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)

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.

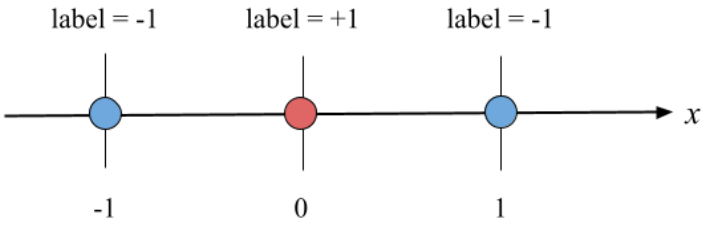


Figure 1: Three data points considered in Problem 1

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?

Answer:
They can't be separated by a linear classifier in 1-D space. Consider a decision boundary $x = c$, where we label point to +1 if $x > c$ and -1 if $x \leq c$. Then, to classify these three point, we have

$$c < 0, c \geq 1, c \geq -1$$

There is no valid c and it's also true if we label inversely, so it's impossible to separate them by a 1-D linear classifier.

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?

Answer:
There is a linear model, e.g. with $\mathbf{w} = [0, -2]^T$ and $b = 1$, which can correctly separate the three points as shown in Fig. 2.

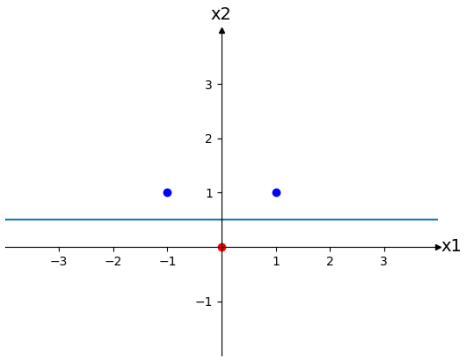


Figure 2: 2-D feature space

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.

Answer:

$$\begin{aligned}\mathbf{K} = \Phi\Phi^T &= \begin{bmatrix}-1 & 1\\ 1 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}-1 & 1 & 0\\ 1 & 1 & 0\end{bmatrix}\\ &= \begin{bmatrix}2 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 0\end{bmatrix}\end{aligned}$$

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).

Answer:
Because the data is separable, so we have
Primal:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \text{ s.t. } y_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i) + b) \geq 1,\ \forall i \in [3]$$

Dual:

$$\max_{\{\alpha_i\}} \sum_{i=1}^{3}\alpha_i - \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi}(\mathbf{x}_i)^T\boldsymbol{\phi}(\mathbf{x}_j) \text{ s.t. } \sum_{i=1}^{3}\alpha_i y_i = 0, \text{ and } \alpha_i \geq 0,\ \forall i \in [3]$$

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.

Answer:
According to the dual form, we have

$$\begin{gathered}-\alpha_1 - \alpha_2 + \alpha_3 = 0\\ \alpha_i \geq 0,\ \forall i \in [3]\\ \max_{\{\alpha_i\}}\ \alpha_1 + \alpha_2 + \alpha_3 - \alpha_1^2 - \alpha_2^2\end{gathered}$$

Replace α_3 with $\alpha_1 + \alpha_2$ in the last equation and utilize the inequality $\frac{\alpha_1+\alpha_2}{2} \leq \sqrt{\frac{\alpha_1^2+\alpha_2^2}{2}}$, we have

$$2(\alpha_1 + \alpha_2) - (\alpha_1^2 + \alpha_2^2) \leq 2(\alpha_1 + \alpha_2) - \frac{(\alpha_1 + \alpha_2)^2}{2}$$

Thus, to maximize this quadratic function, $\alpha_1 + \alpha_2 = 2$. Note that the equality happens when $\alpha_1 = \alpha_2$, so $\alpha_1 = \alpha_2 = 1$ and $\alpha_3 = 2$. Then, we have

$$\begin{aligned}\mathbf{w}^* &= \sum_{i=1}^{3}\alpha_i y_i \boldsymbol{\phi}(\mathbf{x}_i) = [0, -2]^T\\ b^* &= y_1 - \mathbf{w}^*\boldsymbol{\phi}(\mathbf{x}_1) = 1\end{aligned}$$

