign(X_ac(4,a)); % Limit in unit cube if absolute value
```

```
                                        >1
    end

end
X_crc = [ X_crc;ones(1,col)];    % Add rows of 1 at the end
end
```

Function to keep W within unit cube of 5D –

```
function W_crc = Wcheck_cube(W) %function to keep W within unit sphere of 5D

if (abs(W(1,1))<=1)
    W_crc(1,1) = W(1,1);        % Keep same W if absolute value <=1
else
    W_crc(1,1) = sign(W(1,1)); % Limit in unit cube if absolute value >1
end

if (abs(W(2,1))<=1)
    W_crc(2,1) = W(2,1);        % Keep same W if absolute value <=1
else
    W_crc(2,1) = sign(W(2,1)); % Limit in unit cube if absolute value >1
end

if (abs(W(3,1))<=1)
    W_crc(3,1) = W(3,1);        % Keep same W if absolute value <=1
else
    W_crc(3,1) = sign(W(3,1)); % Limit in unit cube if absolute value >1
End
```

Function to keep X within unit sphere of 4D –

```
function X_crc = Xcheck_ball(X) % function to keep X within unit sphere of 4D

X_ac = X(1:4,:);              % First 4 rows has coordinates
col = size(X_ac,2);           % No. of samples
X_crc = zeros(4,col);         % Restricted X
for a = 1:col                 % Loop through the samples
    if (norm(X_ac(:,a))<=1)
        X_crc(:,a) = X_ac(:,a);      % Keep same X if norm <=1
    else

        X_crc(:,a) = X_ac(:,a)/norm(X_ac(:,a)); % Limit in unit sphere if
                                                  norm >1
    end
end
X_crc = [ X_crc;ones(1,col)];    % Add rows of 1 at the end
End
```

Function to keep W within unit sphere of 5D –

```
function W_crc = Wcheck_ball(W) % function to keep W within unit sphere of 5D

if (norm(W)<=1)
    W_crc = W;          % Keep same W if norm <=1
else
    W_crc = W/norm(W);  % Limit in unit sphere if norm >1
end

end
```

Code to generate test Data-

```
clear;clc;close all;

sigma = 0.3 * eye(4);          % Standard Deviation
N = 400;                        % No.of test samples
mu_neg = [-1/4 -1/4 -1/4 -1/4]; % Mean of negative label test samples
mu_pos = [1/4 1/4 1/4 1/4];     % Mean of positive label test sample

%Test Data
R1_tst = mvnrnd(mu_neg,sigma,N/2);
X1_tst = R1_tst';               % Genarated negative label test samples

R2_tst = mvnrnd(mu_pos,sigma,N/2);
X2_tst = R2_tst';               % Genarated positive label test samples

X_tst1 = [X1_tst X2_tst;ones(1,N)];      % Extended Test Samples
Y_tst1 = [-1*ones(1,N/2) ones(1,N/2)];   % Labels

Dt1 = [X_tst1;Y_tst1];            % Create Data Matrix of samples and labels

cols = size(Dt1,2);                    % Shuffle the matrix
ran = randperm(cols);
Dt = Dt1(:,ran);                       % Matrix after reshuffling

Xb_tst = Dt(1:5,:);                    % First 5 rows is X
X_tst = Xcheck_ball(Xb_tst);           % Make sure X lies within unit sphere
Y_tst = Dt(6,:);

save('test.mat','X_tst','Y_tst')            % Save the data in mat file
```

Main SGD Code -

```
clear;clc;close all;

%Define constants
M= 2;                                   % M-bound
rho = sqrt(2);                          % Lipchitzness
sigma = 0.3 * eye(4);                   % Standard Deviation
n = [50 100 500 1000];                  % No.of training samples
N = 400;                                % No.of test samples
mu_neg = [-1/4 -1/4 -1/4 -1/4];         % Mean of negative label test samples
mu_pos = [1/4 1/4 1/4 1/4];             % Mean of positive label test sample
Itr = 30;                               % Number of iterations

%Test Data
load('test.mat')                        % Load Test Samples

L_tot = zeros(1,30);                    % Initialize risk error
Err_tot = zeros(1,30);                  % Initialize classification error

