5. no collaborators

$$(P_R)\ \min_{w \in \mathbb{R}^{d+1}} \frac{1}{N}\sum_{n=1}^{N}(w^T x_n - y_n) + \frac{\lambda}{N}\sum_{i=0}^{d}\alpha_i w_i^2$$

$$= (P_R)\ \min_{w \in \mathbb{R}^{d+1}} \frac{1}{N}\sum_{n=1}^{N}(w^T x_n - y_n) + \frac{\lambda}{N}((\alpha w)^T(\alpha w)), \quad \alpha = \begin{bmatrix} \sqrt{\alpha_0} & 0 & \cdots & 0 \\ 0 & \sqrt{\alpha_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\alpha_d} \end{bmatrix}$$

$\Rightarrow$ solving $\nabla E_{in}(w_{REG}) + \frac{2\lambda}{N}\alpha\, w_{REG} = 0$

$$\Rightarrow \frac{2}{N}(X^T X w_{REG} - X^T y) + \frac{2\lambda}{N}\alpha\, w_{REG} = 0, \quad X = \underbrace{\begin{bmatrix} — x_1^T — \\ — x_2^T — \\ \vdots \\ — x_N^T — \end{bmatrix}}_{N \times (d+1)}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$\Rightarrow (X^T X + \lambda\alpha)\, w_{REG} = X^T y$$

$$\Rightarrow w_{REG} = (X^T X + \lambda\alpha)^{-1} X^T y$$

$$\tilde{x}_k = [0, \ldots, 0, \sqrt{\lambda\alpha_{k-1}}, 0, \ldots, 0]^T \in \mathbb{R}^{d+1}$$

$$= \sqrt{\lambda\alpha_{k-1}}\, e_{k-1}$$

$$\frac{1}{N+K}\left(\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \sum_{k=1}^{K}(w^T \tilde{x}_k - \tilde{y}_k)^2\right)$$

$$= \frac{1}{N+K}\left(\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \sum_{k=1}^{K}(w^T \sqrt{\lambda\alpha_{k-1}}\, e_{k-1})^2\right)$$

$$= \frac{1}{N+K}\left(\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \sum_{k=1}^{K}(w_{k-1}\sqrt{\lambda\alpha_{k-1}})^2\right)$$

$$= \frac{1}{N+d+1}\left(\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \lambda\sum_{k=1}^{d+1}\alpha_{k-1} w_{k-1}^2\right)$$

$$= \frac{1}{N}\left(\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \lambda\sum_{i=0}^{d}\alpha_i w_i^2\right) \times \frac{N}{N+d+1}$$

$$(P_V)\ \min_{w \in \mathbb{R}^{d+1}} \frac{1}{N}\left(\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \lambda\sum_{i=0}^{d}\alpha_i w_i^2\right) \times \frac{N}{N+d+1}$$

$$= (P_V)\ \min_{w \in \mathbb{R}^{d+1}} \frac{1}{N}\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \frac{\lambda}{N}\sum_{i=0}^{d}\alpha_i w_i^2$$

which is same as (P_R)

$\therefore$ optimal w^* obtained by (P_V) is the same as the optimal solution

$(= w_{REG} = (X^T X + \lambda\alpha)^{-1} X^T y)$ obtained by solving (P_R)

6.

no collaborators

$$\tilde{E}_{aug}(\underline{w}) = \tilde{E}_{in}(\underline{w}) + \frac{\lambda}{N}\|\underline{w}\|^2$$

$$= E_{in}(\underline{w}^*) + \frac{1}{2}(\underline{w}-\underline{w}^*)^T \underline{\underline{H}}(\underline{w}-\underline{w}^*) + \frac{\lambda}{N}\|\underline{w}\|^2$$

$$= E_{in}(\underline{w}^*) + \frac{1}{2}\left(\underline{w}^T\underline{\underline{H}}\underline{w} - \underline{w}^T\underline{\underline{H}}\underline{w}^* - \underline{w}^{*T}\underline{\underline{H}}\underline{w} + \underline{w}^{*T}\underline{\underline{H}}\underline{w}^*\right) + \frac{\lambda}{N}\underline{w}^T\underline{w}$$

$$\nabla\tilde{E}_{aug}(\underline{w}) = \nabla E_{in}(\underline{w}^*) + \frac{1}{2}\left(2\underline{\underline{H}}\underline{w} - \underline{\underline{H}}\underline{w}^* - \underline{\underline{H}}\underline{w}^*\right) + \frac{2\lambda}{N}\underline{w}$$

$$= \underline{\underline{H}}(\underline{w}-\underline{w}^*) + \frac{2\lambda}{N}\underline{w}$$

