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]

Answers are in blue font.

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?
If x_i breaks into the margin but is on the correct side, then $0 < \xi_i < 1$.

(b) [1 point] What values ξ_i can take when the training example $\mathbf{x}_i$ stays on or outside the margin?
If x_i is correctly classified, then $\xi_i = 0$. If x_i is wrongly classified ($\xi_i > 1$), it is still possible to stay outside the margin but on the wrong side (For clearity, we say *a sample breaks into margin* if it breaks into the margin but is still on the correct side of the separating hyperplane, excluding the case where *a point breaks into the margin but lies on the wrong side of the plane.*).

(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?
The regularization term tries to maximize the margin while the empirical loss penalizes the mistake, i.e., we do not want too many points to break into the margin (even get misclassified), or the total amount of violation (summation of slack variables). If the second term is removed, then we can maximize the margin allowing as many points to break into the margin, which makes no sense.

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.)
The Lagrangian form of the primal is

$$\min_{\mathbf{w},b,\{\xi_i\}} \max_{\{\alpha_i \geq 0, \beta_i \geq 0\}} L := \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i + \sum_i \beta_i(-\xi_i) + \sum_i \alpha_i[1 - \xi_i - y_i(\mathbf{w}^T x_i + b)] \quad (1)$$

and the corresponding dual form is

$$\max_{\{\alpha_i \geq 0, \beta_i \geq 0\}} \min_{\mathbf{w},b,\{\xi_i\}} L := \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i + \sum_i \beta_i(-\xi_i) + \sum_i \alpha_i[1 - \xi_i - y_i(\mathbf{w}^T x_i + b)] \quad (2)$$

Since the objective function is differentiable (and convex) w.r.t $\mathbf{w}, b, \{\xi_i\}$, we set the partial derivative to be zero to get the minimum value,

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \frac{\partial\left(\sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i\right)}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \quad (3)$$

$$\Rightarrow \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad (4)$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0 \Rightarrow \sum_i \alpha_i y_i = 0 \quad (5)$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \Rightarrow \alpha_i + \beta_i = C, \forall i \quad (6)$$

Substituting $\mathbf{w}$ and the two constraints into the Lagrangian form, we obtain the new objective as

$$L' = \frac{1}{2}\left(\sum_i \alpha_i y_i \mathbf{x}_i\right)^T \left(\sum_i \alpha_i y_i \mathbf{x}_i\right) - \sum_i \alpha_i y_i \left(\sum_i \alpha_i y_i \mathbf{x}_i\right)^T \mathbf{x}_i + \sum_i \alpha_i \quad (7)$$

$$= -\frac{1}{2}\left(\sum_i \alpha_i y_i \mathbf{x}_i\right)^T \left(\sum_i \alpha_i y_i \mathbf{x}_i\right) + \sum_i \alpha_i \quad (8)$$

$$= -\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_i \alpha_i \quad (9)$$

Hence, the standard dual form is

$$\max_{\{\alpha_i \geq 0, \beta_i \geq 0\}} \quad L' := -\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_i \alpha_i \quad (10)$$

$$s.t. \quad \sum_i \alpha_i y_i = 0, \forall i \quad (11)$$

$$\alpha_i + \beta_i = C, \forall i \quad (12)$$

which can be further simplified as

$$\min \quad L'' := \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_i \alpha_i \quad (13)$$

$$s.t. \quad \sum_i \alpha_i y_i = 0, \forall i \in [N] \quad (14)$$

$$0 \leq \alpha_i \leq C, \forall i \in [N] \quad (15)$$

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?
   $\alpha_i^* = 0$ indicates the sample $\mathbf{x}_i$ stays outside the margin since support vectors $S = \{\mathbf{x}_j | \alpha_j^* > 0\}$ stay inside or on 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?
   After solving the dual problem, we obtain $\{\alpha_i^*\}_{i\in[N]}$. From the KKT condition and complementary slackness, we know that if $\alpha_i^* \in (0, C)$, then $\mathbf{x}_i$ lies on the margin and if $\alpha_i^* = C$ then $\mathbf{x}_i$ stays inside the margin. However, if $\alpha_i^* = 0$, we are not sure whether $\mathbf{x}_i$ lies inside/outside/on the margin. On the other hand, if we solve the original problem and obtain $\{\xi_i^*\}_{i\in[N]}$, then the relevant positions of samples are completely determined by $\{\xi_i^*\}_{i\in[N]}$.

