

## Homework 2

Instructor: Vatsal Sharan

Due: February 25 by 11:59 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:** 64 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.
- $\|\cdot\|$  means L2-norm unless specified otherwise i.e.  $\|\cdot\| = \|\cdot\|_2$

## Instructions

We recommend that you use LaTeX to write up your homework solution. However, you can also scan handwritten notes. The detailed submission instructions are as follows:

1. We use Gradescope for homework submission.
2. There are 3 assignments on Gradescope: the first is for the HW report, the second is to submit code, and the third is for the bonus question.
  - The homework report should be a single PDF containing all written solutions to problems, *including all rendered plots/figures*. (Hand drawing plots is permitted only for Problem 1.)
    - Be sure to mark which page corresponds to each question when submission to Gradescope.
  - The HW code should be submitted as a Python file Q3.py containing only the Python code for Problem 3.
    - Q3.py should contain functions part\_1(), part\_2(), part\_3(), part\_4(), and part\_5() containing the solutions to each subproblem. This is the only formatting requirement for Q3.py.
    - All explanations and/or rendered plots for Problem 3 should all be in the homework report, *not* the code submission.
  - The optional submission for the Bonus problem should be a zip file containing Q5.py and Q5.pdf. Q5.py will contain all relevant code used for the bonus problem, while Q5.pdf will contain the written responses and the rendered plot.
3. Be sure to mark any group members on the submission page, following [these instructions](#).

## 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  $(x_1, y_1) = (-1, -1)$ ,  $(x_2, y_2) = (0, -1)$ , and  $(x_3, y_3) = (1, 1)$ , shown in Figure 1.

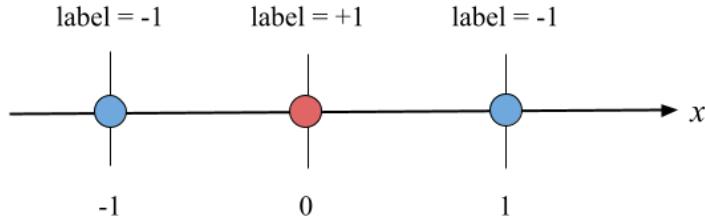


Figure 1: Three data points considered in Problem 1

**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?

**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?

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

**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 ( $\xi$ ) in the primal formulation have to be 0 (and thus can be removed from the optimization).

**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.

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

### Problem 2: Kernel Composition (6pts)

Prove that if  $k_1, k_2 : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  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  $\phi_1$  and  $\phi_2$  are the corresponding mappings for  $k_1$  and  $k_2$  respectively. Construct the mapping  $\phi$  that certifies  $k$  being a kernel function.

## 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  $\geq 3.7$ ) 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  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$  with real-valued labels  $y_1, \dots, y_n \in \mathbb{R}$ , 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  $f_i(\mathbf{w}) = (\mathbf{w}^T \mathbf{x}_i - y_i)^2$  denotes the squared error of the  $i$ th 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, and sampling  $w_{true}$ ) 10 times, and find the

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

**3.2 (7pts)** We will now examine  $\ell_2$  regularization as a means to prevent overfitting. The  $\ell_2$  regularized objective function is given by the following expression:

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

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

*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  $\ell_2$  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, this is `w_true` in the code.)

**3.4 (10pts)** We will now examine the behavior of SGD in more detail. Let  $\mathbf{w}^{(t)}$  refer to the SGD solution at the  $t$ -th iteration. For step sizes  $\{0.00005, 0.005\}$  and 1,000,000 iterations of SGD (and a single trial),

- (i) Plot the normalized training error of the SGD iterate  $\mathbf{w}^{(t)}$  vs. the iteration number  $t$ . 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 of the SGD iterate  $\mathbf{w}^{(t)}$  vs. the iteration number  $t$ . 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  $\ell_2$  norm of the SGD iterate  $\mathbf{w}^{(t)}$  vs. the iteration number  $t$ .

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  $\ell_2$  norm of the final solution?

**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  $\ell_2$  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  $\ell_2$  regularization coefficients. Can you provide an explanation for the behavior seen in this plot?

#### Problem 4: 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/spring26/cc\\_hw2.zip](https://vatsalsharan.github.io/spring26/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  $6 \times 7$  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.

**4.1 (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.

**4.2 (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?

**4.3 (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?

**4.4 (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?

### [Bonus] Problem 5: Logistic Regression (8 pts)

In this problem we will consider a simple binary classification task. We are given  $d$ -dimensional input data  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$  along with labels  $y_1, \dots, y_n \in \{-1, +1\}$ . 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  $f_i(\mathbf{w}) = \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$  denotes the logistic loss of the  $i$ th 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  $[0.12, 0.12, \dots, 0.12] \in \mathbb{R}^d$  for the class labelled as  $+1$ . Similarly, points are drawn from a Gaussian distribution with mean  $[-0.12, -0.12, \dots, -0.12] \in \mathbb{R}^d$  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* (with a batch size of  $b = 1$ ) 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  $\{0.0005, 0.005, 0.05\}$ , initializing with the all zeros vector.

**5.1 (4pts)** 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.

**5.2 (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.

**5.3 (2pts)** Comment on how well the logistic loss act as a surrogate for the 0-1 loss.