CSE 5523 Project: *Stochastic Gradient Descent*

Yun Chen Yen
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
`yen.156@osu.edu`

Yunqi Zhang
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
`zhang.8678@osu.edu`

1 Introduction

1.1 Overall Goals

In this mini-project, we implement the stochastic gradient descent (SGD for short) algorithm to better understand it. In a nutshell, we have the following goals.

- To design and build our own version of SGD using Python.
- To conduct SGD experiments with a naive example.
- To introduce, explain, and visualize our experiment results.

1.2 Procedure Description

Algorithm 1 SGD Learner

```
1: for $\sigma = 0.1, 0.35$ do
2:     $(xTest, yTest) = GenerateTestSet(\sigma, N)$
3:     for $n = 50, 100, 500, 1000$ do
4:         $TotalLoss = EmptyList$
5:         $TotalError = EmptyList$
6:         for $t = 0, 1, 2, \ldots 29$ do
7:             $Ws = ComputeWs(\sigma, n)$
8:             $(Loss, Error) = GetLossAndError(Ws, xTest, yTest)$
9:             $TotalLoss.append(Loss)$
10:            $TotalError.append(Error)$
11:        end for
12:        $LogisticLossMean = Average(TotalLoss)$
13:        $LogisticLossStandardDeviation = StandardDeviation(TotalLoss)$
14:        $LogisticLossMin = Min(TotalLoss)$
15:        $ExcessRisk = LogisticLossMean - LogisticlossMin$
16:        $ClassificationErrorMean = Average(TotalError)$
17:        $ClassificationErrorStandardDeviation = StandardDeviation(TotalError)$
18:    end for
19: end for
```

For each σ (i.e., 0.1 and 0.35), we first generate the test set of size N (i.e., 400). Then, for each training set size (i.e., 50, 100, 500, and 1000), we conduct 30 trials. The pseudo-code above describes our SGD learner algorithm from a high level.

2 Experiments

In the previous section, we introduced our SGD learner without going through the details. In this section, we address the details (e.g., functions and parameters) in our SGD learner. Specifially, we focus on two things:

- The generation of our training datasets and test dataset.
- The inner $for\ loop$ (line 6 to 11) in Algorithm 1 SGD Learner.

We focus on these two items because we believe that the remaining pseudo-code in Algorithm 1 is relatively straightforward.

2.1 Data Generation

To generate our training set, we first randomly pick a number from $[-1, 1]$ as our **y** value. We then randomly pick $d - 1$ floats from a normal distribution with mean of **y** and standard deviation of σ to fill a $(d - 1)$-dimensional Gaussian vector **u**. If **u** falls outside of the $(d - 1)$-dimensional unit ball, we project **u** onto the unit ball (which we will address later in this section) and use the projected vector as our **x**. If **u** falls into the unit ball, we simply use **u** as our **x**. We repeat the above procedure until we have n tuples in our training set (n is our training set size or "number of trials").

To generate our test set, we basically reuse the function from our training set generation. The only difference is that we repeat the data generation procedure for **y** and **x** until we have N tuples in our test set (N is our test set size).

The projection in our data generation process is done by computing the norm of the input vector (**u** in our cases). If the result norm is greater than 1, we consider **u** falls outside of the unit ball. To get the projected vector, we divide **u** by the norm.

2.2 Details of Our SGD Learner

In this section, we focus on two functions from the inner $for\ loop$ (Algorithm 1 SGD Learner line 6 to 11): $ComputeWs$ and $GetLossAndError$.

The following pseudo-code (Algorithm 2 Compute Ws) explains our $ComputeWs$ function in line 7 of Algorithm 1 SGD Learner.

Algorithm 2 Compute Ws

```
1:  $(X, Y) = GenerateData(\sigma, n)$            ▷ Training data generation as mentioned in §2.1
2:  $(Rows, Columns) = n + 1, d$
3:  $Ws = d - dimensional\ vector\ with\ all\ zeros$            ▷ Initialize w1 with 0
4:  for $t = 1, 2, 3, \ldots n - 1$ do
5:      $Xt = Projection(X[t])$            ▷ Details about projection can be found in §2.1
6:      $Draw\ a\ fresh\ sample\ z_t$            ▷ According to class slides: SGD for Convex Learning
7:      $Compute\ Gradient\ G_t$            ▷ According to class slides: SGD for Convex Learning
8:      $Update\ W_{t+1}\ and\ apply\ projection\ if\ needed$
9:  end for
10: return $\hat{w}_s = \frac{1}{n}\sum_{t=1}^{n} w_t$            ▷ According to class slides: SGD for Convex Learning
```

The $GetLossAndError$ function in line 8 of Algorithm 1 SGD Learner simply applies the logistic loss function and the binary classification error function we defined in class (as well as in the project) to compute the logistic loss and the classification error.

3 Analysis of Lipschitzness and Boundedness Properties

This section attempts to provide an answer to the following question.

