

Implementing Support Vector Machines

Sara Farahani

February 19, 2024

Abstract

In this project we will implement support vector machines. Firstly, we try to find optimal values by solving primal problem. After that, we will solve dual problem and use Lagrange multipliers to find primal problem optimal values.

1 Linear SVM by The Primal Problem

The aim of this part is to solve the primal problem, using CVXOPT package. The primal problem for svm is as following formula:

$$\min_{\mathbf{w}, b, \xi_i} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad \text{Subject to: } y_i(w^T x_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \text{for } i = 1, 2, \dots, n$$

1.1 Introduction

Convex optimization problems is in the following form:

$$\text{Minimize } \frac{1}{2} x^T P x + q^T x \quad \text{Subject to: } Gx \leq h, \quad Ax = b$$

We need to obtain the values of P, q, G, h, A, b based on the provided formula. Let's Consider $x = \begin{bmatrix} \mathbf{w}_{(n,1)} & \mathbf{b}_{(1,1)} & \xi_{(m,1)}^T \end{bmatrix}$ and substitute in above formula.

$$\|w\|^2 + C \sum_{i=1}^N \xi_i = \frac{1}{2} \begin{bmatrix} \mathbf{w} & \mathbf{b} & \xi \end{bmatrix} \underbrace{\begin{bmatrix} 2 * I_{(n,n)} & 0_{(n,m+1)} \\ 0_{(m+1,n)} & 0_{(m+1,m+1)} \end{bmatrix}}_P \begin{bmatrix} \mathbf{w} \\ \mathbf{b} \\ \xi \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{0}_{(1,n+1)} & \mathbf{C}_{(1,m)} \end{bmatrix}}_q \begin{bmatrix} \mathbf{w} \\ \mathbf{b} \\ \xi \end{bmatrix}$$
$$\begin{array}{l} y_i(w^T x_i - b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{array} \Rightarrow \begin{array}{l} -y_i(w^T x_i - b) - \xi_i \leq -1 \\ -\xi_i \leq 0 \end{array} \Rightarrow \underbrace{\begin{bmatrix} -\mathbf{y}\mathbf{x}_{(m,n)} & \mathbf{y}_{(m,1)} & -\mathbf{I}_{(m,m)} \\ \mathbf{0}_{(m,n)} & \mathbf{0}_{(m,1)} & -\mathbf{I}_{(m,m)} \end{bmatrix}}_G \begin{bmatrix} \mathbf{w} \\ \mathbf{b} \\ \xi \end{bmatrix} \leq \underbrace{\begin{bmatrix} -\mathbf{1}_{(m)} \\ \mathbf{0}_{(m)} \end{bmatrix}}_h$$

1.2 Implementation

For implementation, we finalize the linear_svm_primal function in svm_primal.py.

```
def linear_svm_primal(X, y, C=1):
    m, n = X.shape

    # define values of P, q, G, h, A, b
    P = np.block([[2*np.identity(n), np.zeros((n, m+1))],
                  [np.zeros((m+1, n)), np.zeros((m+1, m+1))]])

    q = np.block([np.zeros(n+1), C*np.ones(m)])
```

```

G = np.block([[ -X*np.reshape(y, (-1, 1)), y.reshape(-1,1), -np.identity(m)],
              [np.zeros((m, n+1)), -np.identity(m)]])

h = np.block([ -np.ones(m), np.zeros(m)])

# solve problem using cvxopt.solvers.qp
sol = solvers.qp(matrix(P, tc='d'), matrix(q, tc='d'), matrix(G, tc='d'),
                 matrix(h, tc='d'))

# extract weights, bias and distances
weights = np.array(sol['x'][0:n])
bias = float(sol['x'][n])
distances = np.array(sol['x'][n+1:])

return weights, bias, distances

```

1.3 Evaluation

In this part we train the model on two datasets, including "without slack" given dataset and Iris dataset. The following results in figure 1 show Precision, Recall and F1-score for "without slack" dataset. Also, the heat map of the confusion matrix is shown in figure 2. For visualization, decision boundaries are plotted in figure 3.

```

Primal problem Evaluation :
Class -1.0 - Precision: 0.8000, Recall: 0.8000, F1-score: 0.8000
Class 1.0 - Precision: 0.8000, Recall: 0.8000, F1-score: 0.8000
Accuracy: 0.8000

```

Figure 1: With slack primal problem Acc.

