HW 2

Karl Hiner

October 4, 2023

1 Boosting the perceptron

1.1 a

Since the data are not linearly separable, we do not expect the algorithm to converge. Describe how you choose your stopping criterion.

Answer:

Since convergence is not expected, and since we have a small number of training samples with which to do cross-validation, I chose to simply stop at a fixed number of epochs.

I ran a simple search over an epoch count between 50 and 350, in increments of 25, and found that 50 epochs gave the best performance on the test set. Setting the epochs to 50 gave a test accuracy of 96.49%. I will note, however, that even with one epoch, the perceptron acheived 92.11% accuracy, and the accuracy values fluctuate rather than monotonicly increasing with the number of epochs. This leads me to conclude that the perceptron does not truly learn much after the first pass through the data. However, I still chose to use 50 epochs as the stopping criterion, since it gave the best performance, and it is still fast to execute.

Here is the code I used to run the search:

```
# Search for the number of epochs that gives the best accuracy over the test set.
def find_best_epochs(X_train, y_train, X_test, y_test, min = 50, max = 350, step = 25):
    best_accuracy = 0
    best_epoch = 0

    for epochs in range(min, max + 1, step):
        perceptron = Perceptron(epochs=epochs)
        perceptron.fit(X_train, y_train)
        y_pred = perceptron.predict(X_test)
        accuracy = np.mean(y_pred == y_test)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_epoch = epochs

        print(f"Epochs: {epochs}, Test Accuracy: {accuracy * 100:.2f}%")

    print(f"Best Epochs: {best_epoch}, Best Test Accuracy: {best_accuracy * 100:.2f}%")
    return best_epoch
```

1.2 b

Check your accuracy against a standard implementation such as `sklearn`'s Perceptron function. In particular, check your implementation's accuracy against the classification accuracy (0-1 loss) of `sklearn`'s implementation on the same test data (from above). Explain any hyperparameter choices you make in calling `sklearn`'s Perceptron, and whether these cause a discrepancy in the accuracy.

Answer: After some experimentation, I found a configuration that produced the exact same predictions as `sklearn`'s perceptron model, with any number of epochs:

```
# Custom:
Perceptron(learning_rate=learning_rate, epochs=epochs)

# sklearn:
SklearnPerceptron(eta0=learning_rate, max_iter=epochs, tol=None, shuffle=False)
```

Here, `learning_rate` is used by my custom `Perceptron` class to the update multiplier, and `epochs` is the number of epochs run. To get the same results as `sklearn`'s perceptron, I had to set `tol=None` and `shuffle=False`, since my perceptron does not shuffle the data or do early stopping.

With this configuration, and with 50 epochs over the training data, both my custom perceptron and `sklearn` acheive 96.49% accuracy over the test set.

1.3 c

Consider the data $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m]^T$, where $\boldsymbol{x}_i \sim \mathcal{D}$ are iid. For this part alone, assume that, with probability 1, the data are linearly separable with margin ρ and that $\|\boldsymbol{x}\| \leq R$. Prove that the Perceptron converges in at most $(R/\rho)^2$ steps.

Answer:

1. **Update improvements**: After each update (misclassification),

$$\begin{aligned}
\langle \boldsymbol{w}_{t+1}, \boldsymbol{w}^* \rangle &= \langle \boldsymbol{w}_t + y_t \boldsymbol{x}_t, \boldsymbol{w}^* \rangle \\
&= \langle \boldsymbol{w}_t, \boldsymbol{w}^* \rangle + y_t \langle \boldsymbol{x}_t, \boldsymbol{w}^* \rangle && \text{Inner product is distributive} \\
&\geq \langle \boldsymbol{w}_t, \boldsymbol{w}^* \rangle + \rho && \boldsymbol{X} \text{ linearly separable with margin } \rho
\end{aligned}$$

   This implies that $\langle \boldsymbol{w}_{t+1}, \boldsymbol{w}^* \rangle \geq t\rho$.

2. **Norm increase**: After each update, the squared norm of the weight vector increases by at most R^2:

$$\|\boldsymbol{w}_{t+1}\|^2 = \|\boldsymbol{w}_t\|^2 + 2y_t \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle + \|\boldsymbol{x}_t\|^2 \leq \|\boldsymbol{w}_t\|^2 + R^2$$

   This implies that $\|\boldsymbol{w}_{t+1}\|^2 \leq tR^2$.

