# CS 5350/6350: Machine Learning Fall 2024

Homework 4

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

- 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 $\xi_i$ can take when the training example $\mathbf{x}_i$ breaks into the margin?

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

(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?

**Solution:**

(a) When a training example $\mathbf{x}_i$ breaks into the margin, it violates the margin constraint:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1.$$

From the constraint

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

and since $1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0$, it follows that $\xi_i > 0$. Therefore, the slack variable $\xi_i$ takes **positive values** when $\mathbf{x}_i$ breaks into the margin.

(b) When a training example $\mathbf{x}_i$ stays on or outside the margin, it satisfies the margin constraint:

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

This implies that

$$1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b) \leq 0.$$

Given the constraints $\xi_i \geq 0$ and $1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b) \leq \xi_i$, the only value that satisfies both is $\xi_i = 0$. Therefore, the slack variable $\xi_i$ takes the value **zero** when $\mathbf{x}_i$ stays on or outside the margin.

(c) The term $C \sum_i \xi_i$ in the objective function penalizes the total amount of margin violation across all training examples. Including this term ensures a balance between:

- **Maximizing the margin**: Achieved by minimizing $\frac{1}{2}\|\mathbf{w}\|^2$.
- **Minimizing classification errors or margin violations**: Achieved by minimizing $C \sum_i \xi_i$.

If we omit this term, there would be no penalty for large $\xi_i$, allowing the optimizer to choose large slack variables without any cost. This could lead to a trivial solution where the model ignores misclassifications and margin breaches, resulting in poor generalization and defeating the purpose of the Support Vector Machine. Therefore, incorporating $C \sum_i \xi_i$ is essential to enforce the trade-off between margin maximization and error minimization.

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.)

**Solution:**

The dual optimization problem for soft SVMs is formulated as:

$$\max_{\alpha_i} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j,$$

subject to the constraints:

$$\sum_{i=1}^{N} \alpha_i y_i = 0,$$
$$0 \leq \alpha_i \leq C \quad \forall i \in \{1, 2, \ldots, N\}.$$

**Explanation:**

The dual problem is derived by introducing Lagrange multipliers to transform the original primal optimization problem into its dual form. Here's the step-by-step process:

1. **Primal Problem**: The primal soft SVM optimization problem is:

$$\min_{\mathbf{w}, b, \xi_i} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i,$$

subject to:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

2. **Lagrangian Formulation**: To handle the inequality constraints, we introduce dual variables $\alpha_i$ (for the margin constraint) and $\mu_i$ (for the slack variables), leading to the Lagrangian:

$$L(\mathbf{w}, b, \xi, \alpha, \mu) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{N} \mu_i \xi_i.$$

3. **Stationarity Conditions**: Taking derivatives of $L$ with respect to $\mathbf{w}$, $b$, and $\xi_i$ and setting them to zero yields:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \quad \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i + \mu_i = C.$$

4. **Eliminate Primal Variables**: Substituting these conditions into the Lagrangian eliminates $\mathbf{w}$, $b$, and $\xi_i$, resulting in the dual problem.

5. **Constraints**: The constraints $\alpha_i \geq 0$ and $\alpha_i + \mu_i = C$ imply $0 \leq \alpha_i \leq C$.

Thus, the dual optimization problem is obtained as stated above. The dual form has fewer variables and is often more efficient to solve, especially when the feature space is high-dimensional.

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?

(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.

**Solution:**

(a) To determine if an example stays outside the margin, examine the dual variable $\alpha_i$. If $\alpha_i = 0$, it indicates that the corresponding training example does not contribute to the optimization and lies outside the margin, as it is correctly classified and does not violate the margin constraints.

(b) To find training examples that just sit on the margin (neither inside nor outside), we need to look for $\alpha_i$ values that satisfy $0 < \alpha_i < C$. This range indicates that the corresponding training examples are support vectors that lie exactly on the margin boundary. This determination is based on the Kuhn-Tucker conditions in the optimization process.

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

**Solution:**

The kernel trick allows SVMs to perform nonlinear classification by implicitly mapping the input data from its original space into a higher-dimensional feature space, where a linear decision boundary can separate the data. This mapping is achieved without explicitly computing the transformation, using a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ that calculates the inner product in the higher-dimensional feature space.

(a) **Kernel Function**: Replace the inner product $\mathbf{x}_i^\top \mathbf{x}_j$ in the dual optimization problem with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. Common kernel functions include:

- Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + c)^d$,
- Gaussian (RBF) kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$,
- Sigmoid kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i^\top \mathbf{x}_j + c)$.

(b) **Dual Optimization Problem**: Using the kernel trick, the dual optimization problem becomes:

$$\max_{\alpha_i} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j),$$

subject to the constraints:

$$\sum_{i=1}^{N} \alpha_i y_i = 0,$$

$$0 \leq \alpha_i \leq C \quad \forall i \in \{1, 2, \ldots, N\}.$$

(c) **Nonlinear Decision Boundary**: Once the optimal dual variables $\alpha_i$ are obtained, the decision function in the transformed space is:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b,$$

where $K(\mathbf{x}_i, \mathbf{x})$ captures the nonlinear relationship between the data points.