Let $\underline{w_{aug}}$ is the minimizer of $\tilde{E}_{aug}(\underline{w})$

$$\nabla\tilde{E}_{aug}(\underline{w_{aug}}) = \underline{0}$$

$$\Rightarrow \underline{\underline{H}}(\underline{w_{aug}} - \underline{w}^*) + \frac{2\lambda}{N}\underline{w_{aug}} = \underline{0}$$

$$\Rightarrow \left(\underline{\underline{H}} + \frac{2\lambda}{N}\underline{\underline{I}}\right)\underline{w_{aug}} = \underline{\underline{H}}\underline{w}^*$$

$\because \underline{\underline{H}} + \frac{2\lambda}{N}\underline{\underline{I}}$ is positive definite

$$\Rightarrow \underline{w_{aug}} = \left(\underline{\underline{H}} + \frac{2\lambda}{N}\underline{\underline{I}}\right)^{-1}\underline{\underline{H}}\underline{w}^*$$

7.

collaborator:
B11901073 林禹融

$$\mathbb{E}\left(\frac{1}{K}\sum_{n=N-K+1}^{N}(y_n-0)^2\right)=\sigma^2$$

$$\frac{1}{K}\sum_{n=N-K+1}^{N}(y_n-\bar{y})^2$$
$$=\frac{1}{K}\sum_{n=N-K+1}^{N}(y_n^2-2y_n\bar{y}+\bar{y}^2)$$
$$=\frac{1}{K}\left(\sum_{n=N-K+1}^{N}y_n^2-2\bar{y}\sum_{n=N-K+1}^{N}y_n+\sum_{n=N-K+1}^{N}\bar{y}^2\right)$$
$$=\frac{1}{K}\sum_{n=N-K+1}^{N}y_n^2+\frac{1}{K}\left[-2\bar{y}\cdot(0-(N-K)\cdot\bar{y})+K\bar{y}^2\right]$$
$$=\frac{1}{K}\sum_{n=N-K+1}^{N}y_n^2+\frac{1}{K}(2N-K)\bar{y}^2$$

$$\sum_{n=1}^{N}y_n=0$$
$$\Rightarrow\sum_{n=1}^{N-K}y_n+\sum_{n=N-K+1}^{N}y_n=0$$
$$\Rightarrow\sum_{n=N-K+1}^{N}y_n=-\sum_{n=1}^{N-K}y_n$$
$$=-(N-K)\bar{y}$$

$$\mathbb{E}\left(\frac{1}{K}\sum_{n=N-K+1}^{N}y_n^2+\frac{1}{K}(2N-K)\bar{y}^2\right)$$
$$=\mathbb{E}\left(\frac{1}{K}\sum_{n=N-K+1}^{N}y_n^2\right)+\mathbb{E}\left(\frac{2N-K}{K}\bar{y}^2\right)$$
$$=\sigma^2+\frac{2N-K}{K}\times\mathbb{E}(\bar{y}^2)$$
$$=\sigma^2+\frac{2N-K}{K}\mathrm{Var}(\bar{y})$$
$$=\sigma^2+\frac{2N-K}{K}\left(\frac{1}{N-K}\sigma^2\right)$$
$$=\left(1+\frac{2N-K}{K(N-K)}\right)\sigma^2$$

$$\mathrm{Var}(\bar{y})=\mathrm{Var}\left(\frac{1}{N-K}\sum_{n=1}^{N-K}y_n\right)$$
$$=\left(\frac{1}{N-K}\right)^2\mathrm{Var}\left(\sum_{n=1}^{N-K}y_n\right)$$
$$=\left(\frac{1}{N-K}\right)^2\sum_{n=1}^{N-K}\mathrm{Var}(y_n)$$
$$=\left(\frac{1}{N-K}\right)^2(N-K)\sigma^2$$
$$=\frac{1}{N-K}\sigma^2$$