(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.
Using the complementary slackness , we have

$$\beta_i^* \xi_i^* = 0, \forall i \in [N] \tag{16}$$
$$\alpha_i^* [y_i(\mathbf{w}^{*T} \mathbf{x}_i + b^*) - 1 + \xi_i^*] = 0, \forall i \in [N] \tag{17}$$
$$\alpha_i^* + \beta_i^* = 0, \forall i \in [N] \tag{18}$$

We see that if $\alpha_i^* > 0, \xi_i^* = 0$ then $\mathbf{x}_i$ stays on the margin. One sufficient condition is that 1) $\alpha_i^* > 0, \beta_i^* > 0$ (i.e., $0 < \alpha_i^* < C$), which enforces $\xi_i^* = 0$ due to $\beta_i^* \xi_i^* = 0, \forall i \in [N]$ and we do not need to verify the functional margin value after solving the dual (only obtaining $\{\alpha_i^*\}_{i \in [N]}$). On the other hand, 2) if $\xi_i^* > 0$, we have $\beta_i^* = 0 \Rightarrow \alpha_i^* = C$, implying that $\mathbf{x}_i$ stays inside the margin. 1)and 2) together suggests that support vectors lie on or inside the margin.

4. [3 points] How can we use the kernel trick to enable SVMs to perform nonlinear classification? What is the corresponding optimization problem?
Since the prediction for linear SVM is $\text{sgn}(\sum_i \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x})$ for testing sample $\mathbf{x}$, we directly use the kernel to replace the dot product, obtaining $\text{sgn}(\sum_i \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}))$ and the corresponding OPT problem (dual form) is

$$\min_{\{\alpha_i \in [0,C]\}, \sum_i \alpha_i y_i = 0} \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i \tag{19}$$

Note that this problem is still a constrained quadratic OPT problem, which is easy to solve. Also note that we do not need to explicitly construct the feature mapping $\phi(\mathbf{x})$.

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

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

Proof: We first prove that [Lemma] $f_3 := \max(f_1, f_2)$ *is convex if* f_1 *and* f_2 *are convex.* For $x_1, x_2 \in \text{dom}(f_1) \cap \text{dom}(f_2)$, and $\lambda \in [0, 1]$, we have

$$\begin{aligned}
f_3(\lambda x_1 + (1-\lambda) x_2) &= \max\left(f_1(\lambda x_1 + (1-\lambda) x_2), f_2(\lambda x_1 + (1-\lambda) x_2)\right) && (20)\\
&\leq f_j(\lambda x_1 + (1-\lambda) x_2) \text{ for some } j \in [2] && (21)\\
&\leq \lambda f_j(x_1) + (1-\lambda) f_j(x_2) && (22)\\
&\leq \lambda \max(f_1(x_1), f_2(x_2)) + (1-\lambda) \max(f_1(x_2), f_2(x_2)) && (23)\\
&= \lambda f_3(x_1) + (1-\lambda) f_3(x_2) && (24)
\end{aligned}$$

The hinge loss in affine w.r.t $\mathbf{w}$ and linear w.r.t. b. Since affine and linear functions are both convex and concave, we can easily see that the hinge loss is convex w.r.t. both $\mathbf{w}$ and b. Also, the quadratic term is convex and summation of convex functions are also convex. Hence, the SVM objective is convex w.r.t. both $\mathbf{w}$ and b.

