CS 5350/6350: Machine Learning Spring 2019

Homework 4

Handed out: 20 Mar, 2019
Due date: 11:59pm, 5 Apr, 2019

- 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]

1. [3 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) [1 point] What values ξ_i can take when the training example $\mathbf{x}_i$ breaks into the margin?

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

(b) [1 point] What values ξ_i can take when the training example $\mathbf{x}_i$ stays on or outside the margin?

$\xi_i = 0$

(c) [1 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?

This term determines the trade off between a generally large margin and the total penalty of examples in or behind the margin. Without this term the model might over-fit the data instead of using a larger margin and generalize poorly.

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.)

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ over $\mathbf{w}$ and minimize $\sum_i \xi_i$ over $\mathbf{w}$ and b. The constraints on this are that $\forall i, y_i(\mathbf{w}^\top\mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$. This means that every example has a value ξ which is the amount by which it 'breaks into' the margin. By using hinge loss this becomes an accumulation of the penalty of a weight vector.

3. [8 points] Continue with the dual form. Suppose after the training procedure, you have obtained the optimal parameters.

(a) [2 points] What parameter values can indicate if an example stays outside the margin?

If $a*_j = 0$ then example j is outside the margin.

(b) [3 points] What are common and what are different when we use these parameters and the slack variables $\{\xi_i\}$ to determine the relevant positions of the training examples to the margin?

For both a value of 0 denotes an example outside the margin, and for both a value above 0 denotes something inside the margin. One main difference is that $\{\xi_i\}$ is bounded below by $max(0, 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b))$ and above by nothing, while α_i is bounded below by 0 and above by the hyperparameter C.

(c) [3 points] Now 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.

A point which is simply on the margin will have the highest value for α_i. Therefore, to find points on the margin find all values for which $\alpha_i = max(\alpha)$.

4. [3 points] How can we use the kernel trick to enable SVMs to perform nonlinear classification? What is the corresponding optimization problem?

To enable SVM's to perform nonlinear classification replace $\mathbf{x}_i^T\mathbf{x}_j$ with some nonlinear kernel $K(\mathbf{x}_i, \mathbf{x}_j)$. The new optimization problem is simply

$$\min_{\{0\le\alpha_i\le C\},\sum_i \alpha_i y_i = 0} \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i$$

5. [5 points] Prove that the primal objective function of the soft SVM is convex to $\mathbf{w}$ and b,

$$\frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_i \max\big(0, 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b)\big).$$

The objective function can be rewritten as $f_1(\mathbf{w}) + C\sum_i f_2(\mathbf{w})$, where $f_1(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top\mathbf{w}$, $f_2(\mathbf{w}) = max(0, f_3(\mathbf{w}))$, and $f_3(\mathbf{w}) = 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b)$.

$f_1 = \frac{1}{2}w_0^2 + \frac{1}{2}w_n^2$

$f_3 = 1 - y_i w_0 x_{i0} - y_i w_n x_{in} - y_i b$

$$\nabla f_1(\mathbf{w}) = \begin{Bmatrix} \mathbf{w}_0 \\ \vdots \\ \mathbf{w}_n \end{Bmatrix}$$

$\nabla^2 f_1(\mathbf{w}) =$ n $\times$ n identity matrix

$$\nabla f_3(\mathbf{w}) = \begin{Bmatrix} -y_i\mathbf{x}_{i0} \\ \vdots \\ -y_i\mathbf{x}_{in} \end{Bmatrix}$$

$$\nabla^2 f_3(\mathbf{w}) = \begin{Bmatrix} 0 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 0 \end{Bmatrix}$$

$\nabla^2 f_3(\mathbf{w})$ is a 0-matrix, therefore positive semi-definite, and therefore convex. f_3 and 0 are both convex functions, therefore $f_2 = max(0, f_3)$ preserves convexity. $\nabla^2 f_1$ is an n $\times$ n identity matrix, and is therefore also positive semi-definite and convex. Therefore the objective function is a summation of weighted convex functions, and is itself convex.

With respect to b:

$f_1''(b) = 0$

$f_3''(b) = 0$

The previous proof holds with respect to b. The objective function is convex with respect to $\mathbf{w}$ and b.

