## Question 1

The hyperplane in d dimension $\mathbb{R}^{\mathbf{d}}$ is given as $w^T x + b = 0$.

**Decision Variables**

- $w$ (weight vector): Determines the orientation and direction of the hyperplane.
- $b$ (bias scalar): Determines hyperplane offset. If $b$ is positive, hyperplane shifts away from the origin.

**Objective**

To maximise the margin between the two classes, by maximizing the distance between the two parallel hyperplanes $H^+$and $H^-$. Since that is given as $\frac{2}{||w||}$, we can minimize $||w||$.

Thus, the objective function is $\min_{w,b} \frac{1}{2}||w||^2$

**Constraints**

The two classes are labelled as $y_i \in \{-1, +1\}$ and the correct classification is as follows:

$$y_i(w^T x_i + b) \geq 1, \qquad i = 1, \dots, N$$

For when $y_i = +1$ (point belongs to class +1 in hyperplane $H^+$), then $w^T x + b \geq 1$

And when $y_i = -1$ (point belongs to class –1 in hyperplane $H^-$), then $w^T x + b \leq -1$

## Question 2

**Step 1: Map the problem**

Decision variables: $z = (w, b)$

Objective function: To minimize $f(z) = \frac{1}{2}||w||^2$

Feasible set $Z$ :

1. Original SVM constraints: $y_i(w^T x_i + b) \geq 1,\ i = 1, \dots, N$
2. Additional bound constraints: $-M \leq w_j \leq M$ for $j = 1, \dots, d$
   $-M \leq b \leq M$ Where $M > 0$

**Step 2: Find the initial feasible solution**

The first step is to initialize $w_0$ and $b_0$ within the feasible region. To find a feasible solution, we solve a linear programming (LP) feasibility problem, with a dummy objective since a feasible solution is needed, not the optimization of a specific value.

Objective: $$\min_{w,b} 0$$

subject to: $$y_i(w^T x_i + b) \geq 1,\ \forall i$$

If a feasible solution is found, we use it as $(w_0, b_0)$. Otherwise, we increase $M$ *and retry.*

**Step 3: Compute gradient of the objective function for direction**

The gradient provides the direction of the steepest descent, so this is a minimization problem. To get the gradient $\nabla f(z)$ for the objective function, we must take the partial derivative:

- $\nabla f(z)$ with respect to $w$ is $\partial f / \partial w = w$ & with respect to $b$ is $\partial f / \partial b = 0$
- $\nabla f(z) = (w, 0)$

At the iteration k, as we are solving for $d_k$, the gradient is $\nabla f(z) = (w_k, 0)$

**Step 4: Compute feasible directions**

The direction at k iteration is given as $d_k = v_k - z_k$. We first need to solve for $v_k$, by finding:

$$\min_{v \in Z} f(z_k)^T v$$

The vector $v$ is a candidate solution in the feasible set $Z$ which can be written as

$$v = (v_w, v_b)$$

And since $\nabla f(z_k) = (w_k, 0)$, the problem can be rewritten as:

$$\min_{v \in Z} \nabla f(z_k)^T v = w_k^T v_w + 0 \cdot v_b = w_k^T v_w$$

After computing the problem to solve for $v_k$, we can then solve

$$d_k = v_k - z_k$$

**Step 5: Compute step size**

Solve for $\tau_k$ using $$\min_{\tau \in [0,1]} f(z_k + \tau \cdot d_k)$$

Which can be written as $$\min_{\tau \in [0,1]} \frac{1}{2} \left|\left| w_k + \tau \cdot d_k \right|\right|^2$$

To find the value of $\tau_k$, take derivative with respect to $\tau$

$$\frac{d}{d\tau}\left(\frac{1}{2} \left|\left| w_k + \tau \cdot d_k \right|\right|^2\right) = 0$$

$$\frac{1}{2} \cdot 2(w_k + \tau d_k)^T d_k = 0$$

$$\tau_k = -\frac{w_k^T d_k}{d_k^T d_k}$$

**Step 6: Update solution with $\boldsymbol{\tau_k}$**

Repeat until convergence $$z_{k+1} = z_k + \tau_k \cdot d_k$$

**Step 7: Check for convergence**

The algorithm continues until $$\left|\left| z_k - z_{k-1} \right|\right| \leq \epsilon$$

where $\epsilon$ is a small tolerance value (e.g., $10^{-5}$). If convergence is achieved, then the optimal solution ($w^*$, $b^*$) is found.

