CS 5350/6350: Machine Learning Fall 2024

Homework 4

Handed out: 12 Nov, 2024
Due date: 11:59pm, 26 Nov, 2024

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?

   If $\mathbf{x}_i$ breaks into the margin, then $y_i(\mathbf{w}^\top\mathbf{x}_i + b) < 1$ so $\xi_i > 0$.

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

   If $\mathbf{x}_i$ is on or outside the margin, then $y_i(\mathbf{w}^\top\mathbf{x}_i + b) \ge 1$ so $\xi_i = 0$.

   (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?

   This term represents how well the classifier fits the data. C controls the balance between correct classification and margin maximization. If we remove this term, then the optimal w is the zero vector.

2. [6 points] Write down the dual optimization problem for soft SVMs. Please clearly indicate the constraints, and explain how it is derived.

We start with the SVM problem formulation:

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

Using Lagrange multipliers, this is equivalent to:

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

We can switch the max and min to get the dual optimization problem. It is equivalent to this primal problem because it meets Slater's condtion.

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

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?

   You can compute $y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right)$ and see if it is greater than 1. If so, the example is outside the margin. You can also determine this by looking at the α_i values. Any that are zero indicate examples outside the margin.

   (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.

   You can examine the α_i and $\xi_i$values. Any α_i values that are nonzero correspond to examples that are on or inside the margin. Any ξ_i values that are zero correspond to examples that are on or outside the margin. Therefore, examples exactly on the margin will have $\alpha_i > 0$ and $\xi_i = 0$.

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

   The optimation problem for linear SVM is

$$\min_{0\le\alpha_i\le C} \frac{1}{2}\sum_i\sum_j y_iy_j\alpha_i\alpha_j\mathbf{x}_i^\top\mathbf{x}_j - \sum_i \alpha_i$$

   Notice how this requires computing the dot product between 2 examples $\mathbf{x}_i$ and $\mathbf{x}_j$. We can introduce nonlinearity by replacing that dot product with a kernel function that efficiently computes a dot product of the examples

as if they have been mapped to a higher dimensional space. We do this in such a way so that we do not explicitly compute the mapping. This is referred to as the "kernel trick". The resulting optimization problem is

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

5. [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 $\{0.01, 0.005, 0.0025\}$ and hyperparameter $C = 1$. 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

I am assuming that "first three steps" means first three epochs since 3 learning rates are given and the SGD algorithm changes the learning rate after each epoch. I have named the three training examples $\mathbf{x}_A$, $\mathbf{x}_B$, and $\mathbf{x}_C$ and augmented them with a constant feature 1.

$$\mathbf{x}_A = \{0.5, -1, 0.3, 1\} \quad y_A = 1$$

$$\mathbf{x}_B = \{-1, -2, -2, 1\} \quad y_B = -1$$

$$\mathbf{x}_C = \{1.5, 0.2, -2.5, 1\} \quad y_C = 1$$

$$\mathbf{w}^0 = [0, 0, 0]$$

$$\mathbf{w} = [\mathbf{w}^0; 0]$$

EPOCH 1
Shuffled data $= \{\mathbf{x}_B, \mathbf{x}_C, \mathbf{x}_A\}$**. Loop through all examples.**
Compute $y_B \cdot \mathbf{w}^\top \mathbf{x}_B = -1 \cdot 0 = 0 \leq 1$
Compute $\nabla J = [\mathbf{w}^0; 0] - CNy_B\mathbf{x}_B = \mathbf{0} - 1 \cdot 3 \cdot -1 \cdot \mathbf{x}_B = [-3, -6, -6, 3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = \mathbf{0} - 0.01 \cdot [-3, -6, -6, 3] = [0.03, 0.06, 0.06, -0.03]$

Compute $y_C \cdot \mathbf{w}^\top \mathbf{x}_C = 1 \cdot -0.123 = -0.123 \leq 1$
Compute $\nabla J = [0.03, 0.06, 0.06, 0] - 1 \cdot 3 \cdot 1 \cdot \mathbf{x}_C = [-4.47, -0.54, 7.56, -3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.03, 0.06, 0.06, -0.03] - 0.01 \cdot \nabla J = [0.0747, 0.0654, -0.0156, 0]$

Compute $y_A \cdot \mathbf{w}^\top \mathbf{x}_A = 1 \cdot -0.0327 = -0.0327 \leq 1$
Compute $\nabla J = [0.0747, 0.0654, -0.0156, 0] - 1 \cdot 3 \cdot 1 \cdot \mathbf{x}_A = [-2.4253, 3.0654, -0.9156, -3]$

Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.0747, 0.0654, -0.0156, 0] - 0.01 \cdot \nabla J = [0.0989, 0.0347, -0.0247, 0.03]$

EPOCH 2
Shuffled data $= \{\mathbf{x}_A, \mathbf{x}_C, \mathbf{x}_B\}$**. Loop through all examples.**

