Project: Stochastic Gradient Descent

Yifei Zhang (.5387), Yue Lin (.3326)

November 2020

1 Introduction

The goal of this project is to implement stochastic gradient descent (SGD) algorithm and conduct a set of experiments considering two scenarios to evaluate the performance of SGD. The logistic loss and binary classification error are used in evaluation.

In this project, we implement an SGD using the following steps.

- Initialize the network weights $\mathbf{w}_1$ to random numbers between -1 and 1.
- For each input feature vector $\tilde{\mathbf{x}}_t$ and its label y_t, calculate the gradient

$$\nabla f\left(\mathbf{w}_t\right)=\frac{-y_t \tilde{\mathbf{x}}_t \exp \left(-y_t\left\langle\mathbf{w}_t, \tilde{\mathbf{x}}_t\right\rangle\right)}{1+\exp \left(-y_t\left\langle\mathbf{w}_t, \tilde{\mathbf{x}}_t\right\rangle\right)}$$

- Take a projected GD step

$$\mathbf{w}_{t+1}=\Pi_C\left(\mathbf{w}_t-\alpha \nabla f\left(\mathbf{w}_t\right)\right)$$

- Repeat steps 2 and 3 until all n samples are trained, and then output

$$\hat{\mathbf{w}}=\frac{1}{n} \sum_{t=1}^n \mathbf{w}_t$$

2 Experiments

This project is developed in Python. For each of the two scenarios, we test the experimental results using two $\sigma \in \{0.1, 0.35\}$ to generate the Gaussian distribution for both domain and parameter sets. For each σ value, we vary the size of the training set n as $50, 100, 500, 1000$. For each set of the parameters (σ, n), the following experiment is conducted.

- Calculate step size α based on the ρ-Lipschitz properties for each scenario described in the upcoming section.
- Generate data for model training and testing. For each example $(\mathbf{x}, y)$, y is randomly sampled from {-1,1} with equal probability. When $y=-1$, a 4-dimensional vector $\mathbf{u}=(u_1, u_2, u_3, u_4)$ is generated. Each element of $\mathbf{u}$ is randomly picked from Gaussian distribution with mean $-1/4$ and variance σ^2. When $y=1$, each element of $\mathbf{u}$ is randomly picked from Gaussian distribution with mean 1/4 and variance σ^2(σ is specified above, $\sigma \in \{0.1, 0.35\}$).

- Perform Euclidean projection on domain set χ.
  - Scenario 1: For $\mathbf{u}$ located outside of χ, each element u_i in $\mathbf{u}$ will be scaled into [-1,1] by choosing the closest value to u_i on χ, which is calculated as $sign\left(u_i\right) * min\left(\left|u_i\right|, 1\right)$. For $\mathbf{u}$ within χ, no projection is needed.
  - Scenario 2: For $\mathbf{u}$ located outside of χ, each element u_i in $\mathbf{u}$ will be shrunk so that the norm is 1 ,which is calculated as $\frac{u_i}{\|\mathbf{u}\|}$. For $\mathbf{u}$ within χ, no projection is needed.
- Run the SGD algorithm and evaluate the output $\hat{\mathbf{w}}$ on the test set using two types of metrics: logistic loss and classification error. The logistic loss is computed as

  $$.l_{logist}\left(\mathbf{w},(\mathbf{x}, y)\right)=\ln \Big(1+\exp \left(-y\left\langle\mathbf{w}, \tilde{\mathbf{x}}\right\rangle\right)\Big)$$

  The classification error is computed as

  $$\mathbf{1}\Big(\operatorname{sign}\left(\langle\mathbf{w}, \tilde{\mathbf{x}}\rangle\right) \neq y\Big)$$

- Repeat steps 2 and 3 for 30 trials (as determined in the project description). Calculate the below statistics for the specific (σ, n) setting.
  - Calculate the mean, minimum, and standard deviation of the 30 loss estimates. Calculate the difference between the mean and the minimum of the loss estimates as the expected excess risk.
  - Calculate the mean and standard deviation of the 30 binary classification error estimates.