L_avg = zeros(1,4);                     % Initialize Average Risk
L_min = zeros(1,4);                     % Initialize Minimum Risk
L_exp = zeros(1,4);                     % Initialize Excess Risk
L_std = zeros(1,4);                     % Initialize Std Deviation of Risk

Err_avg = zeros(1,4);                   % Initialize Average Classification Error
Err_std = zeros(1,4); % Initialize Standard Deviation of Classification Error

for m = 1:size(n,2)          % Run for training sample size of 50,100,500,1000
for j=1:Itr                 % Run Itr number of iterations for each sample
size

%Train Data
R1_trn = mvnrnd(mu_neg,sigma,n(m)/2);
X1_trn = R1_trn';                       % Genarated negative label train samples

R2_trn = mvnrnd(mu_pos,sigma,n(m)/2);
X2_trn = R2_trn';                       % Genarated positive label train samples

X_trn1 = [X1_trn X2_trn ;ones(1,n(m))];    % Extended Train samples

Y_trn1 = [-1*ones(1,n(m)/2) ones(1,n(m)/2)]; % Labels

Dt1 = [X_trn1;Y_trn1];              % Create Data Matrix of samples and labels

cols = size(Dt1,2);                     % Shuffle the matrix
ran = randperm(cols);
Dt = Dt1(:,ran);                        % Matrix after reshuffling

Xb_trn = Dt(1:5,:);                     % First 5 rows is X
X_trn = Xcheck_ball(Xb_trn);    % Make sure X lies within unit sphere/cube
Y_trn = Dt(6,:);                        % Last row is label
```

```
%SGD start
w = zeros(5,1);                          % Initialize parameter vector to 0
alpha = M/(rho*sqrt(n(m)));              % Step-size
W = zeros(5,n(m)+1);                     % Store parameter vector in W

for i=1:n(m)                             % Run loop for training samples
    c1 = exp(-Y_trn(i)*dot(w,X_trn(:,i)));
    c2 = 1/(1+c1)*c1*-Y_trn(i);
    w_del = c2*X_trn(:,i);               % Gradient computed
    wb = w - alpha*w_del;                % Generate new w
    w = Wcheck_ball(wb);           % Make sure W lies within unit sphere/cube
    W(:,i+1) = w;                        % Store w
end

W_fin = mean(W,2);                       % Take mean

% Compute error from testing samples
l = zeros(1,N);                          % Risk error
err = zeros(1,N);                        % Classification error

%Test
for k=1:N                                % Run for all test samples
   l(k)=log(1+exp(-Y_tst(k)*dot(W_fin,X_tst(:,k)))); % Store logostic loss
   c = sign(dot(W_fin,X_tst(:,k)))-Y_tst(k);  % Store classification error
   if(c ==0)
   err(k) = 0;
   else
   err(k) = 1;
   end
end

L = mean(l);             % Expected logistic loss
Err = mean(err);         % Expected classification

L_tot(j) = L;            % Store for each iteration
Err_tot(j) = Err;

end

L_avg(m) = mean(L_tot);              % Average of risk for each sample size
L_min(m) = min(L_tot);               % Minimum of risk for each sample size
L_exp(m) = L_avg(m) - L_min(m);      % Excess risk for each sample size
L_std(m) = std(L_tot);          % Std Deviation of risk for each sample size

Err_avg(m) = mean(Err_tot);  % Average of Classification Error for each
                               sample size
Err_std(m) = std(Err_tot);   % Std Deviation of Classification Error for each
                               sample size

end

figure(1)
errorbar(n, L_exp, L_std,'k')
```

```
hold on
plot(n,L_exp,'k*')
xlim([0 1200]);
xticks(0:100:1200);
xlabel('Number of Iterations')
ylabel('Expected Excess Risk')
title('Expected Excess Risk with Standard Deviation')

figure(2)
errorbar(n, Err_avg, Err_std,'k')
hold on
plot(n,Err_avg,'k*')
xlim([0 1200]);
xticks(0:100:1200);
xlabel('Number of Iterations')
ylabel('Expected Classification Error')
title('Expected Classification Error with Standard Deviation')
```