Project: Stochastic Gradient Descent

Zilin Wang.10455 and Jude Rajasekera.3

1 Introduction

The goal of this project was to implement the stochastic gradient descent algorithm for logistic regression and test it in different scenarios. By completing this project, we gained a better understanding of the SGD algorithm we learned in class.

Essentially, the SGD algorithm takes some inputs, updates the weight vector of the logistic regression model iteratively, and returns the final weight vector. The inputs in our case include the logistic loss function, a data generator, a scenario, number of iterations, and learning rate. To start, our algorithm initializes the weight of the logistic regression model to the all-zero vector in the given dimension. After this, it repeats the following procedure for a given number of iterations: generate a fresh data sample that fits the given scenario from the data generator; compute the gradient according to the logistic loss function; update the weight vector by subtracting it by the product of the gradient with the learning rate; update the weight again by projecting it onto the parameter set given by the scenario. Finally, our algorithm returns the average of the updated weight vectors in every iteration.

The workload was divided evenly and every design or detail of the model was finalized with discussions between both team members. For the coding part, Jude was mainly responsible for projecting examples onto the domain sets, setting up experiments, evaluating model performance, and plotting the graphs, while Zilin mainly worked on data generation, the SGD algorithm and logistic regression model. For this report, Jude explained how we designed our experiments, and Zilin analyzed the M-bound, ρ-Lipschitz, and proved convexity for both scenarios. The rest of the sections in this report were composed by both team members in collaboration.

2 Experiments

Before explaining the conducted experiments, we will explain the various settings and scenarios. Each experiment belonged to one of two scenarios which characterized both the data and parameter spaces on which the implementation of SGD would be trained and tested. Scenario 1 required that the domain set *X* be a 4-dimensional hypercube centered at the origin with edge length 2, with the parameter set *C* being a 5-dimensional hypercube centered around the origin with edge length 2. Scenario 2 required that the domain set *X* be a 4-dimensional unit ball centered around the origin, with the parameter set *C* being a 5-dimensional unit ball centered around the origin.

The data distribution *D* was defined as follows: with probability ½, set y = -1 and x = u, where u is a 4-dimensional Gaussian vector generated from the distribution $N(\mu_0, \sigma^2 I_4)$, with μ_0 = (-¼, -¼, -¼, -¼), σ = specified as experiment setting, I_4 = identity matrix of rank 4. With the remaining probability, set y = 1 and x = u, where u is generated from a similar distribution to the one above, except with mean μ_0 = (¼, ¼, ¼, ¼). This step in data generation is common to both scenarios, and was implemented in the `fresh_data(num)` method. This method generates `num` Example objects. For each object, a random float between 0 and 1 is generated. If that random float is greater than 0.5, the example's y is set to -1 and a 4-dimensional vector u is generated from the gaussian distribution described above where μ_0 = (-¼, -¼, -¼, -¼). If the random float is less than 0.5, a similar process if followed, except y is set to +1 and u is generated from the gaussian distribution where μ_0 = (¼, ¼, ¼, ¼). Finally, vector u is augmented so that u_augmented = (u, 1) and an example object is created with `Example(u_augmented, y)`. The function returns a list of generated examples.

The next step in data generation is projecting vector u onto the according domain set *X* for each scenario. In the case of scenario 1, the projection of u onto X is just setting the max value = 1 and the min value = -1 for all values in vector u. In other words, if any value in vector u is greater than 1, it is set to 1, and if any value is less than -1, it is set to -1. This logic is defined in function `project_scene1(feature_vector)`. For scenario 2, the projection of u onto X is defined as follows: if the euclidean norm of u is greater than 1, all values in u are divided by the original euclidean norm. This logic is defined in function `project_scene2(feature_vector)`.

In order to test the performance of the SGD learner in different scenarios with different settings, we ran a set of 16 experiments. Each experiment consisted of 30 trials, each trial having the same settings and scenario. Since the training data was generated randomly during each trial, the performance would differ slightly between trials. The aggregation of the trials were used to calculate the expected performance for each experiment.

In total, 16 unique experiments were conducted. 8 experiments were conducted for each of the two specified scenarios. For each scenario, half of the experiments were run with σ = 0.1, while the other half were run with σ = 0.35. For each set of 4 experiments with the same scenario and σ setting, a unique n was picked from a set of values: {50, 100, 500, 100}, representing the number of examples in the training set for that experiment. Therefore the parameters we were able to control in the experiment were the scenario, σ, and n. The `train_logistic_regression(std_dev, scenario, n)` function was used to create and train a `LogisticRegressionClassifier` object we could then evaluate the performance of. This method generated training examples one by one, in accordance with the passed in parameters, and calculated a final weight vector. The resulting model was then tested using the `evaluate_logistic_regression(model, data_file, target_file)` function, against a test dataset that was generated beforehand. This function returned a logistic loss estimate and a classification error. As mentioned before, these values would be used to calculate the expected performance over 30 trials. The test datasets were generated using the `renew_test_files(files, std_devs, N)` function, which saves the data to text files so that the test data would be consistent between all experiment runs.

3 Analysis of -Lipschitz properties

According to *Intro to Convex Learning* slides, we know that the logistic loss is convex and $||\tilde{x}||$-Lipschitz, where $||\tilde{x}||$ is the largest possible L2-norm of the augmented data points in the particular domain set of a scenario.

For scenario 1, the domain set is the 4-dimensional hypercube with edge length 2. It is easy to see that the data points that has the largest L2-norm is at each corner of the hypercube, one of which is $(1, 1, 1, 1)$. So, we can calculate the L2-norm of the augmented version of this point as $\sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2} = \sqrt{5}$. Thus, **the loss function is convex and $\sqrt{5}$-Lipschitz in scenario1.** Next, I will prove the parameter set C is convex and find the M-bound for it. For scenario1, $C = [-1, 1]^5$. Let $u, v \in C$ and u_j, v_j be attributes in u, v. We need to prove that for any $\lambda \in [0, 1]$, we have $\lambda u + (1 - \lambda)v \in C$. First, subtract 1 by $\lambda u_j + (1 - \lambda)v_j$, we have $(1 - \lambda u_j) + (\lambda - 1)v_j$. To minimize this value, we need maximum $u_j$and v_j, which is 1. Thus, $(1 - \lambda u_j) + (\lambda - 1)v_j > 0$, meaning $\lambda u_j + (1 - \lambda)v_j < 1$. Similarly, by subtracting -1 by $\lambda u_j + (1 - \lambda)v_j$ and maximizing the result, we can show that $\lambda u_j + (1 - \lambda)v_j > -1$. So, every attribute in $\lambda u + (1 - \lambda)v$ is in $[-1, 1]$. In other words, $\lambda u + (1 - \lambda)v \in C$. Hence, **the parameter set C is convex in scenario1**. It is also easy to see that the longest distance between two points in C is the distance between two corner points in the longest diagonal, such as $(-1, -1, -1, -1, -1)$ and $(1, 1, 1, 1, 1)$. **Thus, $M = \sqrt{5 * (1 - (-1))^2} = 2\sqrt{5}$ in scenario1.**

For scenario 2, the domain set is the 4-dimensional unit ball. It is easy to see that the data points that has the largest L2-norm is on the shell of the unit ball, one of which is $(1, 0, 0, 0)$. So, we can calculate the L2-norm of the augmented version of this point as $\sqrt{1^2 + 0^2 + 0^2 + 0^2 + 1^2} = \sqrt{2}$. Thus, **the loss function is convex and $\sqrt{2}$-Lipschitz in scenario2**. Next, I will prove the parameter set C is convex and find the M-bound for it. For scenario2, $C = \{w \in R^5 : ||w|| \leq 1\}$. Let $u, v \in C$ and u_j, v_j are attributes in u, v. We need to prove that for any $\lambda \in [0, 1]$, we have $\lambda u + (1 - \lambda)v \in C$. By property of L2-norm, we know that $||\lambda u + (1 - \lambda)v|| \leq \lambda||u|| + (1 - \lambda)||v||$. Note that $||u||, ||v|| \in [0, 1]$. Using the similar method in the previous paragraph, we can show that