## Question 3

**Data preparation**

The iris dataset has 3 classes of plants, but we are only interested in identifying whether the plant is an *iris setosa.* The class *iris setosa* is transformed to be labelled as +1 and everything else as -1, in line with the labels given $y_i$.

**Initial feasible solution**

Linear programming initialization method was chosen to find the initial feasible solution $z_0 = (w_0, b_0)$ that satisfies the constraints and is within the bounds $M$. The LP solver prioritizes feasibility over optimality, and the initial solution it gave pushes the solution to the bounds. Though the initial values are unusual, it is corrected by the feasible direction method.

**Feasible direction method**

The implemented feasible direction method in the code closely follows the steps outlined in question 2. It starts with an initial feasible solution that satisfies all constraints, then iteratively improves upon itself by finding the direction and calculating the step size that reduces the objective function. The algorithm stops when it reaches the tolerance parameter $\epsilon$. The code finds the optimal solution at varying levels of $M$ and $\epsilon$ to explore their effects on the solution.

**Setting tolerance parameter $\epsilon$**

As $\epsilon$ is a small number close to 0, we tested values from a larger 0.01 to a minute $10^{-8}$. When $\epsilon$ is larger, the optimal value and $\tau_k$ is larger than the outcome with smaller $\epsilon$, meaning, it controls the precision of the solution. A larger $\epsilon$ value also requires less iterations and therefore has a faster runtime. A smaller $\epsilon$ value leads to more accurate solution but requires more iterations.

**Setting M**

The $M$ value defines the bounds for $w$ and $b$. We tested the values of $M$ from 1 to 100. $M = 1$ means that the feasible region is small, and it violates the constraints and converges much faster. When $M$ is larger (10, 50, 100), solver was able to find optimal solutions without violating the constraints. In addition, the objective solution gets smaller and therefore, is more optimal with larger $M$. As $M$ increases, it approaches the unbounded question 1. The solution no longer hits the bounds at larger $M = 10, 50, 100$ and therefore is also the answer to question 1.

## Question 4

To classify whether a record belongs to the class *iris versicolor*, we can set up an SVM classifier using a one-vs-all approach, where we treat *iris versicolor* as the positive class (+1) and the other two (Setosa and Virginica) as the negative class (-1).

The following is the given SVM optimization problem:

$$\min_{w,b,u} \frac{1}{2}\left||w|\right|^2 + C.\sum_{i=1}^{N} u_i$$

subject to:
$$y_i(w^T x_i + b) \geq 1 - u_i, \quad i = 1,2,3, \dots, N,$$
$$u_i \geq 0, \qquad where\ i = 1, \dots, N$$

where:

- $w$ is the weight vector (defining the decision boundary).
- $b$ is the bias term.
- $y_i$ is the label of each training sample (+1 for versicolor, -1 for others).
- $u_i$ are slack variables (allowing some misclassification).
- $C > 0$ is a regularization parameter that controls the trade-off between margin maximization and misclassification.

Compared to a previous hard-margin SVM (which requires perfect separability), this formulation introduces:

I. **Slack Variables ($u_i$)**

- These variables allow misclassification by relaxing the strict constraint $y_i(w^T x_i + b)$.
- If $u_i = 0$ , the point is correctly classified and lies on or outside the margin.
- If $0 < u_i < 1$ , the point is correctly classified but inside the margin.
- If $u_i > 1$ , the point is misclassified.

II. **The Term $C.\sum u_i$ in the Objective Function**

- This penalizes misclassification by adding the total slack variable sum to the objective.
- Higher $C \rightarrow$ less tolerance for misclassification, leading to a narrower margin.
- Lower $C \rightarrow$ more tolerance for misclassification, leading to a wider margin.

$C$ is a hyperparameter that needs to be tuned via cross-validation. The following factors are to be considered while choosing the value of $C$

- A small $C$ allows more flexibility but can lead to a high error rate (underfitting).
- A large $C$ forces fewer misclassifications but can lead to poor generalization (overfitting).
- The optimal $C$ is determined using grid search or validation techniques.

