CSCI567: Machine Learning	USC, Fall 2022
Homework 2	
Instructor: Vatsal Sharan	*Due: September 28 by 2:00 pm PST*

We would like to thank previous 567 staff, and Gregory Valiant (Stanford) for kindly sharing many of the problems with us.

A reminder on collaboration policy and academic integrity: Our goal is to maintain an optimal learning environment. You can discuss the homework problems at a high level with other groups, but you should not look at any other group's solutions. Trying to find solutions online or from any other sources for any homework or project is prohibited, will result in zero grade and will be reported. To prevent any future plagiarism, uploading any material from the course (your solutions, quizzes etc.) on the internet is prohibited, and any violations will also be reported. Please be considerate, and help us help everyone get the best out of this course.

Please remember the Student Conduct Code (Section 11.00 of the USC Student Guidebook). General principles of academic honesty include the concept of respect for the intellectual property of others, the expectation that individual work will be submitted unless otherwise allowed by an instructor, and the obligations both to protect one's own academic work from misuse by others as well as to avoid using another's work as one's own. All students are expected to understand and abide by these principles. Students will be referred to the Office of Student Judicial Affairs and Community Standards for further review, should there be any suspicion of academic dishonesty.

Total points: 75 points

Notes on notation:

- Unless stated otherwise, scalars are denoted by small letter in normal font, vectors are denoted by small letters in bold font and matrices are denoted by capital letters in bold font.
- $||.||$ means L2-norm unless specified otherwise i.e. $||.|| = ||.||_2$

Instructions

We recommend that you use LaTeX to write up your homework solution. However, you can also scan handwritten notes. We will announce detailed submission instructions later.

Theory-based Questions

Problem 1: Support Vector Machines (19pts) (Partial Solutions)

Consider a dataset consisting of points in the form of (x, y), where x is a real value, and $y \in \{-1,\ 1\}$ is the class label. There are only three points $(x_1, y_1) = (-1,\ -1)$, $(x_2, y_2) = (1, -1)$, and $(x_3, y_3) = (0,\ 1)$, shown in Figure 1.

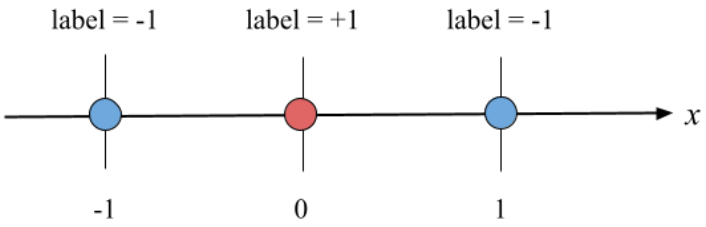


Figure 1: Three data points considered in Problem 1

1.1 (2pts) Can these three points in their current one-dimensional feature space be perfectly separated with a linear classifier? Why or why not?

No. A one-dimensional linear model $\text{sgn}(wx + b)$ is equivalent to a simple threshold function of the form

$$f(x) = \begin{cases} +1 & \text{if } x \geq \theta \\ -1 & \text{else} \end{cases} \quad \text{or} \quad f(x) = \begin{cases} -1 & \text{if } x \geq \theta \\ +1 & \text{else} \end{cases}$$

for some threshold $\theta \in \mathbb{R}$. It is thus clear that if we want points x_1 and x_2 to be correctly classified, then x_3 must be incorrectly classified. (Other correct reasoning gets full points as well.) (2 points)

1.2 (3pts) Now we define a simple feature mapping $\phi(x) = [x, x^2]^T$ to transform the three points from one-dimensional to two-dimensional feature space. Plot the transformed points in the new two-dimensional feature space. Is there a linear model $\mathbf{w}^T\mathbf{x} + b$ for some $\mathbf{w} \in \mathbb{R}^2$ and $b \in \mathbb{R}$ that can correctly separate the three points in this new feature space? Why or why not?

See Figure 2 for the plot. (1 points)
Yes. For example, any horizontal line with an intercept between 0 and 1 can correctly separate the data, which corresponds to $\mathbf{w} = (0, 1)$ and any $b \in (-1, 0)$. (It is enough to give one correct example.) (2 points)

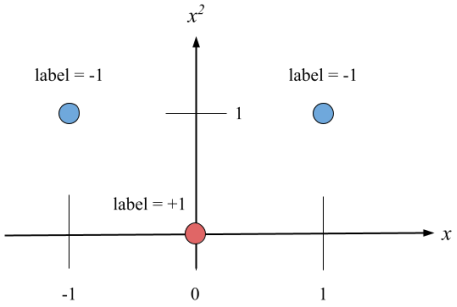


Figure 2: Plot for Q1.2: data points in new 2D space

1.3 (2pts) Given the feature mapping $\phi(x) = [x, x^2]^T$, write down the 3×3 kernel/Gram matrix $\mathbf{K}$ for this dataset.

The kernel function is $k(x, x') = \phi(x)^T\phi(x) = xx' + (xx')^2$, so the Gram matrix is $\mathbf{K} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

1.4 (4pts) Now write down the primal and dual formulations of SVM for this dataset in the two-dimensional feature space. Note that when the data is separable, we set the hyperparameter C to be $+\infty$ which makes sure that all slack variables (ξ) in the primal formulation have to be 0 (and thus can be removed from the optimization).

