SVM (fully from scratch)

Mehran Tamjidi*

*K.N Toosi University of Technology

April 18, 2024

note: *The results correspond to the code provided*

1 Linear SVM by The Primal Problem

1.1 What is the primal problem?

formulation of the primal soft margin problem: Although the coefficient in the question provided neglected we add $\frac{1}{2}$ to the function to follow the common illustration and avoid multiplying coefficients which induces extra calculation in our problem.

Given training data $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i$ are the feature vectors and y_i are the corresponding class labels ($y_i \in \{-1, 1\}$), the objective is to find the optimal hyperplane defined by $\mathbf{w} \cdot \mathbf{x} + b = 0$ that maximizes the margin while minimizing the classification error. This is formulated as the following optimization problem:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \xi_i$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \ldots, N$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, N$$

Where:

- $\mathbf{w}$ is the weight vector perpendicular to the hyperplane,
- b is the bias term,
- $\boldsymbol{\xi} = [\xi_1, \xi_2, \ldots, \xi_N]$ are slack variables that measure the classification error for each sample $(\mathbf{x}_i, y_i)$,
- C is the regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error,
- N is the number of training samples.

1.2 Reformulating primal problem

In this formulation, the first term $\frac{1}{2}\|\mathbf{w}\|^2$ represents the margin maximization objective, and the second term $C\sum_{i=1}^{N} \xi_i$ penalizes the classification error. The parameter C determines the relative importance of these two objectives.

In this section the aim is to formulate the linear SVM primal problem in the form of standard CVXOPT optimization library.

CVXOPT, however, expects that the problem is expressed in the following form:

$$\min_x \frac{1}{2}x^T Px + q^T x$$

subject to

$$Ax = b$$

$$Gx \preceq h$$

We need to rewrite (6) to match the above format. Let's define a matrix P such that $P_{ij} = y_i y_j < x_i, x_j >$. We can then rewrite our original problem in vector form. We also multiply both the objective and the constraints by -1, which turns it into a minimization task. we take X to be

$$x = \begin{bmatrix} w \\ \text{bias} \\ \xi \end{bmatrix}$$

Now we try to find matrix A,b and G,h
We will find out the matrices: the matrix G is constructed by vertically stacking temp1 and temp2.

$$\text{tmp1} = \begin{bmatrix} -x_{11} * y_1 & \cdots & -x_{1n} * y_1 & -y_1 & -1 & 0 & \cdots & 0 \\ -x_{21} * y_2 & \cdots & -x_{2n} * y_2 & -y_2 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -x_{m1} * y_m & \cdots & -x_{mn} * y_m & -y_m & 0 & 0 & \cdots & -1 \end{bmatrix}_{(m)\times(m+n+1)}$$

$$\text{tmp2} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & -1 \end{bmatrix}_{(m)\times(m+n+1)}$$

We can vertically concatenate these two matrix for reaching to matrix G: in the form of

$$G = \begin{bmatrix} \text{tmp1} \\ \text{tmp2} \end{bmatrix}$$

the primal matrix of h and P and q also can be presented as bellow:

$$h = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \qquad q = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ C \\ C \\ \vdots \\ C \end{bmatrix}$$

$P = [[I0][00]]$ (it's first m value are one) $\implies ((m+n+1) \times (m+n+1))$
$q =$ (n+1 zero value and then m C value) $\implies ((m+n+1) \times 1)$
$G =$ is more complex matrix shown in the code format $\implies ((2m) \times (m+n+1))$
$h = ([-np.ones(m), np.zeros(m)]) \implies ((2m \times 1)$

in the matrix P the I is n by n and matrix and the matrix q has m+1 zero value and m value equal to C

```
    # construct P, q, A, b, G, h matrices for CVXOPT
    P = cvxopt.matrix(np.diag(np.hstack([np.ones((n)), np.zeros
((m + 1))])))
    q = cvxopt.matrix(np.vstack([np.zeros((n+1, 1)), C * np.ones
((m, 1))]))
    h = np.hstack([-np.ones(m), np.zeros(m)])
    h = matrix(h, tc='d')
    tmp1 = np.hstack([-y[:, np.newaxis] * X, -y.reshape(-1, 1),
-np.eye(m)])
    tmp2 = np.hstack([np.zeros((m, n + 1)), -np.eye(m)])
    G = np.vstack([tmp1, tmp2])
    G = matrix(G, tc='d')

```

1.3 Visualization and Evaluation

By obtaining the primal parameters (which we will find it's formula in the next section), we can plot and evaluate our implementation.

