CS 5350/6350: Machine Learning Fall 2021

Homework 4

Handed out: 4 Nov, 2021
Due date: 11:59pm, 19 Nov, 2021

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free to discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You do not need to include original problem descriptions in your solutions. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 15 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* **You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.**

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

1 Paper Problems [40 points + 10 bonus]

1. [9 points] The learning of soft SVMs is formulated as the following optimization problem,

$$\begin{aligned} \min_{\mathbf{w},b,\{\xi_i\}} \quad & \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_i \xi_i \\ \text{s.t. } \forall 1 \le i \le N, \quad & y_i(\mathbf{w}^\top\mathbf{x}_i + b) \ge 1 - \xi_i, \\ & \xi_i \ge 0 \end{aligned}$$

   where N is the number of the training examples. As we discussed in the class, the slack variables $\{\xi_i\}$ are introduced to allow the training examples to break into the margin so that we can learn a linear classifier even when the data is not linearly separable.

(a) [3 point] What values ξ_i can take when the training example $\mathbf{x}_i$ breaks into the margin?

Answer We can find the answer through simple algebra

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$=> \xi_i \geq 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)$$

(b) [3 point] What values ξ_i can take when the training example $\mathbf{x}_i$ stays on or outside the margin?

Answer

ξ_i will be zero because no slack is necessary in this case.

(c) [3 point] Why do we incorporate the term $C \cdot \sum_i \xi_i$ in the objective function? What will happen if we throw out this term?

Answer

$C \cdot \sum_i \xi_i$ means that we are trying to minimize the total slack in a trade-off with maximizing the margin, where C is the trade-off term. Without slack, we will not be able to create a model for cases when the data is not linearly separable.

2. [6 points] Write down the dual optimization problem for soft SVMs. Please clearly indicate the constraints, and explain how it is derived. (Note: do NOT directly copy slides content, write down your own understanding.)

Answer

First, the optimization problem statement is summed up as the following:

$$\min_{\{0 \leq \alpha_i \leq C\}, \sum_i \alpha_i y_i = 0} \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j \mathbf{x}^\top \mathbf{x} - \sum_i \alpha_i$$

The constraints are that we want to find the minimum α_i such that it lies between 0 and our hyper-parameter C, and also such that the dot product between our vectors of α and $\mathbf{y}$ will sum to 0.

To derive, start with the Primal problem:

$$\min_{\mathbf{w}, b, \{\xi_i\}} \max_{\{\alpha_i \geq 0, \beta_i \geq 0\}} \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_i \xi_i - \sum_i \beta_i \xi_i - \sum_i \alpha_i (y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i)$$

Swap min and max to get the Dual form, which will be equivalent:

$$\max_{\{\alpha_i \geq 0, \beta_i \geq 0\}} \min_{\mathbf{w}, b, \{\xi_i\}} \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_i \xi_i - \sum_i \beta_i \xi_i - \sum_i \alpha_i (y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i)$$

To get the minimization problem, derive using Lagrangian multipliers and then taking derivatives with respect to the weight vector, the bias, and ξ, our slack variable and setting these derivatives to 0, doing some algebra and resubbing the results back into the original L equation.

3. [10 points] Continue with the dual form. Suppose after the training procedure, you have obtained the optimal parameters.

   (a) [4 points] What parameter values can indicate if an example stays outside the margin?

   Answer

   The parameter values that would indicate that a given vector/example is outside the margin is 0.

   (b) [6 points] if we want to find out which training examples just sit on the margin (neither inside nor outside), what shall we do? Note you are not allowed to examine if the functional margin (i.e., $y_i(\mathbf{w}^\top \mathbf{x}_i + b)$) is 1.

   Answer

   Since any vectors outside the margin have $\alpha = 0$, I would expect that any values just slightly bigger than 0 (i.e. any $\delta > 0$) likely sit approximately on the margin.

4. [6 points] How can we use the kernel trick to enable SVMs to perform nonlinear classification? What is the corresponding optimization problem?

   Answer

   We apply a kernel to our examples to create a gram matrix. This puts our training data into a new space, making this a linear classification problem. The corresponding optimization problem we use is Dual SVM.