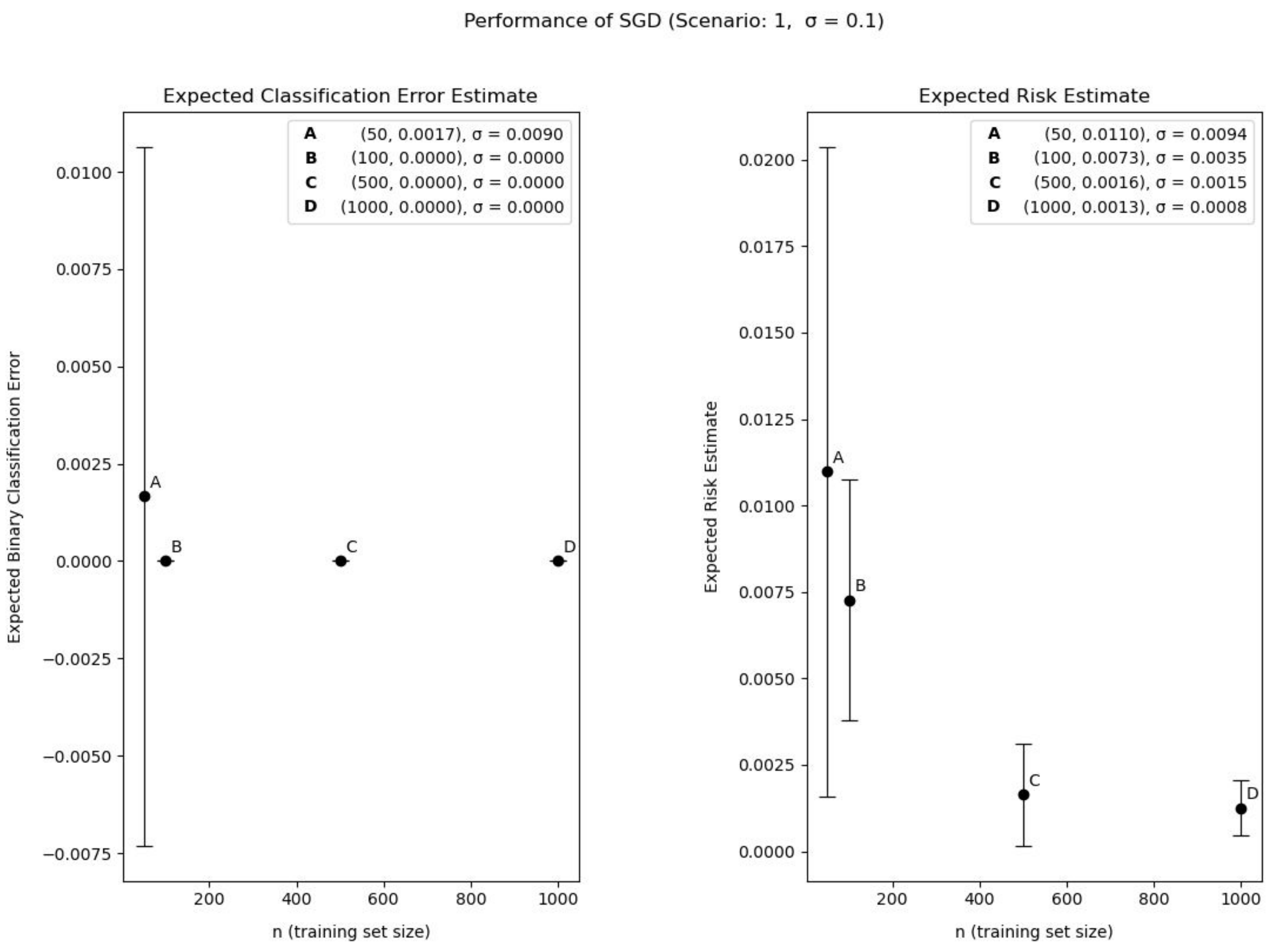
$\lambda||u|| + (1-\lambda)||v|| \leq 1$. Thus, $||\lambda u + (1-\lambda)v|| \leq 1$. In other words, $\lambda u + (1-\lambda)u \in C$. Hence, **the parameter set C is convex in scenario2**. It is again easy to see that the longest distance between two points is the diameter of this 5-dimensional unit ball, which is 2. **Thus, $M = 2$ in scenario2.**

We used constant learning rates for experiments that have the same training size (number of iterations) and in the same scenarios (ρ and M). Specifically, given number of iterations T, ρ-Lipschitz, and M-bound, the learning rate we used was: $\frac{M}{\rho\sqrt{T}}$

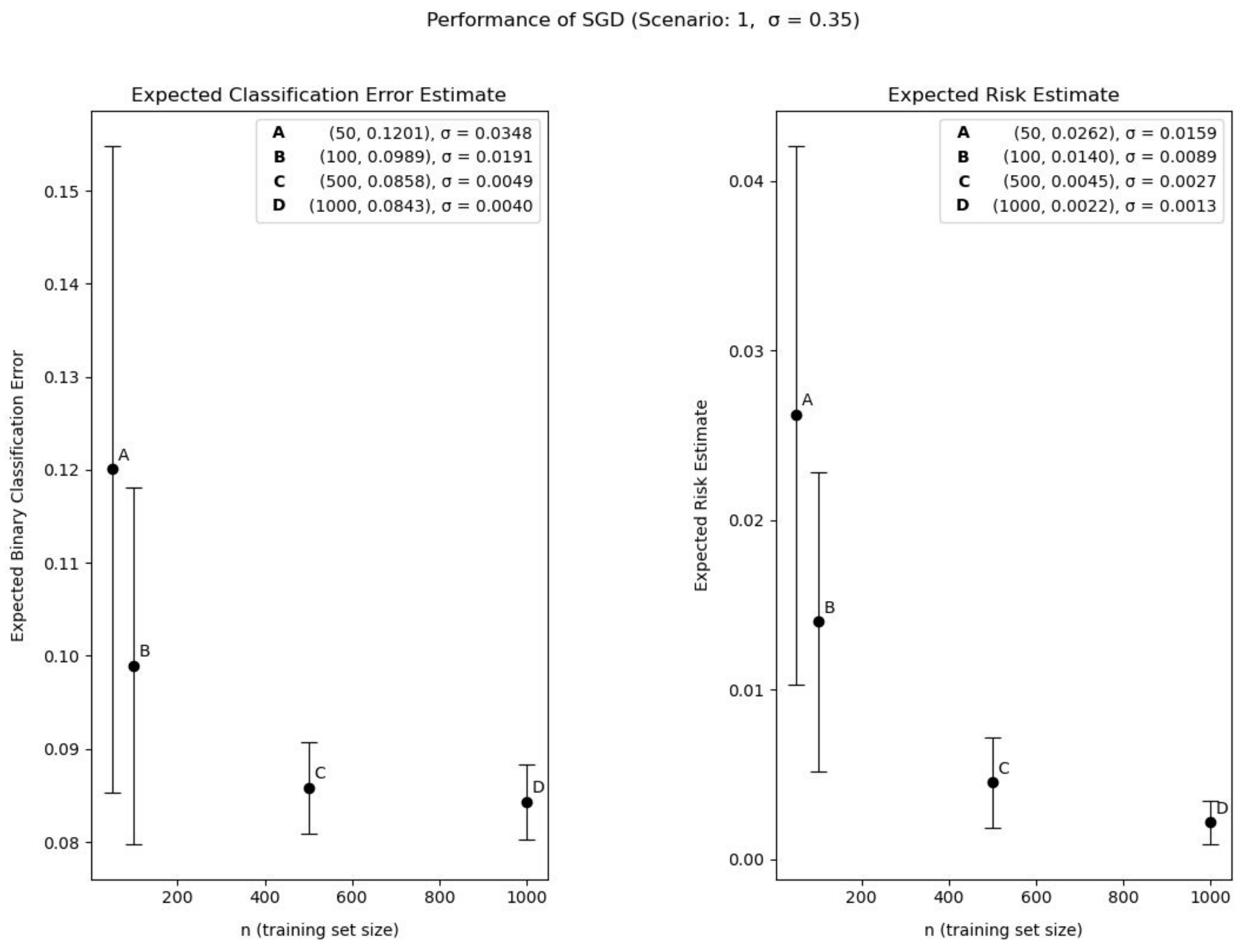
4 Results

Below is a table showing the results of the 16 experiments we conducted. The two following pages contain plots of our model's performance for each unique setting of scenario and σ.