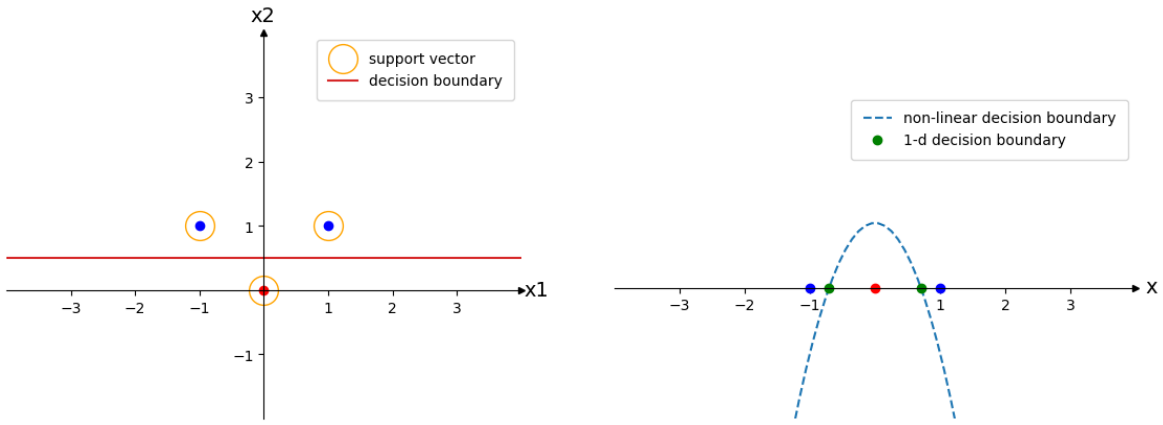
For primal form, we have

$$\begin{aligned}-(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_1) + b) &\geq 1\\ -(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_2) + b) &\geq 1\\ \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_3) + b) &\geq 1\\ \min_{\mathbf{w},b}\ \frac{1}{2}(w_1^2 + w_2^2)&\end{aligned}$$

From the first three inequality, we can derive $-w_1 + w_2 \leq -1 - b$, $w_1 + w_2 \leq -1 - b$ and $b \geq 1$. Thus, $w_2 \leq -1 - b \leq -2$, $w_1 = 0$. Because we want to minimize $\frac{1}{2}(w_1^2 + w_2^2)$, $b = 1$ and $w_2 = -2$. In conclusion, $\mathbf{w} = [0, -2]^T$ and $b = 1$.

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 ${\mathbf{w}^*}^T\phi(x) + b^* = 0$).

Answer:
The 2-D plot is in Fig. 3(a) and 1-D plot is shown in Fig. 3(b). For the 1-D plot, the intersection on the 1-D line has values of $-\sqrt{2}/2$ and $+\sqrt{2}/2$.



(a) 2-D feature space

(b) 1-D feature space

Figure 3: Decision Boundary

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.
Answer:
According to the problem, we have

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \\ &= \boldsymbol{\phi}_1(\mathbf{x})^T\boldsymbol{\phi}_1(\mathbf{x}')\boldsymbol{\phi}_2(\mathbf{x})^T\boldsymbol{\phi}_2(\mathbf{x}') \\ &= (\sum_i \boldsymbol{\phi}_{1,i}(\mathbf{x})\boldsymbol{\phi}_{1,i}(\mathbf{x}'))(\sum_j \boldsymbol{\phi}_{2,j}(\mathbf{x})\boldsymbol{\phi}_{2,j}(\mathbf{x}')) \\ &= \sum_i \sum_j (\boldsymbol{\phi}_{1,i}(\mathbf{x})\boldsymbol{\phi}_{2,j}(\mathbf{x}))(\boldsymbol{\phi}_{1,i}(\mathbf{x}')\boldsymbol{\phi}_{2,j}(\mathbf{x}')) \\ &= \boldsymbol{\phi}(\mathbf{x})\boldsymbol{\phi}(\mathbf{x}') \end{aligned}$$

,where $\boldsymbol{\phi}_{1,i}(\mathbf{x})$ is the i-th element of $\boldsymbol{\phi}_1(\mathbf{x})$ and $\boldsymbol{\phi}_{2,j}(\mathbf{x})$ is the j-th element of $\boldsymbol{\phi}_2(\mathbf{x})$, and