5. [9 points] Suppose we have the training dataset shown in Table 1. We want to learn a SVM classifier. We initialize all the model parameters with 0. We set the learning rates for the first three steps to $\{0.01, 0.005, 0.0025\}$. Please list the sub-gradients of the SVM objective w.r.t the model parameters for the first three steps, when using the stochastic sub-gradient descent algorithm.

x_1	x_2	x_3	y
0.5	-1	0.3	1
-1	-2	-2	-1
1.5	0.2	-2.5	1

Table 1: Dataset

   Answer

   We use the code in Listing 1 to generate the results seen in Tables 2-4, which show how the weight vectors change over each iteration of the sub-gradient descent. The following statements were printed out:

```
PART 1 QUESTION 5
X matrix:
 [[ 0.5 -1.   0.3]
 [-1.  -2.  -2. ]
```

```
 [ 1.5  0.2 -2.5]]
y vector:
 [[ 1]
 [-1]
 [ 1]]
hyper parameter C: 0.3333333333333333
initial weight vector:
 [0 0 0]
initial bias: 0

Gamma for iteration 1: 0.01
Case is <= 1? True
Update according to case 1
Iteration 1 update:
New weight vector:
 [ 0.005 -0.01   0.015]
New bias: 0.009999999999999998

Gamma for iteration 2: 0.005
Case is <= 1? True
Update according to case 1
Iteration 2 update:
New weight vector:
 [9.9750e-03 5.0000e-05 1.3925e-02]
New bias: 0.004999999999999999

Gamma for iteration 3: 0.0025
Case is <= 1? True
Update according to case 1
Iteration 3 update:
New weight vector:
 [ 0.01070006 -0.00495012  0.00764019]
New bias: 0.007499999999999999
```

Listing 1: Functions used to generate results for Part 1 Question 5

```
import numpy as np


def q5():
    print('PART 1 QUESTION 5')
    x1 = np.array([[0.5], [-1], [1.5]])
    x2 = np.array([[-1], [-2], [0.2]])
    x3 = np.array([[0.3], [-2], [-2.5]])
    y = np.array([[1], [-1], [1]])
```

```
w = np.array([0, 0, 0])
b = 0
X = np.hstack([x1, x2, x3])
C = 1/3
gamma = [0.01, 0.005, 0.0025]
print('X matrix:\n', X)
print('y vector:\n', y)
print('hyper parameter C:', C)
print('initial weight vector:\n', w)
print('initial bias:', b)

for i in range(3):
    gamma_i = gamma[i]
    print(f'\nGamma for iteration {i + 1}: {gamma_i}')
    pass_check = compute_check_case(X.T[i], w, y[i])
    print('Case is <= 1?', pass_check)
    if pass_check:
        print('Update according to case 1')
        w, b = case_1_update(w, gamma_i, b, C, len(X[i]), y[i], X.T[i
    else:
        print('Update according to case 2')
        w = case_2_update(w, gamma_t)
    print(f'Iteration {i + 1} update:')
    print('New weight vector:\n', w)
    print('New bias:', b)


def compute_check_case(x_i: np.ndarray, w: np.ndarray, y_i: int) -> bool:
    return (y_i * w.T * x_i <= 1)[0]


def case_1_update(w: np.ndarray, gamma_t: float, b: float, C: float, N: i
    return (w - gamma_t * w + gamma_t * C * N * y_i * x_i), (b + gamma_t


def case_2_update(w: np.ndarray, gamma_t: float):
    return (1 - gamma_t) * w
```

6. [**Bonus**][10 points] Let us derive a dual form for Perceptron. Recall, in each step of Perceptron, we add to the current weights $\mathbf{w}$ (including the bias parameter) $y_i\mathbf{x}_i$ for some misclassified example $(\mathbf{x}_i, y_i)$. We initialize $\mathbf{w}$ with $\mathbf{0}$. So, instead of updating $\mathbf{w}$, we can maintain for each training example i a mistake count c_i — the number of times the data point $(\mathbf{x}_i, y_i)$ has been misclassified.

   - [2 points] Given the mistake counts of all the training examples, $\{c_1, \ldots, c_N\}$,

x	w_1
0.5	0.005
-1	-0.01
1.5	0.015
1	0.0100

Table 2: Part 1 Question 5 Pass 1

x	w_2
-1	$9.975e-03$
-2	$5.000e-05$
0.2	$1.393e-02$
1	0.0050

Table 3: Part 1 Question 5 Pass 2

how can we recover $\mathbf{w}$? How can we make predictions with these mistake counts?

- [3 points] Can you develop an algorithm that uses mistake counts to learn the Perceptron? Please list the pseudo code.
- [5 points] Can you apply the kernel trick to develop an nonlinear Perceptron? If so, how do you conduct classification? Can you give the pseudo code fo learning this kernel Perceptron?

2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of Perceptron, voted Perceptron and average Perceptron algorithms. Remember last time you created the folders "Perceptron". You can commit your code into the corresponding folders now. Please also supplement README.md with concise descriptions about how to use your code to run these algorithms (how to call the command, set the parameters, etc). Please create a new folder "SVM" in the same level as these folders.

   Answer

   The GitHub repo is not the same as previous projects and can be found by clicking here.