Using the above inequalities:

$$\begin{aligned}
\langle \boldsymbol{w}_{t+1}, \boldsymbol{w}^* \rangle &\leq \|\boldsymbol{w}_{t+1}\| \|\boldsymbol{w}^*\| && \text{Cauchy-Schwarz inequality} \\
t\rho &\leq \|\boldsymbol{w}_{t+1}\| \|\boldsymbol{w}^*\| && \text{Inequality 1 above} \\
t\rho &\leq \sqrt{t} R \|\boldsymbol{w}^*\| && \text{Inequality 2 above} \\
t\rho^2 &\leq R^2 \|\boldsymbol{w}^*\|^2 && \text{Simplify} \\
t &\leq \frac{R^2}{\rho^2} && \text{WLOG, set} \|\boldsymbol{w}^*\| = 1
\end{aligned}$$

Note that we can set the norm of the optimal weight vector to 1 without loss of generality because we can scale the weight vector and the margin by the same amount without changing the classification. That is, if $\boldsymbol{w}^*$ is a vector achieving margin ρ, then for some $c \in \mathbb{R}$, $c\boldsymbol{w}^*$ will achieve margin ρ/c. In particular, this will not affect the ratio R/ρ.

Thus, the Perceptron algorithm will converge in at most $(R/\rho)^2$ steps.

1.4 d

Now, we shall implement AdaBoost with your Perceptron code from part (b) as the weak learner.

Give your reasoning for how you would now choose the size of the training dataset for the weak learner. Recall that a weak learner should have an error $< 1/2 - \gamma$, with $\gamma > 0$. Derive a bound for γ in terms of the sample size such that the probability of the weak learner making an error of $1/2 - \gamma$ is $> 1 - \delta$, for some $\delta > 0$. State any PAC learnability assumptions you make.

Answer:

We make two PAC assumptions:

1. **Realizability:** The data distribution allows for a weak learner to achieve an error rate $< 1/2 - \gamma$.
2. **IID:** The data is independently and identically drawn from this distribution.

Let Z_i be the indicator random variable with $Z_i = 1$ if the i-th example is misclassified by the weak learner, and 0 otherwise. Let $\hat{e} = \frac{1}{m}\sum_{i=1}^{m} Z_i$ be the empirical error, and let e be the true error.

Hoeffding's inequality tells us that for any $\varepsilon > 0$,

$$P(|\hat{e} - e| > \varepsilon) \leq 2\exp\left(-2m\varepsilon^2\right).$$

Our objective is to have a true error $e < 1/2 - \gamma$. However, we only have control over the empirical error $\hat{e}$. To link this desired true error with the empirical error, we can freely set $\varepsilon = \gamma$. That is, we aim to ensure the empirical error $\hat{e}$ is within γ of the desired $e < 1/2 - \gamma$, with high probability. Specifically, we want the probability of this not happening to be less than δ:

$$\begin{aligned} 2\exp\left(-2m\gamma^2\right) &\leq \delta \\ -2m\gamma^2 &\leq \ln(\delta/2) \\ \gamma^2 &\geq \frac{-\ln(\delta/2)}{2m} \\ \gamma &\geq \sqrt{-\frac{\ln(\delta/2)}{2m}} \end{aligned}$$

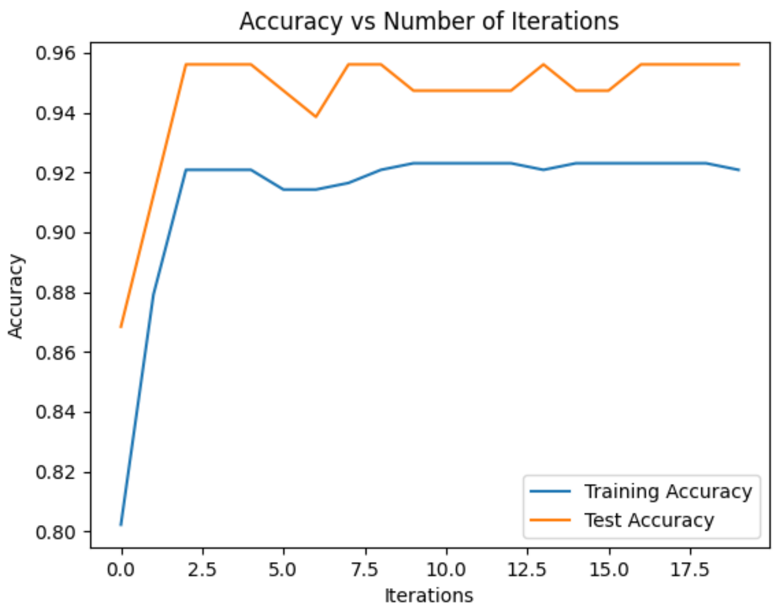
which is the bound for γ in terms of the sample size m and the confidence parameter δ that we are looking for.