Settings					Logistic Loss				Classification error	
Scenario	σ	*n*	N	Trials	Mean	Std Dev	Min	Excess Risk	Mean	Std Dev
1	0.1	50	400	30	0.42889	0.00940	0.41792	0.01097	0.00067	0.00167
1	0.1	100	400	30	0.39272	0.00348	0.38546	0.00726	0.00000	0.00000
1	0.1	500	400	30	0.35082	0.00147	0.34918	0.00163	0.00000	0.00000
1	0.1	1000	400	30	0.34150	0.00079	0.34025	0.00125	0.00000	0.00000
1	0.35	50	400	30	0.46439	0.01588	0.43819	0.02620	0.12008	0.03476
1	0.35	100	400	30	0.43079	0.00886	0.41680	0.01400	0.09892	0.01913
1	0.35	500	400	30	0.38886	0.00266	0.38434	0.00453	0.08583	0.00489
1	0.35	1000	400	30	0.38020	0.00127	0.37804	0.00216	0.08433	0.00403
2	0.1	50	400	30	0.52293	0.00556	0.51556	0.00737	0.00858	0.01883
2	0.1	100	400	30	0.50861	0.00492	0.50328	0.00533	0.00683	0.02002
2	0.1	500	400	30	0.49028	0.00225	0.48767	0.00261	0.00008	0.00044
2	0.1	1000	400	30	0.48588	0.00115	0.48427	0.00161	0.00000	0.00000
2	0.35	50	400	30	0.55154	0.01137	0.53498	0.01657	0.13200	0.05988
2	0.35	100	400	30	0.53484	0.00681	0.52612	0.00873	0.11000	0.03041
2	0.35	500	400	30	0.51481	0.00210	0.51077	0.00404	0.08658	0.00485
2	0.35	1000	400	30	0.51113	0.00190	0.50844	0.00269	0.08650	0.00587

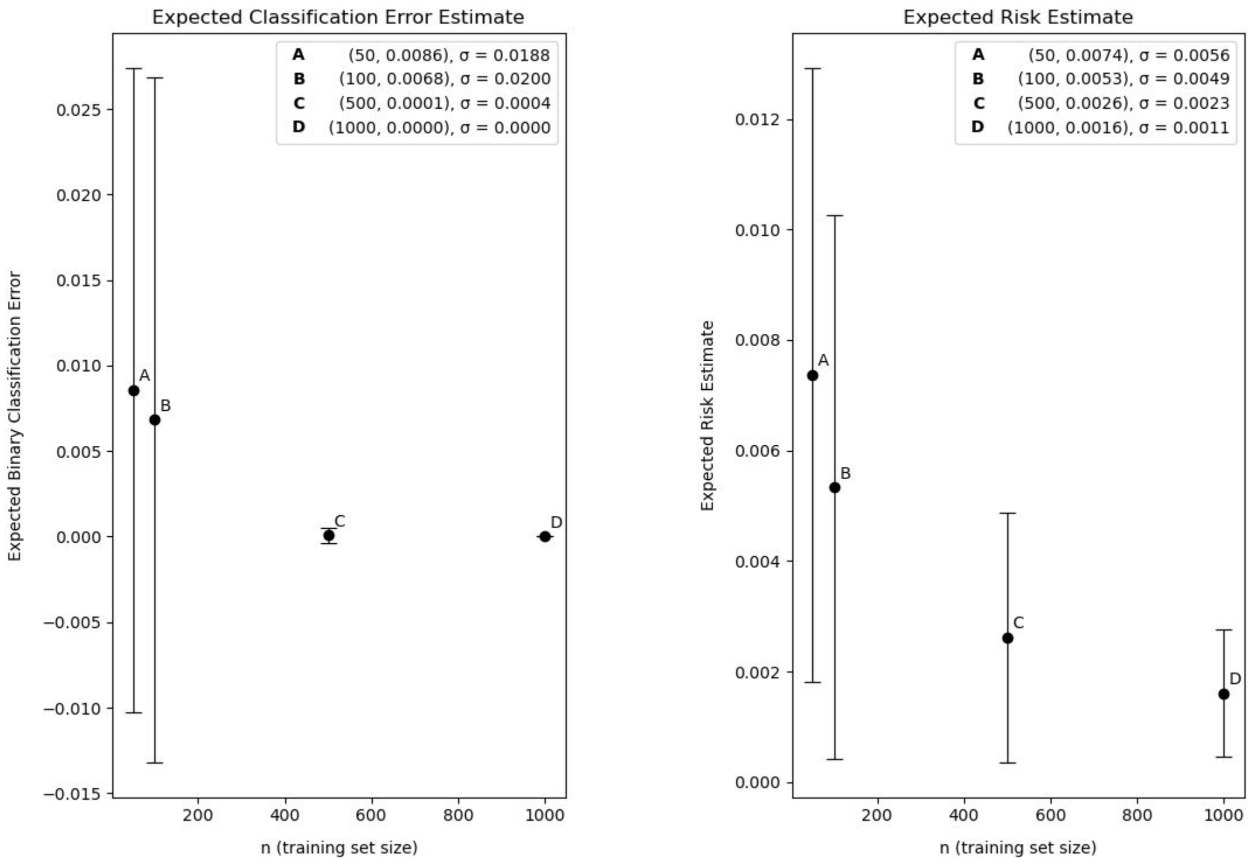


Performance of SGD (Scenario: 1, σ = 0.1)
Expected Classification Error Estimate
A (50, 0.0017), σ = 0.0090
B (100, 0.0000), σ = 0.0000
C (500, 0.0000), σ = 0.0000
D (1000, 0.0000), σ = 0.0000
Expected Binary Classification Error
n (training set size)
Expected Risk Estimate
A (50, 0.0110), σ = 0.0094
B (100, 0.0073), σ = 0.0035
C (500, 0.0016), σ = 0.0015
D (1000, 0.0013), σ = 0.0008
Expected Risk Estimate
n (training set size)