General primal formulation of SVM for separable data is:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||_2^2 \\ \text{s.t.} \quad & y_n[\mathbf{w}^T\phi(\mathbf{x}_n) + b] \geq 1, \ \forall n \end{aligned}$$

Plugging in the specific dataset gives: (2 points)

$$\begin{aligned} \min_{w_1,w_2,b} \quad & \frac{1}{2}(w_1^2 + w_2^2) \\ \text{s.t.} \quad & w_1 - w_2 - b \geq 1 \\ & w_1 + w_2 + b \leq -1 \\ & b \geq 1 \end{aligned}$$

General dual formulation of SVM for separable data is:

$$\begin{aligned} \max_{\alpha} \quad & \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n k(\mathbf{x}_m, \mathbf{x}_n) \\ \text{s.t.} \quad & \alpha_n \geq 0, \ \forall n \\ & \sum_n \alpha_n y_n = 0 \end{aligned}$$

Plugging in the specific dataset gives: (2 points)

$$\begin{aligned} \max_{\alpha_1,\alpha_2,\alpha_3 \geq 0} \quad & \alpha_1 + \alpha_2 + \alpha_3 - \alpha_1^2 - \alpha_2^2 \\ \text{s.t.} \quad & \alpha_1 + \alpha_2 = \alpha_3 \end{aligned}$$

- Okay to write the solution directly without first writing down the general form.
- For each formulation, 1 point for the objective and 1 point for the constraints.
- Do not deduct points if the mistake is solely due to the incorrect Gram matrix from the last question.

1.5 (5pts) Next, solve the dual formulation exactly (note: while this is not generally feasible, the simple form of this dataset makes it possible). Based on that, calculate the primal solution.

Eliminating the dependence on α_3 using the constraint $\alpha_1 + \alpha_2 = \alpha_3$, we arrive at the objective

$$\max_{\alpha_1,\alpha_2 \geq 0} 2\alpha_1 - \alpha_1^2 + 2\alpha_2 - \alpha_2^2.$$

Clearly we can maximize over α_1 and α_2 separately, which gives $\alpha_1^* = \alpha_2^* = 1$ and thus $\alpha_3^* = 2$. (3 points)

The primal solution can be found by

$$(w_1^*, w_2^*)^T = \sum_{n=1}^{3} y_n \alpha_n^* \phi(x_n) = (0, -2)^T, \quad \text{(1 point)}$$

$$b^* = y_1 - \mathbf{w}^{*T}\phi(x_1) = 1. \quad \text{(1 point)}$$

(Note that to calculate b^*, one can use any of the three examples, since they all satisfy $0 < \alpha_n < C = +\infty$.)

1.6 (3pts) Plot the decision boundary (which is a line) of the linear model ${\mathbf{w}^*}^T\mathbf{x} + b^*$ in the two-dimensional feature space, where $\mathbf{w}^*$ and b^* are the primal solution you got from the previous question. Then circle all support vectors. Finally, plot the corresponding decision boundary in the original one-dimensional space (recall that the decision boundary is just the set of all points x such that ${\mathbf{w}^*}^T\phi(x) + b^* = 0$).

The decision boundary for the two-dimensional space is a horizontal line with intercept $1/2$. All three training points are support vectors. (2 points)

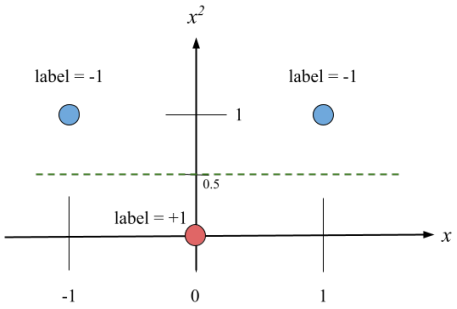


Figure 3: Plot for Q1.6: decision boundary in 2D space

The decision boundary for the one-dimensional space consists of two points $\frac{\sqrt{2}}{2}$ and $-\frac{\sqrt{2}}{2}$ (obtained by solving ${\mathbf{w}^*}^T\phi(\mathbf{x}) + b^* = -2x^2 + 1 = 0$). (1 points)

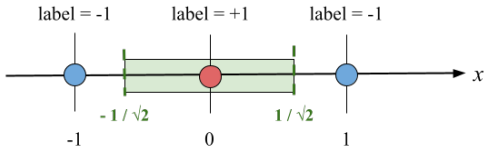


Figure 4: Plot for Q1.6: decision boundary in 1D space

Similarly, for Q1.5 and Q1.6, do not deduct points for mistakes inherited from previous questions.

Problem 2: Kernel Composition (6pts)

Prove that if $k_1, k_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are both kernel functions, then $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ is a kernel function too. Specifically, suppose that $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are the corresponding mappings for k_1 and k_2 respectively. Construct the mapping $\boldsymbol{\phi}$ that certifies k being a kernel function.