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\phi}_{1,1}(\mathbf{x})\boldsymbol{\phi}_{2,1}(\mathbf{x}) \\ \boldsymbol{\phi}_{1,1}(\mathbf{x})\boldsymbol{\phi}_{2,2}(\mathbf{x}) \\ \vdots \\ \boldsymbol{\phi}_{1,\mathcal{D}_1}(\mathbf{x})\boldsymbol{\phi}_{2,\mathcal{D}_2}(\mathbf{x})) \end{bmatrix}$$

, $\mathcal{D}_1$ is the dimension of $\boldsymbol{\phi}_1(\mathbf{x})$ and $\mathcal{D}_2$ is the dimension of $\boldsymbol{\phi}_2(\mathbf{x})$. Thus, $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^{\mathcal{D}_1\mathcal{D}_2}$, which has $\boldsymbol{\phi}_{(i,j)}(\mathbf{x}) = \boldsymbol{\phi}_{1,i}(\mathbf{x})\boldsymbol{\phi}_{2,j}(\mathbf{x})$ for each pair (i, j). Hence proved that $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ is a kernel function too, with a corresponding mappings $\boldsymbol{\phi}$.

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 $f_i(\mathbf{w}) = (\mathbf{w^T x_i} - y_i)^2$ 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 $\mathbf{w} = \mathbf{X^{-1}y}$ 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

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

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.

Answer:
The averaged normalized training error of closed-form solution is 1.1892900793080501e-13, and the averaged normalized test error of closed-form solution is 10.820324934464205.

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:

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

As discussed in class, this has a closed-form solution $\mathbf{w} = (\mathbf{X^T X} + \lambda\mathbf{I})^{-1}\mathbf{X^T y}$. 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**).

Answer:
The plot of averaged normalized training error and test error is shown in Fig. 4.

The training error is always increasing when λ becomes bigger, while the test error goes down first, reach a minima and then goes up. Because regularization always decreases the variance, the training error is always increasing. When λ is too small, the model is over-fitting, so the test error is large. As λ increases, the test error goes down and regularization helps the generalization. But when λ gets too large, the model becomes under-fitting, so test error goes up.

Compared to (**3.1**), regularization effectively reduces the test error. However, it also increases the training error. But with a good choice of λ (like $\lambda = 0.5$ in this case), we can have both good training error and test error.

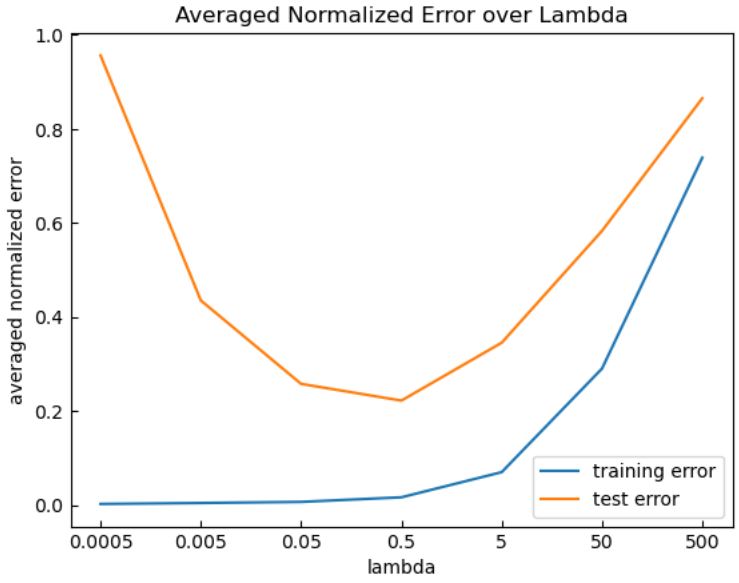


Figure 4: Averaged normalized error over λ

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.)

Answer:
The averaged normalized training error and test error of SGD with step size 0.00005, 0.0005, 0.005 are, respectively, 0.014118939139409953 and 0.22734056683425047, 0.006842851092035152 and 0.2487551459545382, and 0.006247209266946751 and 0.39664549601869215.