3 Analysis of ρ-Lipschitz properties

For logistic loss function

$$l_{logist}\left(\mathbf{w},(\mathbf{x}, y)\right)=\ln \Big(1+\exp \left(-y\left\langle\mathbf{w}, \tilde{\mathbf{x}}\right\rangle\right)\Big)$$

where $\mathbf{x} \in \chi \subset \mathbb{R}^{d-1}$, $\tilde{\mathbf{x}} \triangleq(\mathbf{x}, 1)$, $y \in\{-1,+1\}$, $\mathbf{w} \in \mathbb{R}^d$, and $z=(\mathbf{x}, y)$.

Let's define $g_1(\mathbf{w}) \triangleq y\langle\mathbf{w}, \tilde{\mathbf{x}}\rangle$, $g_2(a)=\log \big(1+\exp (-a)\big)$.

Note $l_{logist}\left(\cdot,(\mathbf{x}, y)\right)=g_2 \circ g_1$, g_1 is linear and $\|\tilde{\mathbf{x}}\|$-Lipschitz, g_2 is convex and 1-Lipschitz. Hence,$l_{logist}\left(\cdot,(\mathbf{x}, y)\right)$ is convex and $\|\tilde{\mathbf{x}}\|$-Lipschitz.

- For Scenario 1, $\|\tilde{\mathbf{x}}\|$ is bounded by $\rho=\sqrt{5}$. C is 5-dimensional hypercuble which is a convex set and bounded by $M=\sqrt{5 * 2^2}=2 \sqrt{5}$
- For Scenario 2, $\|\tilde{\mathbf{x}}\|$ is bounded by $\rho=\sqrt{5}$. C is 5-dimensional unit ball which is a convex set and bounded by $M=2$

4 Results

The experimental results are shown in Table 1. The expected excess risk and the standard deviation of risks can be found in Figure 1. The classification error and its standard deviation can be found in Figure 2.

					Logistic loss				Classification error	
Scenario	σ	n	N	#trials	Mean	Std Dev	Min	Excess Risk	Mean	Std Dev
1	0.10	50	30	0.478278	0.067501	0.352964	0.125314	0.217750	0.195819	0.234416
1	0.10	100	30	0.416993	0.037942	0.334157	0.082836	0.179667	0.172073	0.148509
1	0.10	500	30	0.357439	0.018623	0.319678	0.037761	0.022583	0.019132	0.004687
1	0.10	1000	30	0.344334	0.011261	0.319196	0.025138	0.009000	0.008103	0.000000
1	0.35	50	30	0.496651	0.057732	0.388830	0.107821	0.258500	0.106536	0.102299
1	0.35	100	30	0.454405	0.039142	0.384569	0.069836	0.173167	0.067924	0.131237
1	0.35	500	30	0.388599	0.014437	0.356599	0.032000	0.162667	0.029197	0.019663
1	0.35	1000	30	0.365337	0.007391	0.348272	0.017064	0.107167	0.017317	0.012599
2	0.10	50	30	0.574050	0.060937	0.487055	0.086995	0.384500	0.195184	0.253846
2	0.10	100	30	0.546844	0.038856	0.487513	0.059332	0.303417	0.169155	0.234262
2	0.10	500	30	0.501091	0.013411	0.479484	0.021607	0.350667	0.114501	0.022411
2	0.10	1000	30	0.494299	0.011344	0.477639	0.016660	0.351667	0.115174	0.000000
2	0.35	50	30	0.593439	0.047311	0.511337	0.082102	0.378667	0.119815	0.198564
2	0.35	100	30	0.578969	0.037193	0.512053	0.066917	0.374667	0.106254	0.172125
2	0.35	500	30	0.534508	0.014530	0.516074	0.018434	0.327083	0.053612	0.085103
2	0.35	1000	30	0.519265	0.010103	0.501096	0.018169	0.301333	0.048044	0.044888

Table 1: Experimental results.

Figure 1: Expected excess risk and standard deviation of risks.

