In [1]:

```python
import numpy as np
from scipy.optimize import minimize
import pandas as pd

# ---------------- 数据加载与预处理 ----------------
def load_sonar_data(file_path):
    data = pd.read_csv(file_path, header=None)
    X = data.iloc[:, :-1].values
    y_labels = data.iloc[:, -1].values
    y = np.array([1 if label == 'M' else -1 for label in y_labels])
    return X, y

# ---------------- 核函数 ----------------
def rbf_kernel(x1, x2, gamma):
    return np.exp(-gamma * np.dot(x1 - x2, x1 - x2))

def compute_kernel_matrix(X, gamma):
    sq_dists = np.sum(X**2, axis=1)[:, None] + np.sum(X**2, axis=1)[None,
    return np.exp(-gamma * sq_dists)

# ---------------- SVM 训练函数 ----------------
def train_svm(X_train, y_train, C, gamma, tol=1e-5):
    n = len(X_train)
    K = compute_kernel_matrix(X_train, gamma)
    Q = np.outer(y_train, y_train) * K

    def objective(alpha):
        return 0.5 * alpha @ Q @ alpha - np.sum(alpha)

    def gradient(alpha):
        return Q @ alpha - np.ones(n)

    constraints = {'type': 'eq', 'fun': lambda a: np.dot(a, y_train), 'ja
    bounds = [(0, C)] * n
    alpha0 = np.zeros(n)

    res = minimize(objective, alpha0, jac=gradient, bounds=bounds, constr
                   method='SLSQP', tol=1e-6, options={'disp': False, 'max

    if not res.success:
        print("SLSQP 优化失败:", res.message)
        return np.array([]), np.array([]), np.array([]), 0.0, np.zeros(n)

    alphas = res.x
    sv_mask = alphas > tol
    alpha_sv = alphas[sv_mask]
    X_sv = X_train[sv_mask]
    y_sv = y_train[sv_mask]

    if len(alpha_sv) == 0:
        return alpha_sv, X_sv, y_sv, 0.0, alphas

    margin_mask = (alphas > tol) & (alphas < C - tol)
    b_list = []
    for i in np.where(margin_mask)[0]:
        s = sum(alpha_sv[j] * y_sv[j] * rbf_kernel(X_sv[j], X_train[i], g
        b_list.append(y_train[i] - s)
```

```
    b = np.mean(b_list) if b_list else 0.0
    return alpha_sv, X_sv, y_sv, b, alphas

# ---------------- SVM 预测函数 ----------------
def predict_svm(X_eval, alpha_sv, X_sv, y_sv, b, gamma):
    if len(alpha_sv) == 0:
        scores = np.full(X_eval.shape[0], b)
        preds = np.sign(scores)
        preds[preds == 0] = 1
        return preds, scores

    scores = np.array([
        sum(alpha_sv[j] * y_sv[j] * rbf_kernel(X_sv[j], x, gamma) for j i
        for x in X_eval
    ])
    preds = np.sign(scores)
    preds[preds == 0] = 1
    return preds, scores

# ---------------- 主程序（任务1~3） ----------------
if __name__ == "__main__":
    X, y = load_sonar_data("sonar.all-data")
    n = len(X)
    n_splits = 13
    epsilon = 1e-4

    # ---------- 任务1 ----------
    print("任务1: C=10, gamma=1")
    C, gamma = 10, 1
    train_errors, test_errors, sv_counts, correct_outliers = [], [], [],

    for i in range(n_splits):
        test_idx = np.arange(i, n, n_splits)
        train_idx = np.setdiff1d(np.arange(n), test_idx)
        X_train, y_train = X[train_idx], y[train_idx]
        X_test, y_test = X[test_idx], y[test_idx]

        alpha_sv, X_sv, y_sv, b, alphas = train_svm(X_train, y_train, C,
        if len(alpha_sv) == 0:
            train_errors.append(0.5)
            test_errors.append(0.5)
            sv_counts.append(0)
            correct_outliers.append(0)
            continue

        train_preds, _ = predict_svm(X_train, alpha_sv, X_sv, y_sv, b, ga
        test_preds, _ = predict_svm(X_test, alpha_sv, X_sv, y_sv, b, gamm
        train_errors.append(np.mean(train_preds != y_train))
        test_errors.append(np.mean(test_preds != y_test))
        sv_counts.append(len(alpha_sv))

        outlier_mask = np.isclose(alphas, C, atol=epsilon)
        correct = sum(train_preds[i] == y_train[i] for i in range(len(y_t
        correct_outliers.append(correct)

    print(f"平均训练误差: {np.mean(train_errors):.4f}, 标准差: {np.std(train
    print(f"平均测试误差: {np.mean(test_errors):.4f}, 标准差: {np.std(test_e