The training error and test error are comparable to the error of ℓ_2 regularization. Compared to the solution in (**3.1**), the training error increases and test error decreases. Actually, the averaged normalized training error and test error of the true model are 0.04969876282457394 and 0.049927427359123575. Thus, the solution in (**3.1**) is an over-fitting model, which is theoretically optimal on training set and will have a very small training error but a high test error. While the ℓ_2 regularization and implicit regularization of SGD prevent the over-fitting and reduce the test error.

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 $\mathbf{w}^*$.

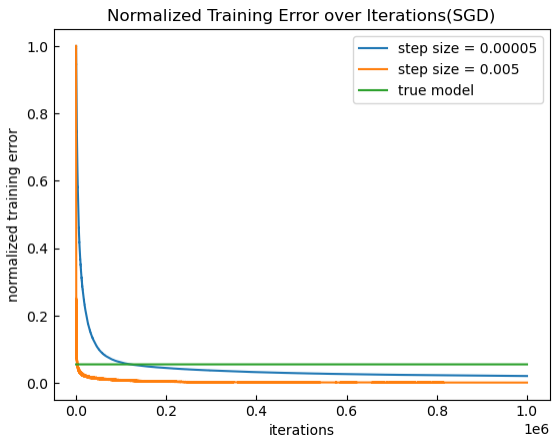
(ii) Plot the normalized test error vs. the iteration number. Your code might take a long time to run if you compute the test error after every SGD step—feel free to compute the test error every 100 iterations of SGD to make the plots.

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

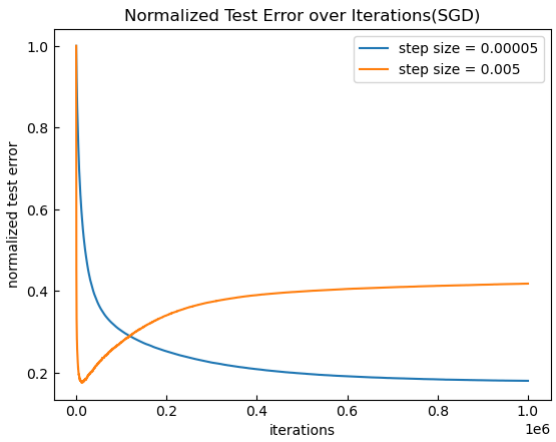
Comment on the plots. What can you say about the generalization ability of SGD with different step sizes? Does the plot correspond to the intuition that a learning algorithm starts to overfit when the training error becomes too small, i.e. smaller than the noise level of the true model so that the model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

Answer:
The plots are shown in Fig. 5(a), Fig. 5(b) and Fig. 6.

As the plots show, SGD with step size of 0.00005 has smaller test error. Thus, the generalization ability is better for smaller step size. For the step size 0.005, we found that it starts to overfit because the test error increases when the training error goes below the true model's training error. But for step size 0.00005, when the training error goes below the true model's training error, the test error is still decreasing. The generalization ability is better for the solution with a smaller ℓ_2 norm of $\mathbf{w}$.



(a) Normalized training error over iterations



(b) Normalized test error over iterations

Figure 5: Averaged normalized error over iterations

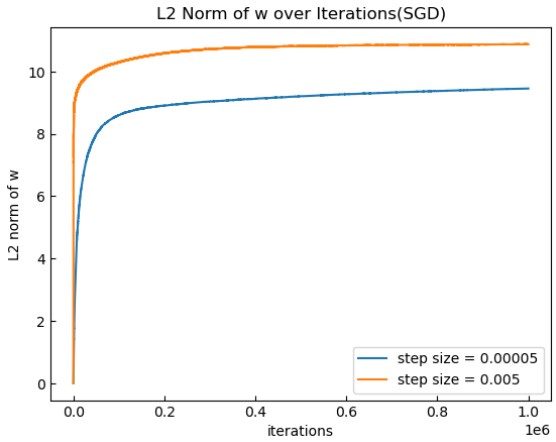


Figure 6: ℓ_2 norm of $\mathbf{w}^{(t)}$ over iterations

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?