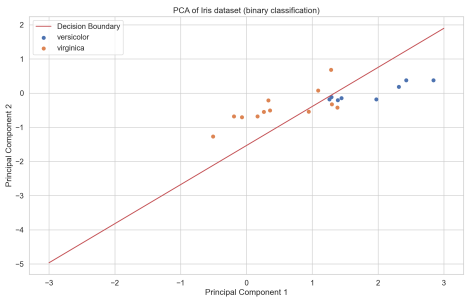

Figure 1: decision boundaries on test set for primal solution

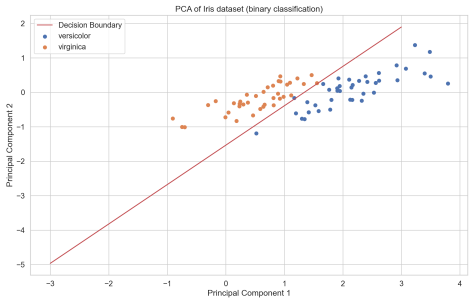

Figure 2: decision boundaries on train set for primal solution

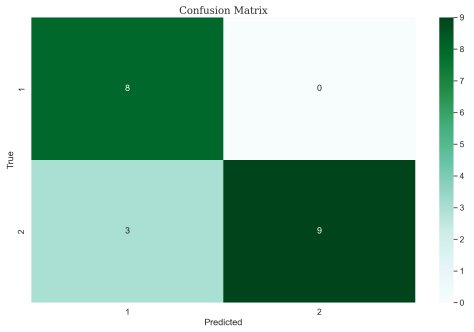

Figure 3: Confusion matrix for primal solutions

```
    Class 1 - Precision: 0.7273, Recall: 1.0000, F1-score:
    0.8421
    Class 2 - Precision: 1.0000, Recall: 0.7500, F1-score:
    0.8571
    Accuracy: 0.8500

```

We found W and b such that:

```
    weights= [[ 4.48283622][-3.92057491]]
    bias= -6.012898756852167

```

you can also check out for the slack parameters printed from the implemented function in 'part 1 primal problem.py'.

2 Dual Problem Derivation

The Lagrangian function is defined as:

$$L(x, y, \alpha, \beta) = f(x, y) - \alpha(g_1(x, y) - c_1) - \beta(g_2(x, y) - c_2)$$

We know that the primal problem being a QP implies that the dual problem is also a QP. The solution is to find (x, y, α, β) such that $\Delta L = 0$ and the derivatives of lagrangian function are zero: $\delta_{x,y\ ,\alpha\ ,\beta}\, L(x, y, \alpha, \beta) = 0$.
According to the primal problem and the constraints we write the corresponding Lagrangian function for soft margin SVM as follow:

$$\mathcal{L}(w, b, \xi, \alpha, \mu) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(w^T x_i + b) - 1 + \xi_i] - \sum_{i=1}^{N}\beta_i\xi_i$$

The dual problem is obtained by maximizing the Lagrangian function with respect to α, subject to the constraints $\alpha_i \geq 0$. The dual problem can be formulated as:

$$z^* = \max_{\alpha,\beta} \left(\min_{w,b,\xi} \mathcal{L}(w, b, \xi, \alpha, \mu) \right)$$

we took the derivatives where $z^*(\alpha, \beta)$ is the dual function.

$$\frac{\partial J(w, b, \xi, \alpha, \beta)}{\partial w} = 0, \text{ which yields } w = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{1}$$

$$\frac{\partial J(w, b, \xi, \alpha, \beta)}{\partial b} = 0, \text{ which yields } \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2}$$

and also we have:

$$\frac{\partial J(w, b, \xi, \alpha, \beta)}{\partial \xi} = 0 \implies C = \alpha_i + \beta_i \tag{3}$$

$$\begin{aligned} L(x, y, \alpha, \beta) =& \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ &+ \underbrace{b \sum_{i=1}^{N} \alpha_i y_i}_{\text{zero by(2)}} + \sum_{i=1}^{N} \alpha_i + \underbrace{C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} (\beta_i + \alpha_i) \xi_i}_{\text{zero by(3)}} \end{aligned} \tag{4}$$

The third term in this equation is zero because of the constraint on b, which gives us the final formulation of the problem. The first two terms are simplified and the last two terms by (3) are equal to each other. so the terms including ξ_is are eliminated and the resultant equation are demonstrated in (5).finally dual problem is that:

$$z^* = \max_{\alpha} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \right) \tag{5}$$

subject to the constraints $0 \leq \alpha_i \leq C$ for $i = 1, \ldots, N$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$. and also we find w and b as following equation:

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i \quad , \quad b = y_i - \sum_{j} \alpha_j y_j x_j^T x_i$$

In the section 6 we talk about the proof of obtaining w and b from the KKT.

3 KKT Condition Derivation

In this section we provide general Karush-Kuhn-Tucker (KKT) conditions for a constrained optimization problem:

Stationarity:

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{i=1}^{p} \mu_i^* \nabla h_i(x^*) = 0$$

Primal Feasibility:

$$g_i(x^*) \leq 0 \quad \text{for } i = 1, \ldots, m$$
$$h_i(x^*) = 0 \quad \text{for } i = 1, \ldots, p$$

Dual Feasibility:

$$\lambda_i^* \geq 0 \quad \text{for } i = 1, \ldots, m$$

Complementary Slackness:

$$\lambda_i^* g_i(x^*) = 0 \quad \text{for } i = 1, \ldots, m$$

Here, x^* is the optimal solution, f is the objective function, g_i are the inequality constraints, h_i are the equality constraints, λ_i^* and μ_i^* are the Lagrange multipliers associated with the constraints g_i and h_i, respectively. m is the number of inequality constraints, and p is the number of equality constraints.