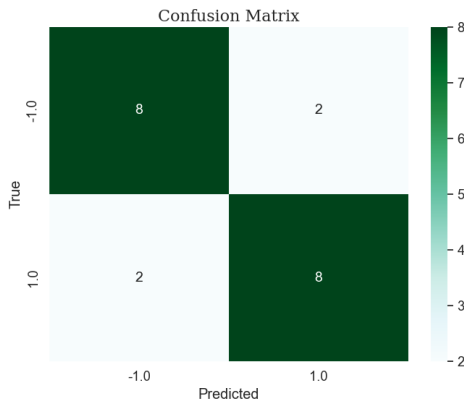


Figure 2: With slack primal problem CM

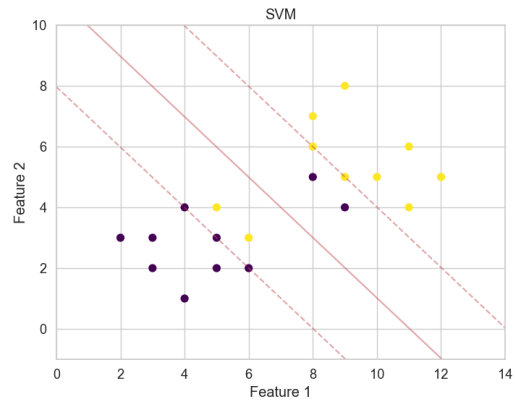


Figure 3: With slack decision boundaries

The following results in 4 are related to Iris train data. Also, the heat map of the confusion matrix is shown in 5. For visualization, we need to reduce 4 dimensions of features to 2. The figure 6 illustrates the classification results.

The following results in figure 7 are related to Iris test data. Also, the heat map of the confusion matrix is shown in figure 8. Decision boundaries are plotted in figure 9.

```

Primal problem Evaluation :
Class -1 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Accuracy: 1.0000

```

Figure 4: Iris train data Acc.

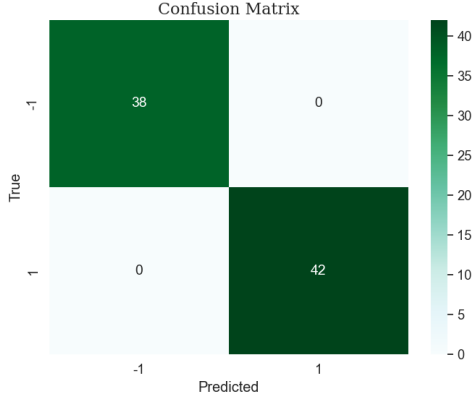


Figure 5: Iris train data CM

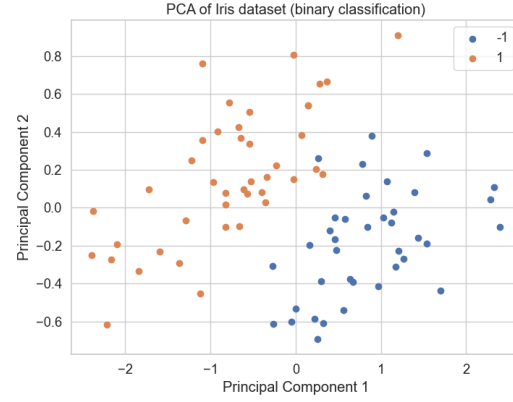


Figure 6: Iris train data PCA

```

Primal problem Evaluation :
Class -1 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Class 1 - Precision: 0.8000, Recall: 1.0000, F1-score: 0.8889
Accuracy: 0.9000

```

Figure 7: Iris test data Acc.

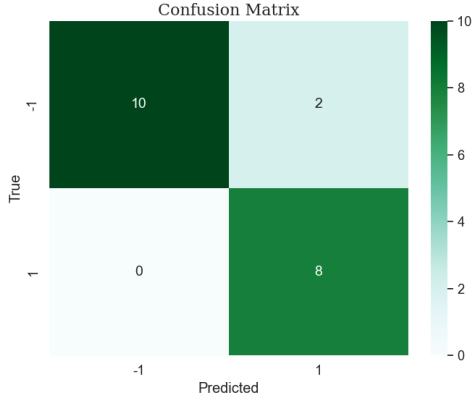


Figure 8: Iris test data CM

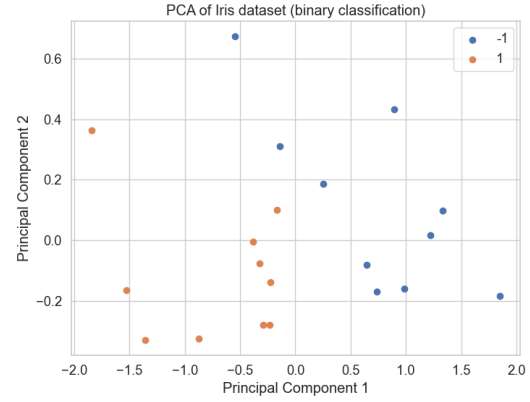


Figure 9: Iris test data PCA

2 Dual Problem Derivation

In this part we derive the primal problem by forming the Lagrangian and minimizing with respect to the primal variables.