Answer:
The plot is shown in Fig. 7. We see that as r increases, the training error slightly increases but test error increases a lot. Combined with part (**3.2**), the ℓ_2 regularization is to make $\mathbf{w}$ close to the origin, so starting point with a high ℓ_2 norm in SGD can be expected to have a similar effect as a small λ value in ℓ_2 regularization. In the plots, we can see that larger starting points more negatively affect the test error than the training error, though it eventually has a significantly negative effect on both. The negative effect is not as strong on the training error may caused by that larger starting points can lead to more quickly fitting (and over-fitting) on the training set.

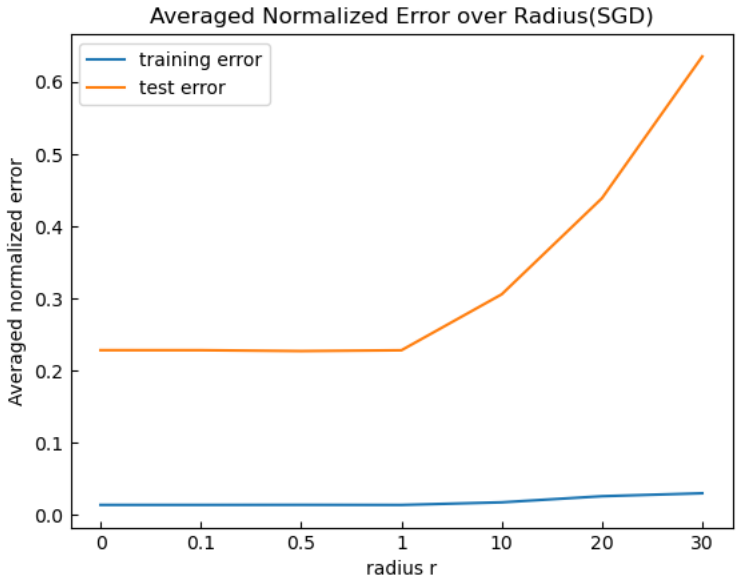


Figure 7: Averaged normalized error over iterations

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.

Answer:
The plot is shown in Fig. 8. For step size 0.0005, the curves of objective function values on the train and test data are almost overlapped. They decrease in the same speed as iterations increase. For step size 0.005, the objective function values on the train data decrease more quickly than that on the test data. For step size 0.05, the objective function values on the train data and test data will not converge. They oscillate or even diverge. Overall, the objective function values on the train data are smaller than that on the test data.

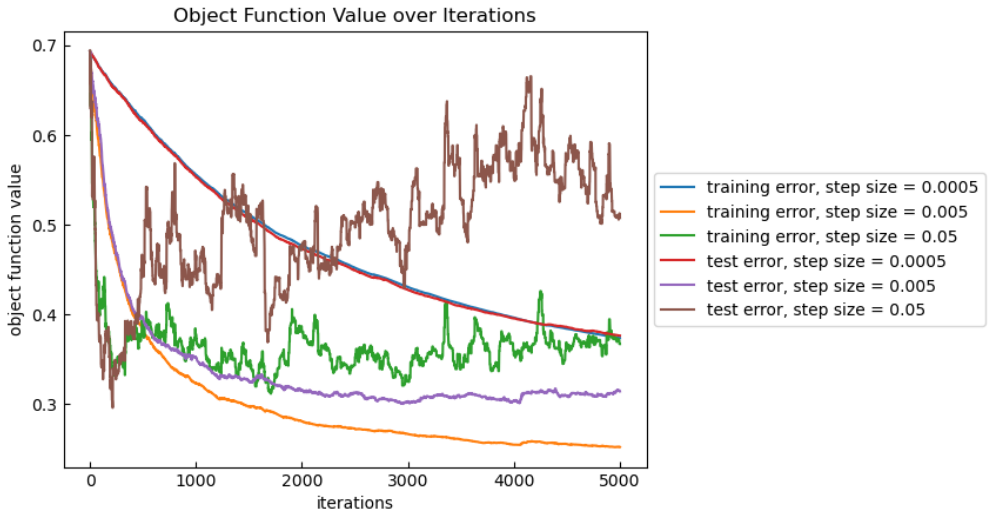


Figure 8: Objective function value over iterations

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.

Answer:
The plot is shown in Fig. 9. The step size 0.0005 has the lowest 0-1 loss on the test set of 0.11.