2. [28 points] We will first implement SVM in the primal domain with stochastic subgradient descent. We will reuse the dataset for Perceptron implementation, namely, "bank-note.zip" in Canvas. The features and labels are listed in the file "classification/data-desc.txt". The training data are stored in the file "classification/train.csv", consisting of 872 examples. The test data are stored in "classification/test.csv", and comprise of 500 examples. In both the training and test datasets, feature values and labels are separated by commas. Set the maximum epochs T to 100. Don't forget to shuffle the training examples at the start of each epoch. Use the curve of the objective function

x	w_3
0.3	0.0107
-2	-0.0050
-2.5	0.0076
1	0.0075

Table 4: Part 1 Question 5 Pass 3

(along with the number of updates) to diagnosis the convergence. Try the hyperparameter C from $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. Don't forget to convert the labels to be in $\{1, -1\}$.

(a) [12 points] Use the schedule of learning rate: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{a}t}$. Please tune γ_0 and a to ensure convergence. For each setting of C, report your training and test error.

Answer

The value for γ_0 I used was 0.1, and the value for a was 0.001. The Training and Test Errors can be found in Table 5.

$W_{wave.var.}$	$W_{wave.skew}$	$W_{wave.curt.}$	$W_{img.entr.}$	$Bias$	Train. Err.	Test Err.	C
-0.6357	-0.2895	-0.3109	-0.1002	0.1681	0.0298	0.0340	0.1145
-2.5356	-1.1667	-1.4354	-0.3031	0.4562	0.0275	0.0380	0.5727
-3.5380	-1.6206	-2.0162	-0.4104	0.6019	0.0275	0.0400	0.8018

Table 5: Training and test errors for part 2 q2a.

(b) [12 points] Use the schedule $\gamma_t = \frac{\gamma_0}{1+t}$. Report the training and test error for each setting of C.

Answer

For this, I simply needed to pass in the same value for both γ_0 and a, as they would cancel each other out to 1 if both were > 0. The Training and Test Errors can be found in Table 6.

$W_{wave.var.}$	$W_{wave.skew}$	$W_{wave.curt.}$	$W_{img.entr.}$	$Bias$	Train. Err.	Test Err.	C
-1.4807	-0.8737	-1.0926	0.0873	2.4854	0.0172	0.0180	0.1145
-6.5171	-3.9263	-4.9219	1.6185	13.0606	0.0183	0.0180	0.5727
-9.0353	-5.3508	-6.7507	2.4361	18.4582	0.0195	0.0180	0.8018

Table 6: Training and test errors for part 2 q2b.

(c) [6 points] For each C, report the differences between the model parameters learned from the two learning rate schedules, as well as the differences between the training/test errors. What can you conclude?

Answer

Given that the results are generally better in Table 6, I would conclude that damping how quickly the learning rate γ_0 change may not necessarily help yield

better results (3% error on Test Data in Table 5 vs 1.8% error on Test Data in Table 6). I had tuned γ_0 for Table 5 so that the objective function would converge, but based on these results, it would seem that doing so was not necessary, since simply using a basic schedule yielded better results than what my tuned model did. However, this could be due to bad tuning, and so perhaps re-tuning γ_0 would yield similar or better results in Table 5.

As for the differences between the parameters themselves, they are not so different on a case by case basis, though the parameters in Table 6 are generally larger than those in Table 5. There are two notable differences, however. The $W_{img.etr}$ and $Bias$ are significantly different. The $Bias$ is quite a bit larger (on a different scale even) than in Table 6, while $W_{img.etr}$ is negative and small in Table 5 but positive and 10 times larger in Table 6. For $W_{img.etr}$ this may be due simply to where the epochs ended, since the objective function oscillates, and these values relative to the overall margin is actually reasonably small, which seems plausible given how similar the errors are. The $Bias$ may have ended up this way for a similar reason, or alternatively it could be that due to how much more quickly γ_t changed in Table 6, that there were simply more cases where the $Bias$ had to be updated, and the rest of the weight vectors changed in response. It may be useful to plot each linear function against the actual data points to do a "spot check" on how well it maximized the margin, though for the sake of time I will not do so for this project.

3. [30 points] Now let us implement SVM in the dual domain. We use the same dataset, "bank-note.zip". You can utilize existing constrained optimization libraries. For Python, we recommend using "scipy.optimize.minimize", and you can learn how to use this API from the document at `https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.minimize.html`. We recommend using SLSQP to incorporate the equality constraints. For Matlab, we recommend using the internal function "fmincon"; the document and examples are given at `https://www.mathworks.com/help/optim/ug/fmincon.html`. For R, we recommend using the "nloptr" package with detailed documentation at `https://cran.r-project.org/web/packages/nloptr/nloptr.pdf`.

   (a) [10 points] First, run your dual SVM learning algorithm with C in $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. Recover the feature weights $\mathbf{w}$ and the bias b. Compare with the parameters learned with stochastic sub-gradient descent in the primal domain (in Problem 2) and the same settings of C, what can you observe? What do you conclude and why?