Let M_1 and M_2 be the dimension of the output of ϕ_1 and ϕ_2 respectively. Then we have

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{M_1} \phi_1(\mathbf{x})_i \phi_1(\mathbf{x}')_i\right)\left(\sum_{j=1}^{M_2} \phi_2(\mathbf{x})_j \phi_2(\mathbf{x}')_j\right) \quad \text{(2 points)}$$

$$= \sum_{i=1}^{M_1}\sum_{j=1}^{M_2} \phi_1(\mathbf{x})_i \phi_1(\mathbf{x}')_i \phi_2(\mathbf{x})_j \phi_2(\mathbf{x}')_j = \sum_{i=1}^{M_1}\sum_{j=1}^{M_2} \left(\phi_1(\mathbf{x})_i \phi_2(\mathbf{x})_j\right)\left(\phi_1(\mathbf{x}')_i \phi_2(\mathbf{x}')_j\right) \quad \text{(2 points)}$$

Therefore, $\phi(\mathbf{x}) = \phi_1(\mathbf{x})\phi_2(\mathbf{x})^T$. Writing it as an M_1M_2-dimensional vector is acceptable. (2 points)

Programming-based Questions

A reminder of the instructions for the programming part. To solve the programming based questions, you need to first set up the coding environment. We use python3 (version ≥ 3.7) in our programming-based questions. There are multiple ways you can install python3, for example:

- You can use conda to configure a python3 environment for all programming assignments.
- Alternatively, you can also use virtualenv to configure a python3 environment for all programming assignments

After you have a python3 environment, you will need to install the following python packages:

- numpy
- matplotlib (for you plotting figures)
- scikit-learn (only for Problem 5)

Note: You are **not allowed** to use other packages, such as *tensorflow, pytorch, keras, scipy*, etc. to help you implement the algorithms you learned. If you have other package requests, please ask first before using them. Note that you will be using *scikit-learn* in the provided code for Problem 5; but you are **not allowed** to use *scikit-learn* for the remaining problems.

Problem 3: Regularization (31pts)

This problem is a continuation of the linear regression problem from the previous homework (HW1 Problem 5). Let us recall the setup. Given d-dimensional input data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ with real-valued labels $y_1, \ldots, y_n \in \mathbb{R}$, the goal is to find the coefficient vector $\mathbf{w}$ that minimizes the sum of the squared errors. The total squared error of $\mathbf{w}$ can be written as $f(\mathbf{w}) = \sum_{i=1}^{n} f_i(\mathbf{w})$, where $f_i(\mathbf{w}) = (\mathbf{w}^\mathbf{T}\mathbf{x_i} - y_i)^2$ denotes the squared error of the ith data point. We will refer to $f(\mathbf{w})$ as the *objective function* for the problem.

Last homework, we considered the scenario where the number of data points was much larger than the number of dimensions and hence we did not worry too much about generalization. (If you remember, the gap between training and test accuracies in 5.1 of the last HW was not too large). We will now consider the setting where $d = n$, and examine the test error along with the training error. Use the following Python code for generating the training data and test data.

```
import numpy as np

train_n = 100
test_n = 1000
d = 100

X_train = np.random.normal(0,1, size=(train_n,d))
w_true = np.random.normal(0,1, size=(d,1))
y_train = X_train.dot(w_true) + np.random.normal(0,0.5,size=(train_n,1))

X_test = np.random.normal(0,1, size=(test_n,d))
y_test = X_test.dot(w_true) + np.random.normal(0,0.5,size=(test_n,1))
```

3.1 (2pts) We will first setup a baseline, by finding the test error of the linear regression solution $\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$ without any regularization. This is the closed-form solution for the minimizer of the objective function $f(\mathbf{w})$. (Note the formula is simpler than what we saw in the last homework because now $\mathbf{X}$ is square as $d = n$). Report the training error and test error of this approach, *averaged over 10 trials*. For better interpretability, report the normalized error $\hat{f}(\mathbf{w})$ rather than the value of the objective function $f(\mathbf{w})$, where we define $\hat{f}(\mathbf{w})$ as

$$\hat{f}(\mathbf{w}) = \frac{\| \mathbf{X}\mathbf{w} - \mathbf{y} \|_2}{\| \mathbf{y} \|_2}.$$

Note on averaging over multiple trials: We're doing this to get a better estimate of the performance of the algorithm. To do this, simply run the entire process (including data generation) 10 times, and find the average value of

$\hat{f}(\mathbf{w})$ over these 10 trials.

Normalized train error (linalg soln): 4.66310e-14 (more generally in range magnitude 1e-13 to 1e-14) (1 point)
Normalized test error (linalg soln): 3.30159e+00 (more generally in range of 7 to 0.5) (1 point)

3.2 (7pts) We will now examine ℓ_2 regularization as a means to prevent overfitting. The ℓ_2 regularized objective function is given by the following expression:

$$\sum_{i=1}^{m}(\mathbf{w}^{\mathbf{T}}\mathbf{x_i} - y_i)^2 + \lambda\|\mathbf{w}\|_2^2.$$

As discussed in class, this has a closed-form solution $\mathbf{w} = (\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{y}$. Using this closed-form solution, present a plot of the normalized training error and normalized test error $\hat{f}(\mathbf{w})$ for $\lambda = \{0.0005, 0.005, 0.05, 0.5, 5, 50, 500\}$. As before, you should average over 10 trials. Discuss the characteristics of your plot, and also compare it to your answer to (**3.1**).