$$L(\mathbf{w}, \mathbf{b}, \xi, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x) = \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i + \sum_{i=1}^m \lambda_i (-y_i \mathbf{w}^T x_i + y_i b - \xi_i + 1) - \sum_{i=1}^m \mu_i \xi_i$$

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{w} - \sum_{i=1}^m \lambda_i \mathbf{y}_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \frac{1}{2} \sum_{i=1}^m \lambda_i \mathbf{y}_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^m \lambda_i \mathbf{y}_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \Rightarrow \mu_i = C - \lambda_i$$

$$\begin{aligned} & \max_{\lambda} \frac{1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j + C \sum_{i=1}^m \xi_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^m \lambda_i \mathbf{y}_i b - \sum_{i=1}^m \lambda_i \xi_i + \sum_{i=1}^m \lambda_i - \sum_{i=1}^m (C - \lambda_i) \xi_i \\ &= \max_{\lambda} \frac{1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j + C \sum_{i=1}^m \xi_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^m \lambda_i \mathbf{y}_i b - \sum_{i=1}^m \lambda_i \xi_i + \sum_{i=1}^m \lambda_i - \sum_{i=1}^m (C - \lambda_i) \xi_i \\ &= \max_{\lambda} \frac{-1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^m \lambda_i = \min_{\lambda} \frac{1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i^T \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j \lambda_j - \sum_{i=1}^m \lambda_i \quad \text{subject to } \sum_{i=1}^m \lambda_i \mathbf{y}_i = 0 \end{aligned}$$

3 Karush-Kuhn-Tucker (KKT) Condition Derivation

3.1 Stationarity

$$L(\mathbf{w}, \mathbf{b}, \xi, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x) = \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i + \sum_{i=1}^m \lambda_i (-y_i \mathbf{w}^T x_i + y_i b - \xi_i + 1) - \sum_{i=1}^m \mu_i \xi_i$$

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{w} - \sum_{i=1}^m \lambda_i \mathbf{y}_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \frac{1}{2} \sum_{i=1}^m \lambda_i \mathbf{y}_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^m \lambda_i \mathbf{y}_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \Rightarrow \mu_i = C - \lambda_i$$

3.2 Primal feasibility

$$\begin{cases} 1. -y_i(w^T x_i - b) + 1 - \xi_i \leq 0 & \text{for } i = 1, 2, \dots, N \\ 2. \xi_i \geq 0 & \text{for } i = 1, 2, \dots, N \end{cases}$$

3.3 Dual feasibility

$$\begin{cases} 1. \lambda_i \geq 0 & \text{for } i = 1, 2, \dots, N \\ 2. \mu_i \geq 0 & \text{for } i = 1, 2, \dots, N \end{cases}$$

3.4 Complementary slackness

$$\begin{cases} 1. \lambda_i (-y_i(w^T x_i - b) + 1 - \xi_i) = 0 & \text{for } i = 1, 2, \dots, N \Rightarrow \lambda_i = 0 \text{ or } (-y_i(w^T x_i - b) + 1 - \xi_i) = 0 \\ 2. -\mu_i \xi_i = 0 & \text{for } i = 1, 2, \dots, N \Rightarrow \mu_i = 0 \text{ or } \xi_i = 0 \end{cases}$$

4 Dual Problem Derivation

In this task we will find optimal values of dual problem using CVXOPT.solvers.qp function.

4.1 Introduction

The same as part 1, we need to find P,q, G, h, A, b values.

$$\min_{\lambda} \frac{1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i^T \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j \lambda - \sum_{i=1}^m \lambda_i = \min_{\lambda} \frac{1}{2} \lambda^T \underbrace{\left(\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \mathbf{y}_i \mathbf{y}_j \mathbf{x}_i^T \mathbf{x}_j \right)}_P \lambda - \underbrace{\mathbf{1}^T}_{\mathbf{q}^T} \lambda \quad (1)$$

From previos part in KKT third stationary conditions we have : $\mu_i = C - \lambda_i$ }
 From previos part in KKT first dual feasibility condition we have : $\lambda_i \geq 0$ } \Rightarrow

$$\Rightarrow 0 \leq \lambda_i \leq C \Rightarrow -\lambda \leq 0, \lambda \leq C \Rightarrow \quad \mathbf{G}_{2m \times m} = \begin{bmatrix} -\mathbf{I}_{(m,m)} \\ \mathbf{I}_{(m,m)} \end{bmatrix} \quad \text{and} \quad \mathbf{h}_{2m} = \begin{bmatrix} \mathbf{0}_{(m)} \\ \mathbf{C}_{(m)} \end{bmatrix} \quad (2)$$

$$\text{From KKT second stationary conditions we have : } \mathbf{y}^T \lambda = 0 \Rightarrow \mathbf{A}_{1 \times n} = \mathbf{y}^T \quad \text{and} \quad b = 0 \quad (3)$$

