CS 5350/6350: Machine Learning Fall 2023

Homework 4

By: Milena Belianovich
Date: 11/20/2023

- 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?

(b) [3 point] What values ξ_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 means that the example is either on the wrong side of the margin or has crossed over to the wrong side of the decision boundary. Therefore, $\xi_i \geq 0$.

If $\mathbf{x}_i$ is on the wrong side of the decision boundary, $\xi_i \geq 1$, since $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0$ making $1 - \xi_i < 0$. The further into the margin or the wrong side $\mathbf{x}_i$ is, the larger the ξ_i gets/.

(b) For training examples that are correctly classified and either on the margin or outside the margin (but on the correct side), $\xi_i \approx 0$. In this situation, $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 0$ and therefore $1 - \xi_i \leq 1$, which holds true if $\xi_i = 0$.

(c) The term $C \cdot \sum_i \xi_i$ is added to the objective function to penalize the misclassification or the margin violations. C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.

If we remove this term, the optimization would solely focus on maximizing the margin without considering the classification errors or violations of the margin, i.e. the SVM would revert to a hard-margin SVM, which requires perfect linear separability of the data. This would likely result in a model that does not generalize well to unseen data, especially if the training 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.)

Solution. The dual problem is derived from the primal problem (mentioned in the previous problem) by applying the Lagrange multipliers method. The primal problem involves direct optimization over the weights $\mathbf{w}$ and bias b, along with the slack variables ξ_i. In the dual problem, we use Lagrange multipliers for the constraints of the primal problem, leading to an optimization problem in terms of these multipliers.

Formulation:

The dual problem for a soft-margin SVM can be formulated as the following:

$$\max_{\alpha}[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j-1}^{N} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle] \tag{1}$$

with the following constraints:

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

$$0 \leq \alpha_i \leq C, \forall i = 1, \ldots, N$$

Here α_i are the Lagrange multipliers for the training examples, and C is the regularization parameter from the primal problem.

Constraint explanation:

The constraint $\sum_{i=1}^{N} \alpha_i y_i = 0$ comes from the condition for the optimality regarding the bias term b in the primal problem. It ensures that the solution lies in the decision boundary hyperplane.

The constraints $0 \leq \alpha_i \leq C$ are derived from the primal problem's constraints related to the slack variables ξ_i. They ensure that each multiplier respects the margin violation penalty and the regularization parameter C.

Derivation:

1. Lagrangian Formulation: Start with the primal problem and introduce Lagrange multipliers for each of the constraints. This results in the Lagrangian, which is a function of $\mathbf{w}$, b, $\{\xi_i\}$, and the Lagrange multipliers.

2. Conditions for Optimality: Compute the partial derivatives of the Lagrangian with respect to $\mathbf{w}$, b, and $\{\xi_i\}$, and set them to zero. These are the KKT (Karush-Kuhn-Tucker) conditions for optimality.

3. Substituting Back into the Lagrangian: Substitute the expressions obtained from the conditions for optimality back into the Lagrangian. This step eliminates $\mathbf{w}$, b, and $\{\xi_i\}$ from the problem, resulting in a formulation that depends only on the Lagrange multipliers α_i.

4. Dual Problem: What results is the dual problem, which is a maximization problem in terms of the α_is. This dual problem is easier to solve computationally, especially when dealing with kernels for non-linear classification.

The dual formulation is particularly powerful because it allows the use of kernel functions to enable SVMs to perform non-linear classification. This approach is computationally more efficient, especially for large datasets, and it inherently supports the extension to non-linear classifiers through the kernel trick.

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) The value of the Lagrange multiplier α_i for each training example $\mathbf{x}_i$ indicates its position relative to the margin. For a training example that lies outside the margin (correctly classified and not a support vector), the corresponding Lagrange multiplier α_i will be 0. This is because such examples do not contribute to the decision boundary in SVMs. In other words, these are the examples that the model is most confident about, lying beyond the reach of the margin and not affecting the position or orientation of the decision boundary.