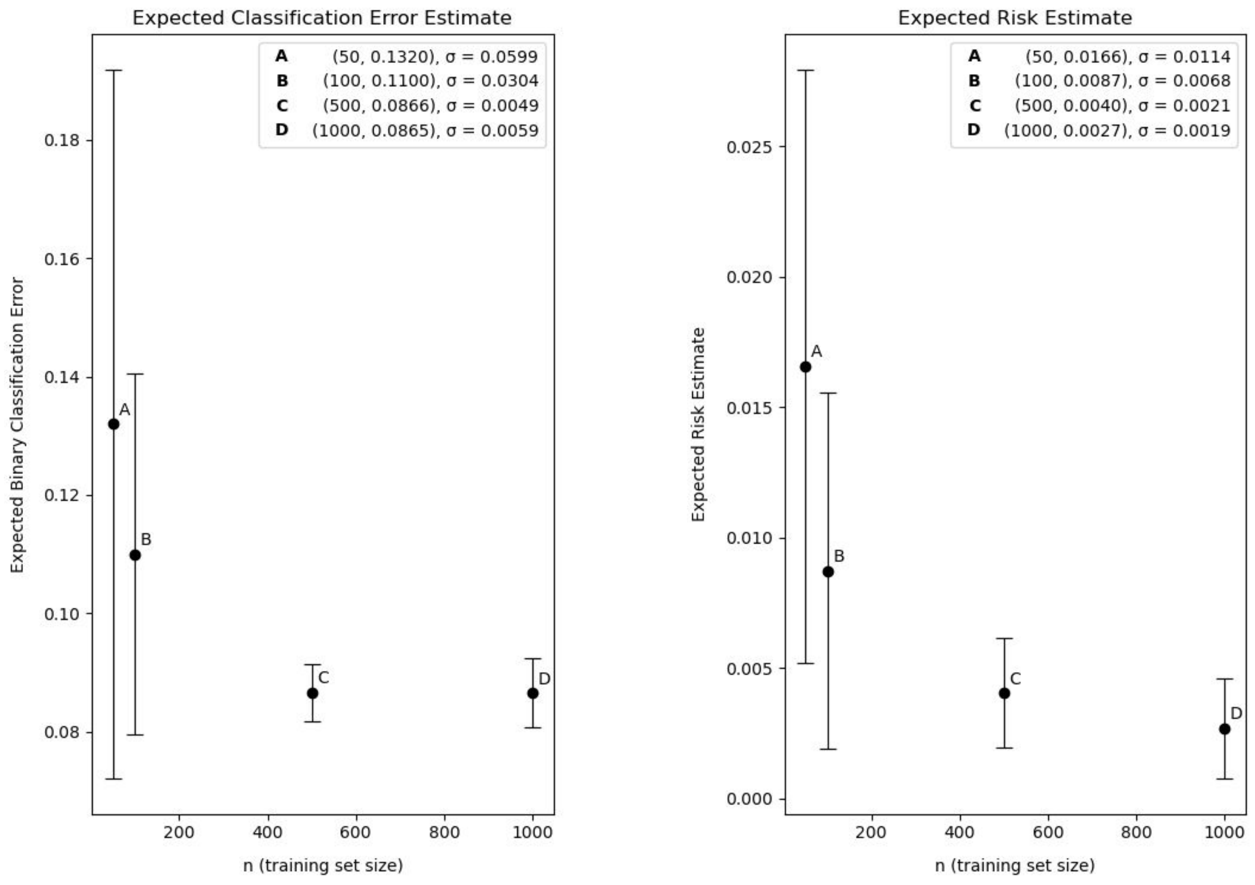


Performance of SGD (Scenario: 1, σ = 0.35)
Expected Classification Error Estimate
A (50, 0.1201), σ = 0.0348
B (100, 0.0989), σ = 0.0191
C (500, 0.0858), σ = 0.0049
D (1000, 0.0843), σ = 0.0040
Expected Binary Classification Error
n (training set size)
Expected Risk Estimate
A (50, 0.0262), σ = 0.0159
B (100, 0.0140), σ = 0.0089
C (500, 0.0045), σ = 0.0027
D (1000, 0.0022), σ = 0.0013
Expected Risk Estimate
n (training set size)

Performance of SGD (Scenario: 2, σ = 0.1)



Performance of SGD (Scenario: 2, σ = 0.35)



5 Conclusion

The theorem we learned in class about the learnability of bounded-convex-Lipschitz problems states that: if a learning problem is (M, ρ) bounded-convex-Lipschitz, then there is an efficient algorithm such that for any $\epsilon > 0$, if that algorithm is given a training set S of size $n = n_H(\epsilon) \approx \frac{M^2\rho^2}{\epsilon^2}$ i.i.d examples (frome some unknown distribution D over Z), it outputs a parameter vector $\widehat{W}_S \in C$ that satisfies:

$$E_{S\sim D^n}[L(\widehat{W}_S; D)] \leq \min_{w\in C} L(w; D) + \epsilon.$$

One of the efficient algorithms is stochastic gradient descent, which we utilized in this project. Thus, we can get ϵ in terms of n, M, and ρ from the first equation with $\epsilon \approx M\rho\sqrt{\frac{1}{n}}$. Then, plugging in ϵ to the second equation, we can rewrite the inequality as $E_{S\sim D^n}[L(\widehat{W}_S; D)] - \min_{w\in C} L(w; D) \leq M\rho\sqrt{\frac{1}{n}}$. The left side of the inequality represents an estimate of the expected excess risk of our SGD learner, which we found during our experiments.

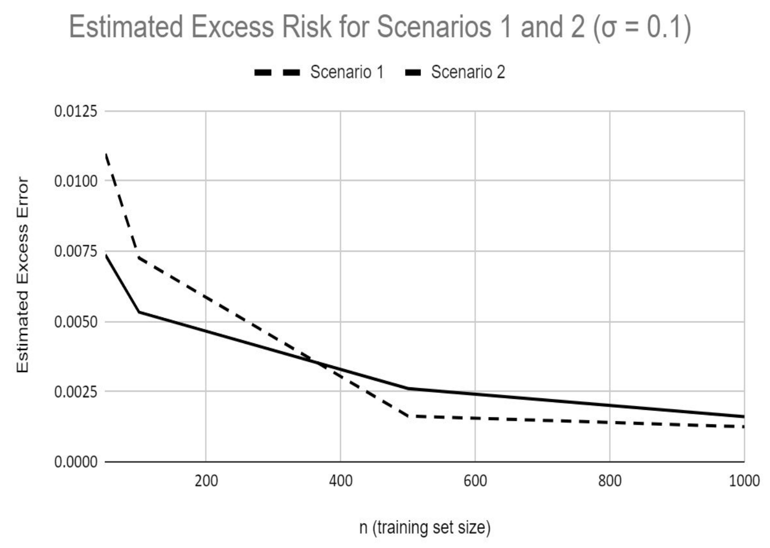
$$Estimate\ of\ Expected\ Excess\ Risk \leq M\rho\sqrt{\frac{1}{n}}.$$

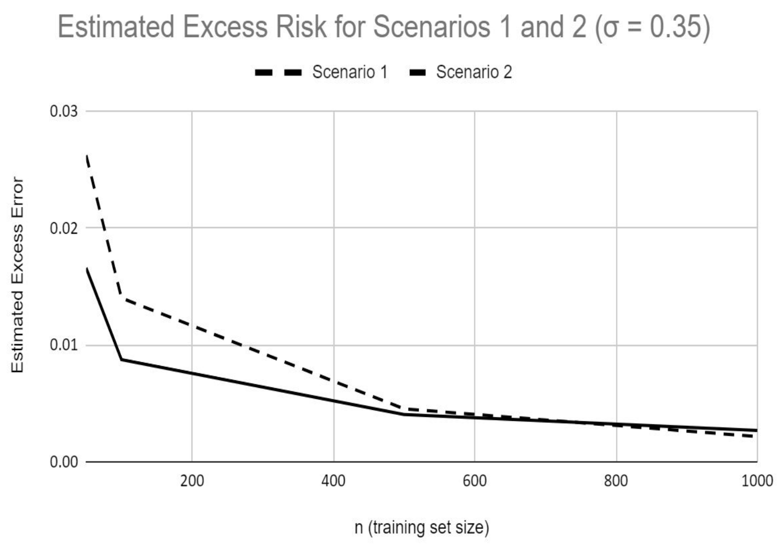
Using this inequality we can prove that for each of our experiments, the excess risk is bounded. The table on the next page shows the calculated expected excess risk upper bound and estimated expected excess risk for each experiment. For all experiments, the estimate of expected risk was below the upper bound. Therefore our results are in agreement with the theoretical results we derived in class, with respect to the excess error bound.

Scenario	σ	n	Expected Excess Risk Upper Bound	Estimate of Expected Excess Risk	Estimate less than bound?	Difference
1	0.10	50	1.41421	0.01097	Yes	1.40324
1	0.10	100	1.00000	0.00726	Yes	0.99274
1	0.10	500	0.44721	0.00163	Yes	0.44558
1	0.10	1000	0.31623	0.00125	Yes	0.31498
1	0.35	50	1.41421	0.02620	Yes	1.38801
1	0.35	100	1.00000	0.01400	Yes	0.98600
1	0.35	500	0.44721	0.00453	Yes	0.44268
1	0.35	1000	0.31623	0.00216	Yes	0.31407
2	0.10	50	0.40000	0.00737	Yes	0.39263
2	0.10	100	0.28284	0.00533	Yes	0.27751
2	0.10	500	0.12649	0.00261	Yes	0.12388
2	0.10	1000	0.08944	0.00161	Yes	0.08783
2	0.35	50	0.40000	0.01657	Yes	0.38343
2	0.35	100	0.28284	0.00873	Yes	0.27411
2	0.35	500	0.12649	0.00404	Yes	0.12245
2	0.35	1000	0.08944	0.00269	Yes	0.08675

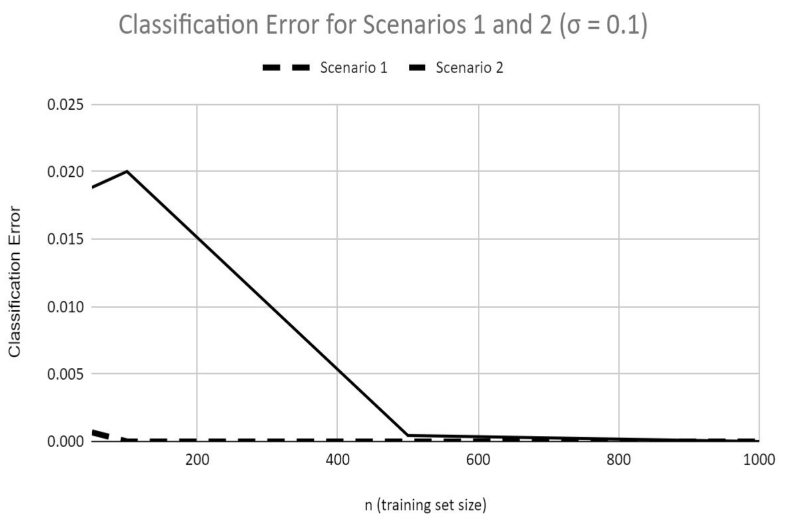
Note that in all cases, the expected risk estimates clearly decreased as the number of examples in the training set increased. This indicates that our stochastic gradient descent algorithm worked as desired on our logistic regression model. As for the expected classification error estimates, although there were a little fluctuations shown in the figure, they were generally decreasing as the training size was increasing. These fluctuations were within expectations due to the stochastic nature of our algorithm.