These conditions characterize a point that is feasible with respect to the constraints, and at which the gradient of the objective function and the gradients of the constraints are aligned in a certain way. They are essential for identifying optimal solutions to constrained optimization problems. **The KKT conditions for the soft margin SVM problem are:**

Stationarity:

$$\nabla_w L = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0$$
$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0$$
$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \quad \text{for } i = 1, \ldots, N$$

Primal Feasibility:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, N$$
$$\xi_i \geq 0 \quad \text{for } i = 1, \ldots, N$$

Dual Feasibility:

$$\alpha_i \geq 0 \quad \text{for } i = 1, \ldots, N$$
$$\beta_i \geq 0 \quad \text{for } i = 1, \ldots, N$$

Complementary Slackness:

$$\alpha_i[y_i(w^T x_i + b) - 1 + \xi_i] = 0 \quad \text{for } i = 1, \ldots, N$$
$$\beta_i \xi_i = 0 \quad \text{for } i = 1, \ldots, N$$

These conditions, when satisfied simultaneously along with the primal feasibility conditions, characterize the optimal solution to the soft margin SVM problem.

4 The Dual Problem

4.1 Reformulating dual problem

The dual problem computed in (5) have to be in the standard format of CVXOPT library first we modify the (5) to minimize it instead of maximizing it.

$$z^* = \min_{\alpha} \left(\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{N} \alpha_i \right) \tag{6}$$

subject to the constraints $0 \leq \alpha_i \leq C$ for $i = 1, \ldots, N$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$.

CVXOPT, however, expects that the problem is expressed in the following form:

$$\min_x \frac{1}{2} x^T P x + q^T x$$

subject to

$$Ax = b$$

$$Gx \preceq h$$

We need to rewrite (6) to match the above format. Let's define a matrix P such that $P_{ij} = y_i y_j < x_i, x_j >$. We can then rewrite our original problem in vector form. We also multiply both the objective and the constraints by -1, which turns it into a minimization task.

$$\min_\alpha \left(\frac{1}{2} \alpha^T P \alpha - 1^T \alpha \right)$$

subject to

$$y^T \alpha = 0$$

$$-\alpha_i \leq 0 \quad \forall i$$

$$\alpha_i \leq C \quad \forall i$$

with the comparison we found all of the matrices and it's dimension.we will find out the matrices:

$$\begin{aligned}
P &= YY^T K \quad \Longrightarrow \text{ (K is kernel)} \\
q &= -\text{eye(m)} \Longrightarrow (m \times m) \\
A &= Y^T \Longrightarrow (1 \times m) \\
b &= 0 \quad \Longrightarrow \text{ (scalar)} \\
G &= \text{vstack}(-\text{eye(m)}, \text{eye(m)}) \quad \Longrightarrow (2m \times m) \\
h &= (\text{m zero value and then m C value}) \Longrightarrow (2m \times 1)
\end{aligned}$$

```
# construct P, q, A, b, G, h matrices for CVXOPT
P = cvxopt.matrix(np.outer(y,y) * K)
q = cvxopt.matrix(np.ones(n_samples) * -1)
A = cvxopt.matrix(y, (1,n_samples))
```

```
    b = cvxopt.matrix(0.0)
    G = cvxopt.matrix(np.vstack((np.diag(np.ones(n_samples) *
-1), np.identity(n_samples))))
    h = cvxopt.matrix(np.hstack((np.zeros(n_samples), np.ones(
n_samples) * C)))
    # solve QP problem
    print('finding quadratic programming')
    solution = cvxopt.solvers.qp(P, q, G, h, A, b)

```

Then we can return the Lagrangian multipliers from the function implemented in "part 5 The Dual Problem.py"

```
    Lagrangian multipliers= [2.84999011e-089.99999983e
+002.64132980e+00 7.10115538e-08
    9.58400141e-08 6.51494152e-08 4.46317109e-07 1.28767344e-07
    7.56558807e-08 2.52156437e-08 7.41622824e-08 7.80581100e-08
    4.17268237e-08 2.20709756e-07 5.03645013e-08 3.01591050e-07
    1.06407886e-07 1.52911514e-07 8.01209104e-08 2.33894100e-07
    7.37898898e-08 3.09485187e-07 3.86263085e-08 1.65358000e-07
    9.99999980e+00 6.71982850e-08 7.75619563e-08 7.67139355e-08
    9.02138124e-08 7.45402461e-07 7.22841426e-08 1.65999999e-07
    6.45885745e-08 6.00617284e-08 1.38801018e-07 5.17679692e-08
    1.40614491e-07 7.78320346e-08 1.50922558e-07 1.62311047e-07
    6.93634654e+00 4.59348626e-07 4.67818762e-08 4.06542401e-08
    3.95023620e-08 4.20607836e-08 7.81039857e-08 1.93859560e-08
    8.84418678e-08 2.62581733e-07 8.83686620e-08 1.86117732e-07
    3.98755231e-08 2.83389385e-08 1.53671048e-07 3.62943279e-08
    6.67770968e-08 5.42294851e-08 6.41118857e-08 5.60664119e-08
    8.38388272e-08 5.90166884e-08 9.99999419e+00 3.51140677e-08
    8.33736034e-08 5.05574683e-08 5.50903448e-08 3.89899729e-07
    6.80618510e-07 1.23492785e-07 5.70497900e+00 8.86093740e-08
    8.19083621e-08 4.90400654e-08 9.99999936e+00 9.99999949e+00
    2.24369193e-07 3.62260994e-08 1.07757916e-07 1.07757916e-07]