For the project scenario (Project Description and Template page 2), show that there is a constant ρ such that for all $z \in \mathcal{X} \times \{1, +1\}$, $\ell_{logist}(\cdot, z)$ is convex and ρ-Lipschitz over C and that C is M-bounded for some M > 0. Specify ρ and M.

$$\ell_{logist}(w, (x, y)) = ln(1 + exp(-y < w, \hat{x} >))$$

$$g_1(w) \triangleq y < x, \hat{x} >$$

$$g_2(a) = ln(1 + exp(-a))$$

We have $\ell_{logist}(\cdot, (x, y)) = g_2 \circ g_1$. Note that g_1 is linear and $||\hat{x}||$-Lipschitz. g_2 is convex and 1-Lipschitz. Hence, $\ell_{logist}(\cdot, (x, y))$ is convex and $||\hat{x}||$-Lipschitz.

By definition, $||\hat{x}|| = ||(x, 1)||$. Since $||x|| \leq 1$, we have $\rho = \sqrt{1 + 1} = \sqrt{2}$.

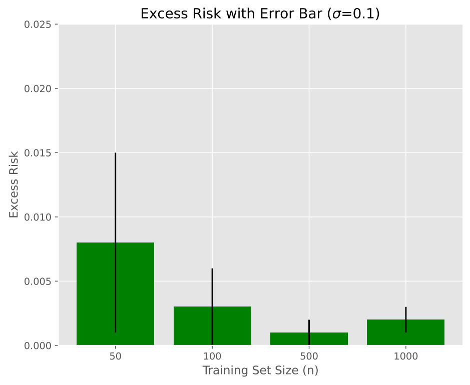
The parameter space $C = \{w \in R^d : ||w|| \leq 1\}$ is M-bounded. Thus, M = 2.

4 Results

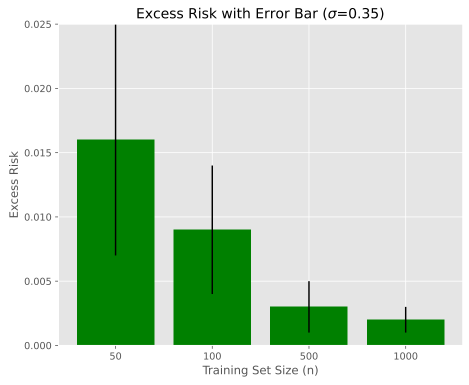
The following table shows our main experiment results.

σ	n	N	# trials	Logistic loss				Classification error	
				Mean	Std Dev	Min	Excess Risk	Mean	Std Dev
0.1	50	400	30	0.525	0.007	0.518	0.008	0.004	0.010
0.1	100	400	30	0.510	0.003	0.507	0.003	0.001	0.001
0.1	500	400	30	0.492	0.001	0.490	0.001	0.000	0.000
0.1	1000	400	30	0.488	0.001	0.486	0.002	0.000	0.000
0.35	50	400	30	0.541	0.009	0.525	0.016	0.111	0.036
0.35	100	400	30	0.523	0.005	0.514	0.009	0.095	0.011
0.35	500	400	30	0.503	0.002	0.501	0.003	0.096	0.005
0.35	1000	400	30	0.499	0.001	0.497	0.002	0.095	0.005

The following plots visualize our experiment results.

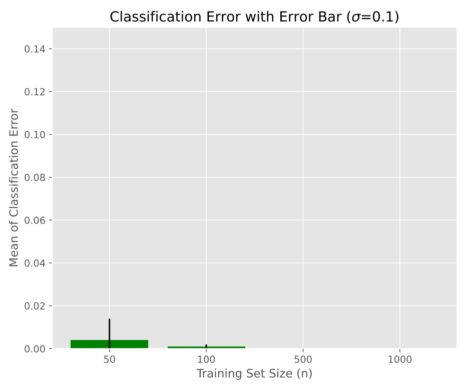


(a) $\sigma = 0.1$

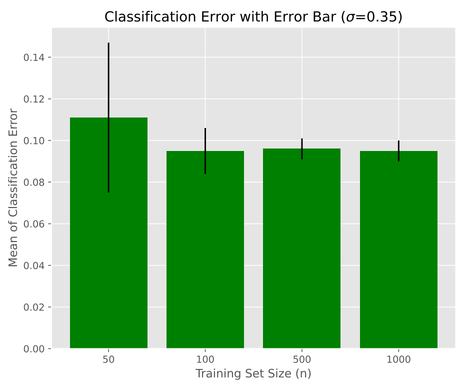


(b) $\sigma = 0.35$

Figure 1: excess risks with error bars for $\sigma = 0.1$ and $\sigma = 0.35$



(a) $\sigma = 0.1$



(b) $\sigma = 0.35$

Figure 2: means of classification errors with error bars for $\sigma = 0.1$ and $\sigma = 0.35$

5 Conclusion

To conclude our experiments, generally, we believe that our experiment results are similar to the theoretical results we derived from class (Stochastic Gradient Descent for Convex Learning page 37 - Remarks).

To be specific, fixing σ, we find out that as we increase the training set size (n), the corresponding excess risk will be lower as the excess risk is upper bounded by $\frac{\rho \times M}{\sqrt{n}}$. We also notice that, for fixed n, change in σ has greater impact on classification errors than excess risks. One potential explanation is that, in our experiments, the sample range is much larger for $\sigma = 0.35$.