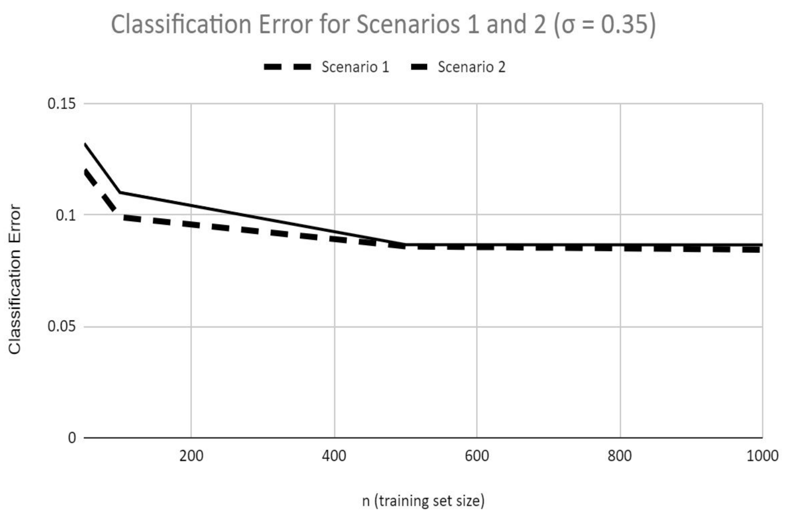
Each scenario required a different domain set and parameter set, which is why results varied between scenarios even when all other settings are held constant. Scenario 1 seemed to have a higher estimated excess risk compared to scenario 2 for small values of n; however, scenario 1 reduced estimated excess risk at a faster rate than scenario 2 (rate meaning the risk reduced per additional training set example). This can be observed on the plots on the following page. We think that this difference exists between scenarios because they have different values of (M, ρ); for scenario 1 we have $(M_{S1} * \rho_{S1}) = 10$ and for scenario 2 we have $(M_{S2} * \rho_{S2}) = 2\sqrt{2}$. Since excess risk is bounded by $M\rho\sqrt{\frac{1}{n}}$, it is apparent why excess risk is larger for smaller values of n in scenario 1 and why the excess risks for each scenario get closer to each other as n increases.





Scenario 2 always had a greater classification error than scenario 1, but as the value of n increased, the classification errors of the two scenarios quickly got closer to each other. This can be observed on the plots below.





Within the same scenario setting, our models for both standard deviation settings of the Gaussian data distribution converged quickly. However, the models for smaller standard deviation seemed to achieve slightly lower values in all four logistic loss parameters: mean, standard deviation, min and expected risk. They also seemed to perform better on expected classification error, achieving 0 classification error after 1000 training epochs while neither models for the larger standard deviation got this accuracy. After discussing this observation, we believe the reason behind this is that, with smaller standard deviation, data points with the same labels lie closer to each. Thus, if we are able to visualize the data points, these data points will form two clusters that are more distinct than those formed by data points generated by a larger standard deviation Gaussian distribution. Hence, the models could learn to classify them more easily.

A Appendix: Symbol Listing

x - data points (4 dimensional)

$\tilde{x}$ - augmented data points with an extra 1 appended (5 dimensional)

y - data label {-1,1}

w - weight of model

$\hat{w}$ - ouput predictor

$\mathcal{X}$ - domain set of each scenario

$\mathcal{C}$ - parameter set of each scenario

M - M-bound of parameter set

ρ - Lipchitz of loss function

λ - a constant in $[0, 1]$

T - number of training iterations

n - size of training set

N - size of test set

D - true data distribution

σ - standard deviation of the true data distribution

μ_0 - mean of data points with -1 label

μ_1 - mean of data points with 1 label

Π - Euclidean projection

u - vector from data distribution directly without projection

`Trial.scenario` - scenario setting of that trial instance $\{1, 2\}$

`Trial.std_dev` - data distribution standard deviation setting of that trial instance

`Trial.n` - size of training set for that trial instance

`Trial.excess_risk_estimate` - average logistic loss for that trial instance

`Trial.classification_error_estimate` - classification error for that trial instance

`Experiment.scenario` - scenario setting of that experiment instance $\{1, 2\}$

`Experiment.std_dev` - data distribution standard deviation setting of that experiment instance

`Experiment.n` - size of training set for each trial in that experiment instance

`Experiment.trials` - list of trial instances belonging to that experiment instance

`Experiment.excess_risk_estimate_mean` - expected logistic loss over all trials in that experiment instance

`Experiment.excess_risk_estimate_std_dev` - standard deviation of the average loss in that experiment instance

`Experiment.excess_risk_estimate_min` - minimum logistic loss average over all trials in that experiment instance

`Experiment.expected_excess_risk` - expected logistic loss minus minimum logistic loss average over all trials in that experiment instance

`Experiment.classification_error_estimate_mean` - mean classification error over all trials in that experiment instance

`Experiment.classification_error_estimate_std_dev` - standard deviation of the mean classification error over all trials in that experiment instance

B Appendix: Library Routines

We built our model and experiments from scratch and only used libraries such as numpy and matplotlib whose functionalities are obvious from the function names.

C Appendix: Code

Below is the code we wrote to generate data, train/evaluate the model, and plot the results.

main.py

```
import argparse
import numpy as np
from typing import List
from experiment import *


"""
    Command-line arguments to the system.
    :return: the parsed args bundle
"""
def _parse_args() -> argparse.Namespace:

    parser = argparse.ArgumentParser(description='main.py')

    # Renew testing data
    parser.add_argument('--renew_test', default=False, action='store_true',
help='generate new test data sets')

    # SGD model hyperparameters
    parser.add_argument('--lr', type=float, default=.01, help='learning rate of SGD
model')
    parser.add_argument('--N', type=int, default=400, help='number of testing data
examples(normally should not be changed)')

    # Data files and target files for each combination of scenarios and sigma
    parser.add_argument('--data_s1_sig1', type=str,
default='test_data/data_s1_sig1.txt', help='path to data of scenario 1, sigma value 1')
    parser.add_argument('--target_s1_sig1', type=str,
default='test_data/target_s1_sig1.txt', help='path to targets of scenario 1, sigma value
1')

    parser.add_argument('--data_s1_sig2', type=str,
default='test_data/data_s1_sig2.txt', help='path to data of scenario 1, sigma value 2')
    parser.add_argument('--target_s1_sig2', type=str,
default='test_data/target_s1_sig2.txt', help='path to targets of scenario 1, sigma value
2')

    parser.add_argument('--data_s2_sig1', type=str,
default='test_data/data_s2_sig1.txt', help='path to data of scenario 2, sigma value 1')
    parser.add_argument('--target_s2_sig1', type=str,
default='test_data/target_s2_sig1.txt', help='path to targets of scenario 2, sigma value
1')

    parser.add_argument('--data_s2_sig2', type=str,
default='test_data/data_s2_sig2.txt', help='path to data of scenario 2, sigma value 2')
    parser.add_argument('--target_s2_sig2', type=str,
default='test_data/target_s2_sig2.txt', help='path to targets of scenario 2, sigma value
2')
```

```
    # May be useful later
    # parser.add_argument('--d', type=int, default=5, help='data dimensionality')
    # parser.add_argument('--n', type=int, default=500, help='number of training data 
examples')
    # parser.add_argument('--sigma', type=float, default=0.1, help='standard deviation 
of the data distribution')
    # parser.add_argument('--epoch', type=int, default=10, help='Number of iterations to 
train on the whole train data set')

    args = parser.parse_args()
    return args

"""
    Main function
"""
def main():

    # Read argument from command line and display
    args = _parse_args()
    print("\nNamespace:")
    for arg in vars(args):
        print(f'{arg.ljust(15)}: {getattr(args, arg)}')

    # 3d list for files
    files = [[[args.data_s1_sig1, args.target_s1_sig1],[args.data_s1_sig2, 
args.target_s1_sig2]],[[args.data_s2_sig1, args.target_s2_sig1],[args.data_s2_sig2, 
args.target_s2_sig2]]]

    # Define standard deviations
    std_devs = [0.1, 0.35]
    scenarios = [1,2]

    # Renew test set if needed
    if args.renew_test:
        renew_test_files(files, std_devs, args.N)
        print("\nTest files renewed!")

    # Do experiments
    for i,scenario in enumerate(scenarios):
        for j,std_dev in enumerate(std_devs):
            exp_set = ExperimentSet(scenario, std_dev, args.lr, files[i][j][0], 
files[i][j][1])
            exp_set.run()
            exp_set.plot()

    print("\nCompleted All Experiments\n")
"""
    Only run when executed as main class
"""
if __name__ == "__main__":
    main()
```