```

5 Strong Duality Verification

To verify the strong duality we have to find weights and biases from the both dual and primal cases.Then using the function 'np.allclose()' we check the vicinity of the answers. Here is the output of the function "checking strong duality" in the "part 6 Strong duality verification.py" file.

```
  ----------------------------------------
```

```
weights_primal= [[ 4.48283622]
[-3.92057491]] bias_primal= -6.012898756852167
w_dual= [ 4.48283305 -3.92057581] bias_dual= -5.976280206093889
----------------------------------------
[1.14341297] 1.14341190375244
[1.53367781] 1.524337367339736
Are weight of dual and primal equal? True
Are bias of dual and primal equal? True
```

As we see the weights obtained from dual and primal completely matches to each other hence the strong duality holds!

6 Solve Primal via Dual

To find the primal solution from the dual solution, we can use the Karush-Kuhn-Tucker (KKT) conditions or minimize the Lagrangian at the optimal dual variables with respect to the primal variables.

One approach is to use the KKT conditions. The KKT conditions provide necessary conditions for optimality in constrained optimization problems. By satisfying the KKT conditions, we ensure that both the primal and dual solutions are optimal.

Alternatively, we can minimize the Lagrangian function with respect to the primal variables at the optimal dual variables. This involves setting the derivatives of the Lagrangian function with respect to the primal variables to zero and solving for those variables. hence we do that to directly obtain the primal solution from the dual solution without explicitly solving the primal problem(this happens when storng duality holds).

6.1 At the first step we write down the KKT conditions:

Stationarity:

$$\frac{\partial}{\partial w_i} L(w, b, \xi, \alpha, \beta) = 0 \quad \text{for } i = 1, 2, ..., n \tag{7}$$

$$\frac{\partial}{\partial b} L(w, b, \xi, \alpha, \beta) = 0 \tag{8}$$

$$\frac{\partial}{\partial \xi_i} L(w, b, \xi, \alpha, \beta) = 0 \quad \text{for } i = 1, 2, ..., m \tag{9}$$

Primal Feasibility:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, 2, ..., m \tag{10}$$
$$\xi_i \geq 0 \quad \text{for } i = 1, 2, ..., m \tag{11}$$

Dual Feasibility:

$$\alpha_i \geq 0 \quad \text{for } i = 1, 2, ..., m \tag{12}$$
$$\beta_i \geq 0 \quad \text{for } i = 1, 2, ..., m \tag{13}$$

Complementary Slackness:

$$\alpha_i[y_i(w^T x_i + b) - 1 + \xi_i] = 0 \quad \text{for } i = 1, 2, ..., m \tag{14}$$
$$\beta_i \xi_i = 0 \quad \text{for } i = 1, 2, ..., m \tag{15}$$

6.2 Solving KKT for SVM with slack

now we solve the KKT condition for SVM problem with slack. Given the optimal dual variables α^* and β^*, we can find the primal solution as follows: At the first step we try to satisfy stationary condition of SVM it will be done by computing derivative with respect to Lagrangian variable w^* and b^* using the following equations:

$$w^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i \tag{16}$$

$$b^* = y_k - \sum_{i=1}^{m} \alpha_i^* y_i (x_i \cdot x_k) \quad \text{for some } \alpha_k^* > 0 \tag{17}$$

where k is any index such that $0 < \alpha_k^* < C$. Use the complementary slackness condition to find the support vectors. Any data point x_i with $\alpha_i^* > 0$ is a support vector. Calculate the bias term b^* using one of the support vectors:

$$b^* = y_k - \sum_{i=1}^{m} \alpha_i^* y_i (x_i \cdot x_k) \tag{18}$$

By satisfying the KKT conditions and finding the primal variables w^* and b^* using the optimal dual variables α^* and β^*, we obtain the solution to the primal problem.