6. [2 points] Illustrate how the Occam's Razor principle is reflected in the SVM objective and optimization in Problem 5.
The regularization term $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ prefers weight vectors with smaller squared norms, which implies that smoother linear functions are preferred. This actually reduces the hypothesis space somehow and only focus on slowly-changing functions.

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

Assume all vectors are in the form of column vectors, i.e., $\mathbf{w} \in \mathbf{R}^{d+1}(augmented), \mathbf{w}_0 \in \mathbf{R}^d(original)\mathbf{x}_i \in \mathbf{R}^{d+1}$ where d is the dimension of the samples.
The initial model parameters are $\mathbf{w}_0 = [0,0,0]^T, b = 0, \mathbf{w}^0 = [0,0,0,0]^T$ (compact form $\mathbf{w} = [w_1, w_2, w_3, b]^T$ and augment all samples with constant feature 1) and the SVM loss with SGD is $J = \frac{1}{2}\mathbf{w}^T\mathbf{w} + CN\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i), N = 3$ and the sub-gradient is $\nabla J = [\mathbf{w}_0^T, 0]^T$ if $1 - y_i\mathbf{w}_i^T\mathbf{x}_i \leq 0$ and $\nabla J = [\mathbf{w}_0^T, 0]^T - Cy_i\mathbf{x}_i$ if $1 - y_i\mathbf{w}_i^T\mathbf{x}_i > 0$.

The update procedure is as follows.
For $(\mathbf{x}_1, y_1) = [0.5, -1, 0.3, 1, 1]^T$ ($\mathbf{x}_1$ is augmented with constant feature), since $1 - y_1\mathbf{w}^{0T}\mathbf{x}_1 = 1 > 0$, the sub-gradient is (assuming $C = \frac{1}{N} = \frac{1}{3}$).

$$\begin{aligned} \nabla J &= [\mathbf{w}_0, 0] - 3Cy_1\mathbf{x}_1 = [0,0,0,0]^T - 3C[0.5, -1, 0.3, 1]^T = [-0.5, 1, -0.3, -1]^T \\ \mathbf{w}^1 &= \mathbf{w}^0 - 0.01[-0.5, 1, -0.3, -1]^T = [0.005, -0.01, 0.003, 0.01]^T \end{aligned}$$

For $\mathbf{x}_2$, $1 - y_2\mathbf{w}^{1T}\mathbf{x}_2 = 1 - (-1)[0.005, -0.01, 0.003, 0.01][-1, -2, -2, 1]^T = 0.979 > 0$, we have

$$\begin{aligned} \nabla J &= [0.005, -0.01, 0.003, 0] - CNy_2\mathbf{x}_2 = [-0.995, -2.01, -1.997, 1]^T \\ \mathbf{w}^2 &= \mathbf{w}^1 - 0.005[-0.995, -2.01, -1.997, 1]^T = [-0.9900, -1.9999, -1.9870, 0.9950]^T \end{aligned}$$

For $\mathbf{x}_3$, $1 - y_3\mathbf{w}^{2T}\mathbf{x}_3 = 1 - [-0.9900, -1.9999, -1.9870, 0.9950][1.5, 0.2, -2.5, 1]^T = 1 - 4.0775 < 0$, we have

$$\begin{aligned} \nabla J &= [\mathbf{w}^{2T}, 0]^T = [-0.9900, -1.9999, -1.9870, 0]^T \\ \mathbf{w}^3 &= \mathbf{w}^2 - 0.0025[-0.9900, -1.9999, -1.9870, 0]^T = [-0.9875, -1.9949, -1.9820, 0.9950]^T \end{aligned}$$