experiment.py

```
import random
import numpy as np
import matplotlib.pyplot as plt
from model import *
from matplotlib.legend_handler import HandlerBase
from matplotlib.text import Text
from matplotlib.legend import Legend
from string import ascii_uppercase

"""
    Single Trial. Represents a trial for an experiment for a given scenario, standard
deviation, and n.
    After a trial is run, it will contain estimates for excess risk and classification
error
"""
class Trial:
    trial_counter = 0

    """
        Constructor
        Parameters:
            scenario : experiment scenario {1,2}
            std_dev : distribution standard deviation
            n : training set size
    """
    def __init__(self, scenario, std_dev, n):
        self.scenario = scenario
        self.std_dev = std_dev
        self.n = n
        self.excess_risk_estimate = None
        self.classification_error_estimate = None

        Trial.trial_counter += 1
        self.id = Trial.trial_counter


    """
        Run trial (Generate data, train model, test model, and calculate risk + error)
    """
    def run(self):
        # Information needed for training and evaluation
        data_file = ExperimentSet.data_file
        target_file = ExperimentSet.target_file

        # Train model
        model = train_logistic_regression(self.std_dev, self.scenario, self.n)

        # Evaluate model, calculate excess_risk_estimate & classification_error_estimate
        self.excess_risk_estimate, self.classification_error_estimate =
evaluate_logistic_regression(model, data_file, target_file)


    def __repr__(self):
        return "Trial ID:{} --> Scenario: {} | Std Dev: {} | n: {} | Excess Risk
```

```
Estimate = {} | Classification Error Estimate = {}"\
            .format(self.id,self.scenario, self.std_dev, self.n,
self.excess_risk_estimate, self.classification_error_estimate)

    def __str__(self):
        return repr(self)

"""
    Single Experiment. Represents an experiment for a given scenario, standard
deviation, and n.
    Each experiment consists of 30 trials. After an experiment is run, it will contain
the mean,
    min, and standard deviation of the excess risk from the 30 trials, as well as the
expected
    excess risk.
"""
class Experiment:

    """
        Constructor
        Parameters:
            scenario : experiment scenario {1,2}
            std_dev : distribution standard deviation
            n : training set size
    """
    def __init__(self, scenario, std_dev, n):
        self.scenario = scenario
        self.std_dev = std_dev
        self.n = n

        # create 30 trials
        self.trials = [Trial(self.scenario, self.std_dev, self.n) for _ in range(30)]

        self.excess_risk_estimate_mean = None
        self.excess_risk_estimate_std_dev = None # standard deviation of the 30
estimates around their mean
        self.excess_risk_estimate_min = None
        self.expected_excess_risk = None   #(mean - min)

        self. classification_error_estimate_mean = None
        self. classification_error_estimate_std_dev = None


    """
        Run trial (Generate data, train model, test model, and calculate risk + error)
    """
    def run(self):
        # Run all trials
        excess_risk_estimate_array = np.zeros(30)
        classification_error_estimate_array = np.zeros(30)
        for i, trial in enumerate(self.trials):
            trial.run()
            excess_risk_estimate_array[i] = trial.excess_risk_estimate
            classification_error_estimate_array[i] = trial.classification_error_estimate

        # calculate risk and classification error stats
        self.excess_risk_estimate_mean = np.mean(excess_risk_estimate_array)
        self.excess_risk_estimate_std_dev = np.std(excess_risk_estimate_array)
        self.excess_risk_estimate_min = np.min(excess_risk_estimate_array)
        self.expected_excess_risk = self.excess_risk_estimate_mean - 
self.excess_risk_estimate_min
        self. classification_error_estimate_mean = 
np.mean(classification_error_estimate_array)
        self. classification_error_estimate_std_dev = 
np.std(classification_error_estimate_array)



    def __repr__(self):
        return ("____________________Experiment____________________"
                "\nScenario: {} | Standard Deviation: {} | n: {}"
                "\nExcess Risk Estimate Mean = {} "
                "\nExcess Risk Estimate Std Dev = {} "
                "\nExcess Risk Estimate Std Min = {} "
                "\nExpected Excess Risk = {} "
                "\nClassification Error Estimate Mean = {}"
                "\nClassification Error Estimate Std Dev = {}")\
            .format(self.scenario, self.std_dev, self.n, self.excess_risk_estimate_mean, 
self.excess_risk_estimate_std_dev, 
self.excess_risk_estimate_min,self.expected_excess_risk, self. 
classification_error_estimate_mean, self. classification_error_estimate_std_dev)

    def __str__(self):
        return repr(self)



"""
    Single ExperimentSet. Represents a set of 4 experiments with the same scenario and 
standard
    deviation, where each experiment has a unique n from [50, 100, 500, 1000]. When 
ExperimentSet
    is run, the the 4 experiments are run, and the risk + classification error 
statistics are
    calculated and printed.
"""
class ExperimentSet:

    N_ARRAY = np.array([50, 100, 500, 1000])

    """
        Constructor
        Parameters:
            scenario : experiment scenario {1,2}
            std_dev : distribution standard deviation
    """
    def __init__(self, scenario, std_dev, lr, data_file, target_file):
        self.scenario = scenario
        self.std_dev = std_dev
        self.experiment_array = [Experiment(self.scenario, self.std_dev, n) for n in 
self.N_ARRAY]
        self.expected_risk_estimate_array = np.zeros(4)
        self.expected_risk_estimate_std_dev_array = np.zeros(4)
        self.expected_classification_error_estimate_array = np.zeros(4)
        self.expected_classification_error_estimate_std_dev_array = np.zeros(4)

        # Static parameters for Trial to use
        ExperimentSet.data_file = data_file
        ExperimentSet.target_file = target_file

    """
        Run ExperimentSet. Runs each experiment, collects and prints performance
metrics.
    """
    def run(self):
        print("\n\n*************************** Experiment Set
***************************")
        print("Scenario: {} | Standard Deviation: {}".format(self.scenario,
self.std_dev))

        # run experiments in set and collect performance metrics
        for i, experiment in enumerate(self.experiment_array):
            experiment.run()

            self.expected_risk_estimate_array[i] = experiment.expected_excess_risk
            self.expected_risk_estimate_std_dev_array[i] =
experiment.excess_risk_estimate_std_dev
            self.expected_classification_error_estimate_array[i] =
experiment.classification_error_estimate_mean
            self.expected_classification_error_estimate_std_dev_array[i] =
experiment.classification_error_estimate_std_dev

            print(experiment)

        print("**********************************************************************")


    """
        Creates and shows plots of the ExperimentSet.
        The plot consists of two subplots:
            1. Expected Risk Estimate
            2. Expected Classification Error Estimate
    """
    def plot(self):
        print("\nExit Plot Window To Continue...")

        # Create plot
        fig, (class_err_plot, risk_plt) = plt.subplots(1,2)
        fig.suptitle('Performance of SGD (Scenario: {},  σ = {})'.format(self.scenario,
self.std_dev))
        fig.set_size_inches(12.5, 8.5)
        fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.5,
hspace=None)

        # create risk subplot
        risk_plt_title = 'Expected Risk Estimate'
        x_label = 'n (training set size)'
        risk_plt_y_label = 'Expected Risk Estimate'
        create_subplot(fig,
                       risk_plt,
                       self.N_ARRAY,
                       self.expected_risk_estimate_array,
                       self.expected_risk_estimate_std_dev_array,
                       risk_plt_title,
                       x_label,
                       risk_plt_y_label)


        # create classification error subplot
        class_err_plt_title = 'Expected Classification Error Estimate'
        class_err_y_label = 'Expected Binary Classification Error'
        create_subplot(fig,
                       class_err_plot,
                       self.N_ARRAY,
                       self.expected_classification_error_estimate_array,
                       self.expected_classification_error_estimate_std_dev_array,
                       class_err_plt_title,
                       x_label,
                       class_err_y_label)

        plt.show()


"""
    Create and format error subplot
    Parameters:
        subplot : subplot on which to create the error plot
        x : np array of x data
        y : np array of y data
        y_err : np array of y error
        title : subplot title
        x_label : x axis label
        y_label : y axis label
"""
def create_subplot(figure, subplot, x, y, y_err, title, x_label, y_label):
    error_line_width = 0.85
    error_bar_cap_size = 5
    error_bar_cap_thickness = 0.85
    x_label_offset_factor = 85.0
    y_label_offset_factor = 85.0
    axis_label_padding = 10

    # create error subplot
    subplot.errorbar(x,
                     y,
                     yerr = y_err,
                     elinewidth = error_line_width,
                     capsize = error_bar_cap_size,
                     capthick = error_bar_cap_thickness,
                     marker='o',
                     color="black",
                     linestyle='None')

    # calculate label offset y
    subplot_range_min, subplot_range_max  = subplot.get_ylim()
    subplot_range = subplot_range_max - subplot_range_min
    subplot_y_label_offset = subplot_range/y_label_offset_factor

    # calculate label offset x
    subplot_domain_min, subplot_domain_max  = subplot.get_xlim()
    subplot_domain = subplot_domain_max - subplot_domain_min
    subplot_x_label_offset = subplot_domain/x_label_offset_factor

    # label data points
    l = 0
    pt_labels_alpha = []
    pt_labels_numeric = []
    for i, j, err in zip(x, y, y_err):
        i_rounded = round(i,0)
        j_rounded = round(j,4)
        err_rounded = round(err, 4)
        point_str = '({}, {:.4f}), σ = {:.4f}'.format(i_rounded, j_rounded, err_rounded)
        alpha_str = ascii_uppercase[l]

subplot.annotate(alpha_str,xy=(i+subplot_x_label_offset,j+subplot_y_label_offset))
        pt_labels_alpha.append(ascii_uppercase[l])
        pt_labels_numeric.append(point_str)
        l+=1

    # add legend
    Legend.update_default_handler_map({str : CustomTextHandler()})
    legend = subplot.legend(handles=pt_labels_alpha, labels=pt_labels_numeric)

    #align legend labels right
    renderer = figure.canvas.get_renderer()
    shift = max([text.get_window_extent(renderer).width for text in legend.get_texts()])
    for text in legend.get_texts():
        text.set_ha('right')
        text.set_position((shift,0))

    # add axis labels and title
    subplot.set_xlabel(x_label, labelpad = axis_label_padding)
    subplot.set_ylabel(y_label, labelpad = axis_label_padding)
    subplot.set_title(title)


"""
    Custom text handler class to print text in legend
"""
class CustomTextHandler(HandlerBase):
    def create_artists(self, legend, text, xdescent, ydescent,
                       width, height, fontsize, trans):
        text_artist = Text(width/2., height/2, text,
                           fontsize=fontsize, ha="center",
                           va="center", fontweight="bold")
        return [text_artist]
```