To find the primal solution from the dual solution, we can start by taking the derivative of the Lagrangian function with respect to the primal variables $\mathbf{w}$ and b. The Lagrangian function for the SVM primal problem is given by:

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{m}\beta_i\xi_i \tag{19}$$

We take the partial derivatives of L with respect to $\mathbf{w}$ and b and set them to zero to find the stationary points:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i \mathbf{x}_i = 0 \tag{20}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{21}$$

From Equation (20), we can solve for $\mathbf{w}$ to obtain:

$$\mathbf{w} = \sum_{i=1}^{m}\alpha_i y_i \mathbf{x}_i \tag{22}$$

Next, from Equation (21), we solve for b to get:

$$\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{23}$$

We can then use these expressions for $\mathbf{w}$ and b to derive the primal solution from the dual solution. $\mathbf{w}$ has already calculated now we use the (23) to find b^* we choose an Index i with $0 < \alpha_i^* < C$: then, we use Complementary Slackness that for each support vector i, one of the following must hold:

$$\alpha_i^* = 0 \quad \text{if} \quad y_i(w^T x_i + b^*) \geq 1 \tag{24}$$

$$\alpha_i^* = C \quad \text{if} \quad y_i(w^T x_i + b^*) \leq 1 \tag{25}$$

From complementary slackness, we have $\alpha_k^*(y_k(b^* + \mathbf{w}^{*T}\mathbf{x}_k) - 1 + \xi_k) = 0$. Since $\alpha_k^* > 0$ (from the condition $0 < \alpha_k^* < C$), this simplifies to $y_k(b^* + \mathbf{w}^{*T}\mathbf{x}_k) = 1$. Solve for b^*: from (22) We know that $\mathbf{w}^{*T}\mathbf{x}_k = \sum_{i=1}^{n}\alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x}_k \rangle$. now substitute this expression and finally we obtain:

$$y_k(b^* + \sum_{i=1}^{n}\alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x}_k \rangle) = 1 \tag{26}$$

we use the decision function for a support vector machine (SVM) is given by:

$$y_k(w^T x_k + b^*) = \sum_{i=1}^{n} \alpha_i^* y_i \langle x_i, x_k \rangle + b^* \tag{27}$$

Now, we solve for b^*: For each support vector i on the margin, the complementary slackness condition tells us that

$$\alpha_i^*(y_i(w^T x_i + b^*) - 1 + \xi_i) = 0. \tag{28}$$

Since $\alpha_i^* > 0$ (from the condition $0 < \alpha_i^* < C$), this simplifies to

$$y_i(w^T x_i + b^*) = 1. \tag{29}$$

Calculate Bias Term b^*: Since $y_i = \pm 1$, we have

$$b^* = y_i - w^T x_i \quad \text{for any support vector on the margin.} \tag{30}$$

we can rewrite equation (30) as below:

$$b^* = y_k - \sum_{i=1}^{n} \alpha_i^* y_i \langle x_i, x_k \rangle \tag{31}$$

This equation gives us the value of b^* that satisfies the condition for the chosen support vector k. We can use this b^* along with the computed w^* to obtain the primal solution from the dual solution.

If we change to soft-margin SVM with slack variable C, and changing the notation of label. We could generate a run time comparison betwenn Primal solution and Dual solution. The run time for Dual solution is almost one-half to the Primal in changing of slack variable or changing sample size, which makes it a more convenient way of solving SVM.thats one of the draws back of SVM.

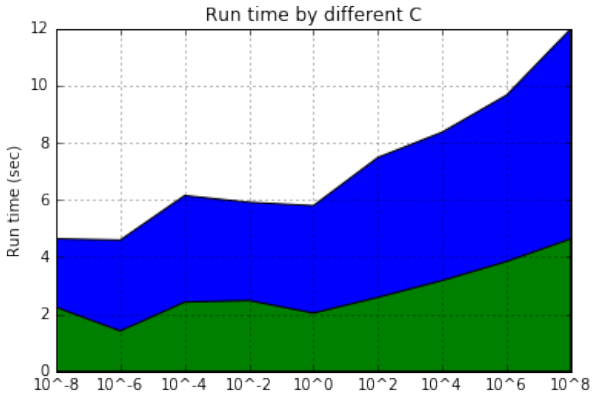

Figure 4: run time comaprison

7 Linear SVM by The Dual Problem

We use the implemented dual function in the previous sections for this part. Therefore in this section we only illustrate visualization, results, and parameters of the dual problem.

7.1 Visualization and Results

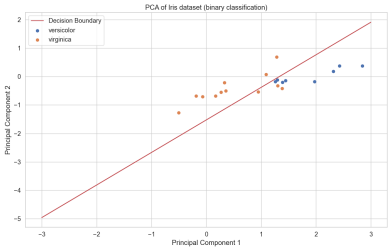

Figure 5: Separator for dual problem on test set

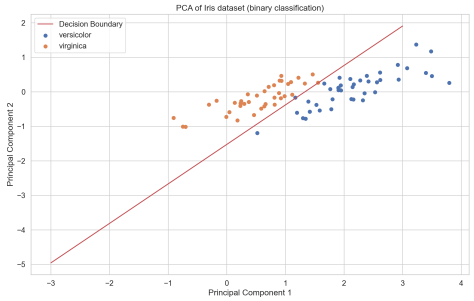

Figure 6: Separator for dual problem on train set

```
Class 1 - Precision: 0.7273, Recall: 1.0000, F1-score: 0.8421
Class 2 - Precision: 1.0000, Recall: 0.7500, F1-score: 0.8571
Accuracy: 0.8500

```

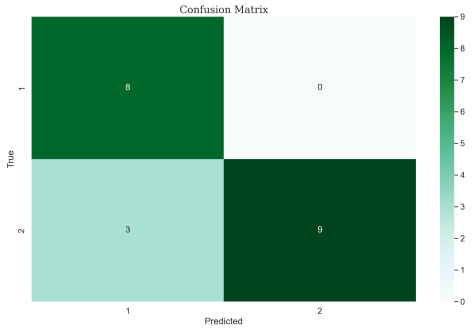

Figure 7: Confusion matrix of dual problem

we also can check the value of the weights and bias obtained from the dual problem as follows:

```
w= [ 4.48283305 -3.92057581] bias= -5.976280206093889
```

8 Implement Kernel SVM

we often use the some of the regular kernel to map our data to higher dimension like linaer,RBF ,Polynomial and Sigmoid.