4.2 Implementation

For implementation, we finalize the linear_svm_dual function in svm_dual.py.

```
def linear_svm_dual(X, y, C=1):
    m, n = X.shape

    # define values of P, q, G, h, A, b
    linear_kernel = np.dot(X, X.T)
    P = 0.5*np.outer(y, y) * linear_kernel

    q = -np.ones(m)

    G = np.block([[ -np.identity(m)],
                  [ np.identity(m) ]])

    h = np.block([ -np.zeros(m),
                  C*np.ones(m) ])

    A = np.reshape(y, (1, m))

    b = 0.0

    # solve problem using cvxopt.solvers.qp
    sol = solvers.qp(matrix(P, tc='d'), matrix(q, tc='d'), matrix(G, tc='d'),
                    matrix(h, tc='d'), matrix(A, (1, m), tc='d'), matrix(b, tc='d'))

    # extract Lagrange multipliers
    alphas = np.ravel(sol['x'])

    return alphas
```

5 Strong Duality Verification

To verify strong duality, we use results of previous part and compute primal problem and dual problem values to check weather they are equal or not.

5.1 Implementation

For implementation, we finalize the StrongDuality function in main.py.

```
def StrongDuality(X, y, C=1):
    # find Lagrange multipliers from dual problem
    alpha = linear_svm_dual(X, y, C)

    # find weights and bias from primal problem
    w, b, distances = linear_svm_primal(X, y, C)

    # compute primal problem optimal value
    primal_optimal_value = np.dot(w.T, w) + C*np.sum(distances)

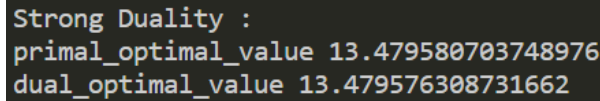
    # compute dual problem optimal value
    linear_kernel = X*np.reshape(y, (-1,1))
    P = np.dot(linear_kernel, linear_kernel.T)
    dual_optimal_value = -0.25*(np.dot(np.dot(alpha.T, P), alpha)) + np.sum(alpha)

    print("\nStrong-Duality:-")
    print("primal_optimal_value", float(primal_optimal_value))
    print("dual_optimal_value", float(dual_optimal_value))

    return
```

5.2 Evaluation

The output of implemented function is shown in figure 10. As we can see, the optimal values of primal and dual are the same.



```
Strong Duality :
primal_optimal_value 13.479580703748976
dual_optimal_value 13.479576308731662
```

Figure 10: Strong duality

6 Solve Primal via Dual

Now that we have optimal values from dual problem, we can compute weights using the first condition in KKT.

$$\frac{\partial L}{\partial w} = 2\mathbf{w} - \sum_{i=1}^m \lambda_i \mathbf{y}_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \frac{1}{2} \sum_{i=1}^m \lambda_i \mathbf{y}_i \mathbf{x}_i$$

For support vectors $\xi_i = 0$, and we have $y_i(w^T x_i - b) = 1 \Rightarrow y_i^2(w^T x_i - b) = y_i$ and since $y_i^2 = 1$ (labels are +1 and -1), we can find support vectors and compute $bias = \frac{1}{m} \sum_{i=1}^m (w^T x_i - y_i)$, where m is the number of support vectors.

7 Linear SVM by The Dual Problem

Now we bring everything together and implement linear svm by the dual problem.

7.1 Implementation

For implementation, we finalize the `Linear_SVM_Primal.via.Dual` function in `main.py`.

```
def linear_svm_primal_via_dual(X, y, C=1):
    # find Lagrange multipliers from dual problem
    alpha = linear_svm_dual(X, y, C)

    # find support vectors
    support_vectors = np.where((alpha > 1e-4) & (alpha <= C))[0]
    # compute weights using dual problem results
    w_dual = 0.5 * np.dot((y[support_vectors] * alpha[support_vectors]).T,
                          X[support_vectors, :])
    # compute bias using dual problem results
    b_dual = np.mean(X[support_vectors, :].dot(w_dual) - y[support_vectors])

    # compute weights and bias directly from primal problem
    w_primal, b_primal, _ = linear_svm_primal(X, y, C=1)

    # reshape and print weights to check whether they are the same or not
    w_primal = np.reshape(w_primal, (-1))
    w_dual = np.reshape(w_dual, (-1))
    print("\nPrimal problem weights:-", w_primal, "bias:-", b_primal)
    print("Dual problem weights:-", w_dual, "bias:-", b_dual)

    return w_dual, b_dual
```

7.2 Evaluation

In this part we train the model on two datasets, including "without slack" given dataset and Iris dataset. The following results in figure 11 show Precision, Recall and F1-score for "without slack" dataset. Also, the heat map of the confusion matrix is shown in figure 12. For visualization, decision boundaries are plotted in figure 13.

```
Primal via Dual Evaluation :
Class -1.0 - Precision: 0.8000, Recall: 0.8000, F1-score: 0.8000
Class 1.0 - Precision: 0.8000, Recall: 0.8000, F1-score: 0.8000
Accuracy: 0.8000
```

Figure 11: With slack dual problem Acc.