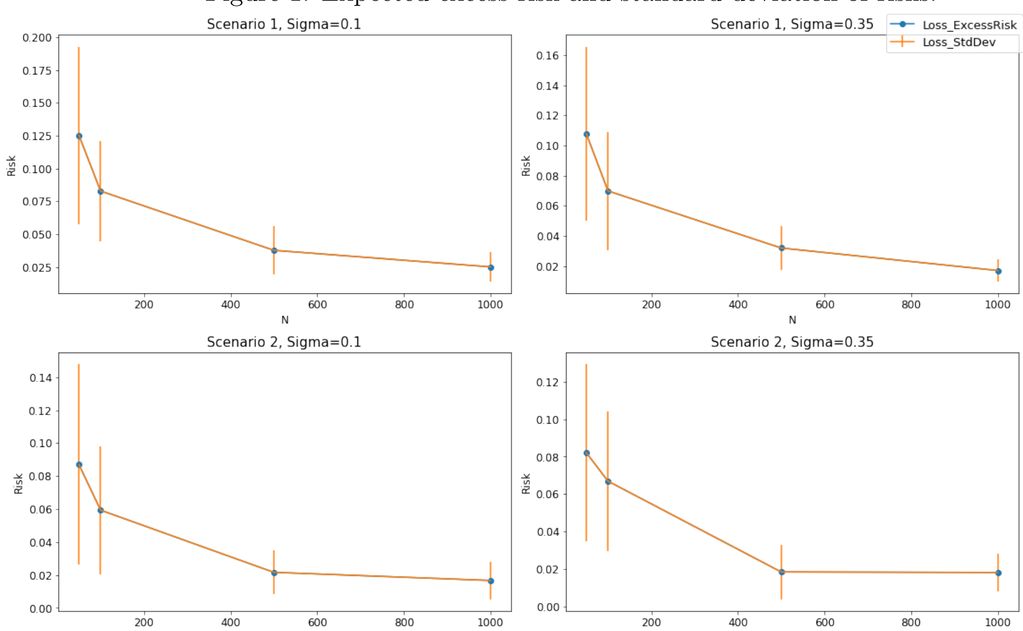
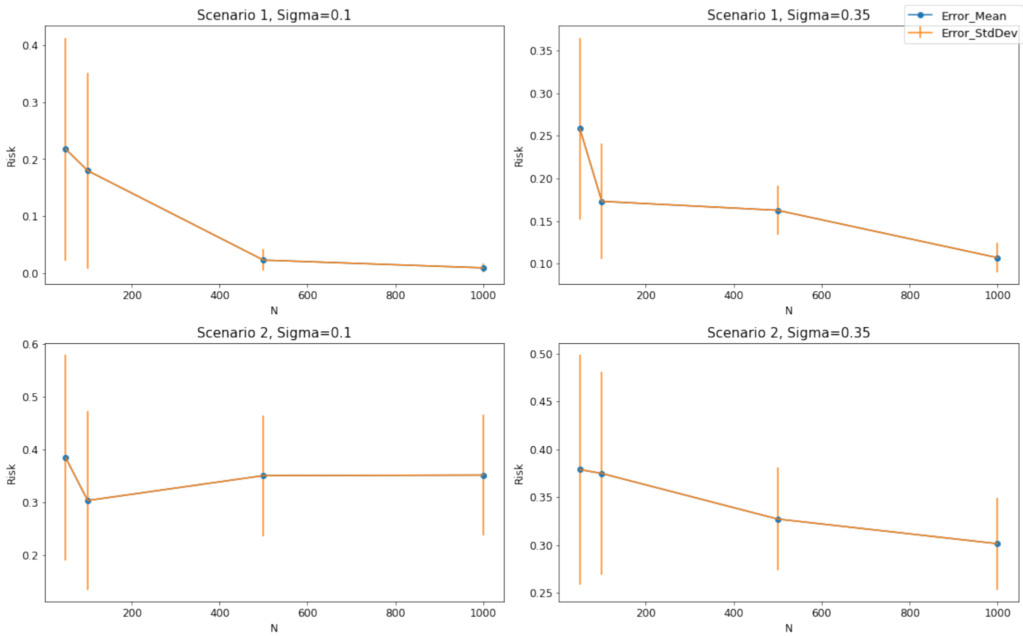


Figure 2: Classification error and its standard deviation.