(b) To find out which training examples are exactly on the margin without examining the functional margin $y_i(\mathbf{w}^T\mathbf{x}_i + b)$, we can look at the values of α_i. Training examples that sit exactly on the margin are the support vectors that define the decision boundary. They are characterized by Lagrange multipliers α_i that are greater than 0 but less than C, i.e. $0 < \alpha_i < C$. The reason for this is that these examples are marginally compliant with the classification condition. They are pivotal in defining the margin and the decision boundary. The constraint $0 < \alpha_i < C$ is a direct consequence of the slack variables ξ_i being exactly 0 for these points (since they are on the margin). and the dual formulation constraints.

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

   Solution. Using the kernel trick in SVMs to enable nonlinear classification involves modifying the dual optimization problem. The key idea is to replace the linear inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ in the dual problem with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. This kernel function implicitly maps the data to a higher-dimensional space where it might be linearly separable.

   The dual optimization problem with the kernel trick for nonlinear classification is formulated as follows:

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

   with the following constraints:

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

$$0 \leq \alpha_i \leq C, \forall i = 1, \ldots, N$$

   where $K(x_i, x_j))$ is the kernel function, which computes the dot product between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the transformed feature space.

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

Solution. The primal form of the linear SVM objective function is:

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

The sub-gradients of the SVM objective with respect to the model parameters $\mathbf{w}$ and b for the first three steps of the stochastic sub-gradient descent algorithm:

1. Step 1 learning rate = 0.01:

- Sub-gradient w.r.t $\mathbf{w}$: $[-0.5, 1.0, -0.3]$

- Sub-gradient w.r.t b: -1.0

2. Step 2 learning rate = 0.005:

- Sub-gradient w.r.t $\mathbf{w}$: $[-0.995, -2.01, -1.997]$

- Sub-gradient w.r.t b: 1.0

3.. Step 3 learning rate = 0.005:

- Sub-gradient w.r.t $\mathbf{w}$: $[-1.490025, -0.199995, 2.512985]$

- Sub-gradient w.r.t b: -1.0

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 for learning this kernel Perceptron?

Solution.

(a) Given the mistake counts $\{c_1, \ldots, c_N\}$ for all training examples, the weight vector $\mathbf{w}$ can be the following: $\mathbf{w} = \sum_{i=1}^{N} c_i y_i \mathbf{x}_i$. To make predictions for a new example $\mathbf{x}$, we use the sign of $\mathbf{w}^T\mathbf{x}$, which is equivalent to the sign of $\sum_{i=1}^{N} c_i y_i (\mathbf{x}_i^T\mathbf{x})$

(b) The pseudo code for a Perceptron algorithm that uses mistake counts is shown below.

Data: $c_i = 0$ for all examples
for *each epoch or until convergence* **do**
  for *each training example* (x_i, y_i) **do**
    if $y_i *$ *(sum over* $j(c_j * y_j * dot(x_j, x_i))) \leftarrow 0$ **then**
      $c_i = c_i + 1$
    end
  end
end

Algorithm 1: Problem b

(c) Applying the kernel trick to develop a nonlinear Perceptron involves using a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ to compute the dot product in a transformed feature space. This allows the algorithm to learn nonlinear decision boundaries.

To conduct classification in this setting, the following expression is used:

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

The pseudo code for learning this kernel Perceptron would be:

Data: $c_i = 0$ for all examples
Choose a kernel function K
for *each epoch or until convergence* **do**
  for *each training example* (x_i, y_i) **do**
    if $y_i *$ *(sum over* $j(c_j * y_j * K(x_j, x_i))) \leftarrow 0$ **then**
      $c_i = c_i + 1$
    end
  end
end

Algorithm 2: Problem a

In this algorithm, the kernel function K allows us to implicitly map the data into a higher-dimensional space and compute dot products in this space without explicitly transforming the data points. This approach can effectively capture complex patterns and nonlinear relationships in the data.

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.

Solution. The changes made to the GitHub repository, can be seen in said GitHub repository: https://github.com/milenabel/CS6350-ML2023.git

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.