8. P17 of lecture 15 no collaborators

$$E_{in}(w_0^*) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{N}\sum_{n=1}^{N} y_n - y_n\right)^2$$
$$= \left(\frac{1}{N}\right)^2\left[\frac{1}{N}\sum_{n=1}^{N}\left(\sum_{n=1}^{N} y_n - N y_n\right)^2\right]$$
$$\Rightarrow N^2 E_{in}(w_0^*) = \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{n=1}^{N} y_n - N y_n\right)^2$$

$$E_{loocv}(A_{avg}) = \frac{1}{N}\sum_{n=1}^{N} e_n$$
$$= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{N-1}\sum_{i\neq n} y_i - y_n\right)^2$$
$$= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{N-1}\left(\sum_{i=1}^{N} y_i - y_n\right) - y_n\right)^2$$
$$= \left(\frac{1}{N-1}\right)^2\left[\frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=1}^{N} y_i - y_n - (N-1)y_n\right)^2\right]$$
$$= \left(\frac{1}{N-1}\right)^2\left[\frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=1}^{N} y_i - N y_n\right)^2\right]$$
$$= \left(\frac{N}{N-1}\right)^2 E_{in}(w_0^*) \quad \text{for } N \geq 2$$

Q.E.D.

9.

no collaborators

$$E_{out}(g) = E_{out}(g_c)$$

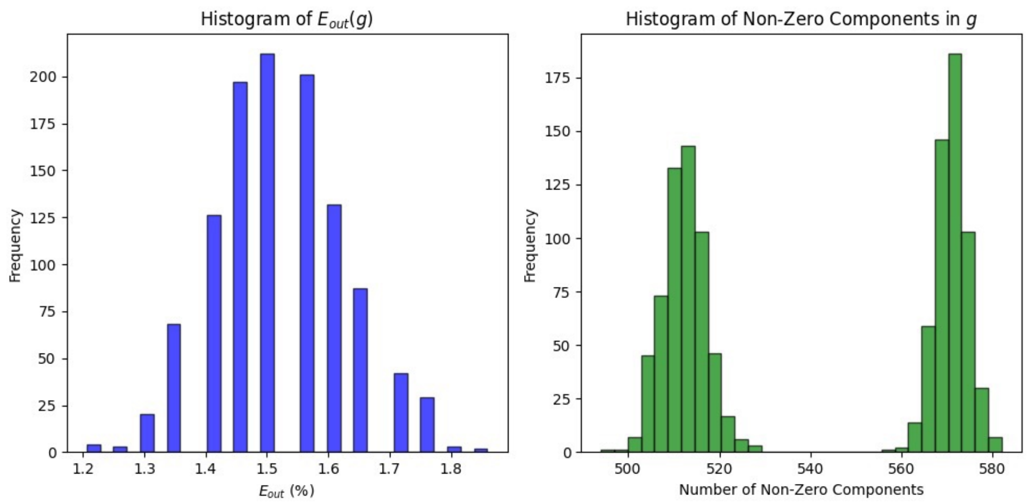
$$\Rightarrow p\epsilon_- + (1-p)\epsilon_+ = p$$

$$\Rightarrow p(\epsilon_- - \epsilon_+ - 1) = -\epsilon_+$$

$$\Rightarrow p = \frac{\epsilon_+}{1+\epsilon_+ - \epsilon_-}$$

10.