1.5 e

e1

Implement your AdaBoost algorithm. Plot the accuracy (training and test) as a function of number of iterations.

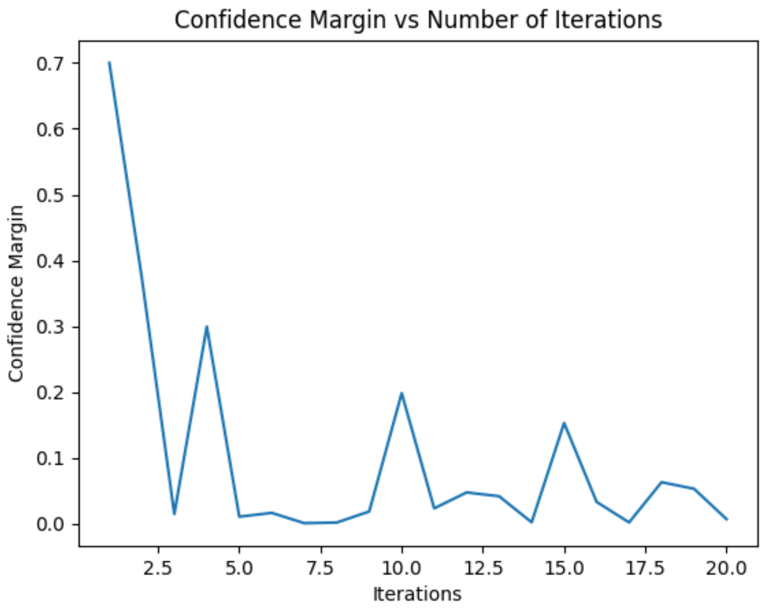
Plotted below. I cannot figure out why my test accuracy is better than my training accuracy, but it casts doubt on my implementation. I'm curious to compare with other implementations or if one is released in the answer guide.

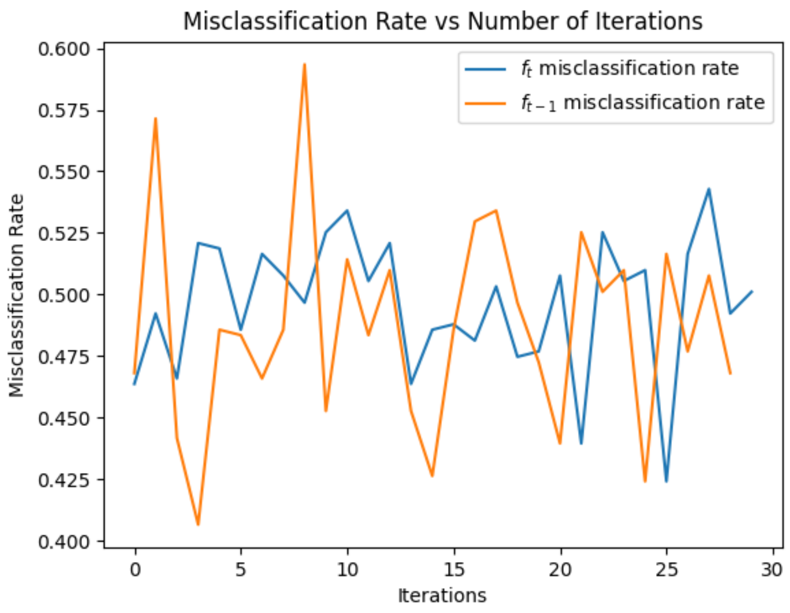


e2

Plot the confidence margin, $\min_{x \in S} \|h(x)\|$, of your predictions vs number of iterations.

Plotted below. I've made many changes to my algorithm, with different results, but the basic shape is consistent - the confidence margin drops down to a relatively consistent near-zero value. I interpret this as AdaBoost finding weak learners that contribute to the accuracy until it reaches the limit of its capacity, at which point it is unable to find any more weak learners that contribute to the accuracy.


Confidence Margin vs Number of Iterations
Confidence Margin
Iterations
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



1.6 f

Let f_t be your Perceptron output at iteration t and D_{t+1} be the updated distribution of the training points. Generate iid samples from the D_{t+1} in your code and plot the percentage of points misclassified by f_t, f_{t-1} at each t. Analytically derive what this should be.