Figure 9: Objective function value over iterations

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

Answer:
Compared to 0-1 loss, logistic loss is convex and continuous, which could be optimized efficiently. Note that when the 0-1 loss reaches optimal minima for step size 0.0005, the logistic loss is still decreasing, which means that it is trying to push the different classes even further apart to improve its robustness. The logistic loss is able to learn more even when the 0-1 loss has already reached the optimal loss in some cases.

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.

Answer:
Linear classifiers do classify MOON dataset with a reasonable accuracy like a little higher than 0.80 for both the train and test accuracy. However, they classify badly on CIRCLE dataset with low accuracies like 0.57 and 0.40 on the train set and test set. Conversely, SVM with RBF kernel classify MOON and CIRCLE great enough with approximately 0.90 accuracy in average, though it has a relatively low accuracy on CIRCLE test set with $C = 0.25$. While linear classifiers fail to classify the CIRCLE dataset, the SVM with RBF kernel classifies that data well.

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?

Answer:

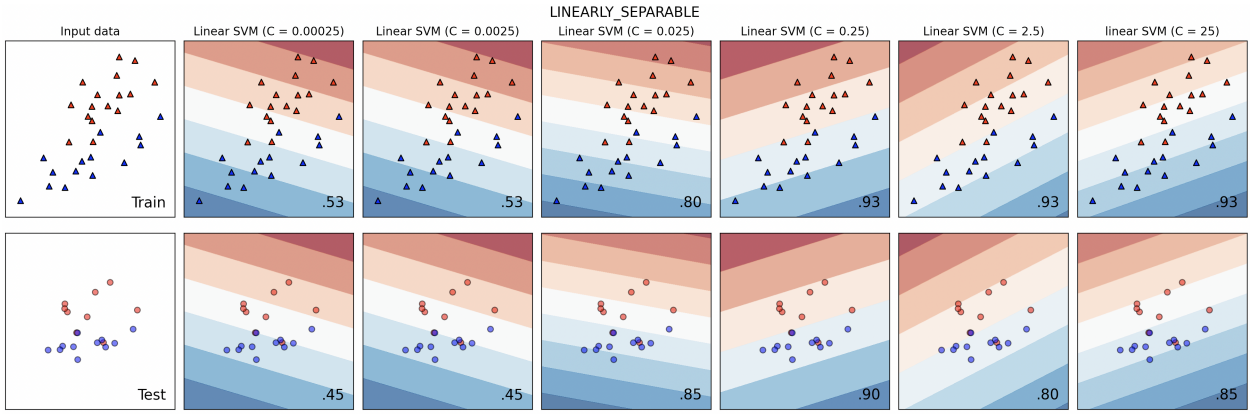


Figure 10: Linear SVM on LINEARLY SEPARABLE dataset for various values of C

As shown in Fig. 10, besides the initial C values, four more different C values are tested for this SVM with linear kernel. The C values are C = 0.00025, 0.0025, 0.025, 0.25, 2.5, and 25. On the LINEARLY SEPARABLE dataset, the train and test accuracy does closely related to the change in C. As C decreases from a large value, the training accuracy goes down, while the test accuracy goes up, reaches an optimal maxima, and then goes down. Because the strength of regularization is inversely proportional to C, the relationship is aligned well with the impact of increasing the strength of regularization. When C is large, the decision boundary tends to fit the training data well and have a big slope. As C decreases from a large value, the decision boundary begins to rotate clockwise and finally reaches an optimal state which is dominated by the regularization.

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?

Answer:

As shown in Fig. 11, besides the initial C values, four more different C values are tested for this SVM with RBF kernel. The C values are C = 0.0025, 0.025, 0.25, 1.0, 2.5, and 25. As C decreases from a large value, the training accuracy goes down, while the test accuracy goes up, reaches an optimal maxima, and then goes down. Because the strength of regularization is inversely proportional to C, the relationship is aligned well with the impact of increasing the strength of regularization. When C is large, the decision boundary tends to fit the training data well and becomes more complex. As C decreases from a large value, the decision boundary becomes less complex and smoother. Finally, it reaches an optimal state which is dominated by the regularization.