5 Conclusion

A Appendix: Symbol Listing

Symbol	Description	Variable name in code
$\mathbf{w}_t$	weight vector at t	w_t
$\tilde{\mathbf{x}}_t$	extended feature vector at t	X
y_t	label at t	y
$\nabla f(\mathbf{w}_t)$	gradient at t	g
Π_C	Euclidean projection onto C	prj_grad
α	step size	l_rate
n	training set size	bs
N	test set size	test_n
$\mathbf{w}$	weight vector	w
$\mathbf{x}$	feature vector	X
$\tilde{\mathbf{x}}$	extended feature vector (with '1' appended at the end of x)	X

B Appendix: Library Routines

C Appendix: Code

```
import numpy as np
import random
import matplotlib.pyplot as plt
import pandas as pd


def cube_prj(sample):
    '''
    This function projects both domain and parameter sets to a hypercube.

    sample: features or gradients, 1*d array (d: #dimension)

    return:
        a hypercube with edge length 2 and centered around the origin
    '''
    return [np.sign(i) * min(np.abs(i), 1) for i in sample]


def ball_prj(sample):
    '''
    This function projects both domain and parameter sets to a unit ball.

    sample: features or gradients, 1*d array (d: #dimension)

    return:
        a unit ball centered around the origin
    '''
    ratio = 1 / np.linalg.norm(sample)
    return [i * ratio for i in sample]


def prj_data(X, y, prj_code):
    '''
    This function projects the domain set in terms for two scenarios.

    X: feature vectors, n*d array (n: #sample, d: #dimension)
    y: labels, 1*n array with values of -1 or +1
    prj_code: type of projection, 0 for cube, 1 for ball

    return:
        prj_x: projected feature vectors
        y: labels, same as the input
```

```
    '''
    if prj_code == 0:
        prj_x = np.apply_along_axis(cube_prj, 1, X)
    elif prj_code == 1:
        prj_x = np.apply_along_axis(ball_prj, 1, X)
    else:
        print("Please input correct code for projection type: 0 for cube, 1 for ball.")

    b = np.ones((prj_x.shape[0], 1))
    prj_x = np.append(prj_x, b, axis=1)
    return prj_x, y


def prj_grad(g, prj_code):
    '''
    This function projects the parameter set for two scenarios.

    g: gradients, 1*d array (d: #dimension)
    prj_code: type of projection, 0 for cube, 1 for ball

    return:
        prj_g: projected gradients
    '''
    if prj_code == 0:
        prj_g = cube_prj(g)
    elif prj_code == 1:
        prj_g = ball_prj(g)
    else:
        print("Please input correct code for projection type: 0 for cube, 1 for ball.")
    return prj_g


def gen_data(sig, n, d_dimension):
    '''
    This function generates the data for training and test.

    sig: standard deviation of the Gaussian function
    n: number of samples
    d_dimension: dimensionality of the feature vectors

    Return:
        X: feature vectors, n*d array (n: #sample, d: #dimension)
        y: labels, 1*n array with values of -1 and +1
    '''
    y = np.random.choice([-1, 1], p = [0.5, 0.5], size = n)
    X = np.array([])
```

```
    for i in range(n):
        if y[i] == -1:
            mu = -(1 / 4)
            negvec = np.random.normal(mu, sig, d_dimension)
            X = np.concatenate([X, negvec], axis=0)
        else:
            mu = (1 / 4)
            posvec = np.random.normal(mu, sig, d_dimension)
            X = np.concatenate([X, posvec], axis=0)
    X = np.reshape(X, (n, d_dimension))
    return X, y


def log_loss(X, y, w):
    '''
    This function outputs the logistic loss.

    X: feature vector, 1*d array (d: #dimension)
    y: label
    w: weight vector, 1*d array

    Return: logistic loss
    '''
    return np.log(1 + np.exp(-y * np.dot(w.T, X)))


def err(X, y, w):
    '''
    This function outputs the classification error.

    X: feature vector, 1*d array (d: #dimension)
    y: label
    w: weight vector, 1*d array

    Return: classification error
    '''
    yhat = -1 if np.dot(w.T, X) < 0.5 else 1
    return 0 if yhat == y else 1


def sgd(X, y, w_t, prj_code, l_rate):
    '''