First, the analytical derivation.

In the textbook, in extension 10.3, we are given that

$$\frac{Z_{t+1}}{Z_t} \leq e^{-2\gamma^2},$$

where $Z_t = \frac{1}{m}\sum_{i=1}^{m} e^{-y_i f_t(x_i)}$ is the normalization factor for the distribution D_{t+1}.

They further derive a tighter bound of $\frac{Z_{t+1}}{Z_t} \leq \sqrt{1-4\gamma^2}$. Thus, we should expect to see in the plot a consistent relationship of $\frac{Z_{t+1}}{Z_t} \leq \sqrt{1-4\gamma^2}$.

My empirical results are plotted below. *Clearly, something is wonky about my implementation. I've spent a lot of time trying different variations, introducing randomness to encourage diversity amongst weak learners, and modifying calculations for D sample weights, among other things. But I haven't been able to get the results to align with my interpretation of the theory.*

1.7 g

Describe your stopping criterion for AdaBoost.

Answer:

I simply use a fixed number of iterations as the stopping criterion for AdaBoost.

1.8 h

Explain your plots in part (e) using the results discussed in class about the effect of Boosting on the generalization error and margin.

Answer: It seems clear that my empirical results at this time do not align with the theoretical expectations. However, what we would *expect* to see is that the training accuracy would increase consistently with the number of iterations, since AdaBoost focuses on the mistakes of the weak learners in previous iterations by re-weighting the training samples. Thus, with each iteration, it's more likely to correctly classify previously misclassified samples.

In fact, in general AdaBoost should be able to achieve perfect accuracy by overfitting the training set. Thus, we would expect to see the training accuracy approach 100% as the number of iterations increases, while the test accuracy would reach a peak and then decrease as the model overfits the training data.

For the confidence margin, let's interpret what wthat e *do* see in the results, which is a decreasing margin. This suggests that AdaBoost is struggling to correctly classify challenging instances, and the ensemble's overall confidence in those instances is not improving despite adding more weak learners.

I interpret this as AdaBoost finding weak learners that contribute to the accuracy until it reaches the limit of its capacity, at which point it is unable to find any more weak learners that contribute to the accuracy.

2 Support vectors

Consider a classifier that returns the option 0 (reject) whenever $|yh(x)| \leq \rho$. Recall the true risk/error, when considering the loss associated with a miclassification to be $1 : \mathbb{P}(Yh(X) < 0)$.

2.1 a

Let the class conditional density $\eta(x) = \mathbb{P}(Y = 1|X = x)$. Suppose the loss value associated with returning the reject option, 0, is $c < 1/2$. As with the 0-1 loss, the cost of misclassification, with confidence $yh(x) < -\rho$, is 1. Derive an expression for the generalization error or Bayes risk, $R(h)$.

Answer:

The generalization error or Bayes risk, $R(h)$, for a classifier $h(x)$ can be formally defined as follows:

$$R(h) = \int \mathbb{P}(Yh(X) < 0 \mid X = x)p(x)dx$$

Here, $\mathbb{P}(Yh(X) < 0 \mid X = x)$ is the probability of making an incorrect decision given that the feature vector X is x. The integral runs over the entire input space, weighted by the data distribution $p(x)$.

Let's decompose this term. An error happens in one of two scenarios:

1. $Y = 1$ and $h(x) < -\rho$, occurring with probability $\eta(x)$

2. $Y = 0$ and $h(x) > \rho$, occurring with probability $1 - \eta(x)$

Additionally, the classifier has an option to "reject", producing the output 0, with a cost c for both classes whenever $|h(x)| \leq \rho$.

We will use the following shorthand labels for the three regions of the hypothesis space:

$$\begin{aligned} A &\coloneqq \{x : h(x) < -\rho\} \\ B &\coloneqq \{x : -\rho \leq h(x) \leq \rho\} \\ C &\coloneqq \{x : h(x) > \rho\} \end{aligned}$$