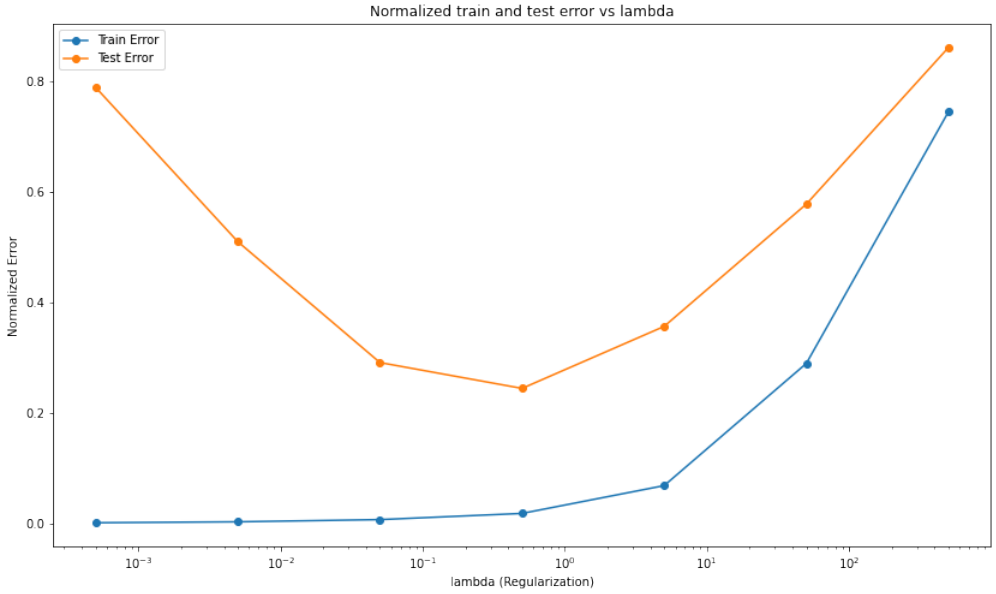


Figure 5: Plot of the normalized training error and normalized test error $\hat{f}(\mathbf{w})$ for varying values of regularization coefficient λ

λ	0.0005	0.005	0.05	0.5	5	50	500
Normalized Train Error	9.39e-4	2.65e-3	6.65e-3	1.78e-2	6.82e-2	2.89e-1	7.45e-1
Normalized Test Error	0.79	0.51	0.29	0.24	0.36	0.58	0.86

We see a U-shaped curve where small values of lambda lead to over-fitting (train error is disproportionately lower than test error) and large values lead to under-fitting (both train and test errors are high). There is an optimal value where both errors are minimized.

Train errors are all higher than the values reported in part (3.1).

The best test error (for λ = 0.5) is lower than the test error in part (3.1).

The following questions explore the concept of implicit regularization. This is a very active topic of research, with the idea being that optimization algorithms such as SGD can themselves act like regularizers (in the sense that they prefer the solutions to the regularized problems instead of just the original problem). There's no one correct answer we're

looking for in many of these questions, and idea is to make you think about what is happening and report your findings.

3.3 (7pts) Run stochastic gradient descent (SGD) on the original objective function $f(\mathbf{w})$, with the initial guess of $\mathbf{w}$ set to be the all 0's vector. Run SGD for 1,000,000 iterations for each different choice of the step size, $\{0.00005, 0.0005, 0.005\}$. Report the normalized training error and the normalized test error for each of these three settings, averaged over 10 trials. How does the SGD solution compare with the solutions obtained using ℓ_2 regularization? Note that SGD is minimizing the original objective function, which does *not* have any regularization. In Part (**3.1**) of this problem, we found the *optimal* solution to the original objective function with respect to the training data. How does the training and test error of the SGD solutions compare with those of the solution in (**3.1**)? Can you explain your observations? (It may be helpful to also compute the normalized training and test error corresponding to the true coefficient vector $\mathbf{w}^*$, for comparison.)

Step Size	5e-5	5e-4	5e-3
Normalized Train Error	1.53e-2	7.02e-3	5.48e-3
Normalized Test Error	0.238	0.237	0.446

The best test errors from the SGD solutions are comparable to the best test errors seen in part (3.2) [this was for the exact (linear algebra) solution to the regularized objective with lambda = 0.5]. Train errors are slightly lower than the train errors seen with the linear algebra solution.

The best test errors from the SGD solutions are better than the best test errors seen in part (3.1). The train errors are significantly higher than those observed in part (3.1).

It appears as though SGD (for appropriate step sizes) is implicitly optimizing the regularized objective.

3.4 (10pts) We will now examine the behavior of SGD in more detail. For step sizes $\{0.00005, 0.005\}$ and 1,000,000 iterations of SGD,

(i) Plot the normalized training error vs. the iteration number. On the plot of training error, draw a line parallel to the x-axis indicating the error $\hat{f}(\mathbf{w}^*)$ of the true model $\mathbf{w}^*$.