Acknowledgments and Disclosure of Funding

This short report is a partial fulfillment of CSE 5523: Machine Learning and Statistical Pattern Recognition (SP 2022) at the Ohio State University. We would like to thank Prof. Raef Bassily for his teaching and the Ohio State University for the resources.

A Appendix: Symbol Listing

Symbol	Description
σ	the standard deviation of Gaussian Distribution
n	the size of training set
N	the size of test set
t	the number of trail that each training set conduct
d	the dimensionality parameter
C	the parameter space
ρ	the Lipschitz constant
M	the bound for the convex set C
ℓ_{logist}	the loss function of logistic regression
$\|\| \cdot \|\|$	the l^2-norm

B Appendix: Library Routines

Syntax	Description
numpy.linalg.norm	The function will return norm of the matrix or vector.
numpy.dot	The function returns dot product of two arrays.
numpy.std	The function computes the standard deviation.
math.sqrt(x)	The function returns the square root of a number.
random.choice	The function returns a randomly selected element from the specified sequence.
sum()	The function returns the sum of all items in an iterable.
min()	The function returns the item with the lowest value.

C Appendix: Code (Python)

```python
import numpy as np
import math
import random
    trials = 30
    N = 400
    d = 5
    M = 2
    Rho = math.sqrt(2)
    def Projection(vector):
      d = np.linalg.norm(vector)
      if d>1 :
        return vector/d
      return vector

    def GenerateData(sigma, T):
      X = []
      Y = []
      for i in range(T):
        y = random.choice([-1, 1])
        x = np.random.normal(y/4, sigma, d-1)
        Y.append(y)
        X.append(x)
      return X, Y

    def ComputeWs(sigma, T):
      X, Y = GenerateData(sigma, T)
      # initialization: w1 = 0
      rows, cols = (T+1, d)
      W = [[0]*cols]*rows
      Ws = np.zeros(d)
      Learning_Rate = M / (Rho * math.sqrt(T))
      for t in range(1, T):
        # <x,1>
        Xt = np.append(Projection(X[t]), [1], axis = 0)

        # y<w,~x>
        z = Y[t] * np.dot(np.array(W[t]), Xt)

        # logistic loss function
        # loss = math.log(1 + math.exp(-z))

      # Compute Gt
      Gt = (-1 * math.exp(-z) / (1 + math.exp(-z))) * Y[t] * Xt

      # Update wt+1
      W[t+1] = W[t] - Learning_Rate * Gt
      W[t+1] = Projection(W[t+1])

    for t in range(1, T+1):
      Ws = np.add(Ws, W[t])
    return Ws/T

  def GetLossAndError(Ws, x, y_test):
    Error = 0
    Loss = 0
    for i in range(N):
      x_test = np.append(Projection(x[i]), [1], axis = 0)
      z = y_test[i] * np.dot(np.array(Ws), x_test)
      loss = math.log(1 + math.exp(-z))
      Loss+=loss

      true_y = y_test[i]
      if np.dot(x_test, Ws) >0:
        pred_y = 1
      else:
        pred_y = -1

      if pred_y!=true_y:
        Error+=1

    Loss = Loss/N
    Error = Error/N

    return Loss, Error

  def SGD_Learner():
    print("{0:^21}|{1:^31}|{2:^20}"
    .format("", "Logistic loss", "Classification error"))
    print("{0:^4}|{1:^4}|{2:^3}|{3:^7}|{4:^5}|
          {5:^7}|{6:^5}|{7:^11}|{8:^10}|{9:^10}"
    .format("","n", "N","#trials","Mean","Std Dev"
          , "Min", "Excess Risk", "Mean", "Std Dev"))
    print("-------------------------------------------
              -------------------------------")

    for sigma in [0.1, 0.35]:
      x_test, y_test = GenerateData(sigma, N)#generate test set
      for n in [50, 100, 500, 1000]:
        Total_Loss = []
        Total_Error = []
        for t in range(30):
          Ws = ComputeWs(sigma, n)
          Loss, Error = GetLossAndError(Ws, x_test, y_test)
          Total_Loss.append(Loss)
          Total_Error.append(Error)
```

```
            L_mean = sum(Total_Loss)/len(Total_Loss)
            L_std = np.std(Total_Loss)
            L_min = min(Total_Loss)
            L_excess_risk = L_mean-L_min

            C_mean = sum(Total_Error)/len(Total_Error)
            C_std = np.std(Total_Error)

            print("{0:^4}|{1:^4}|400|{2:^7}|{3:^5.3f}|{4:^7.3f}|
                  {5:^5.3f}|{6:^11.3f}|{7:^10.3f}|{8:^10.3f}"
            .format(sigma, n, 30, L_mean, L_std, L_min,
                      L_excess_risk, C_mean, C_std))
        print("----------------------------------------
                  ----------------------------------")
    def main():
        SGD_Learner()

    if __name__ == "__main__":
        main()
```