Now, we can express the total generalization risk as follows, where $\mathbf{1}(\cdot)$ is the indicator function, which is 1 its argument is true and 0 otherwise:

$$\begin{aligned} R_1 &\coloneqq \int \eta(x)\mathbf{1}(x \in A)p(x)dx && Y = 1 \text{ and } x \in A \\ R_0 &\coloneqq \int (1-\eta(x))\mathbf{1}(x \in C)p(x)dx && Y = 0 \text{ and } x \in C \\ R_c &\coloneqq c\int [\eta(x)\mathbf{1}(x \in B) + (1-\eta(x))\mathbf{1}(x \in B)]p(x)dx && Y = 0 \text{ or } Y = 1 \text{ and } x \in B \\ &= c\int [\eta(x) + (1-\eta(x))]\mathbf{1}(x \in B)p(x)dx && \text{Simplify} \\ &= c\int \mathbf{1}(x \in B)p(x)dx && \text{Final reject risk (independent of } \eta) \\ R(h) &= R_1 + R_0 + R_c && \text{Total generalization risk,} \end{aligned}$$

2.2 b

The minimizer h^* of the generalization risk is known to be of form

$$h^*(x) = \begin{cases} -1 & \eta(x) < \delta \\ 0 & \delta \leq \eta(x) \leq 1 - \delta \\ 1 & \eta(x) > 1 - \delta \end{cases}$$

Show that $\delta = c$ minimizes the risk from part (a) with minimum risk being $E_X \min\{\eta(x), 1 - \eta(x), c\}$.

Answer:

To find the classifier h^* that minimizes the Bayes risk $R(h)$, we will consider each term in $R(h)$ individually. Recall from subsection (a) that $R(h) = R_1 + R_0 + R_c$.

1. The R_1 risk is minimized when $\eta(x) < \delta$ and $h(x) = -1$. This is the lowest probability of misclassification for the positive class.
2. The R_0 risk is minimized when $\eta(x) > 1 - \delta$ and $h(x) = 1$. This is the lowest probability of misclassification for the negative class.
3. Finally, the R_c risk when choosing to "reject" will be c. Since $c < 1/2$, this is lower than choosing either of the classes if $\delta \leq \eta(x) \leq 1 - \delta$, with $\delta = c$.

Thus, the Bayes risk $R(h^*)$ for the optimal classifier h^* can be found with:

$$\begin{aligned}
R(h^*) &= \min_h (R_1 + R_0 + R_c) \\
&= \min_h \left[\int \eta(x)\mathbf{1}(x \in A)p(x)dx + \int (1-\eta(x))\mathbf{1}(x \in C)p(x)dx + c\int \mathbf{1}(x \in B)p(x)dx \right] \\
&= \int \min\{\eta(x)\mathbf{1}(x \in A), (1-\eta(x))\mathbf{1}(x \in C), c\mathbf{1}(x \in B)\}p(x)dx \\
&= \int \min\{\eta(x), 1-\eta(x), c\}p(x)dx \\
&= E_X[\min\{\eta(x), 1-\eta(x), c\}] \quad \text{QED}
\end{aligned}$$

That is, when the probability $\eta(x)$ is within a range where the risk of committing to a decision is higher than the fixed cost c, the optimal classifier prefers to reject. This essentially enforces the notion that if the cost of making a *wrong decision* is higher than the cost of *not making a decision*, it is better to abstain.

2.3 c

Bartlett and Wegkamp 2008 define the following loss:

$$l(z,h) = \begin{cases} 1 - \frac{(1-c)yh(x)}{c} & yh(x) < 0 \\ 1 - yh(x) & 0 \leq yh(x) \leq 1 \\ 0 & yh(x) > 1 \end{cases}$$

Note that the above loss is greater than the discontinuous loss in part (a). When $c < 1/2 \leq \rho \leq 1-c$, they show that the excess risk with this loss for any h upper bounds the excess risk with the loss in part (a). Write down an optimization problem for the ERM of this $l(z,h)$ loss using bounded, affine functions, i.e., $h_{w,b}(x) = w^Tx + b, \|w\| \leq r$. Show that this optimization is convex.

Answer:

The ERM optimization problem is formulated as:

$$\min_{w,b,\|w\|\leq r} \frac{1}{N}\sum_{i=1}^{N} l(y_i, h_{w,b}(x_i)),$$

where $h_{w,b}(x) = w^Tx + b$ and $\|w\| \leq r$.