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?
The perceptron updating rule is $\mathbf{w}_{t+1} = \mathbf{w}_t + r(y_i\mathbf{x}_i)$ if $y_i \neq \text{sgn}(\mathbf{w}_t^T\mathbf{x}_i)$. If we assume r stays unchanged during the iterations, then the order of making mistakes on each sample during multiple epochs does not affect the final learned weight vector. Hence, with $\mathbf{w}_0 = \mathbf{0}$, we have

$$\mathbf{w}_{\text{learned}} = \mathbf{w}_0 + \sum_{i=1}^{N} rc_i(y_i\mathbf{x}_i) = r\sum_{i=1}^{N} c_i y_i \mathbf{x}_i$$

and the prediction is $\text{sgn}(\mathbf{w}_{\text{learned}}^T\mathbf{x}) = \text{sgn}(r\sum_{i=1}^{N} c_i y_i \mathbf{x}_i^T\mathbf{x})$, which implies that *only the samples with mistakes affect the the learned weight vector*, which is similar to that only the support vectors decides the final weight vector in the SVM. Note that r does not affect the prediction.

- [3 points] Can you develop an algorithm that uses mistake counts to learn the Perceptron? Please list the pseudo code.
INPUT: the set of training data $\mathbf{x}_i \in \mathbf{R}^{n+1}$ (augmented with constant feature 1) and $y_i \in \{-1, +1\}$
• INITIALIZATION: the mistake counters $c_i = 0, \forall i \in [N]$.
• For each sample $(\mathbf{x}_k, y_k)$:
— prediction: $\bar{y}_k = \text{sgn}(r\sum_{i=1}^{N} c_i y_i \mathbf{x}_i^T\mathbf{x}_k)$
— If $\bar{y}_k \neq y_k$: update $c_k = c_k + 1$
• RETURN: final mistake counts $c_i, \forall i \in [N]$ and $\mathbf{w}_{\text{learned}} = r\sum_{i=1}^{N} c_i y_i \mathbf{x}_i$.

- [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?
Since the prediction is the just the dot product of the some training samples and the test sample, we can directly apply the kernel trick $\text{sgn}(\mathbf{w}_{\text{learned}}^T\phi(\mathbf{x})) = \text{sgn}(r\sum_{i=1}^{N} c_i y_i K(\mathbf{x}_i, \mathbf{x}_k))$ for the prediction of $\mathbf{x}_k$. The algorithm is as follows:
INPUT: the set of training data $\mathbf{x}_i \in \mathbf{R}^{n+1}$ (augmented with constant feature 1) and $y_i \in \{-1, +1\}$, and the feature mapping $\phi(.)$.
• INITIALIZATION: the mistake counters $c_i = 0, \forall i \in [N]$.
• For each sample $(\phi(\mathbf{x}_k), y_k)$:
— prediction: $\bar{y}_k = \text{sgn}(r\sum_{i=1}^{N} c_i y_i K(\mathbf{x}_i, \mathbf{x}_k))$
— If $\bar{y}_k \neq y_k$: update $c_k = c_k + 1$
• RETURN: final mistake counts $c_i, \forall i \in [N]$ and $\mathbf{w}_{\text{learned}} = r\sum_{i=1}^{N} c_i y_i \phi(\mathbf{x}_i)$.
Note that we do not need to explicitly compute $\phi(\mathbf{x}_i)$ in $\mathbf{w}_{\text{learned}}$ though it is expresses as the summation of $\phi(\mathbf{x}_i)$s.

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. Please refer to `https://github.com/xiangzhang-122/Machine_Learning`.