   Answer

   The weight vector returned for each C as well as their respective training and test errors can be found in Table 7. The learned parameters and bias, as well as the errors, tend to land somewhere between Table 6 and Table 7. I would conclude that the Dual SVM yields similar results to Primal SVM Sub-Gradient Descent. This makes sense because Primal and Dual SVMs are supposed to be equivalent.

$W_{wave.var.}$	$W_{wave.skew}$	$W_{wave.curt.}$	$W_{img.entr.}$	$Bias$	Train. Err.	Test Err.	C
-0.9429	-0.6515	-0.7337	-0.0410	2.3802	0.0229	0.0240	0.1145
-1.5639	-1.0141	-1.1807	-0.1565	3.8609	0.0310	0.0340	0.5727
-2.0425	-1.2807	-1.5135	-0.2491	4.9736	0.0333	0.0360	0.8018

Table 7: Training and test errors for part 2 q3a.

(b) [15 points] Now, use Gaussian kernel in the dual form to implement the non-linear SVM. Note that you need to modify both the objective function and the prediction. The Gaussian kernel is defined as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\gamma}).$$

Test γ from $\{0.1, 0.5, 1, 5, 100\}$ and the hyperparameter C from $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. List the training and test errors for the combinations of all the γ and C values. What is the best combination? Compared with linear SVM with the same settings of C, what do you observe? What do you conclude and why?

Answer

The training and test errors can be found in Table 8. These results clearly indicate that depending on the hyperparameters, either the model performs much worse, or much better. The best combination in this case based on the Training and Test Errors is both when ($C = 0.5727 = \frac{500}{873}; \gamma = 5$) and ($C = 0.8018 = \frac{700}{873}; \gamma = 5$). I would conclude that for this dataset that the ability to create a non-linear classifier allows for a much better fit, but it requires more tuning and processing time than with the Primal Sub-Gradient Descent or Linear Dual SVM. The reason why there is more variability between the Gaussian and Linear model for the training and test errors is likely linked–fundamentally–to the fact that Gaussian Kernel has an additional degree of freedom (i.e. another hyperparameter to be tuned) that can greatly transform the training data. This degree of freedom, that is to say γ, affects how well the differences between different rows of the training data are represented. In the case when γ is very small, the differences end up being represented as essentially 0, but as γ increases the difference between different vectors are more easily discerned, making it easier for the model to determine which rows should be included as support vectors.

(c) [5 points] Following (b), for each setting of γ and C, list the number of support vectors. When $C = \frac{500}{873}$, report the number of overlapped support vectors between consecutive values of γ, i.e., how many support vectors are the same for $\gamma = 0.01$ and $\gamma = 0.1$; how many are the same for $\gamma = 0.1$ and $\gamma = 0.5$, etc. What do you observe and conclude? Why?

Answer

The overlap results can be seen in Table 9, and the support vector counts can be found in Table 8. It would seem that the overlap between consecutive γ values decreases. More importantly, it looks like as gamma increases, certain support

C	γ	Train. Err.	Test Err.	SV Count
0.1145	0.1	0.0722	0.3940	871
0.1145	0.5	0.0034	0.1040	839
0.1145	1.0	0.0011	0.0260	857
0.1145	5.0	0.0034	0.0040	800
0.1145	100	0.0138	0.0080	704
0.5727	0.1	0.0000	0.2140	872
0.5727	0.5	0.0000	0.0140	767
0.5727	1	0.0000	0.0040	746
0.5727	5	0.0000	0.0000	685
0.5727	100	0.0080	0.0060	394
0.8018	0.1	0.0000	0.1820	869
0.8018	0.5	0.0000	0.0100	786
0.8018	1	0.0000	0.0040	689
0.8018	5	0.0000	0.0000	409
0.8018	100	0.0080	0.0060	560

Table 8: Results for part 2 q3b.

vectors seem to be conserved. For example, all the support vectors for $\gamma = 0.5$, every single one was also present when $\gamma = 0.1$. So, even though with increasing values of γ the number of support vectors decrease, certain support vectors for consecutive values of γ tend be conserved. This is likely because certain vectors that have a large difference between X_i and X_j become larger and fall more firmly outside the margin, making it so that they are no longer support vectors.

γ_i	γ_j	Support Vector Overlap
0.1	0.5	767
0.5	1.0	680
1.0	5.0	605
5.0	100	320

Table 9: Results for part 2 q3c.

(d) [**Bonus**] [10 points] Implement the kernel Perceptron algorithm you developed in Problem 8 (Section 1). Use Gaussian kernel and test γ from $\{0.1, 0.5, 1, 5, 100\}$. List the training and test errors accordingly. Compared with the nonlinear SVM, what do you observe? what do you conclude and why?