(ii) Plot the normalized test error vs. the iteration number. Your code might take a long time to run if you compute the test error after every SGD step—feel free to compute the test error every 100 iterations of SGD to make the plots.

(iii) Plot the ℓ_2 norm of the SGD solution vs. the iteration number.

Comment on the plots. What can you say about the generalization ability of SGD with different step sizes? Does the plot correspond to the intuition that a learning algorithm starts to overfit when the training error becomes too small, i.e. smaller than the noise level of the true model so that the model is fitting the noise in the data? How does the generalization ability of the final solution depend on the ℓ_2 norm of the final solution?

Plot 3.4(i) Fig 6 The training error quickly goes lower than the train error of the true model w_true. This means that the model starts fitting the noise in the data.

Plot 3.4(ii) Fig 7 The test error keeps decreasing with step size 5e-5. However, with step size 5e-3 the test error first decreases and then starts increasing showing that the model over-fits.

Plot 3.4(iii) Fig 8 The norm of the model weights increases quickly but starts to saturate with step size 5e-5. However, with step size 5e-3, the weight norm keeps increasing.

SGD takes longer to converge but generalizes better with smaller learning rates. With larger learning rates it is prone to over-fitting.

Over-fitting can be seen from the trend of training error when the training error drops below the noise level in the data (the train error with the true weight values). However, this does not seem to be true for all step sizes. For step size 5e-5, both train and test error keep decreasing throughout training.

The larger L2 norm of weight vectors is correlated with the tendency to overfit. Models generalize better when the weight norms are lower.

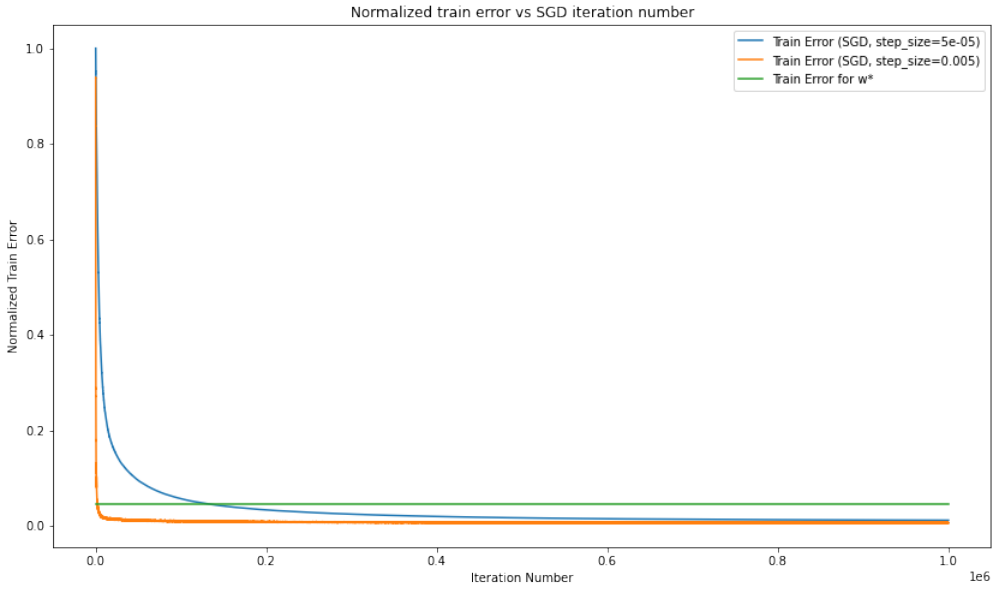


Figure 6: 3.4(i) Plot of the normalized training error of the SGD iterate $\mathbf{w}^{(t)}$ vs. the iteration number t

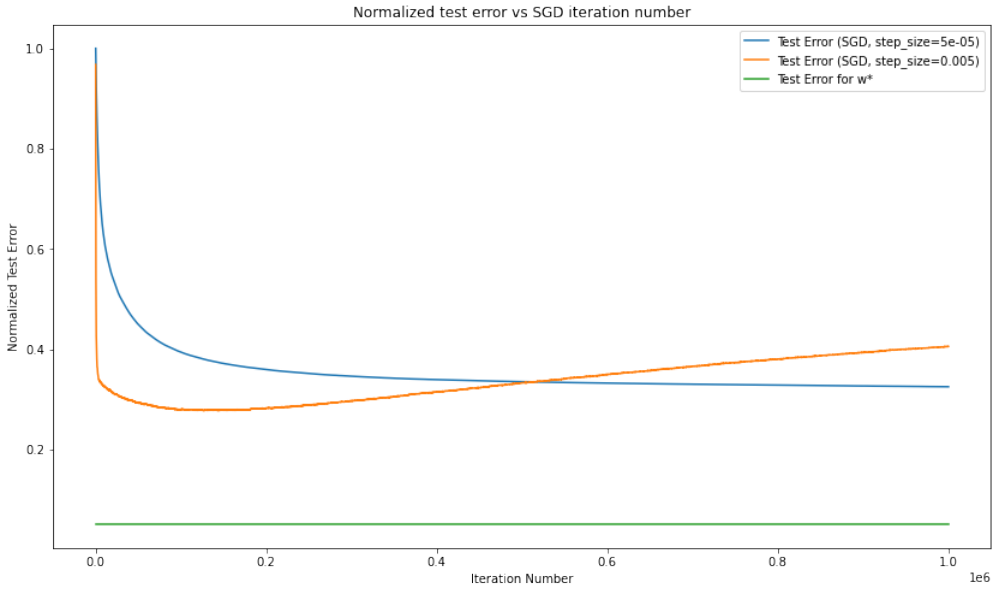


Figure 7: 3.4(ii) Plot of the normalized test error of the SGD iterate $\mathbf{w}^{(t)}$ vs. the iteration number t