8.1 commonly used kernel functions

1. Linear Kernel

   Linear Kernel Formula:

   $$F(x, x_j) = \sum_{i=1}^{n} x_i \cdot x_{ji}$$

   Here, x and x_j represent the data we're trying to classify.

2. Polynomial Kernel

   Polynomial Kernel Formula:

   $$F(x, x_j) = (x \cdot x_j + 1)^d$$

   Here ' $\cdot$ ' denotes the dot product of both the values, and d denotes the degree. $F(x, x_j)$ represents the decision boundary to separate the given classes.

3. Gaussian RBF Kernel

   Gaussian Radial Basis Formula:

$$F(x, x_j) = \exp(-\gamma||x - x_j||^2)$$

   The value of γ varies from 0 to 1. We have to provide the value of γ in the code manually. The most preferred value for γ is 0.1.

4. Sigmoid Kernel

   Sigmoid Kernel Formula:

$$F(x, x_j) = \tanh(\alpha x \cdot y + c)$$

we could see that in the code block format:

```
1
2     def linear_kernel(x1, x2):
3     return np.dot(x1, x2)
4
5     def polynomial_kernel(x, y, C=1, d=3):
6     return (np.dot(x, y) + C) ** d
7
8     def gaussian_kernel(x, y, gamma=0.5):
9     return np.exp(-gamma*linalg.norm(x - y) ** 2)
10
11    def sigmoid_kernel(x, y, alpha=1, C=0):
12    return np.tanh(alpha * np.dot(x, y) + C)
13
```

Then we can compute prediction for each kernel function.

8.2 Visualization and evaluation

At the first step the decision boundaries and confusion matrix for polynomial function with parameter $C = 10$ and polynomial-degree=7 are plotted.

```
1     Class 1 - Precision: 0.7273, Recall: 1.0000, F1-score:
      0.8421
2     Class 2 - Precision: 1.0000, Recall: 0.7500, F1-score:
      0.8571
3     Accuracy: 0.8500
4
```

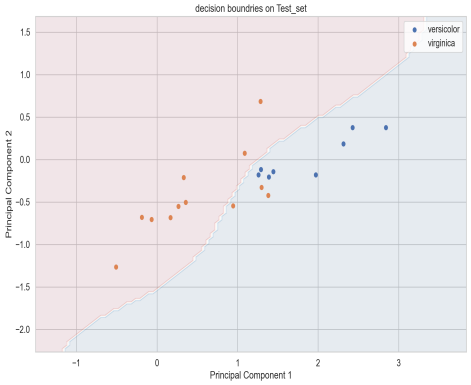

Figure 8: Decision boundaries on train set for C=10 and polynomial-degree=7

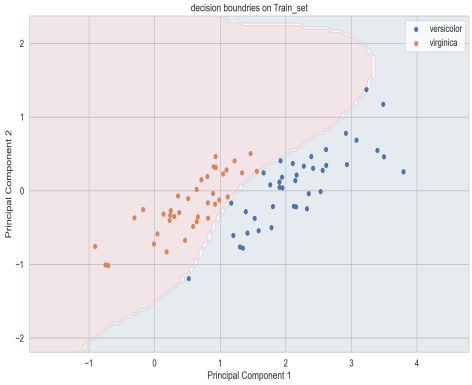

Figure 9: Decision boundaries on test set for C=10 and polynomial-degree=7

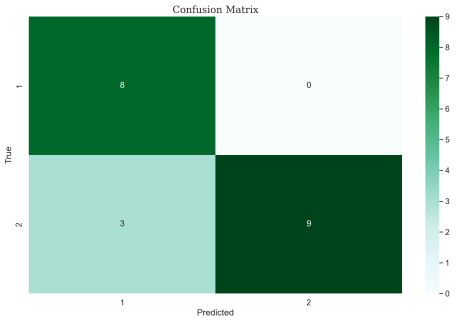

Figure 10: Confusion matrix for C=10 and polynomial-degree=7

to get better result we use RBF function as the kernel function and we consider C=1 and RBF parameter equal to 10.