   Solution. Values for γ_0 and a when converging first happened for each of the C values:

   $C = 0.1145475372279496$, $\gamma_0 = 0.1$, $a = 1$

   $C = 0.572737686139748$, $\gamma_0 = 0.1$, $a = 1$

   $C = 0.8018327605956472$, $\gamma_0 = 0.1$, $a = 1$

C	Training Error	Testing Error
$\frac{100}{873}$	0.15252293577981652	0.17
$\frac{500}{873}$	0.07110091743119266	0.074
$\frac{700}{873}$	0.06995412844036697	0.086

Table 2: learning rate: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{a}t}$

   The rest of the solution can be seen in Table 2.

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

   Solution. Values for γ_0 and a when converging first happened for each of the C values:

   $C = 0.1145475372279496$, $\gamma_0 = 0.1$, $a = 1$

   $C = 0.572737686139748$, $\gamma_0 = 0.1$, $a = 1$

   $C = 0.8018327605956472$, $\gamma_0 = 0.1$, $a = 1$

   The rest of the solution can be seen in Table 3.

   (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	Training Error	Testing Error
$\frac{100}{873}$	0.3795871559633027	0.378
$\frac{500}{873}$	0.30160550458715596	0.284
$\frac{700}{873}$	0.22821100917431192	0.23

Table 3: learning rate: $\gamma_t = \frac{\gamma_0}{1+t}$

Solution.

Parameter Differences:

The parameter differences vary with different values of C. Specifically, for $\frac{500}{873}$, the parameter difference is the highest, indicating that the model parameters are more sensitive to this particular value of C when compared to the other values. For $C = \frac{100}{873}$ and $C = \frac{700}{873}$, the parameter differences are lower, suggesting more stability in the learned model parameters at these regularization strengths.

Training and Testing Error Differences:

The training and testing error differences also vary with C. The error differences are relatively higher for $C = \frac{500}{873}$ and lower for $C = \frac{700}{873}$. This indicates that the model's performance is more variable at the intermediate regularization strength ($C = \frac{700}{873}$) and more stable at the higher regularization strength ($C = \frac{700}{873}$).

The higher error difference at $C = \frac{500}{873}$ suggests that this particular regularization strength may not be as effective in terms of achieving consistent performance across different learning rate schedules.

C	Parameter Difference	Training Error Difference	Testing Error Difference
$\frac{100}{873}$	0.23058077240441846	0.2270642201834862	0.208
$\frac{500}{873}$	0.4184984035737148	0.23050458715596328	0.20999999999999996
$\frac{700}{873}$	0.22365662239461243	0.15825688073394495	0.14400000000000002

Table 4: Comparison

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.

Solution.

Weights and Bias: The weights and bias obtained from the dual domain implementation are quite different from those obtained via stochastic sub-gradient descent. This difference is expected since the primal and dual forms tackle the optimization problem from different perspectives. The primal form directly optimizes the decision boundary, while the dual form focuses on the Lagrange multipliers, which are then used to compute the weights and bias. The magnitudes and directions of the weights can vary significantly, especially depending on the nature of the data and the model complexity.

Errors: There is a notable difference in the training and test errors between the two methods. The stochastic sub-gradient descent method shows varied errors based on the learning rate schedules, while the dual domain implementation's errors are consistent across different values of C. However, overall the dual domain implementation errors are higher than the ones of the single domain. It's important to note that the dual implementation may be more sensitive to the choice of kernel and hyperparameters.

In summary, while both approaches aim to solve the same problem, the differences in methodology lead to variations in the model parameters and performance. The choice between them can depend on factors like the nature of the data, the need for a linear or non-linear decision boundary, and the specific requirements of the application at hand.

(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?

Solution. The comparison of all the combinations of different C and γ_0 values can be seen in Table 5.

Observations.

Effect of γ and C: In the nonlinear SVM, the training and test errors vary significantly with different combinations of γ and C. Particularly, lower values of γ (e.g.,

0.1 and 0.5) generally yield better test performance, especially for $C = 500/873$ and $C = 700/873$. In contrast, as γ increases, the test error tends to increase, indicating potential overfitting.

Comparison with Linear SVM: The linear SVM shows relatively consistent performance across different values of C, but the errors are generally higher compared to the best-performing settings of the nonlinear SVM. This suggests that the nonlinear SVM can capture complex patterns in the data more effectively than the linear SVM.