collaborators:
B11201009 黃勤元
B11901073 林禹融

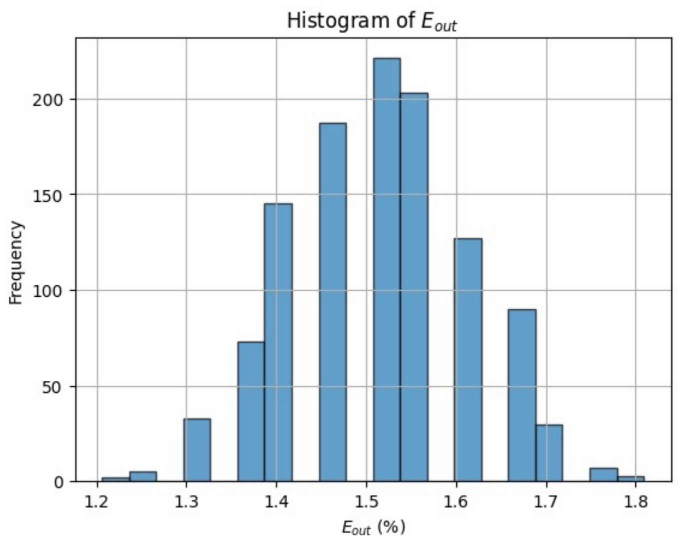


hw5-10.py > ...

```python
import os
import requests
import bz2
import numpy as np
import scipy.sparse
from liblinear.liblinearutil import *
import matplotlib.pyplot as plt

# Step 1: Download and decompress the data
def download_and_extract(url, dest_path):...

def decompress_bz2(file_path, output_path):...

train_url = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.scale.bz2"
test_url = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.scale.t.bz2"
train_file_compressed = "mnist.scale.bz2"
test_file_compressed = "mnist.scale.t.bz2"
train_file = "mnist.scale"
test_file = "mnist.scale.t"

download_and_extract(train_url, train_file_compressed)
download_and_extract(test_url, test_file_compressed)
decompress_bz2(train_file_compressed, train_file)
decompress_bz2(test_file_compressed, test_file)

# Step 2: Load and preprocess the data
y_train, X_train = svm_read_problem(train_file, return_scipy=True)
y_test, X_test = svm_read_problem(test_file, return_scipy=True)

# Filter for classes 2 and 6, and relabel them for binary classification
train_mask = np.isin(y_train, [2, 6])
test_mask = np.isin(y_test, [2, 6])
y_train, X_train = y_train[train_mask], X_train[train_mask]
y_test, X_test = y_test[test_mask], X_test[test_mask]
y_train = np.where(y_train == 2, 1, -1)
y_test = np.where(y_test == 2, 1, -1)

def align_features(train, test):...

# Align feature dimensions between training and test sets
X_train, X_test = align_features(X_train, X_test)

# Scale the data using csr_scale
scale_param = csr_find_scale_param(X_train, lower=0)
X_train = csr_scale(X_train, scale_param)
X_test = csr_scale(X_test, scale_param)
    Ctrl+L to chat, Ctrl+K to generate
# Step 3: Define helper functions
def lambda_to_C(lmbda):
    return 1 / lmbda

lambdas = [0.01, 0.1, 1, 10, 100, 1000]
N = len(y_train)
best_lambda, min_error = None, float('inf')
errors_in = []

# Step 4: Find the best lambda and repeat experiments with 1126 seeds
random_seeds = range(1126)
errors_out, non_zero_counts = [], []

for seed in random_seeds:
    for lmbda in lambdas:
        C = lambda_to_C(lmbda)
        model = train(y_train, X_train, f'-s 6 -c {C} -B 1 -q')
        _, p_acc, _ = predict(y_train, X_train, model)
        E_in = 100 - p_acc[0]
        errors_in.append(E_in)
        if E_in < min_error or (E_in == min_error and lmbda > best_lambda):
            best_lambda, min_error = lmbda, E_in

    C = lambda_to_C(best_lambda)
    np.random.seed(seed)
    model = train(y_train, X_train, f'-s 6 -c {C} -B 1 -q')
    _, p_acc, _ = predict(y_test, X_test, model)
    E_out = 100 - p_acc[0] # accuracy to error
    errors_out.append(E_out)

    # Access weights and bias terms
    weights, bias = model.get_decfun()
    non_zero_counts.append(np.sum(np.abs(weights) > 1e-6))

# Step 6: Plot histograms
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.hist(errors_out, bins=30, color="blue", edgecolor="black", alpha=0.7)
plt.title("Histogram of $E_{out}(g)$")
plt.xlabel("$E_{out}$ (%)")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.hist(non_zero_counts, bins=30, color="green", edgecolor="black", alpha=0.7)
plt.title("Histogram of Non-Zero Components in $g$")
plt.xlabel("Number of Non-Zero Components")
plt.ylabel("Frequency")

plt.tight_layout()
```