model.py

```python
import math
import random
import numpy as np
from data import *

"""
    Logistic regression classifier wrapper class.
    Need to train this in the main method.
"""
class LogisticRegressionClassifier(object):

    """
        Constructor
        For one-path SGD, the initial weight should be an all-zero vector
    """
    def __init__(self, init_weight):
        self.weight = init_weight
        self.final_weight = np.zeros(5)
        self.num_sample = 1
        self.training = True

    """
        Get current weight
    """
    def get_weight(self):
        if self.training:
            return self.weight
        else:
            return self.final_weight

    """
        Update weight by summing current weight with negative of the gradient of loss
function.
        In this project, the gradient of logistic loss
        Parameters:
            gradient : the direction of max increase rate
            scenario : weight domain and feature domain {1,2}
            lr : learning rate
    """
    def renew_weight(self, gradient, scenario: int, lr: float):
        # Update weight
        self.weight = self.weight - lr*gradient

        # Projection
        if scenario == 1:
            self.weight = project_scene1(self.weight)
        else:
            self.weight = project_scene2(self.weight)

        # Record weight
        self.final_weight += self.weight
        self.num_sample += 1

    """
```

```
        Signals that the training has stopped
    """
    def finish_train(self):
        self.training = False
        self.final_weight = self.final_weight / self.num_sample

    """
        Predict the label of a given augmented feature vector (probability > 0.5)
        Returns:
            Either 1 or -1
    """
    def classify(self, feature_vector):
        linear = feature_vector.dot(self.get_weight())
        sigmoid_value = 1/(1+math.e**(-1*linear))
        if sigmoid_value > 0.5:
            return 1
        else:
            return -1

    def linear_prediction(self, feature_vector):
        return feature_vector.dot(self.get_weight())

"""
    Calculate logistic loss value or logistic loss gradient
    Parameters:
        weight : weight vector for logistic regression classifier
        ex : a data example containing augmented feature vector and true label
        gradient : this function returns loss value or loss gradient or not
"""
def logistic_loss_or_gradient(weight, ex, gradient=False):
    x = ex.u
    y = ex.y
    linear_component = y * weight.dot(x)
    if not gradient:
        return np.log(1 + np.exp(-1 * linear_component))
    else:
        gradient = np.zeros(len(x))
        for i in range(len(x)):
            gradient[i] = (-1 * y * x[i] * np.exp(-1 * linear_component)) / (1 +
np.exp(-1 * linear_component))
        return gradient


"""
    Train logistic regression model with **one-path stochastic gradient descent**
    Parameters:
        dg : standard deviation of true data distribution
        scenario : experiment scenario {1,2}
        n : training set size
"""
def train_logistic_regression(std_dev: float, scenario: int, n: int) ->
LogisticRegressionClassifier:
    model = LogisticRegressionClassifier([0]*5)

    #print(f"\nStart training Logistic Regression model by SGD")
    #print(f"std_dev: {std_dev}, scenario: {scenario}, training size: {n}")
    # Initialize model
    dg = DataGenerator(std_dev)
    model = LogisticRegressionClassifier(np.zeros(5))

    # Calculate learning rate for each sample
    lr = get_lr(scenario,n)

    # Train the model with a fresh piece of data point from DataGenerator
    loss = 0
    for i in range(1,n):

        # Generate fresh data
        if scenario == 1:
            data = dg.data_scene1(1)
        elif scenario == 2:
            data = dg.data_scene2(1)

        # Calculate gradient and update weight
        gradient = logistic_loss_or_gradient(model.get_weight(), data, gradient=True)
        model.renew_weight(gradient, scenario, lr)

        # Display result loss value
        loss += logistic_loss_or_gradient(model.get_weight(), data, gradient=False)
        if (i+1) % 50 == 0:
            #print(f'Average loss from Data{i-49} to {i+1}: {loss/50}')
            loss = 0

    # Signal stop training
    model.finish_train()

    return model


"""
    Evaluate logistic regression model with test data
    Parameters:
        model: instance of LogisticRegressionClassifier to evaluate
        data_file : the filename of the test data feature vector file
        target_file : the filename of the test data label file
    Returns:
        Tuple (excess_risk_estimate, expected_classification_err)
"""
def evaluate_logistic_regression(model: LogisticRegressionClassifier, data_file: str,
target_file:str):
    # read data from test data files
    data, target = read_data(data_file, target_file)

    excess_risk_sum = 0.0
    classification_err_sum = 0.0
    for i, feature_vector in enumerate(data):
        label = target[i]

        excess_risk_sum += logistic_loss_or_gradient(model.get_weight(),
Example(feature_vector,label))
```

```
        prediction = model.linear_prediction(feature_vector)
        classification_err_sum += mismatching_signs(prediction, label)


    excess_risk_estimate = excess_risk_sum / len(target)
    expected_classification_err = classification_err_sum / len(target)

    return excess_risk_estimate, expected_classification_err

"""
    Returns 0.0 if signs of the input parameters match, 1.0 otherwise
    Parameters:
        a: first value being compared
        b: first value being compared
    Returns:
        0.0 if sign(a) == sign(b), 1.0 otherwise
"""
def mismatching_signs(a, b):
    if (a >= 0 and b >= 0) or (a < 0 and b < 0):
        return 0.0
    else:
        return 1.0
```