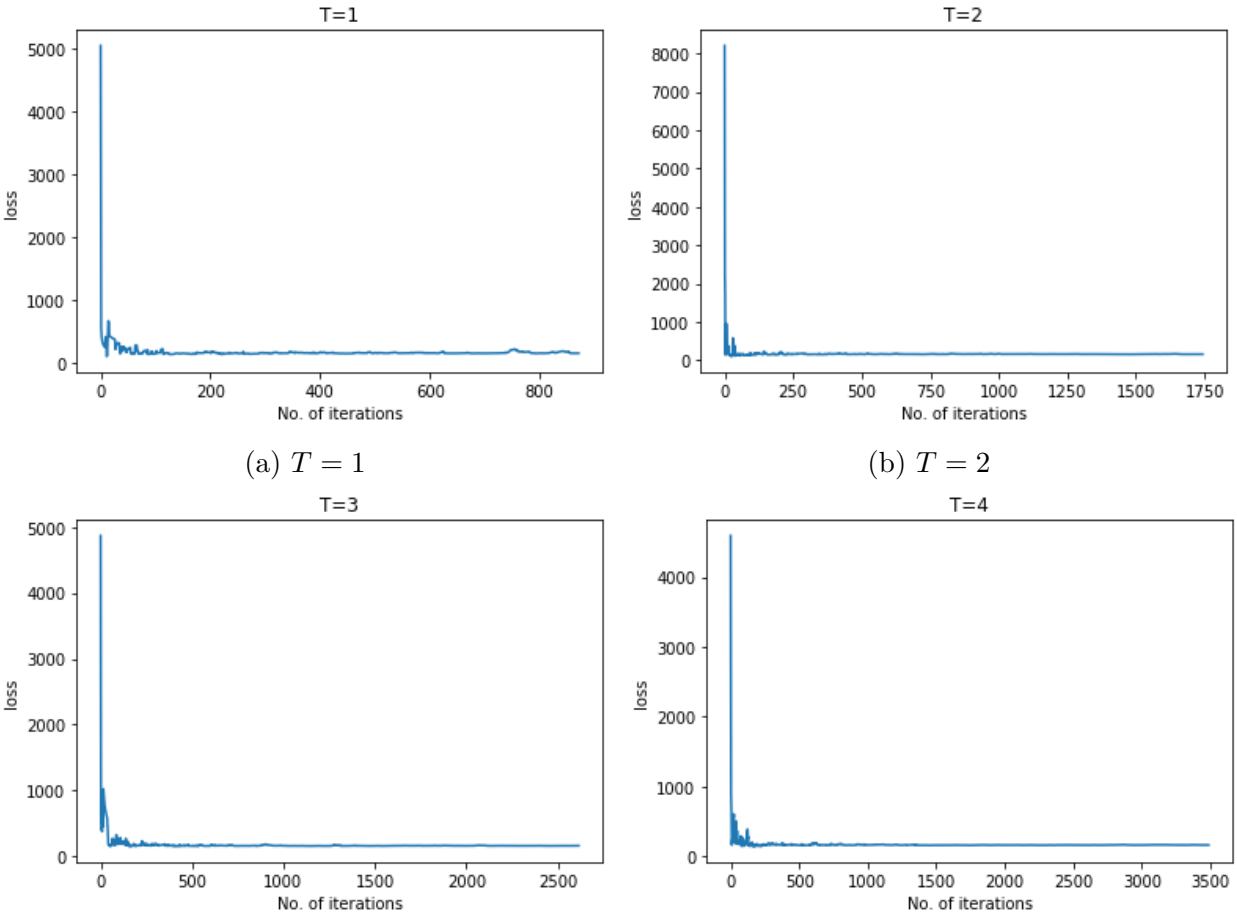
2. [28 points] We will first implement SVM in the primal domain with stochastic subgradient descent. We will reuse the dataset for Perceptron implementation, namely, "bank-note.zip" in Canvas. The features and labels are listed in the file "classification/data-desc.txt". The training data are stored 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 (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. Choose $\gamma_0 = 2.3, d = 1$ and hence $\gamma_t = \frac{2.3}{1+2.3t}$, which guarantees the convergence. The convergence plot is shown below in Fig. 1 for $T = 1, 2, 3, 4$ with $C = \frac{1}{837}$ (we have assumed that this will guarantee convergence for all values of C) and the corresponding weights and loss are