Feel free to run "part 11 kernel svm.py" in the attached files for other kernel functions with different parameters we can make a grid search to find the optimum parameter and kernel for this example.

```
1    Class 1 - Precision: 0.8000, Recall: 1.0000, F1-score:
     0.8889
2    Class 2 - Precision: 1.0000, Recall: 0.8333, F1-score:
     0.9091
3    Accuracy: 0.9000
4
```

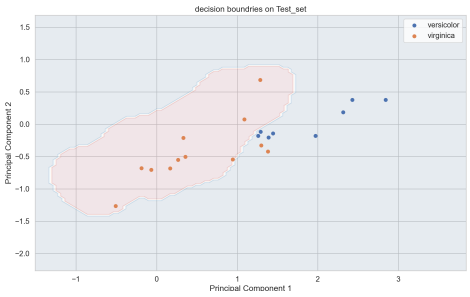

Figure 11: decision boundaries on test set for RBF function with C=1 and RBF parameter equal to 10

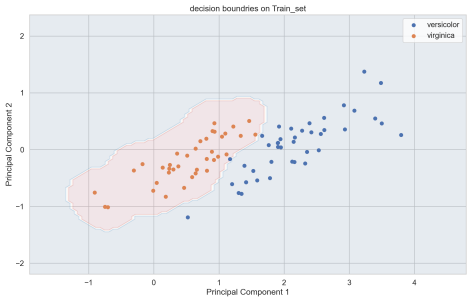

Figure 12: decision boundaries on test set for RBF function with C=1 and RBF parameter equal to 10

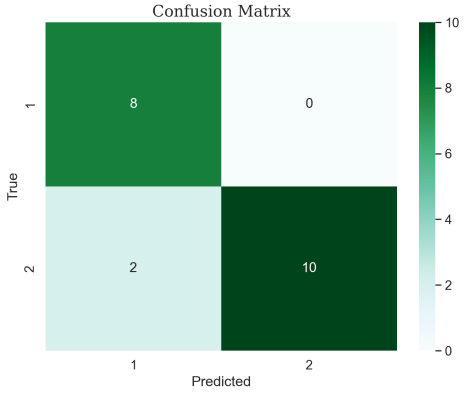

Figure 13: Confusion matrix for RBF function

9 Implement Multiclass SVM

We use the same structure in the previous section for multiclass classification.

9.1 Multiclass Approaches

For multiclass classification there is three main structure:

- **One-vs-One (OvO):**
  - In OvO, a binary classifier is trained for every pair of classes in the dataset.

  - During prediction, each classifier votes for one of the classes, and the class with the most votes is selected as the final prediction.
  - **Pros and cons:** Simple and easy to implement but Requires training $\frac{N(N-1)}{2}$ classifiers for N classes, which can be computationally expensive for large datasets.This approach also has the ambiguous region problem.

- **One-vs-All (OvA or One-vs-Rest):**
  - In OvA, a binary classifier is trained for each class, treating samples from that class as positive examples and samples from all other classes as negative examples.
  - During prediction, the classifier with the highest confidence score is selected as the predicted class.
  - **Pros and cons:** Requires only N classifiers to be trained, making it more computationally efficient than OvO. Imbalanced class distributions may affect classifier performance, especially when dealing with classes of different sizes.This approach also has the ambiguous region problem.

- **Direct Optimization (or Direct Multi-Class Classification):**
  - In direct optimization, a single classifier is trained to directly distinguish between all classes simultaneously.In this method we use K discriminant function and using their magnitude for classification.
  - The classifier learns a decision boundary that separates different classes in the feature space.
  - **Advantage:** Can be computationally efficient, especially for large datasets, as it avoids the need to train multiple binary classifiers.
  - **Disadvantage:** Optimization can be more complex than OvO or OvA, and the decision boundary may be harder to interpret.

In our work we use One-VS-all algorithm because it is computationally more efficient!

For implementation of One-VS-all we should assign a label 1 to one class of data and assign -1 to the rest of the data points for each class. Then we

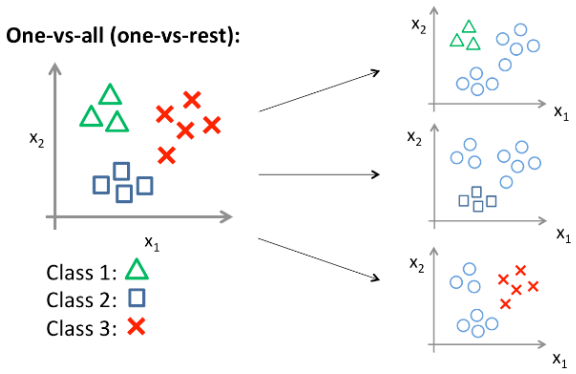

Figure 14: One-VS-all concept

concatenated all of the data points in a one array .The algorithm of One-VS-all is based on majority voting.Which means that we have to return only one class of data and with the maximum confidence.hence we can use argmax values to predict the class of predicted data.

9.2 Visualization and Evaluation

At the first step, we plot multi-class classification on the IRIS dataset considering all of its three classes. We illustrate that our data is working well on separating each class from the rest of the data.