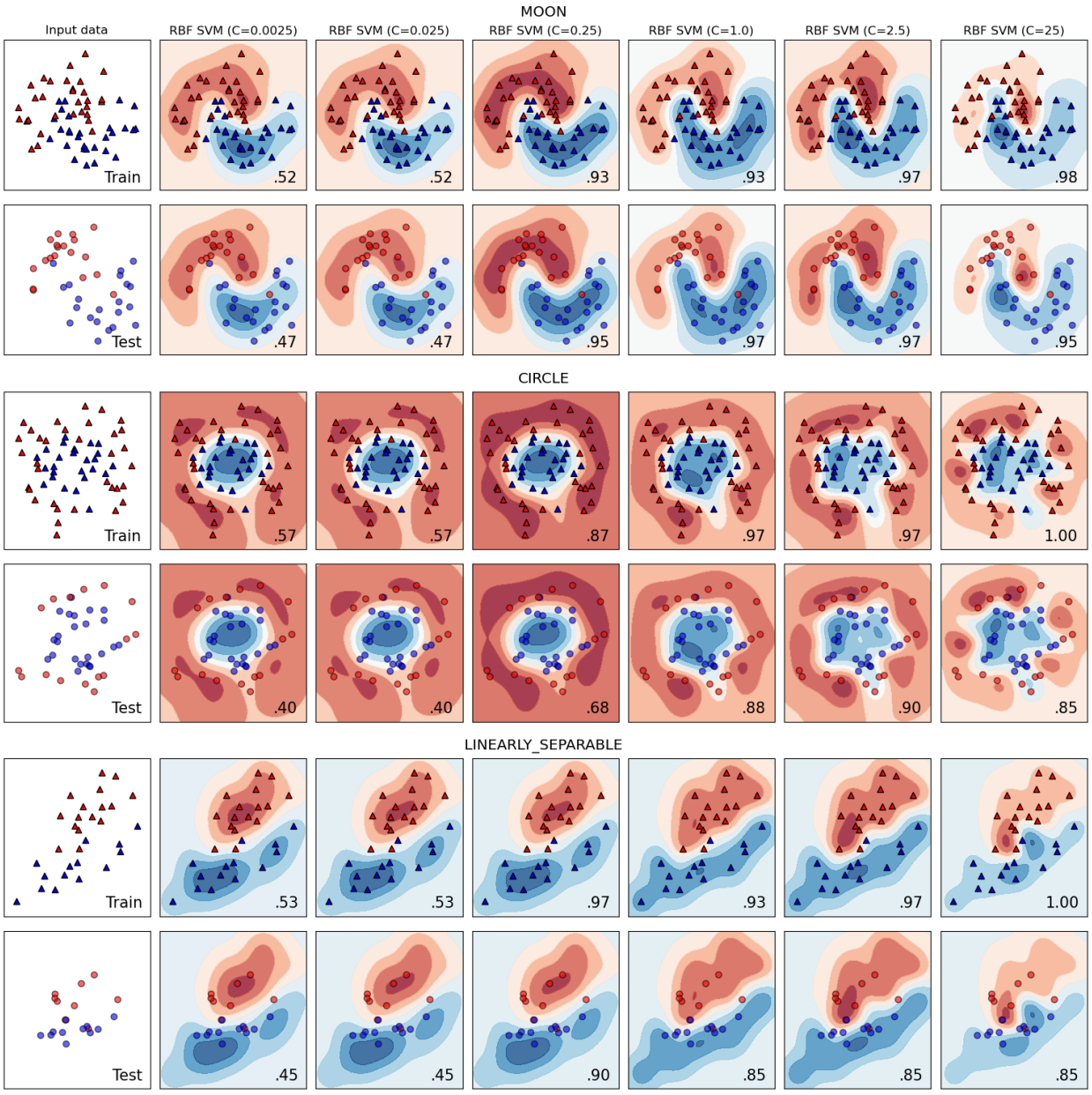


Figure 11: SVM with RBF kernel for various values of C = 0.0025, 0.025, 0.25, 1.0, 2.5, and 25

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?

Answer:
As shown in Fig. 12, various C (indicated by lambda) are tested. The regularization strength is proportional to lambda. As tested and plotted, we can only find the noticing effect of regularization strength on Logistic Regression with the LINEARLY SEPARABLE dataset. As lambda gets larger (C gets smaller), the train accuracy gets worse, while the test accuracy goes up first, reaches an optimal maxima, and then goes down. In conclusion, for logistic regression, when the dataset is not linearly separable, the loss is dominated by classification error, so the regularization term has small effects. For linearly separable dataset, the regularization strength affects the train and test accuracy as supposed.