T	weight	loss
1	[-0.3715, -0.1717, -0.1491, -0.1015, 0.0011]	160.85
2	[-0.3755, -0.1732, -0.1409, -0.0824, 0.0000]	161.73
3	[-0.3768, -0.1599, -0.1488, -0.0868, 0.0000]	162.72
4	[-0.3788, -0.1723, -0.1395, -0.0792, 0.0002]	161.34

   Setting $T = 100, \gamma_0 = 2.3, d = 1$, we obtain the errors:

$C \times 873$	train error %	test error %	weight
1	4.93	7.20	[-0.3734, -0.1701, -0.1459, -0.0788, 1.1454e-05]
10	4.13	4.80	[-0.7288, -0.3666, -0.3699, -0.1932, -0.0001]
50	4.01	4.60	[-1.0979, -0.5910, -0.6374, -0.2538, 0.0]
100	3.89	4.60	[-1.2658, -0.6801, -0.7427, -0.2699, 0.0]
300	3.89	4.60	[-1.5827, -0.8694, -0.9339, -0.3470, 0.0]
500	4.01	0.52	[-1.7736, -0.9528, -1.0838, -0.3492, 0.0]
700	3.89	4.60	[-1.8572, -1.0130, -1.0998, -0.4343, 0.0]



(a) $T = 1$ (b) $T = 2$

(c) $T = 3$ (d) $T = 4$

Figure 1: Evidence of convergence using $\gamma_t = \frac{2.3}{1+2.3t}$.

We can see that the training error decreases as C increases which makes sense because large value of C imposes more penalty on the empirical loss (training error), leading to a smaller training error.

(b) [12 points] Use the schedule $\gamma_t = \frac{\gamma_0}{1+t}$. Report the training and test error for each setting of C.
Setting $T = 100, \gamma_0 = 2.3$, we obtain

$C \times 873$	train error %	test error %	weight
1	5.05	7.20	[-0.3737, -0.1712, -0.1451, -0.0793, 0.0]
10	4.13	4.80	[-0.7286, -0.3692, -0.3729, -0.1969, 0.0]
50	4.13	4.80	[-1.1082, -0.5951, -0.6289, -0.2650, 0.0]
100	4.01	5.00	[-1.2968, -0.6737, -0.7496, -0.3138, 0.0]
300	3.89	4.80	[-1.7072, -1.0008, -1.0295, -0.4302, 0.0]
500	4.01	4.80	[-1.9309, -1.0284, -1.1304, -0.4133, 0.0]
700	6.08	6.20	[-2.1974, -1.0915, -1.4061, -0.6400, 0.0]

(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?
1) First we see that for both schedulings, the training error basically decreases as C increases which makes sense because large value of C imposes more penalty on the empirical loss (training error), leading to a smaller training error.
2) For smaller values of $C(\leq \frac{100}{873})$, the learned weight for both schedulings are very close.For larger values of C, there is some difference between the learned weight vectors between the two scheduling.

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?
The errors and weights are

$C * 873$	train err%	test err%	weight
100	7.11	7.80	[-9.4292e-01 -6.5149e-01 -7.3372e-01 -4.1021e-02 9.62e-11]
500	5.62	6.60	[-1.5639 -1.0140 -1.1806 -1.5651e-01 6.60e-09]
700	5.62	6.20	[-2.0425 -1.2807 -1.5135 -2.4906e-01 -1.61e-09]

It can be seen that, for each value of C, 1) both training and testing error are a little higher than Problem 2. in both part (b) and (c); 2) the final weight is roughly the same (but not very close to) as the weight learned previously. More specifically, w_1, w_2, w_3, b are close, except that w_3 is a bit far from the precious weight. I guess this difference is due to the accuracy of the SLSQP opt algorithm. Fortunately, the errors are not very far from those of the primal form.

(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 $\{0.01, 0.1, 0.5, 1, 2, 5, 10, 100\}$ 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?