6. [2 points] Illustrate how the Occam's Razor principle is reflected in the SVM objective and optimization in Problem 5.

The objective function is able to use a few very simple functions to encode a very complicated optimization problem. The functions are trivial to prove convex, and with it the whole function, meaning a local minimum can always be found.

7. [3 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

Iteration 1:

∇J^1 = [-0.172, 0.344, -0.103]

∇J^2 = [-0.344, -0.687, -0.687]

∇J^3 = [-0.515, -0.069, 0.859]

Iteration 2:

∇J^1 = [-0.172, 0.344, -0.103]

∇J^2 = [-0.344, -0.687, -0.687]

∇J^3 = [-0.515, -0.0687, 0.859]

Iteration 3:

∇J^1 = [-0.172, 0.344, -0.103]

∇J^2 = [-0.344, -0.687, -0.687]

∇J^3 = [-0.515, -0.0687, 0.859]

8. [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, $\{c_1, \ldots, c_N\}$, how can we recover $\mathbf{w}$? How can we make predictions with these mistake counts?
     $\mathbf{w} = \sum_i c_i y_i \mathbf{x}_i$
     Prediction $= sgn(\mathbf{x}_t \sum_i c_i y_i \mathbf{x}_i)$
   - [3 points] Can you develop an algorithm that uses mistake counts to learn the Perceptron? Please list the pseudo code.
     - Initialize $\mathbf{c}_0 = 0 \in \mathbb{R}^n$
     - For each training example $(\mathbf{x}_i, y_i)$:

```
        * predict $y' = sgn(\mathbf{x}_i \sum_j \mathbf{c}_j y_j \mathbf{x}_j)$
        * if $y' \neq y_i$:
            · $\mathbf{c}_i = \mathbf{c}_i + 1$
    – Return $\mathbf{c}$
```

- [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?

```
    – Initialize $\mathbf{c}_0 = 0 \in \mathbb{R}^n$
    – For each training example $(\mathbf{x}_i, y_i)$:
        * predict $y' = sgn(\sum_j \mathbf{c}_j y_j K(\mathbf{x}_i, \mathbf{x}_j))$
        * if $y' \neq y_i$:
            · $\mathbf{c}_i = \mathbf{c}_i + 1$
    – Return $\mathbf{c}$
```

Prediction/classification for $\mathbf{x} = sgn(\sum_i \mathbf{c}_i y_i K(\mathbf{x}_i, \mathbf{x}))$

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.

2. [28 points] We will first implement SVM in the primal domain with stochastic sub-gradient 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 (along with the number of updates) to diagnosis the convergence. Try the hyperparameter C from $\{\frac{1}{873}, \frac{10}{873}, \frac{50}{873}, \frac{100}{873}, \frac{300}{873}, \frac{500}{873}, \frac{700}{873}\}$. Don't forget to convert the labels to be in $\{1, -1\}$.

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

   $\gamma_0 = 0.01$, $d = 0.01$

C =	$\frac{1}{873}$	$\frac{10}{873}$	$\frac{50}{873}$	$\frac{100}{873}$	$\frac{300}{873}$	$\frac{500}{873}$	$\frac{700}{873}$
Training Error	0.050	0.041	0.040	0.040	0.040	0.039	0.040
Testing Error	0.072	0.048	0.046	0.046	0.05	0.048	0.054

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

   $\gamma_0 = 0.01$

C =	$\frac{1}{873}$	$\frac{10}{873}$	$\frac{50}{873}$	$\frac{100}{873}$	$\frac{300}{873}$	$\frac{500}{873}$	$\frac{700}{873}$
Training Error	0.050	0.041	0.039	0.040	0.042	0.046	0.047
Testing Error	0.072	0.052	0.046	0.042	0.048	0.046	0.054

   (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?

   Despite schedule B producing weight vectors of much larger magnitude than schedule A, there is a negligible difference in training and testing error. This large magnitude is likely due to the larger values for γ given by schedule B.