Substituting $l(z,h)$, we get:

$$\min_{w,b,\|w\|\leq r} \frac{1}{N}\sum_{i=1}^{N} \begin{cases} 1 - \frac{(1-c)y_i(w^Tx_i+b)}{c} & y_i(w^Tx_i+b) < 0 \\ 1 - y_i(w^Tx_i+b) & 0 \leq y_i(w^Tx_i+b) \leq 1 \\ 0 & y_i(w^Tx_i+b) > 1 \end{cases}$$

We can restate this using the indicator function $\mathbf{1}(\cdot)$:

$$\begin{aligned}
\min_{w,b} \frac{1}{N}\sum_{i=1}^{N} \Big[&\mathbf{1}(y_i(w^Tx_i+b) < 0)\left(1 - \frac{(1-c)y_i(w^Tx_i+b)}{c}\right) \\
&+\mathbf{1}(0 \leq y_i(w^Tx_i+b) \leq 1)(1 - y_i(w^Tx_i+b))\Big],
\end{aligned}$$

subject to $\|w\| \leq r$.

Note that we simply drop the $y_i(w^T x_i + b) > 1$ component, since the loss in that domain is always 0.

The function inside the sum is piecewise linear.

To determine if this piecewise linear function is convex, set $\alpha := y_i(w^T x_i + b)$, and compare the two lines representing the first and second piece:

1. $y_1 = 1 - \frac{(1-c)\alpha}{c}$
2. $y_2 = 1 - \alpha$

Observe that the slope of y_2 will be less than the slope of y_1 exactly when $c < 1/2$, which is the setting we are considering. Thus, the function is convex.

Note that each term (sample pair) inside the summation of piecewise convex functions is piecewise *over the same intervals* of the yh domain. Thus, the sum is also piecewise linear.

Finally, the constraints $\|w\| - r \leq 0$ are affine. Therefore, the optimization problem is convex.

2.4 d

Derive the KKT conditions for the problem in part (c).

Answer:

The KKT conditions consist of:

1. Stationarity:

$$\nabla_w L = \frac{1}{N}\sum_{i=1}^{N}\left[-\mathbf{1}(y_i(w^T x_i + b) < 0)\frac{(1-c)y_i x_i}{c} - \mathbf{1}(0 \leq y_i(w^T x_i + b) \leq 1)y_i x_i\right] + 2\alpha w = 0,$$
$$\nabla_b L = \frac{1}{N}\sum_{i=1}^{N}\left[-\mathbf{1}(y_i(w^T x_i + b) < 0)\frac{(1-c)y_i}{c} - \mathbf{1}(0 \leq y_i(w^T x_i + b) \leq 1)y_i\right] = 0,$$

2. Primal Feasibility:

$$\|w\| - r \leq 0,$$

3. Dual Feasibility:

$$\alpha \geq 0,$$

4. Complementary:

$$\alpha(\|w\|^2 - r^2) = 0.$$

2.5 e

We are asked not to submit code, but I chose to anyway, since it took a good deal of work, and since we only received instructions for how to adjust our loss based on ρ a day before the due date.

I believe the proposed method could be straightforwardly adapted into the `custom_loss\` function below to penalize the separate conditions based on the deviation of ρ.

The test loss, without adjustment of ρ, is 0.109, which is already better than the 0-1 perceptron loss.

```
import cvxpy as cp
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

data = load_breast_cancer()
X = 2 * data.data - 1
y = 2 * data.target - 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

N, D = X_train.shape
c = 1 / 3
r = 1

w = cp.Variable(D)
b = cp.Variable()
t = cp.Variable(N)

# Set up the constraints
constraints = []
for i in range(N):
    score = y_train[i] * (X_train[i, :] @ w + b)
    constraints.append(t[i] >= 1 - ((1 - c) * score / c))
    constraints.append(t[i] >= 1 - score)
    constraints.append(t[i] >= 0)

constraints.append(cp.norm(w) <= r)

# Set up the objective
objective = cp.Minimize(cp.sum(t) / N)

# Formulate and solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# Extract the optimal values
w_opt = w.value
b_opt = b.value

# print(f"Optimal w: {w_opt}")
# print(f"Optimal b: {b_opt}")

def custom_loss(y, scores, c):
    loss_values = np.zeros(scores.shape)
    # Condition: y * score < 0
    mask1 = y * scores < 0
    loss_values[mask1] = 1 - ((1 - c) * y[mask1] * scores[mask1]) / c
    # Condition: 0 <= y * score <= 1
    mask2 = (y * scores >= 0) & (y * scores <= 1)
    loss_values[mask2] = 1 - y[mask2] * scores[mask2]
    # Condition: y * score > 1
    loss_values[y * scores > 1] = 0

    return np.mean(loss_values)
```

```
scores_test = X_test @ w_opt + b_opt
test_loss = custom_loss(y_test, scores_test, c)

print(f"Test loss: {test_loss}")
```