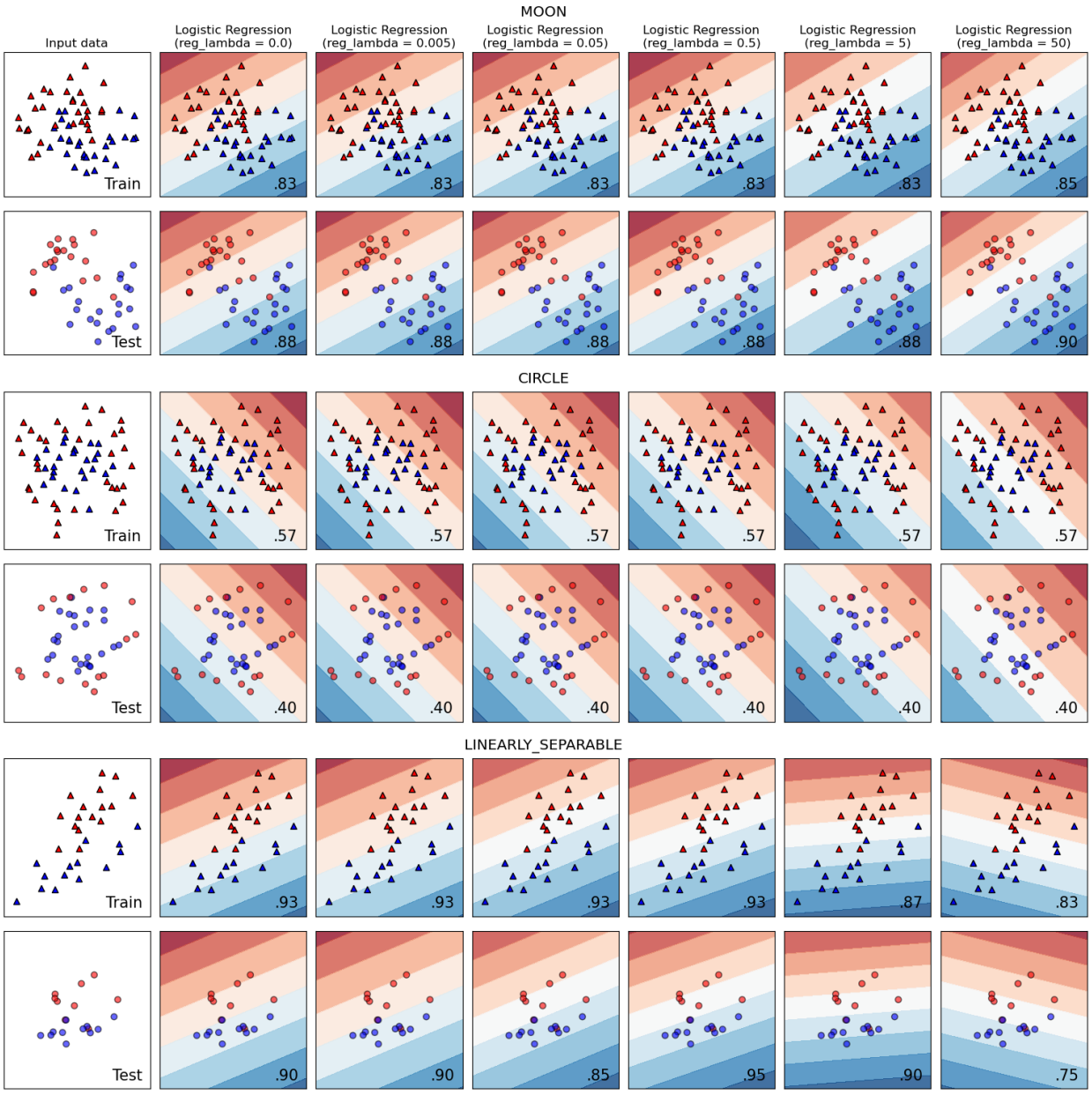


Figure 12: Logistic Regression for various values of C