C	Schedule A	Schedule B	Train Diff	Test Diff
1/873	[-0.373, -0.17, -0.145, -0.079]	[-0.376, -0.169, -0.147, -0.083]	0.0	0.0
10/873	[-0.73, -0.366, -0.371, -0.195]	[-0.781, -0.396, -0.395, -0.209]	0.0	0.003
50/873	[-1.104, -0.593, -0.642, -0.247]	[-1.53, -0.858, -0.926, -0.366]	0.001	0.0
100/873	[-1.266, -0.68, -0.741, -0.282]	[-2.15, -1.241, -1.366, -0.584]	0.0	0.004
300/873	[-1.619, -0.821, -0.942, -0.317]	[-5.709, -3.625, -3.909, -1.789]	0.002	0.002
500/873	[-1.741, -0.908, -1.011, -0.327]	[-10.712, -7.588, -7.898, -3.989]	0.007	0.002
700/873	[-1.797, -0.922, -1.096, -0.351]	[-14.214, -9.392, -10.242, -4.264]	0.007	0.0

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 to use "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`. For Matlab, we recommend to use the internal function "fmincon"; the document and examples are given at `https://www.mathworks.com/help/optim/ug/fmincon.html`. For R, we recommend to use the "nloptr" package with detailed documentation at `https://cran.r-project.org/web/packages/nloptr/nloptr.pdf`. In principle, you can choose any nonlinear optimization algorithm. But we recommend to use L-BFGS or CG for their robustness and excellent performance in practice.

   (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?

C	$\mathbf{w}$	b
100/873	[-0.943, -0.651, -0.734, -0.041]	-5.141
500/873	[-1.564, -1.014, -1.1806, -0.157]	-8.590
700/873	[-2.043, -1.281, -1.514, -0.249]	-11.233

The magnitude of these weight vectors is slightly smaller than those from 2b, and much smaller than those from 2c. This is most likely because of the lack of a bias term from the primal SVM, so weight magnitudes need to be larger to offset this.

   (b) [15 points] Now, use Gaussian kernel in the dual form to implement the nonlinear 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 $\{0.01, 0.1, 0.5, 1, 2, 5, 10, 100\}$ 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?

Training Error:

C \ γ	0.01	0.1	0.5	1	2	5	10	100
100/873	0.0	0.0	0.0	0.0	0.0	0.01	0.01	0.05
500/873	0.0	0.0	0.0	0.0	0.0	0.01	0.01	0.01
700/873	0.0	0.0	0.0	0.0	0.0	0.01	0.01	0.01

Testing Error:

C \ γ	0.01	0.1	0.5	1	2	5	10	100
100/873	0.056	0.024	0.024	0.024	0.03	0.03	0.028	0.88
500/873	0.056	0.024	0.024	0.024	0.024	0.03	0.026	0.026
700/873	0.056	0.024	0.024	0.024	0.024	0.028	0.028	0.018

Training set size was reduced to make run time more manageable. On this train/test data, the best hyperparameter combination is $C = 700/873$ and $\gamma = 100$. Compared to the linear SVM, the dual SVM performs far better as a classifier across the board. Additionally, the convex pattern for C, with the best C values being closer to 0.5, apparent in the linear SVM is not obvious for the dual SVM. These results suggest that C is not a very significant hyperparameter for this dataset in the dual SVM.

(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?

Ratio of Support Vectors:

C \ γ	0.01	0.1	0.5	1	2	5	10	100
100/873	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.97
500/873	1.0	1.0	1.0	1.0	0.94	0.86	0.93	0.51
700/873	1.0	1.0	1.0	1.0	0.93	0.9	0.78	0.96

Ratio of overlapping support vectors:

γ	$0.01 \to 0.1$	$0.1 \to 0.5$	$0.5 \to 1$	$1 \to 2$	$2 \to 5$	$5 \to 10$	$10 \to 100$
	1.0	1.0	0.985	0.945	0.78	0.53	0.513

As γ increases the total number of support vectors decreases, meaning the margin must be shrinking, but that more examples are more confidently classified. We also see this reflected in the number of overlapping support vectors, showing here the ratio decreases as C increases.

(d) [**Bonus**] [10 points] Implement the kernel Perceptron algorithm you developed in Problem 8 (Section 1). Use Gaussian kernel and test γ from $\{0.01, 0.1, 0.5, 1, 2, 5, 10, 100\}$. List the training and test errors accordingly. Compared with the nonlinear SVM, what do you observe? what do you conclude and why?