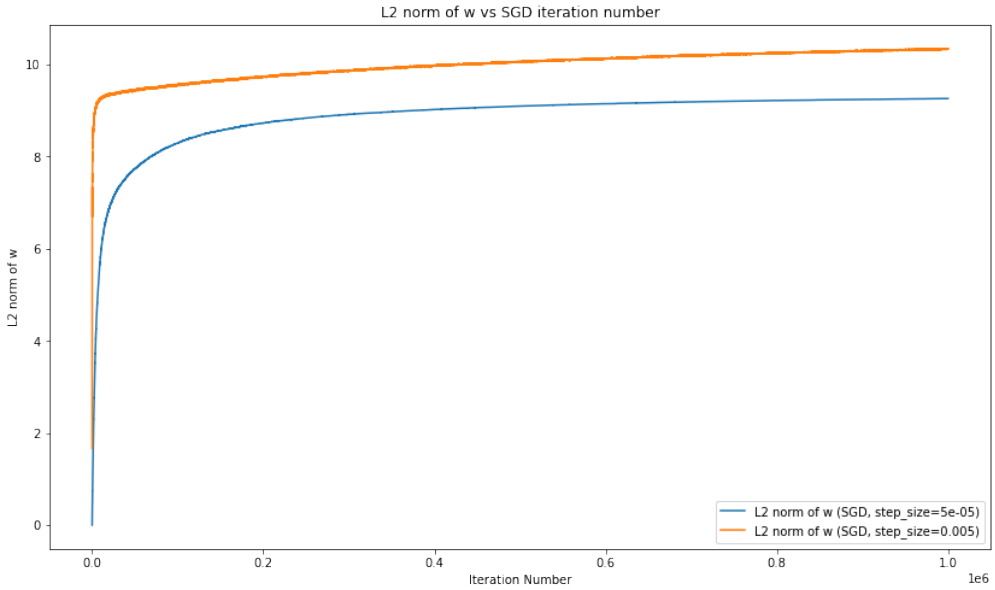


Figure 8: 3.4(iii) Plot of the ℓ_2 norm of the SGD iterate $\mathbf{w}^{(t)}$ vs. the iteration number t

3.5 (5pts) We will now examine the effect of the starting point on the SGD solution. Fixing the step size at 0.00005 and the maximum number of iterations at 1,000,000, choose the initial point randomly from the d-dimensional sphere with radius $r = \{0, 0.1, 0.5, 1, 10, 20, 30\}$ (to do this random initialization, you can sample from the standard Gaussian $N(0, \mathbf{I})$, and then renormalize the sampled point to have ℓ_2 norm r). Plot the average normalized training error and the average normalized test error over 10 trials vs r. Comment on the results, in relation to the results from part (**3.2**) where you explored different ℓ_2 regularization coefficients. Can you provide an explanation for the behavior seen in this plot?

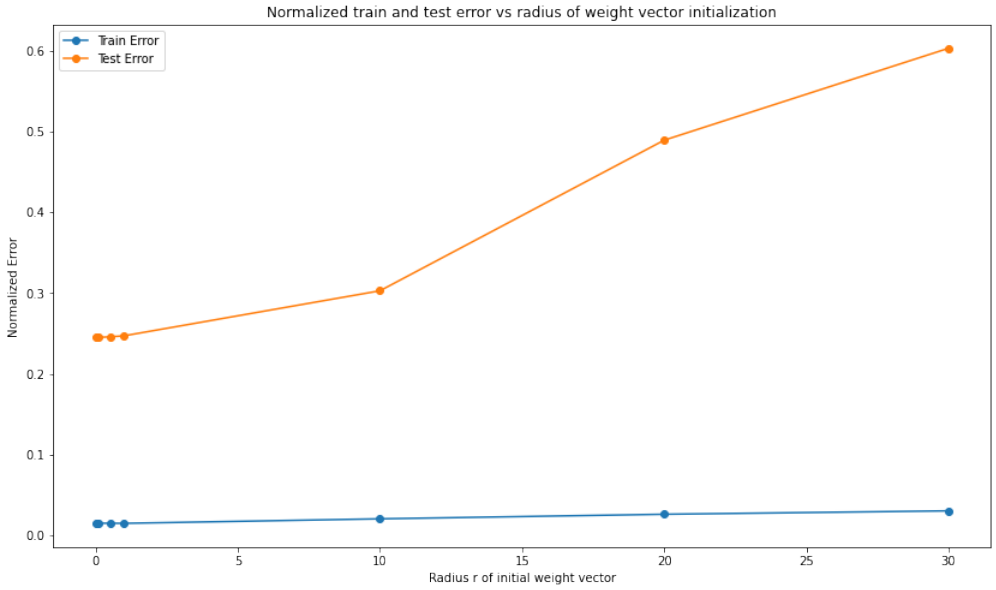


Figure 9: 3.5 Plot of the average normalized training error and the average normalized test error over 10 trials vs r

r	0	0.1	0.5	1	10	20	30
Normalized Train Error	1.49e-2	1.49e-2	1.49e-2	1.48e-2	2.04e-2	2.6e-2	3.03e-2
Normalized Test Error	0.245	0.245	0.246	0.247	0.303	0.489	0.603