Compute $y_A \cdot \mathbf{w}^\top \mathbf{x}_A = 1 \cdot 0.03734 = 0.03734 \leq 1$
Compute $\nabla J = [0.0989, 0.0347, -0.0247, 0] - 1 \cdot 3 \cdot 1 \cdot \mathbf{x}_A = [-1.4011, 3.0347, -0.9247, -3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.0989, 0.0347, -0.0247, 0.03] - 0.005 \cdot \nabla J = [0.1059, 0.0195, -0.0201, 0.045]$

Compute $y_C \cdot \mathbf{w}^\top \mathbf{x}_C = 1 \cdot 0.258 = 0.258 \leq 1$
Compute $\nabla J = [0.1059, 0.0195, -0.0201, 0] - 1 \cdot 3 \cdot 1 \cdot \mathbf{x}_C = [-4.3941, -0.5805, 7.299, -3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.1059, 0.0195, -0.0201, 0.045] - 0.005 \cdot \nabla J = [0.0326, 0.0224, -0.0566, 0.06]$

Compute $y_B \cdot \mathbf{w}^\top \mathbf{x}_B = -1 \cdot 0.0958 = -0.0958 \leq 1$
Compute $\nabla J = [0.0326, 0.0224, -0.0566, 0] - 1 \cdot 3 \cdot -1 \cdot \mathbf{x}_B = [-2.9674, -5.9776, -6.0566, -3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.0326, 0.0224, -0.0566, 0.06] - 0.005 \cdot \nabla J = [0.0474, 0.0524, -0.0263, 0.075]$

EPOCH 3
Shuffled data $= \{\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C\}$**. Loop through all examples.**

Compute $y_A \cdot \mathbf{w}^\top \mathbf{x}_A = 1 \cdot 0.03851 = 0.03851 \leq 1$
Compute $\nabla J = [0.0474, 0.0524, -0.0263, 0] - 1 \cdot 3 \cdot 1 \cdot \mathbf{x}_A = [-1.4526, 3.0523, -0.9263, -3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.0474, 0.0524, -0.0263, 0.075] - 0.0025 \cdot \nabla J = [0.051, 0.0447, -0.0240, 0.08$

Compute $y_B \cdot \mathbf{w}^\top \mathbf{x}_C = -1 \cdot -0.0099 = 0.0099 \leq 1$
Compute $\nabla J = [0.051, 0.0447, -0.0240, 0] - 1 \cdot 3 \cdot -1 \cdot \mathbf{x}_B = [-2.949, -5.9553, -6.024, 3]$
Update $\mathbf{w} = \mathbf{w} - \gamma_t \nabla J = [0.1059, 0.0195, -0.0201, 0.045] - 0.0025 \cdot \nabla J = [0.0584, 0.0596, -0.0073, 0.0$

Compute $y_C \cdot \mathbf{w}^\top \mathbf{x}_B = 1 \cdot 0.19277 = 0.19277 \leq 1$
Compute $\nabla J = [0.0584, 0.0596, -0.0073, 0] - 1 \cdot 3 \cdot 1 \cdot \mathbf{x}_C = [-4.4416, -0.5404, 7.4927, -3]$
Update $\mathbf{w} = [0.0584, 0.0596, -0.0073, 0.075] - 0.0025 \nabla J = [0.0695, 0.0609, -0.026, 0.0825]$

2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library.

   DONE

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 $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. 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 $\gamma_0 > 0$ and $a > 0$ to ensure convergence. For each setting of C, report your training and test error.

After much experimentation, I chose $\gamma_0 = 0.00001$ and $a = 0.000008$. For $C = \frac{100}{873}$, training error is 1.49% and test error is 1.40%. For $C = \frac{500}{873}$, training error is 0.80% and test error is 0.80%. For $C = \frac{700}{873}$, training error is 0.92% and test error is 0.80%.

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

After much experimentation, I chose $\gamma_0 = 0.000005$. For $C = \frac{100}{873}$, training error is 2.18% and test error is 2.00%. For $C = \frac{500}{873}$, training error is 0.92% and test error is 0.80%. For $C = \frac{700}{873}$, training error is 0.80% and test error is 0.80%.

(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?

$C = \frac{100}{873}$
The first schedule learned $[0.519, -0.637, -0.374, -0.408, -0.091]$ **with training error 1.29% and test error 1.40%.**
The second schedule learned $[0.366, -0.579, -0.308, -0.330, -0.103]$ **with training error 2.18% and test error 2.00%.**

$C = \frac{500}{873}$
The first schedule learned $[1.108, -0.839, -0.579, -0.638, -0.067]$ **with training error 0.92% and test error 0.80%.**
The second schedule learned $[0.919, -0.740, -0.502, -0.550, -0.085]$ **with training error 0.92% and test error 0.80%.**

$C = \frac{700}{873}$
The first schedule learned $[1.201, -0.910, -0.631, -0.699, -0.063]$ **with training error 0.92% and test error 0.80%.**
The second schedule learned $[1.040, -0.790, -0.539, -0.595, -0.074]$ **with training error 0.80% and test error 0.80%.**

As C increases, the training error decreases because the algorithm is focusing more on fitting the data. As C increases, the norm of the

weight vector also increases which indicates a smaller margin. Smaller C values lead to larger margin because the algorithm is less focused on fitting the data. Overall, test error decreases as C increases because the learned hyperplane is more generalizable when it has a large margin.