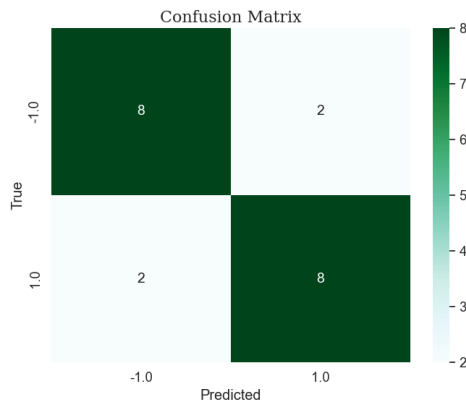


Figure 12: With slack dual problem CM

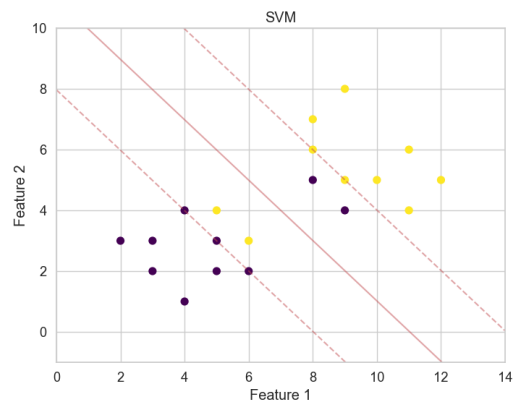


Figure 13: With slack decision boundaries

Same as part 1, the following results in figure14 are related to Iris test dataset. Also, the heat map of the confusion matrix is shown in figure 15. For visualization, we need to reduce 4 dimensions of features to 2. The figure16 illustrates the classification results.

```
Dual via Dual Evaluation :
Class -1 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Class 1 - Precision: 0.8000, Recall: 1.0000, F1-score: 0.8889
Accuracy: 0.9000
```

Figure 14: Iris dual problem Acc.

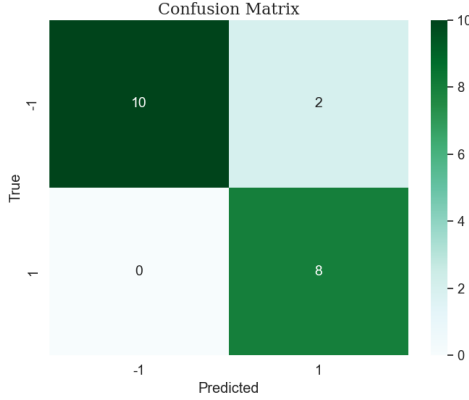


Figure 15: Iris dual problem CM

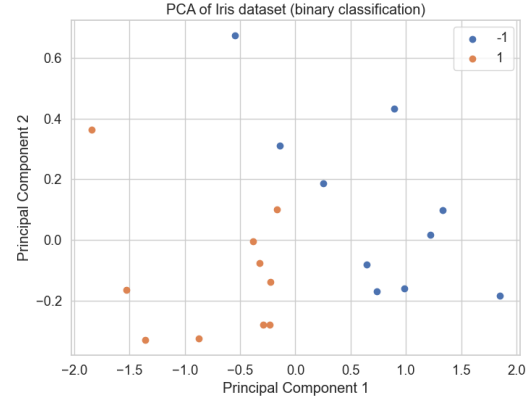


Figure 16: Iris dual problem visualization

It can be seen that the results of solving primal problem via dual problem is the same as the results of solving primal problem directly.

8 Kernel SVM

In this part we try to implement kernel svm to be able to handle non-linearly separable datasets.