    This function implements SGD.

    X: feature vectors, n*d array (n: #sample, d: #dimension)
    y: labels, 1*n array
```

```
    w_t: weights at t, n*d array
    prj_code: type of projection, 0 for cube, 1 for ball
    l_rate: learning rate

    Return:
        w_t: updated weight at t+1
    '''
    w_t = np.array(w_t)
    g = (-y * X * np.exp(-y * np.dot(w_t.T, X)) / (1 + np.exp(-y * np.dot(w_t.T, X))))
    w_t = prj_grad(np.add(w_t, np.multiply(-l_rate, g)), prj_code)
    return w_t


def train(train_x, train_y, test_x, test_y, l_rate, n_epoch, bs, prj_code):
    '''
    This function implements and tests the SGD algorithm for logistic regression.

    train_x: feature vectors for training, n*d array (n: #sample, d: #dimension)
    train_y: labels for training, 1*n array
    test_x: feature vectors for test, n*d array (n: #sample, d: #dimension)
    test_y: labels for test, 1*n array
    l_rate: learning rate
    n_epoch: number of trials
    bs: training set size
    prj_code: type of projection, 0 for cube, 1 for ball

    Return:
        w: final weights
        risk_ave: average risk
        risk_min: minimum of all risks
        risk_var: standard deviation of all risks
        exp_excess_risk: expected excess risk
        cls_err_ave: average classification error
        cls_err_var: standard deviation of all classification errors
    '''
    risk_all = []
    cls_err_all = []

    for epoch in range(n_epoch):
        w_t = np.random.uniform(-1, 1, (train_x.shape[1]))
        risk = cls_err = 0.
        w_all = []
        for idx in range(epoch * bs, (epoch + 1) * bs):
            # Read data
            X = train_x[idx]
            y = train_y[idx]
```

```
            # SGD
            w_t = sgd(X, y, w_t, prj_code, l_rate)
            # Backward propagation
            w_all.append(w_t)

        w = np.average(np.array(w_all), axis=0)

        # Evaluate
        for idx in range(test_x.shape[0]):
            # Read data
            X = test_x[idx]
            y = test_y[idx]
            # Evaluate
            risk += log_loss(X, y, w) / test_x.shape[0]
            cls_err += err(X, y, w) / test_x.shape[0]

        risk_all = np.append(risk_all, risk)
        cls_err_all = np.append(cls_err_all, cls_err)

    # Report risk
    risk_ave = np.average(risk_all)
    risk_min = np.amin(risk_all)
    risk_var = np.sqrt(np.var(risk_all))
    exp_excess_risk = risk_ave - risk_min
    # Report classification error
    cls_err_ave = np.average(cls_err_all)
    cls_err_var = np.sqrt(np.var(cls_err_all))
    return [w, risk_ave, risk_min, risk_var, exp_excess_risk, cls_err_ave, cls_err_var]


# Set up hyperparameters
n_epoch = 30    # training epochs
test_n = 400    # size of test set
d_dimension = 4
train_bs = np.array([50, 100, 500, 1000])  # batch size for each training epoch

np.random.seed(1)

result_list = []
for prj_code in [0, 1]:
    for sigma in [0.1, 0.35]:
        for bs in train_bs:

            if prj_code == 0:
                m = 2 * np.sqrt(d_dimension + 1)
            else:
```

```
    m = 2

rho = d_dimension + 1
l_rate = m / (rho * np.sqrt(bs))

# Generate training data
train_x, train_y = gen_data(sigma, bs * n_epoch, d_dimension)
train_px, train_py = prj_data(train_x, train_y, prj_code)

# Generate test data
test_x, test_y = gen_data(sigma, test_n, d_dimension)
test_px, test_py = prj_data(test_x, test_y, prj_code)

# Train
output = train(train_px, train_py, test_px, test_py, l_rate, n_epoch, bs, prj_code)

print('>scenario=%d, sigma=%.2f, n=%d, lr=%.2f, log_loss_mean=%.3f, \
    log_loss_std_dev=%.3f, log_loss_min=%.3f, \
    excess_risk=%.3f, cls_error_mean=%.3f, cls_error_std_dev=%.3f'
    % (prj_code + 1, sigma, bs, l_rate, output[1], output[3], \
        output[2], output[4], output[5], output[6]))
result = [prj_code + 1, sigma, bs, n_epoch,output[1], output[3], \
    output[2], output[4], output[5], output[6]]
result_list.append(result)
```