3. [30 points] Now let us implement SVM in the dual domain.

   (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? Note that if your code calculates the objective function with a double loop, the optimization can be quite slow. To accelerate, consider writing down the objective in terms of the matrix and vector operations, and treat the Lagrange multipliers that we want to optimize as a vector! Recall, we have discussed about it in our class.

   $C = \frac{100}{873}$
   Dual SVM learned $[2.517, -0.943, -0.651, -0.734, -0.041]$ **with training error 2.64% and test error 3.00%.**

   $C = \frac{500}{873}$
   Dual SVM learned $[3.956, -1.564, -1.014, -1.181, -0.157]$ **with training error 3.10% and test error 3.60%.**

   $C = \frac{700}{873}$
   Dual SVM learned $5.037, -2.043, -1.281, -1.516, -0.249]$ **with training error 3.44% and test error 3.60%.**

   Comparing this to the primal implementation, the learned parameters are very similar. The error is higher for this dual version. The parameters should be the same since primal and dual formulations are equivalent for SVM, however, implementation details can cause them to not be equivalent.

   (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?

For $\gamma = 0.1$ and $C = \frac{100}{873}$ I got training error of 1.61% and a test error of 1.80%.
For $\gamma = 0.1$ and $C = \frac{500}{873}$ I got training error of 1.61% and a test error of 1.80%.
For $\gamma = 0.1$ and $C = \frac{700}{873}$ I got training error of 1.72% and a test error of 1.80%.
For $\gamma = 0.5$ and $C = \frac{100}{873}$ I got training error of 0.34% and a test error of 0.40%.
For $\gamma = 0.5$ and $C = \frac{500}{873}$ I got training error of 0.80% and a test error of 0.60%.
For $\gamma = 0.5$ and $C = \frac{700}{873}$ I got training error of 0.80% and a test error of 0.60%.
For $\gamma = 1$ and $C = \frac{100}{873}$ I got training error of 0% and a test error of 1.60%.
For $\gamma = 1$ and $C = \frac{500}{873}$ I got training error of 0% and a test error of 0%.
For $\gamma = 1$ and $C = \frac{700}{873}$ I got training error of 0% and a test error of 0.40%.
For $\gamma = 5$ and $C = \frac{100}{873}$ I got training error of 20.4% and a test error of 38.8%.
For $\gamma = 5$ and $C = \frac{500}{873}$ I got training error of 29.9% and a test error of 36.2%.
For $\gamma = 5$ and $C = \frac{700}{873}$ I got training error of 30.7% and a test error of 37%.
For $\gamma = 100$ and $C = \frac{100}{873}$ I got training error of 28.9% and a test error of 38.8%.
For $\gamma = 100$ and $C = \frac{500}{873}$ I got training error of 38.2% and a test error of 44.2%.
For $\gamma = 100$ and $C = \frac{700}{873}$ I got training error of 39.0% and a test error of 44.2%.

For $\gamma = 1$ especially, this nonlinear kernel outperforms the linear SVM and manages to get a training and test error of 0. The best combination is $\gamma = 1$ and $C = \frac{500}{873}$. The nonlinear version is more accurate because it can represent data that is not linearly separable. I also noticed that if γ is too high, then the error increases a lot.

(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?

For $\gamma = 0.1$ and $C = \frac{100}{873}$ I got 869 support vectors.
For $\gamma = 0.1$ and $C = \frac{500}{873}$ I got 869 support vectors.
For $\gamma = 0.1$ and $C = \frac{700}{873}$ I got 868 support vectors.
For $\gamma = 0.5$ and $C = \frac{100}{873}$ I got 825 support vectors.

For $\gamma = 0.5$ and $C = \frac{500}{873}$ I got 731 support vectors.
For $\gamma = 0.5$ and $C = \frac{700}{873}$ I got 694 support vectors.
For $\gamma = 1$ and $C = \frac{100}{873}$ I got 805 support vectors.
For $\gamma = 1$ and $C = \frac{500}{873}$ I got 556 support vectors.
For $\gamma = 1$ and $C = \frac{700}{873}$ I got 528 support vectors.
For $\gamma = 5$ and $C = \frac{100}{873}$ I got 442 support vectors.
For $\gamma = 5$ and $C = \frac{500}{873}$ I got 208 support vectors.
For $\gamma = 5$ and $C = \frac{700}{873}$ I got 194 support vectors.
For $\gamma = 100$ and $C = \frac{100}{873}$ I got 290 support vectors.
For $\gamma = 100$ and $C = \frac{500}{873}$ I got 116 support vectors.
For $\gamma = 100$ and $C = \frac{700}{873}$ I got 99 support vectors.

As γ increases, the number of support vectors decreases. Also, C affects the number of support vectors because it determines how large the margin is. A larger margin will include more data points and have more support vectors.