data.py

```
import random
import numpy as np
from typing import List


# Augmented data dimensionality is assumed to be 5 in all functions

"""
    Single data example, containing augmented feature vector and label
"""
class Example(object):
    """
        Constructor
        Parameters:
            u : augmented feature vector
            y : true label
    """
    def __init__(self, u, y):
        self.u = u
        self.y = y

    def __repr__(self):
        return "(" + str(self.u) + "," + str(self.y) + ")"

    def __str__(self):
        return repr(self)


class DataGenerator(object):

    """
        Constructor
        Parameters:
            sigma: standard deviation of data distribution
    """
    def __init__(self, sigma):
        self.sigma = sigma

    """
        Generate some fresh Examples from data distribution
        Parameters:
            num : number of fresh Examples to generate
        Returns:
            A list containing Example objects if num >= 1
    """
    def fresh_data(self, num: int) -> List[Example]:
        result = []
        for _ in range(num):
            # Generate a piece of fresh Example
            prob = random.random()
            if prob > 0.5:
                # label
                y = -1
                # feature vector
                mean = [-1/4] * 4
                cov = np.diag([self.sigma**2] * 4)
                u = np.random.multivariate_normal(mean, cov)
                u_augmented = np.append(u, 1)

                result.append(Example(u_augmented, y))
            else:
                # label
                y = 1
                # feature vector
                mean = [1/4] * 4
                cov = np.diag([self.sigma**2] * 4)
                u = np.random.multivariate_normal(mean, cov)
                u_augmented = np.append(u, 1)

                result.append(Example(u_augmented, y))
        return result

    """
        Generate some fresh Examples for scenario 1
        Parameters:
            num : number of fresh Examples to generate
        Returns:
            A single Example object if num = 1
            A list containing Example objects if num > 1
    """
    def data_scene1(self,num: int):
        data = self.fresh_data(num)

        if len(data) == 1:
            return Example(project_scene1(data[0].u), data[0].y)

        data_projected = []
        for ex in data:
            data_projected.append(Example(project_scene1(ex.u), ex.y))
        return data_projected


    """
        Generate some fresh Examples for scenario 2
        Parameters:
            num : number of fresh Examples to generate
        Returns:
            A single Example object if num = 1
            A list containing Example objects if num > 1
    """
    def data_scene2(self,num: int):
        data = self.fresh_data(num)

        if len(data) == 1:
            return Example(project_scene2(data[0].u), data[0].y)

        data_projected = []
        for ex in data:
            data_projected.append(Example(project_scene2(ex.u), ex.y))
        return data_projected
```

```
"""
    Generate a list of learning rates for given scenario and given size of training
sample
    Parameters:
        scenario : which scenario the data is for , {1,2}
        n : number of samples in training sample
"""
def get_lr(scenario: int, n: int):
    if scenario ==  1:
        M = 2 * np.sqrt(5)
        rho = np.sqrt(5)
    else:
        M = 2
        rho = np.sqrt(2)
    return M/(rho*np.sqrt(n))

"""
    Given a vector of length 5, return the projection of it on the domain set X of
scenario 1.
    The domain set X = [-1, 1]^4, i.e., X is the 4 dimensional hypercube with edge
length 2 centered around the origin.
    Parameters:
        ex : a vector of length 5 to be projected onto X of scenario 1
    Returns:
        the vector projected onto X set of scenario 1
"""
def project_scene1(feature_vector: List[float]) -> List[float]:
    u = feature_vector
    u[u > 1] = 1
    u[u < -1] = -1
    return u

"""
    Given a vector of length 5, return the projection of it on the domain set X of
scenario 2.
    The domain set X = {x ∈ R^(d-1) : ||x|| <= 1 }, i.e., X is the 4-dimensional unit
ball centered around the origin.
    Parameters:
        ex : a vector of length 5 to be projected onto X of scenario 2
    Returns:
        the vector projected onto X set of scenario 2
"""
def project_scene2(feature_vector: List[float]) -> List[float]:
    u = feature_vector[0:4]
    l2_norm = np.linalg.norm(u)

    if l2_norm > 1:
        u = np.divide(u, l2_norm)
    u = np.append(u, feature_vector[4])
    return u

"""
    Renew all the test data
    Parameters:
        args : contains all file name information
        std_devs : contain 2 standard deviation values
"""
def renew_test_files(files, std_devs, N):
    for i in range(len(files)):
        for j in range(len(std_devs)):
            write_data(files[i][j][0], files[i][j][1], i+1, std_devs[j], N)

"""
    Write testing data to given file name
    Parameters:
        file_name : the file to write into
        scenario : which scenario the data is for , {1,2}
        std_dev : standard deviation of the data distribution
        num_data : size of test set needed
"""
def write_data(data_file: str, target_file:str, scenario: int, std_dev: float, num_data: 
int):
    # Generate data
    dg = DataGenerator(std_dev)
    if scenario == 1:
        datas = dg.data_scene1(num_data)
    else:
        datas = dg.data_scene2(num_data)

    # Write feature vectors
    o = open(data_file, 'w',encoding='utf-8')
    for data in datas:
        str_u = [str(feature) for feature in data.u]
        o.write(" ".join(str_u) + "\n")
    o.close()

    # Write labels
    o = open(target_file, 'w',encoding='utf-8')
    for data in datas:
        o.write(str(data.y) + "\n")
    o.close()


"""
    Read testing data (feature vectors + labels) from a given filename
    Parameters:
        data_file : the filename of the feature vector file
        target_file : the filename of the label file
    Returns:
        Tuple of numpy arrays (data, target)
"""
def read_data(data_file: str, target_file:str):
    data = read_file(data_file)
    target = read_file(target_file)
    return data, target


"""
    Read data from given file name.
    :file_name: the file name to read from.
    :return: an numpy array containing all the data from file.
```

```
"""
def read_file(file_name) -> np.ndarray:
    return np.genfromtxt(file_name, delimiter=' ')


########## For Testing Purpose ###########
"""
    Check if a given data example fits scenario 1
"""
def in_scene1(ex: Example) -> bool:
    feature_vector = ex.u

    if len(feature_vector) != 5:
        return False

    for feature in feature_vector:
        if feature > 1 or feature < -1:
            return False
    return True

"""
    Check if a given data example fits scenario 2
"""
def in_scene2(ex: Example) -> bool:
    feature_vector = ex.u

    if len(feature_vector) != 5:
        return False

    # Allow some inaccuracy for float type
    if np.linalg.norm(feature_vector[0:4]) - 1 > 0.001:
        return False

    return True

def test(num_data):
    print(f"\nTesting data_scene1 and data scene2 with {num_data} data points")

    dg = DataGenerator(0.35) # This value shouldn't matter
    data1 = dg.data_scene1(num_data)
    data2 = dg.data_scene2(num_data)

    error_found = False
    for data in data1:
        if not in_scene1(data):
            print("Error in data_scene1:", data)
            error_found = True
    if not error_found:
        print("data_scene1 correct!")

    error_found = False
    for data in data2:
        if not in_scene2(data):
            print("Error in data_scene2:", data)
            error_found = True
    if not error_found:
        print("data_scene2 correct!")

if __name__ == "__main__":
    test(10000)
```