8.1 Introduction

We can use the computed Lagrangian dual problem in part 2 and replace $\mathbf{x}_i^T, \mathbf{x}_j$ with $K(\mathbf{x}_i^T, \mathbf{x}_j)$:

$$\max_{\lambda} \frac{-1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j K(\mathbf{x}_i^T, \mathbf{x}_j) + \sum_{i=1}^m \lambda_i = \min_{\lambda} \frac{1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i^T \mathbf{y}_i \mathbf{y}_j K(\mathbf{x}_i^T, \mathbf{x}_j) \lambda_j - \sum_{i=1}^m \lambda_i \quad \text{subject to} \quad \sum_{i=1}^m \lambda_i \mathbf{y}_i = 0$$

The same as part 1, we need to find P,q, G, h, A, b values.

$$\min_{\lambda} \frac{1}{4} \sum_{i=1}^m \sum_{j=1}^m \lambda_i^T \mathbf{y}_i \mathbf{y}_j K(\mathbf{x}_i^T, \mathbf{x}_j) \lambda_j - \sum_{i=1}^m \lambda_i = \min_{\lambda} \frac{1}{2} \lambda^T \underbrace{\left(\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \mathbf{y}_i \mathbf{y}_j K(\mathbf{x}_i^T, \mathbf{x}_j) \right)}_{\mathbf{P}} \lambda - \underbrace{\mathbf{1}^T}_{\mathbf{q} \cdot \mathbf{T}} \lambda \quad (4)$$

From previos part in KKT third stationary conditions we have : $\mu_i = C - \lambda_i$ }
From previos part in KKT first dual feasibility condition we have : $\lambda_i \geq 0$ } \Rightarrow

$$\Rightarrow 0 \leq \lambda_i \leq C \Rightarrow -\lambda \leq 0, \lambda \leq C \Rightarrow \quad \mathbf{G}_{2m \times m} = \begin{bmatrix} -\mathbf{I}_{(m,m)} \\ \mathbf{I}_{(m,m)} \end{bmatrix} \quad \text{and} \quad \mathbf{h}_{2m} = \begin{bmatrix} \mathbf{0}_{(m)} \\ \mathbf{C}_{(m)} \end{bmatrix} \quad (5)$$

$$\text{From KKT second stationary conditions we have : } \mathbf{y}^T \lambda = 0 \Rightarrow \mathbf{A}_{1 \times n} = \mathbf{y}^T \quad \text{and} \quad b = 0 \quad (6)$$

Now that we have Lagrange multipliers, we can find support vectors and compute bias to make predictions. For any support vector x_i from m support vectors, we have $y_i(\sum_{i=1}^m \frac{1}{2} \lambda_m \mathbf{y}_m K(x_i, x_m) - b) = 1 \Rightarrow y_i^2(\sum_{i=1}^m \frac{1}{2} \lambda_m \mathbf{y}_m K(x_i, x_m) - b) = y_i$ and since $y_i^2 = 1$ (labels are +1 and -1), to compute bias, we can find support vectors and compute $bias = \frac{1}{m} \sum_{i \in SV} (\sum_{m \in SV} \frac{1}{2} \lambda_m \mathbf{y}_m K(x_i, x_m) - y_i)$

8.2 Implementation

For implementation, we finalize the `kernel_svm` and `compute_kernel` functions in `kernel_svm.py`.

```
def compute_kernel(kernel_type, X, X_prime, params):
    if kernel_type == "rbf":
        sigma = params.get("sigma", None)
        if sigma is None:
            sigma = 1.0
            prtin("Compute with default value of sigma=1.0")
        dst = np.sqrt(np.sum((X[:, None, :] - X_prime[None, :, :]) ** 2, axis=2))
        kernel = np.exp(-dst ** 2 / (2 * sigma ** 2))

    elif kernel_type == "polynomial":
        degree = params.get("degree", None)
        if degree is None:
            degree = 3
            prtin("Compute with default value of degree=3")
        kernel = (np.sum(X[:, None] * X_prime, axis=2) + 1) ** degree

    else:
        print(kernel_type + " is not supported!")
        exit()

    return kernel

def kernel_svm(X, y, C, kernel_type, params):
    m, n = X.shape

    # define values of P, q, G, h, A, b
    kernel = compute_kernel(kernel_type, X, X, params)
    P = 0.5 * np.outer(y, y) * kernel

    q = -np.ones(m)

    G = np.block([[ -np.identity(m)],
                  [ np.identity(m) ]])

    h = np.block([ np.zeros(m),
                  C * np.ones(m) ])

    A = np.reshape(y, (1, m))

    b = 0.0

    # solve problem using cvxopt.solvers.qp
    sol = solvers.qp(matrix(P, tc='d'), matrix(q, tc='d'), matrix(G, tc='d'),
                    matrix(h, tc='d'), matrix(A, (1, m), tc='d'), matrix(b, tc='d'))

    # extract Lagrange multipliers
```

```

alpha = np.ravel(sol['x'])

# find support vectors
support_vectors = np.where((alpha > 1e-4))[0]

# compute bias
bias = np.sum(-y[support_vectors] + np.sum(0.5*alpha[support_vectors]
      * y[support_vectors]
      * kernel[:, support_vectors][support_vectors],
      axis=0))

bias /= (len(support_vectors))

return alpha, support_vectors, bias

def predict(alpha, X, X_prime, y, bias, kernel_type, params):

    kernel = compute_kernel(kernel_type, X, X_prime, params)

    # predict scores and predict classes of data points
    y_pred_scores = np.sum(0.5*alpha * y * kernel.T, axis=1) - bias
    y_pred = np.sign(y_pred_scores)

    return y_pred, y_pred_scores