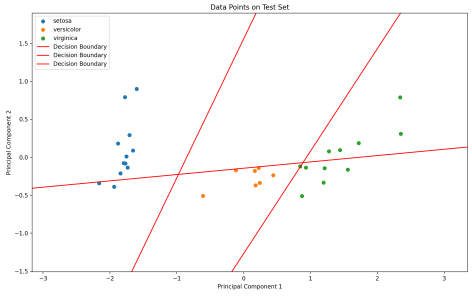

Figure 15: One-VS-all on the test set

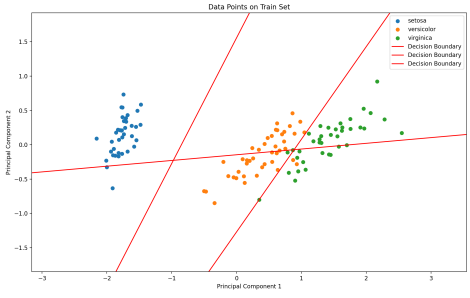

Figure 16: One-VS-all on the train set

This algorithm also works well on the confusion matrix and benchmarks.

But we could use the direct Multi-Class Classification and use the magnitude of prediction instead of the sign to have better accuracy with respect

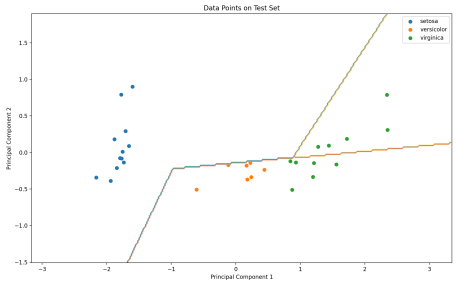

Figure 17: Decision boundaries in linear kernel classifier for the test set

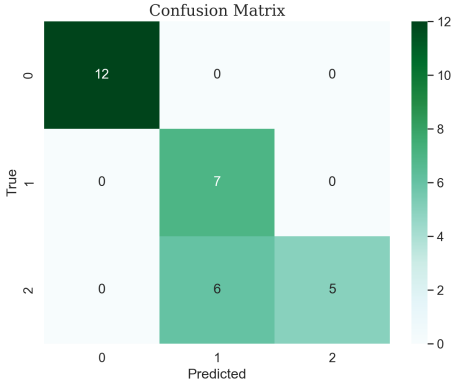

Figure 18: Confusion matrix of linear multi-class SVM using OVA

```
   Class 0 - Precision:
1.0000, Recall: 1.0000,
F1-score: 1.0000
   Class 1 - Precision:
0.5385, Recall: 1.0000,
F1-score: 0.7000
   Class 2 - Precision:
1.0000, Recall: 0.4545,
F1-score: 0.6250
   Accuracy: 0.8000

```

Listing 1: Performance of linear multiclass classifier

```
   Class 0 - Precision:
1.0000, Recall: 1.0000,
F1-score: 1.0000
   Class 1 - Precision:
0.8750, Recall: 1.0000,
F1-score: 0.9333
   Class 2 - Precision:
1.0000, Recall: 0.9091,
F1-score: 0.9524
   Accuracy: 0.9667

```

Listing 2: Performance of for polynomial with$C = 10$ and polynomial-degree=7

to one-vs-all.

For the visualization, if we use the polynomial function with parameters $C = 10$ and polynomial-degree=7, it could yield a promising result!

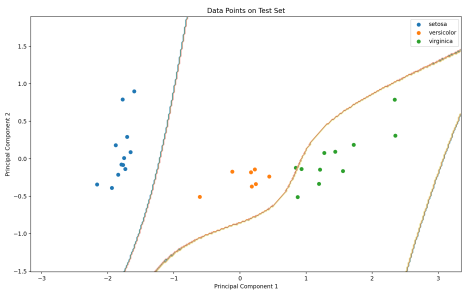

Figure 19: One-VS-all decision boundaries on the train set for $C = 10$ and polynomial-degree=7

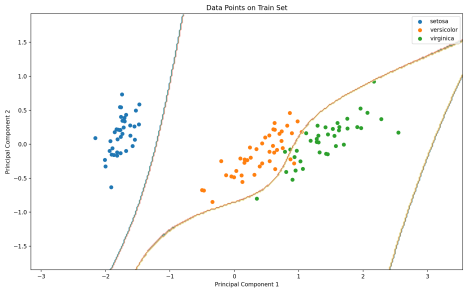

Figure 20: One-VS-all decision boundaries on the test set for $C = 10$ and polynomial-degree=7

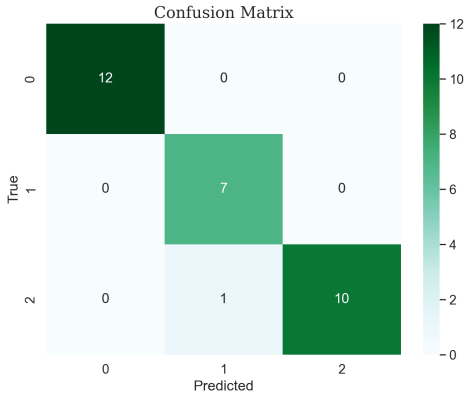

Figure 21: Confusion matrix for $C = 10$ and polynomial-degree=7

Feel free to run "part 12 multiclass classification.py" in the attached files for other kernel functions with different parameters.