The kernel SVM dual is

$$\min \quad \frac{1}{2}\boldsymbol{\alpha}^T \widehat{\mathbf{K}} \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha} \tag{25}$$

$$s.t. \quad \mathbf{0} \leq \boldsymbol{\alpha} \leq C\boldsymbol{e} \tag{26}$$

$$\boldsymbol{y}^T \boldsymbol{\alpha} = 0 \tag{27}$$

where $\widehat{\mathbf{K}} := [\widehat{K}_{ij}]_{N\times N}, \widehat{K}_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) = y_i y_j \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\gamma})$, $\boldsymbol{e}$ denotes the N-dimensional all-one vector and $\boldsymbol{y} := [y_1, y_2, \cdots, y_N]^T$. The decision variable is $\boldsymbol{\alpha} := [\alpha_1, \alpha_2, \cdots, \alpha_N]^T$.

The training and test errors are listed as follows:

train err/test err%	$C = \frac{100}{873}$	$\frac{500}{873}$	$\frac{700}{873}$
$\gamma = 0.01$	0.0/0.0	0.0/0.0	0.0/0.0
0.1	0.0/0.2	0.0/0.2	0.0/0.2
0.5	0.0/0.2	0.0/0.2	0.0/0.2
1	0.0/0.2	0.0/0.2	0.0/0.2
2	0.0/0.2	0.0/0.2	0.0/0.2
5	0.8/0.6	0.34/0.6	0.34/0.6
10	1.49/1.60	1.49/1.8	1.49/1.8
100	31.54/28	29.81/26.60	29.82/26.60

It can be seen that the best combinations (when $\gamma = 0.01$ and for all C) achieve zero training and test error simultaneously, which implies under this feature mapping the data becomes strict linearly separable. Compared with the linear SVM,

non-linear SVM has much smaller training and test errors, implying that the data becomes much more (linearly) separable after the infinite feature mapping of Gaussian kernel.

(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?
The number of support vectors is

Num. support vectors	$C = \frac{100}{873}$	$\frac{500}{873}$	$\frac{700}{873}$
$\gamma = 0.01$	872	872	872
0.1	869	871	870
0.5	858	751	790
1	821	613	781
2	749	619	766
5	722	527	611
10	552	334	514
100	459	444	450

The number of overlapped support vectors between γ_i and $\gamma_{i+1}, \forall i \in [7]$ is

γ	0.01	0.1	0.5	1	2	5	10	100
Num. overlaps	871	750	592	487	398	228	164	–
ratio:=$\frac{\#\text{ overlaps}}{\#\text{ supp}_\text{v}\text{ectors}}$	99.8%	86.1%	78.8 %	79.4%	64.3%	43.3%	49.1%	–

It can be seen that for $C = \frac{500}{873}, \frac{700}{873}$ and for the same γ, the larger C is, the more support vectors we will have. However, this does not hold for $C = \frac{100}{873}$, where the reason might be: in the optimal dual variables $\{\alpha_i^*\}_{i\in[N]}$, some of them have very small values. Though small, these are not counted as support vectors. Hence, the number of counted support vectors might be larger than the actual number.

(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?
The training and test errors (in the form "train error/test error(%)"m the number of epochs for training is $T = 1$) are

γ	0.01	0.1	0.5	1	2	5	10	100
error	0.23/0.0	0.0/0.4	0.0/0.4	0.0/0.4	0.0/0.2	0.0/0.2	0.0/0.2	0.46/0.40

It can be seen that 1) both the training and test errors are very small using the Gaussian kernel. One difference is that nonlinear SVM has much higher errors for large values of γ, which does not seem to affect kernel perceptron very much; 2) these errors are much smaller than the standard perceptron algorithm, implying

that the the application of feature mapping of Gaussian kernel makes the data more linearly separable, leading to smaller training and prediction errors.