```

8.3 Evaluation

In this part we train the model with RBF and polynomial kernels on the moons dataset, which is not linearly separable. Firstly, we try RBF kernel $K(x, x') = \exp(\frac{\|x-x'\|^2}{2\sigma^2})$ with $\sigma = 1$. The following results in figure 17 show Precision, Recall and F1-score. Also, the heat map of the confusion matrix is shown in figure 18. For visualization, decision boundaries are plotted in figure 19.

```

Evaluation :
Class -1.0 - Precision: 1.0000, Recall: 0.9787, F1-score: 0.9892
Class 1.0 - Precision: 0.9706, Recall: 1.0000, F1-score: 0.9851
Accuracy: 0.9875

```

Figure 17: RBF kernel Acc.

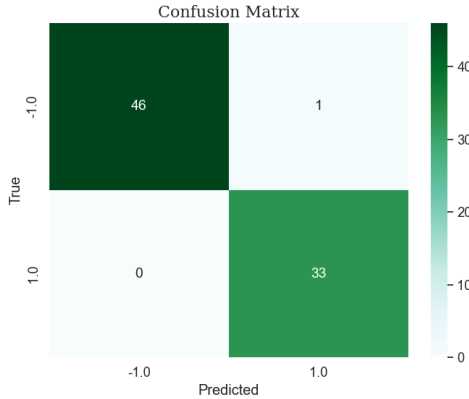


Figure 18: RBF kernel CM

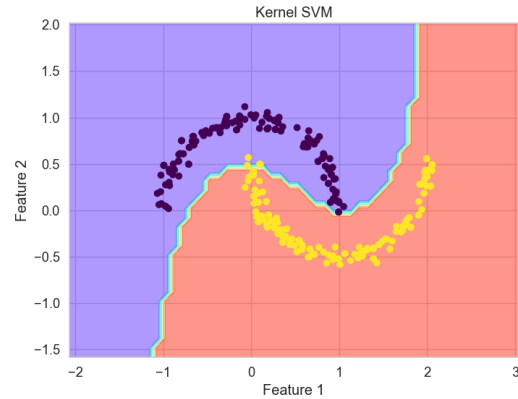


Figure 19: RBF kernel visualized results

Secondly, we use polynomial kernel $K(x, x') = (x^T x + 1)^{\text{degree}}$ with degree=3. The following results

in figure 20 show Precision, Recall and F1-score. Also, the heat map of the confusion matrix is shown in figure 21. For visualization, decision boundaries are plotted in figure 22.

```

Evaluation :
Class -1.0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1.0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Accuracy: 1.0000

```

Figure 20: Polynomial kernel Acc.

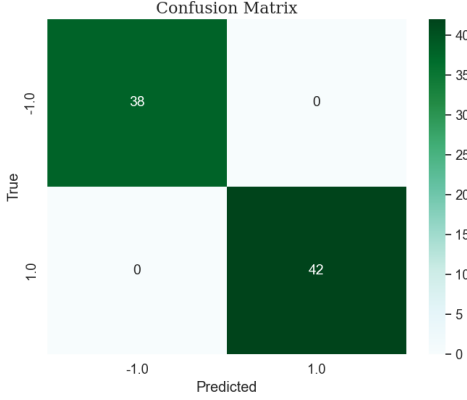


Figure 21: Polynomial kernel CM

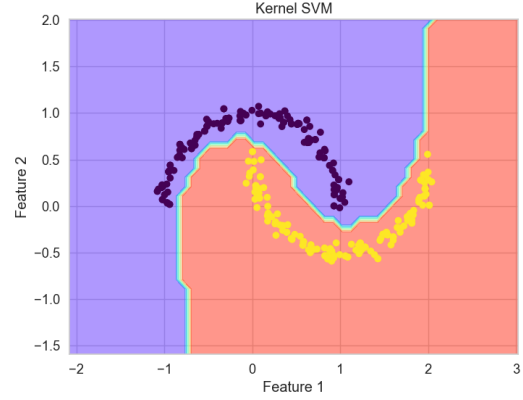


Figure 22: Polynomial kernel visualized results

9 Multiclass SVM

The purpose of this section is to extend the binary classification capability of SVM to handle multiple classes.

9.1 Introduction

One-vs-rest and one-vs-one are two strategies for converting the multi-class problem into multiple binary classification problems. In this project, we will implement one-vs-rest strategy. In this strategy, we need to select each class as a +1 class and other classes as -1 class, and solve dual problem. After that we need to combine binary predictions and make decision for multi-class prediction. In this project, we use the maximum score between scores of different binary predictions to assign a class to each point of dataset.