    print(f"平均支持向量数量: {np.mean(sv_counts):.2f}")
    print(f"平均正确离群点数量: {np.mean(correct_outliers):.2f}")
```

```
# ---------- 任务2 ----------
print("\n任务2: 分析不同 C 值对离群点和支持向量的影响")
C_values = [0.01, 0.1, 1, 10, 50, 100]
result_table = []

for C in C_values:
    svs, correct_outs, incorrect_outs = [], [], []
    for i in range(n_splits):
        test_idx = np.arange(i, n, n_splits)
        train_idx = np.setdiff1d(np.arange(n), test_idx)
        X_train, y_train = X[train_idx], y[train_idx]

        alpha_sv, X_sv, y_sv, b, alphas = train_svm(X_train, y_train,
        if len(alpha_sv) == 0:
            svs.append(0)
            correct_outs.append(0)
            incorrect_outs.append(0)
            continue

        train_preds, _ = predict_svm(X_train, alpha_sv, X_sv, y_sv, b
        outlier_mask = np.isclose(alphas, C, atol=epsilon)
        correct = sum(train_preds[i] == y_train[i] for i in range(len
        incorrect = sum(train_preds[i] != y_train[i] for i in range(l

        svs.append(len(alpha_sv))
        correct_outs.append(correct)
        incorrect_outs.append(incorrect)

    result_table.append({
        "C": C,
        "Avg SVs": np.mean(svs),
        "Correct Outliers": np.mean(correct_outs),
        "Incorrect Outliers": np.mean(incorrect_outs)
    })

df2 = pd.DataFrame(result_table)
print(df2)

# ---------- 任务3 ----------
print("\n任务3: 超参数调优")
gamma_list = [0.01, 0.1, 0.5, 1, 5]
C_list = [0.1, 1, 10, 50, 100]
best_result = {"gamma": None, "C": None, "error": float('inf')}

for gamma in gamma_list:
    for C in C_list:
        test_errors = []
        for i in range(n_splits):
            test_idx = np.arange(i, n, n_splits)
            train_idx = np.setdiff1d(np.arange(n), test_idx)
            X_train, y_train = X[train_idx], y[train_idx]
            X_test, y_test = X[test_idx], y[test_idx]

            alpha_sv, X_sv, y_sv, b, _ = train_svm(X_train, y_train,
            if len(alpha_sv) == 0:
                test_errors.append(0.5)
                continue
            preds, _ = predict_svm(X_test, alpha_sv, X_sv, y_sv, b, g
            test_errors.append(np.mean(preds != y_test))
```

```
            avg_error = np.mean(test_errors)
            if avg_error < best_result["error"]:
                best_result = {"gamma": gamma, "C": C, "error": avg_error

            print(f"Gamma={gamma}, C={C} => 平均测试误差: {avg_error:.4f}")

    print(f"最优结果: Gamma={best_result['gamma']}, C={best_result['C']}, 
```

```
任务1: C=10, gamma=1
平均训练误差: 0.0000, 标准差: 0.0000
平均测试误差: 0.1202, 标准差: 0.0754
平均支持向量数量: 142.85
平均正确离群点数量: 0.00

任务2: 分析不同 C 值对离群点和支持向量的影响
        C     Avg SVs  Correct Outliers  Incorrect Outliers
0    0.01  184.076923         84.307692           89.538462
1    0.10  183.384615         89.923077           84.307692
2    1.00  153.769231         62.538462            1.769231
3   10.00  142.846154          0.000000            0.000000
4   50.00  142.846154          0.000000            0.000000
5  100.00  142.846154          0.000000            0.000000

任务3: 超参数调优
Gamma=0.01, C=0.1 => 平均测试误差: 0.4663
Gamma=0.01, C=1 => 平均测试误差: 0.4375
Gamma=0.01, C=10 => 平均测试误差: 0.2212
Gamma=0.01, C=50 => 平均测试误差: 0.1827
Gamma=0.01, C=100 => 平均测试误差: 0.1827
Gamma=0.1, C=0.1 => 平均测试误差: 0.4663
Gamma=0.1, C=1 => 平均测试误差: 0.2019
Gamma=0.1, C=10 => 平均测试误差: 0.1394
Gamma=0.1, C=50 => 平均测试误差: 0.1250
Gamma=0.1, C=100 => 平均测试误差: 0.1346
Gamma=0.5, C=0.1 => 平均测试误差: 0.3750
Gamma=0.5, C=1 => 平均测试误差: 0.1442
Gamma=0.5, C=10 => 平均测试误差: 0.0962
Gamma=0.5, C=50 => 平均测试误差: 0.0962
Gamma=0.5, C=100 => 平均测试误差: 0.0962
Gamma=1, C=0.1 => 平均测试误差: 0.4519
Gamma=1, C=1 => 平均测试误差: 0.1346
Gamma=1, C=10 => 平均测试误差: 0.1202
Gamma=1, C=50 => 平均测试误差: 0.1202
Gamma=1, C=100 => 平均测试误差: 0.1202
Gamma=5, C=0.1 => 平均测试误差: 0.4663
Gamma=5, C=1 => 平均测试误差: 0.1731
Gamma=5, C=10 => 平均测试误差: 0.1683
Gamma=5, C=50 => 平均测试误差: 0.1683
Gamma=5, C=100 => 平均测试误差: 0.1683
最优结果: Gamma=0.5, C=10, 测试误差=0.0962
```