11. collaborator: B11901073 林禹融

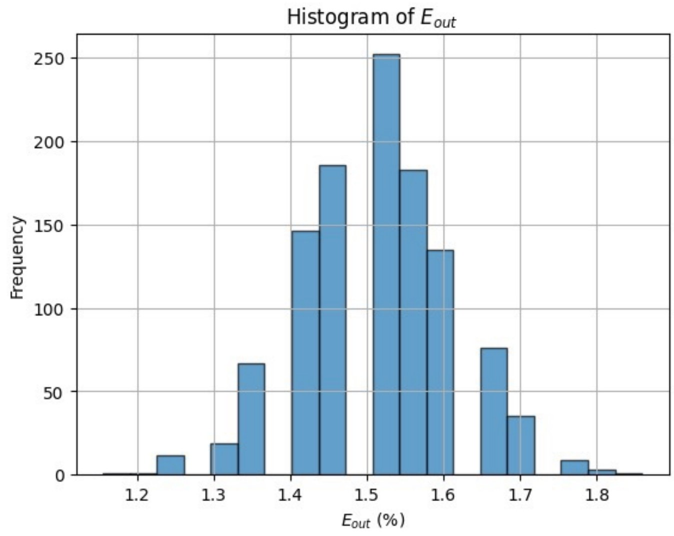
Both histograms are center around $E_{out} \approx 1.5\%$, suggesting that performance is similar between both methods. While the distribution in Problem 11 is slightly wider than that in Problem 10, reflecting that cross-validation approach introduces additional variability because of the random splits into sub-training and validation sets.



hw5-11.py > ...

```python
import os
import requests
import bz2
import numpy as np
from liblinear.liblinearutil import *
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix, hstack

# Constants
LAMBDA_VALUES = [0.01, 0.1, 1, 10, 100, 1000]
N_EXPERIMENTS = 1126
TRAIN_SIZE = 8000

# Step 1: Download and decompress the data
def download_and_extract(url, dest_path):…

def decompress_bz2(file_path, output_path):…

train_url = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.scale.bz2"
test_url = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.scale.t.bz2"
train_file_compressed = "mnist.scale.bz2"
test_file_compressed = "mnist.scale.t.bz2"
train_file = "mnist.scale"
test_file = "mnist.scale.t"

download_and_extract(train_url, train_file_compressed)
download_and_extract(test_url, test_file_compressed)
decompress_bz2(train_file_compressed, train_file)
decompress_bz2(test_file_compressed, test_file)

# Step 2: Load and preprocess the data
y_train, X_train = svm_read_problem(train_file, return_scipy=True)
y_test, X_test = svm_read_problem(test_file, return_scipy=True)

# Filter for classes 2 and 6, and relabel them for binary classification
train_mask = np.isin(y_train, [2, 6])
test_mask = np.isin(y_test, [2, 6])
y_train, X_train = y_train[train_mask], X_train[train_mask]
y_test, X_test = y_test[test_mask], X_test[test_mask]
y_train = np.where(y_train == 2, 1, -1)
y_test = np.where(y_test == 2, 1, -1)

def align_features(train, test):…

# Align feature dimensions between training and test sets
X_train, X_test = align_features(X_train, X_test)

# Scale the data using csr_scale
scale_param = csr_find_scale_param(X_train, lower=0)
X_train = csr_scale(X_train, scale_param)
X_test = csr_scale(X_test, scale_param)

# Function to calculate error
def calculate_error(y_true, y_pred):
    return np.mean(y_true != y_pred) * 100

# Prepare for histogram
E_out_values = []

# Main loop for 1126 experiments
for experiment in range(N_EXPERIMENTS):
    np.random.seed(experiment)
    # Step 1: Split data into sub-training and validation sets
    X_subtrain, X_val, y_subtrain, y_val = train_test_split(
        X_train, y_train, train_size=TRAIN_SIZE, random_state=experiment
    )
    best_lambda, best_E_val = None, float('inf')

    # Step 2: Evaluate each lambda
    for lambda_value in LAMBDA_VALUES:
        C = 1 / lambda_value
        model = train(y_subtrain, X_subtrain, f'-s 6 -c {C} -B 1 -q')
        p_val, _, _ = predict(y_val, X_val, model, '-q')
        E_val = calculate_error(y_val, p_val)
        if E_val < best_E_val or (E_val == best_E_val and lambda_value > best_lambda):
            best_E_val = E_val
            best_lambda = lambda_value

    # Step 3: Re-train with the best lambda on the full training set
    C = 1 / best_lambda
    final_model = train(y_train, X_train, f'-s 6 -c {C} -B 1 -q')

    # Step 4: Evaluate E_out on the test set
    p_test, _, _ = predict(y_test, X_test, final_model, '-q')
    E_out = calculate_error(y_test, p_test)
    E_out_values.append(E_out)

# Step 5: Plot histogram of E_out
plt.hist(E_out_values, bins=20, edgecolor='black', alpha=0.7)
plt.title('Histogram of $E_{out}$')
plt.xlabel('$E_{out}$ (%)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

12. collaborator: B11901073 林禹融



Both histograms are center around $E_{out} \approx 1.5\%$, suggesting that performance is similar between both methods. Additionally, histogram in Problem 12 is more concentrated around the mean. The 3-fold cross validation may be generally a better approach to minize variance in model selection process

hw5-12.py > ...