9.2 Implementation

For implementation, we finalize the `multiclass_svm` and `find_best_prediction` functions in `multiclass_svm.py`.

```

def find_best_prediction(X, classifiers):
    # find the max score to find the max-margin
    predictions = []
    for svm in classifiers:
        w = svm["weights"]
        b = svm["bias"]
        _, y_pred = predict(X, w, b)

```

```

        predictions.append(y_pred)

    best_pred = np.argmax(np.array(predictions), axis=0)

    return best_pred

def multiclass_svm(X, y, C=1):

    ovr_classifiers= []
    num_classes = len(np.unique(y_train))
    # one-vs-rest
    for i in range(num_classes):
        binary_y = y_train.copy()
        binary_y[binary_y != i] = -1
        binary_y[binary_y == i] = 1
        w_dual, b_dual = linear_svm_primal_via_dual(X_train, binary_y, C)
        ovr_classifiers.append({'weights':w_dual, 'bias':b_dual})

    return ovr_classifiers

```

9.3 Evaluation

In this part we train the model on Iris dataset. The following results in figure 23 show Precision, Recall and F1-score. Also, the heat map of the confusion matrix is shown in figure 24.

```

Multiclass Evaluation :
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 0.8571, Recall: 1.0000, F1-score: 0.9231
Class 2 - Precision: 1.0000, Recall: 0.9000, F1-score: 0.9474
Accuracy: 0.9600

```

Figure 23: Multi-class Iris Acc.

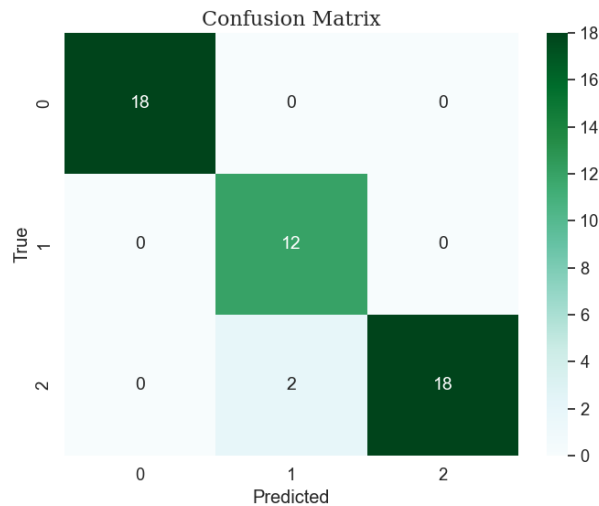


Figure 24: Multi-class Iris CM.

Also we train the model on 2 last features of Iris data points. The following results in figure 25 show Precision, Recall and F1-score for test data. Also, the heat map of the confusion matrix is shown in figure 26. For visualization, decision boundaries are plotted in figure 27. It can be seen that 3 classes of Iris dataset are separated from each other.

```

Multiclass Evaluation :
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 1.0000, Recall: 0.8182, F1-score: 0.9000
Class 2 - Precision: 0.7647, Recall: 1.0000, F1-score: 0.8667
Accuracy: 0.9200

```

Figure 25: Multi-class 2d Iris Acc.

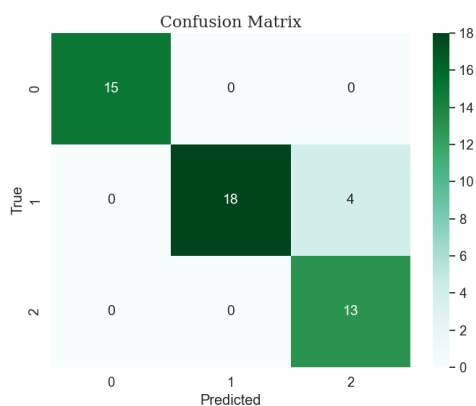


Figure 26: Multi-class 2d Iris CM

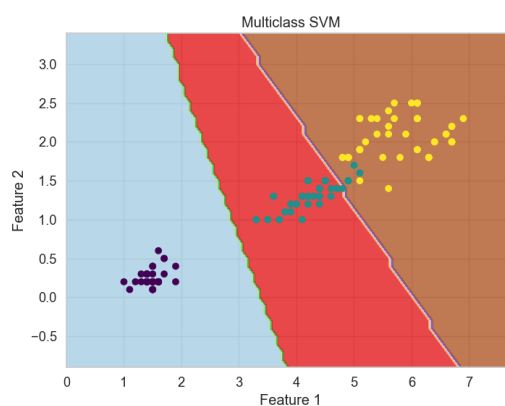


Figure 27: Multi-class 2d Iris visualized results