Task 4 Report: Support Vector Classification on Sonar Dataset

Overview

This report presents the implementation and evaluation of a non-linear support vector machine (SVM) classifier applied to the UCI Sonar dataset. The classifier was implemented from scratch using the SLSQP optimizer with an RBF kernel. Three main tasks were performed:

1. Evaluation of the classifier with fixed parameters `C = 10` and `gamma = 1`
2. Analysis of how varying the penalty parameter `C` affects model characteristics
3. Hyperparameter tuning of `C` and `gamma` to minimize the test error

---

Task 1: Fixed Parameters (C=10, gamma=1)

I evaluated the classifier using 13-fold stratified cross-validation, where each fold uses every 13th instance as the test set. The results were:

- **Average training error**: **0.0000**
- **Standard deviation (training)**: **0.0000**
- **Average test error**: **0.1202**
- **Standard deviation (test)**: **0.0754**
- **Average number of support vectors**: **142.85**
- **Average number of correctly classified outliers**: **0.00**

These results indicate perfect training performance with a reasonable test error, but no support vectors were identified as outliers (i.e., dual variables $\alpha \approx C$).

Task 2: Effect of C on Support Vectors and Outliers

I varied the value of `C` across a wide range and measured the effect on the number of support vectors and correctly/incorrectly classified outliers. The results are summarized below:

C	Avg SVs	Correct Outliers	Incorrect Outliers
0.01	184.08	84.31	89.54
0.10	183.38	89.92	84.31
1.00	153.77	62.54	1.77
10.00	142.85	0.00	0.00
50.00	142.85	0.00	0.00
100.00	142.85	0.00	0.00

Observations:

- As **C increases**, the **number of support vectors decreases**, indicating a tighter decision boundary.

- **Correct and incorrect outliers** are both high for small C, but drop to **zero** for `C ≥ 10`.
- Small `C` leads to underfitting, assigning many training points as support vectors or outliers.
- High `C` encourages fewer support vectors and no slack variables, reducing outliers entirely.

Task 3: Hyperparameter Tuning

A grid search was performed over:

- `gamma`: [0.01, 0.1, 0.5, 1, 5]
- `C`: [0.1, 1, 10, 50, 100]

Each combination was evaluated using 13-fold cross-validation, and the average test error was recorded.

Best Result:

- **Optimal gamma**: **0.5**
- **Optimal C**: **10**
- **Minimum test error**: **0.0962**

Selected Results:

Gamma	C	Avg Test Error
0.5	10	**0.0962**
0.5	50	0.0962
0.5	100	0.0962
0.1	10	0.1394
1.0	10	0.1202
5.0	10	0.1683
0.01	100	0.1827

Observations:

- Moderate values of `gamma` and `C` led to the lowest test error.
- Extremely small `gamma` (e.g., 0.01) underfit the data.
- Extremely large `gamma` (e.g., 5) increased overfitting and degraded performance.
- Test error plateaued across some values of C, suggesting robustness in the optimal region.

Conclusion

The custom SVM implementation using the SLSQP optimizer effectively classified the Sonar dataset with competitive test accuracy. The tuning of hyperparameters revealed that moderate regularization (`C = 10`) and kernel flexibility (`gamma = 0.5`) achieve the best performance. Future improvements could involve incorporating visualization, confidence scores, and more advanced kernels.