We see that the train error is comparable for small initial radii and increases slightly with higher radius. However, test error for smaller radii 0, 0.1, 0.5, 1 are much lower than test errors for larger radii.

The larger initial radius means that the final solutions will also have larger norms. Whereas similar to observations in part 3.4, smaller initial radius means that final solutions also have a smaller norm and the model generalizes better.

The best SGD test errors for small initial weight norm are comparable to performing L2 regularization with the optimal value of λ. This is an indicator that SGD on the original objective with a small initialization norm implicitly performs L2 regularization.

Deliverables for Problem 3: Code for the previous parts as a separate Python file `Q3.py`. Training and test error for part 3.1. Plots for part 3.2, 3.4 and 3.5. Training and test error for different step sizes for part 3.3. Explanation for parts 3.2, 3.3, 3.4, 3.5.

Problem 4: Logistic Regression (11 pts)

In this problem we will consider a simple binary classification task. We are given d-dimensional input data $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ along with labels $y_1, \cdots, y_n \in \{-1, +1\}$. Our goal is to learn a linear classifier $\text{sign}(\mathbf{w}^T\mathbf{x})$ to classify the datapoints $\mathbf{x}$. We will find $\mathbf{w}$ by minimizing the logistic loss. The total logistic loss for any $\mathbf{w}$ can be written as $f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w})$, where $f_i(\mathbf{w}) = \log\left(1 + \exp(-y_i\mathbf{w}^\mathbf{T}\mathbf{x_i})\right)$ denotes the logistic loss of the ith data point $(\mathbf{x}_i, y_i)$. We will refer to $f(\mathbf{w})$ as the *objective function* for the problem.

Use the following Python code for generating the training data and test data. The data consists of points drawn from a Gaussian distribution with mean $[0.12, 0.12, ..., 0.12] \in \mathbb{R}^d$ for the class labelled as $+1$. Similarly, points are drawn from a Gaussian distribution with mean $[-0.12, -0.12, ..., -0.12] \in \mathbb{R}^d$ for the class labelled as -1. The data is then split such that 80% of the points are in the training data and the remaining 20% form the test data.

```
import numpy as np

np.random.seed(42)
d = 100 # dimensions of data
n = 1000 # number of data points
hf_train_sz = int(0.8 * n//2)

X_pos = np.random.normal(size=(n//2, d))
X_pos = X_pos + .12

X_neg = np.random.normal(size=(n//2, d))
X_neg = X_neg - .12

X_train = np.concatenate([X_pos[:hf_train_sz],
                          X_neg[:hf_train_sz]])
X_test = np.concatenate([X_pos[hf_train_sz:],
                         X_neg[hf_train_sz:]])

y_train = np.concatenate([np.ones(hf_train_sz),
                          -1 * np.ones(hf_train_sz)])
y_test = np.concatenate([np.ones(n//2 - hf_train_sz),
                         -1 * np.ones(n//2 - hf_train_sz)])
```

In this part you will implement and run *stochastic gradient descent* to solve for the value of $\mathbf{w}$ that approximately minimizes $f(\mathbf{w})$ over the training data. Recall that in stochastic gradient descent, you pick one training datapoint at random from the training data, say $(\mathbf{x}_i, y_i)$, and update your current value of $\mathbf{w}$ according to the gradient of $f_i(\mathbf{w})$. Run stochastic gradient descent for 5000 iterations using step sizes $\{0.0005, 0.005, 0.05\}$.

4.1 (7pts) As the algorithm proceeds, compute the value of the objective function on the train and test data at each iteration. Plot the objective function value on the training data vs. the iteration number for all 3 step sizes. On the same graph, plot the objective function value on the test data vs. the iteration number for all 3 step sizes. (The deliverable is a single graph with 6 lines and a suitable legend). How do the objective function values on the train and test data relate with each other for different step sizes? Comment in 3-4 sentences.

With step size 0.0005 the test error initially decreases and then increases rapidly. The train error also stops decreasing and then has a slight upward trend i.e. the model does not converge.