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. **Solution:**

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

Table 1: Dataset

To compute the sub-gradients of the SVM objective with respect to the model parameters using stochastic sub-gradient descent, we analyze the hinge loss and regularization terms for each step.

The SVM objective function is:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)).$$

The sub-gradient of the objective function w.r.t. $\mathbf{w}$ and $b$ depends on whether the training example satisfies the margin constraint $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$. For each training example:

$$\mathbf{g}_\mathbf{w} = \begin{cases} \mathbf{w} - Cy_i\mathbf{x}_i, & \text{if } y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1, \\ \mathbf{w}, & \text{otherwise.} \end{cases}$$

$$g_b = \begin{cases} -Cy_i, & \text{if } y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1, \\ 0, & \text{otherwise.} \end{cases}$$

The steps for computing the sub-gradients are as follows:

**1. Step 1:** - Initialize $\mathbf{w}_0 = 0$, $b_0 = 0$. - Compute $\mathbf{w}^\top \mathbf{x}_1 + b$ for the first training example. - Based on the hinge loss condition, calculate $\mathbf{g}_\mathbf{w}$ and $g_b$.

**2. Step 2:** - Update parameters: $\mathbf{w}_1 = \mathbf{w}_0 - \eta_1 \mathbf{g}_\mathbf{w}$, $b_1 = b_0 - \eta_1 g_b$. - Repeat the process for the second training example to compute sub-gradients.

**3. Step 3:** - Update parameters: $\mathbf{w}_2 = \mathbf{w}_1 - \eta_2 \mathbf{g}_\mathbf{w}$, $b_2 = b_1 - \eta_2 g_b$. - Compute sub-gradients for the third example.

For each step, compute the sub-gradients explicitly using the given training data, substituting the appropriate $\mathbf{x}_i$, $y_i$, and learning rate ($\eta_1 = 0.01$, $\eta_2 = 0.005$, $\eta_3 = 0.0025$).

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, $\{c_1, \ldots, c_N\}$, 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?

   **Solution:**

   To derive the dual form for the Perceptron algorithm, we represent the weight vector $\mathbf{w}$ as a linear combination of the training examples, leveraging the mistake counts $c_i$.

   **Derivation: 1. Perceptron Update Rule:** In the standard Perceptron algorithm, the weight vector $\mathbf{w}$ is updated for a misclassified example $(\mathbf{x}_i, y_i)$ as:

$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i.$$

   If the example is misclassified multiple times, the same update is applied repeatedly. Let $c_i$ denote the number of times $(\mathbf{x}_i, y_i)$ has been misclassified during training.

   **2. Representing $\mathbf{w}$ in Terms of $c_i$:** Starting with $\mathbf{w} = \mathbf{0}$, after processing all updates, $\mathbf{w}$ can be expressed as:

$$\mathbf{w} = \sum_{i=1}^{N} c_i y_i \mathbf{x}_i,$$

   where $N$ is the number of training examples.

   **3. Decision Function:** The decision function for the Perceptron is:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x}) = \text{sign}\left( \sum_{i=1}^{N} c_i y_i (\mathbf{x}_i^\top \mathbf{x}) \right).$$

   Here, the dot product $(\mathbf{x}_i^\top \mathbf{x})$ implies that the Perceptron decision rule depends only on the inner product of training and test data points, making it similar to kernel-based methods.

4. Dual Form: In the dual representation: - The weights $\mathbf{w}$ are replaced with the coefficients $c_i$. - The update rule is now interpreted as incrementing $c_i$ whenever $(\mathbf{x}_i, y_i)$ is misclassified:

$$c_i \leftarrow c_i + 1 \quad \text{if } y_i \sum_{j=1}^{N} c_j y_j (\mathbf{x}_j^\top \mathbf{x}_i) \leq 0.$$

5. Advantages of the Dual Form: - The dual form expresses the decision boundary in terms of the training examples, enabling the use of kernel functions to handle nonlinear decision boundaries. - The algorithm can generalize to high-dimensional feature spaces without explicitly mapping the data, by replacing the inner product $(\mathbf{x}_i^\top \mathbf{x}_j)$ with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$.

6. Kernelized Perceptron: By using a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, the decision function becomes:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} c_i y_i K(\mathbf{x}_i, \mathbf{x})\right).$$

The update rule is modified to:

$$c_i \leftarrow c_i + 1 \quad \text{if } y_i \sum_{j=1}^{N} c_j y_j K(\mathbf{x}_j, \mathbf{x}_i) \leq 0.$$

This derivation shows how the Perceptron algorithm can be reformulated into a dual form and extended to nonlinear classification using kernels.

# 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 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 $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. 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.

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

(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?

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? 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.

(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 $\gamma$ 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 $\gamma$ 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?

(c) [5 points] Following (b), for each setting of $\gamma$ and $C$, list the number of support vectors. When $C = \frac{500}{873}$, report the number of overlapped support vectors between consecutive values of $\gamma$, 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?

(d) [**Bonus**] [10 points] Implement the kernel Perceptron algorithm you developed in Problem 8 (Section 1). Use Gaussian kernel and test $\gamma$ 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?