```python
import os
import requests
import bz2
import numpy as np
from liblinear.liblinearutil import *
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix, hstack

# Constants
LAMBDA_VALUES = [0.01, 0.1, 1, 10, 100, 1000]
N_EXPERIMENTS = 1126
K_FOLDS = 3  # 3-fold cross-validation

# Step 1: Download and decompress the data
def download_and_extract(url, dest_path): ...

def decompress_bz2(file_path, output_path): ...

train_url = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.scale.bz2"
test_url = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.scale.t.bz2"
train_file_compressed = "mnist.scale.bz2"
test_file_compressed = "mnist.scale.t.bz2"
train_file = "mnist.scale"
test_file = "mnist.scale.t"

download_and_extract(train_url, train_file_compressed)
download_and_extract(test_url, test_file_compressed)
decompress_bz2(train_file_compressed, train_file)
decompress_bz2(test_file_compressed, test_file)

# Step 2: Load and preprocess the data
y_train, X_train = svm_read_problem(train_file, return_scipy=True)
y_test, X_test = svm_read_problem(test_file, return_scipy=True)

# Filter for classes 2 and 6, and relabel them for binary classification
train_mask = np.isin(y_train, [2, 6])
test_mask = np.isin(y_test, [2, 6])
y_train, X_train = y_train[train_mask], X_train[train_mask]
y_test, X_test = y_test[test_mask], X_test[test_mask]
y_train = np.where(y_train == 2, 1, -1)
y_test = np.where(y_test == 2, 1, -1)

def align_features(train, test): ...

# Align feature dimensions between training and test sets
X_train, X_test = align_features(X_train, X_test)

# Scale the data using csr_scale
scale_param = csr_find_scale_param(X_train, lower=0)
X_train = csr_scale(X_train, scale_param)
X_test = csr_scale(X_test, scale_param)

# Function to calculate error
def calculate_error(y_true, y_pred): ...

# Function for 3-fold cross-validation
def cross_validate(X, y, lambda_value):
    kf = KFold(n_splits=K_FOLDS, shuffle=True, random_state=None)
    total_error = 0

    for train_index, val_index in kf.split(X):
        X_train_fold, X_val_fold = X[train_index], X[val_index]
        y_train_fold, y_val_fold = y[train_index], y[val_index]

        C = 1 / lambda_value
        model = train(y_train_fold, X_train_fold, f'-s 6 -c {C} -B 1 -q')

        p_val, _, _ = predict(y_val_fold, X_val_fold, model, '-q')
        total_error += calculate_error(y_val_fold, p_val)

    # Return average cross-validation error
    return total_error / K_FOLDS

# Prepare for histogram
E_out_values = []

# Main loop for 1126 experiments
for experiment in range(N_EXPERIMENTS):
    np.random.seed(experiment)

    best_lambda, best_E_CV = None, float('inf')

    # Step 1: Perform 3-fold cross-validation for each lambda
    for lambda_value in LAMBDA_VALUES:
        E_CV = cross_validate(X_train, y_train, lambda_value)

        # Update best lambda
        if E_CV < best_E_CV or (E_CV == best_E_CV and lambda_value > best_lambda):
            best_E_CV = E_CV
            best_lambda = lambda_value

    # Step 2: Re-train with the best lambda on the full training set
    C = 1 / best_lambda
    final_model = train(y_train, X_train, f'-s 6 -c {C} -B 1 -q')

```