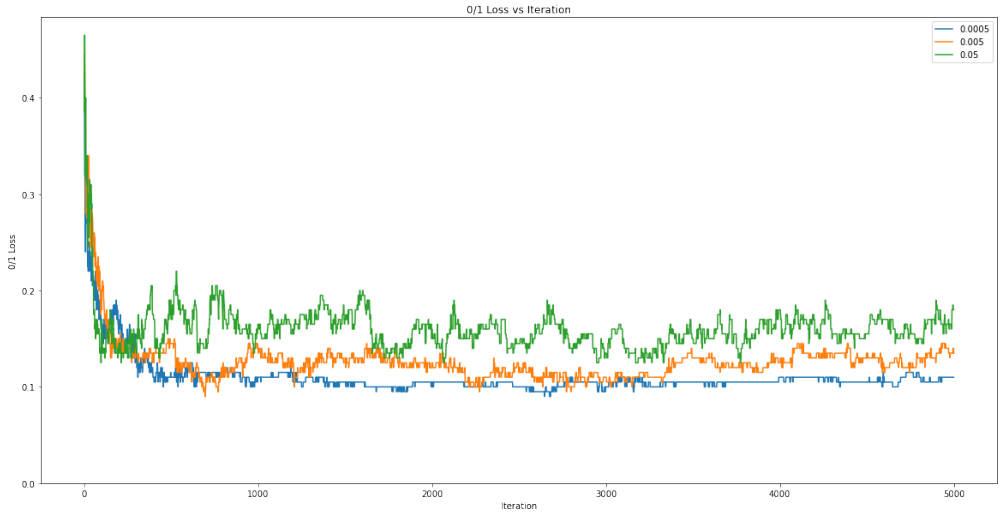
With step size 0.005, the model has the lowest train logistic error but we observe over-fitting i.e. test error stops decreasing while train error continues to decrease.

With step size 0.05 both train and test errors continue to decreases as the model converges in a stable manner.

- Award 1 point for a properly annotated graph with legend (1 point)
- Award 0.5 points for each matching trend line (3 points)
- Award 1 point for correct description of behavior for each step size (3 points)

4.2 (2pts) So far in the problem, we've minimized the logistic loss on the data. However, remember from class that our actual goal with binary classification is to minimize the classification error given by the 0/1 loss, and logistic loss is just a surrogate loss function we work with. We will examine the average 0-1 loss on the test data in this part (note that the average 0-1 loss on the test data is just the fraction of test datapoints that are classified incorrectly). As the SGD algorithm proceeds, plot the average 0-1 loss on the test data vs. the iteration number for all 3 step sizes on the same graph. Also report the step size that had the lowest final 0-1 loss on the test set and the corresponding value of the 0-1 loss.

Step size 0.0005 has the lowest 0/1 loss of 0.11.



- Correct plot (1.5 points)
- Correct value of best learning rate and comparable error (0.5 points)

4.3 (2pts) Comment on how well the logistic loss act as a surrogate for the 0-1 loss.

There is a discrepancy when using the surrogate loss to choose the best learning rate vs using the 0/1 loss. The surrogate loss is necessary for better optimization. However, we shouldn't perform model selection based on the surrogate loss values; the task loss (in this case classification error) is the better choice for selecting the best model.

Deliverables for Problem 4: Code for the previous parts as a separate Python file `Q4.py`. Plot (with 6 lines) and associated discussion for 4.1. Plot (with 3 lines) and associated discussion for 4.2. Discussion for 4.3.

Problem 5: Classifier Comparison (8 pts)

In this part, we will compare the behaviour of Logistic Regression to SVMs with Linear and Radial Basis Function (RBF) kernels. You can find the code on `https://vatsalsharan.github.io/fall22/cc_hw2.zip`.

Take some time to understand the code. Running the code will create 3 datasets: MOON, CIRCLES and LINEARLY_SEPARABLE, train 6 classifiers on each dataset, and generate a graph with 6×7 grid of scatter plots. The first column displays the data while the remaining columns display the learned decision boundary of 6 classifiers. The number in the bottom right of each plot shows the classifier accuracy. Odd rows correspond to the training data while even rows correspond to the test data for each dataset. For the following questions, explain your observations in 2-3 lines.

5.1 (2pts) Notice that MOON and CIRCLES are not linearly separable. Do linear classifiers do well on these? How does SVM with RBF kernel do on these? Comment on the difference.

We can see that the linear classifiers struggle to fit the first two datasets MOON and CIRCLES (which are not linearly separable). The RBF kernel allows SVMs to fit all the 3 datasets. Observe that the linear models perform better than an RBF-kernel SVM on the last LINEARLY_SEPARABLE dataset.

5.2 (2pts) Try various values of the penalty term C for the SVM with linear kernel. On the LINEARLY_SEPARABLE dataset, how does the train and test set accuracy relate to C? On the LINEARLY_SEPARABLE dataset, how does the decision boundary change?

The penalty term describes how many training errors are acceptable when fitting the data. For the linearly separable dataset, very small penalty (stronger regularization) learns a worse decision boundary.

5.3 (2pts) Try various values of the penalty term C for the SVM with RBF kernel. How does the train and test set accuracy relate to C? How does the decision boundary change?

The strength of the penalty term determines how well the decision boundary fits the datasets. The fit improves on the MOON and CIRCLE dataset as the penalty term is increased.

5.4 (2pts) Try various values of C for Logistic Regression (Note: C is the inverse regularization strength). Do you see any effect of regularization strength on Logistic Regression? Hint: Under what circumstances do you expect regularization to affect the behavior of a Logistic Regression classifier?

Due to the low dimensionality and since both dimensions are crucial to classification, regularization does not play a big role and effects are negligible.