Best Performing Combination: For the nonlinear SVM, the best combination appears to be $C = 500/873$ or $C = 700/873$ with $\gamma = 0.5$ or $\gamma = 0.1$, yielding the lowest test errors.

Conclusions: Nonlinearity Advantage: The nonlinear SVM with a Gaussian kernel can model complex, non-linear decision boundaries more effectively than the linear SVM, leading to better generalization in certain cases.

Hyperparameter Sensitivity: The nonlinear SVM is sensitive to the choice of γ and C. Proper tuning of these parameters is crucial for achieving optimal performance.

Risk of Overfitting: Higher values of γ in the nonlinear SVM lead to overfitting, as indicated by increased test errors. This shows the importance of balancing the model's complexity with its ability to generalize.

The Gaussian kernel in the nonlinear SVM allows for capturing nonlinear relationships in the data, which the linear SVM cannot model. The variance parameter γ in the Gaussian kernel controls the level of nonlinearity. Lower γ values lead to smoother decision boundaries, which can generalize better for certain datasets. The linear SVM's performance is limited by its assumption of linear separability, making it less effective for datasets with complex, nonlinear patterns.

(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?

Solution.

The number of support vectors and their overlap between different γ values provide insight into how the model complexity and decision boundary change with γ. A high number of support vectors typically indicates a more complex model, potentially leading to overfitting, especially for higher values of γ (narrower Gaussian kernel). Overlap in support vectors between different γ values suggests similarity in the decision boundaries learned by the model for those values. If the overlap is high, it indicates that the decision boundary does not change significantly between those γ values. Low overlap might indicate a more significant change in the model's behavior as γ changes.

(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,

C	γ_0	Training Error	Testing Error
$\frac{100}{873}$	0.1	0.045871559633027525	0.044
$\frac{100}{873}$	0.5	0.1559633027522936	0.2
$\frac{100}{873}$	1	0.2775229357798165	0.332
$\frac{100}{873}$	5	0.30619266055045874	0.454
$\frac{100}{873}$	100	0.0	0.528
$\frac{500}{873}$	0.1	0.03784403669724771	0.024
$\frac{500}{873}$	0.5	0.008027522935779817	0.008
$\frac{500}{873}$	1	0.005733944954128441	0.012
$\frac{500}{873}$	5	0.18463302752293578	0.408
$\frac{500}{873}$	100	0.0	0.532
$\frac{700}{873}$	0.1	0.10321100917431193	0.076
$\frac{700}{873}$	0.5	0.0	0.002
$\frac{700}{873}$	1	0.0011467889908256881	0.002
$\frac{700}{873}$	5	0.0	0.22
$\frac{700}{873}$	100	0.0	0.532

Table 5: Comparison 2

what do you observe? what do you conclude and why?

Solution.

Model Complexity and Regularization: The kernel Perceptron, lacking a regularization term, might be expected to overfit more than the SVM. However, in this case, it generalizes well, likely due to the nature of the dataset and the effectiveness of the Gaussian kernel at capturing its structure.

Robustness to Hyperparameters: The kernel Perceptron appears more robust to changes in γ compared to the nonlinear SVM, which may be more sensitive to the choice of both C and γ.

Suitability for Different Data Types: The effectiveness of the kernel Perceptron suggests that it can be a strong candidate for certain types of datasets, particularly where the decision boundary is complex but can be effectively captured by the chosen kernel.

Trade-offs in Model Selection: The choice between a kernel Perceptron and an SVM would depend on the specific dataset, the need for regularization, and the sensitivity to hyperparameters. The SVM's regularization term can be crucial for preventing overfitting, especially in cases with more noise or less clear-cut decision boundaries.

The kernel Perceptron's simplicity and lack of regularization make it less prone to the kind of overfitting that might be expected. Its effectiveness on this dataset indicates that the Gaussian kernel is well-suited to capturing the underlying data structure. The nonlinear SVM's varied performance across different γ and C values highlights the importance of hyperparameter tuning in achieving optimal performance. The regularization term in SVM helps control overfitting but also

introduces sensitivity to C and γ.

γ_0	Training Error	Testing Error
0.1	0.0	0.002
0.5	0.0	0.004
1	0.0	0.004
5	0.0	0.002
100	0.0	0.008

Table 6: Bonus