**Using One-vs-All approach to classify a new Iris record**

Approach: One-vs-All (OvA)

- One-vs-All (OvA): Train three SVM classifiers, each separating one class from the others.
  - $\mathrm{SVM}_{\text{setosa}}$ vs $(Versicolor + Virginica)$
  - $\mathrm{SVM}_{\text{versicolor}}$ vs $(Virginica + Setosa)$
  - $\mathrm{SVM}_{\text{virginica}}$ vs $(Setosa + Versicolor)$
- For a new test point $x$, we compute the decision function for each classifier:

$$f_{setosa}(x) = w_{setosa} \cdot x + b_{setosa}$$
$$f_{versicolor}(x) = w_{versicolor} \cdot x + b_{versicolor}$$
$$f_{virginica}(x) = w_{virginica} \cdot x + b_{virginica}$$

- Assign $x$ to the class with the highest decision function value

**Motivation behind the proposed algorithm**:

- Robust Classification: SVMs are well-suited for high-dimensional classification problems like Iris data.
- Handles Overlapping Classes: The soft-margin formulation ensures that even if some data points are misclassified, the model generalizes well.
- Scalability: One-vs-All (OvA) SVMs are computationally efficient for small datasets like the Iris dataset.

## Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 20 13:41:03 2025

@author: AAMA Group 30
"""

import numpy as np
import pandas as pd
import pyomo.environ as pyo
from pyomo.opt import SolverFactory
from scipy.optimize import linprog


# Load Data
file_name = 'Data.csv'
df = pd.read_csv(file_name, header=None, index_col=False)
# Create label y for Iris-setosa=1 and everything else=-1
df['label'] = df[4].apply(lambda x: 1 if x == 'Iris-setosa' else -1)

X = df.iloc[:, :4].values  # Features
y = df['label'].values  # Labels
num_samples, num_features = X.shape


# Find the initial feasible solution with LP
def lp_initialization(X, y, num_features, M):
    # Minimise 0 as objective
    c = np.zeros(num_features + 1)

    # Constraints: y_i * (w^T x_i + b) ≥ 1
    A = -y[:, np.newaxis] * np.hstack((X, np.ones((len(X), 1))))
    b = -np.ones(len(X))  # Reformulated to ≤ constraint for linprog

    # Bounds for w and b
    bounds = [(-M, M)] * num_features + [(-M, M)]

    # Solve the LP problem
    res = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='highs')

    if res.success:
        return res.x[:-1], res.x[-1]  # w, b
    else:
        raise ValueError("No feasible solution found.")


# Compute feasible direction
def direction(w_k, X, y, M):
    model = pyo.ConcreteModel()
    model.j = pyo.RangeSet(len(w_k))
    model.i = pyo.RangeSet(len(X))
    model.v_w = pyo.Var(model.j, bounds=(-M, M))
```

```
    model.v_b = pyo.Var(bounds=(-M, M))

    model.obj = pyo.Objective(expr=sum(w_k[j - 1] * model.v_w[j] for j in
model.j), sense=pyo.minimize)

    def svm_constraint(model, i):
        return y[i - 1] * (sum(model.v_w[j] * X[i - 1, j - 1] for j in
model.j) + model.v_b) >= 1

    model.svm_constr = pyo.Constraint(model.i, rule=svm_constraint)
    solver = SolverFactory('glpk')
    solver.solve(model)

    v_w = np.array([pyo.value(model.v_w[j]) for j in model.j])
    v_b = pyo.value(model.v_b)
    return v_w, v_b

# Compute step size
def step_size(w_k, d_w):
    model = pyo.ConcreteModel()
    model.tau = pyo.Var(bounds=(0, 1))

    def objective_rule(model):
        w_new = w_k + model.tau * d_w
        return 0.5 * sum(w_new[j]**2 for j in range(len(w_new)))

    model.obj = pyo.Objective(rule=objective_rule, sense=pyo.minimize)
    solver = SolverFactory('ipopt')
    solver.solve(model)

    return pyo.value(model.tau)

# Define constraint check function
def check_constraints(X, y, w_opt, b_opt):
    violations = sum(y[i] * (np.dot(X[i], w_opt) + b_opt) < 1 for i in
range(len(X)))
    return violations == 0, violations

# Calculate accuracy
def accuracy(X, y, w, b):
    predictions = np.sign(np.dot(X, w) + b)
    accuracy = np.mean(predictions == y)
    return accuracy

# Update solution and repeat until convergence
M_values = [1, 10, 50, 100]
epsilon_values = [0.01, 1e-4, 1e-6, 1e-8]
results = []

for M in M_values:
    for epsilon in epsilon_values:
        # Use LP-based initialization
        w0, b0 = lp_initialization(X, y, num_features, M)
```

```
        w_k, b_k = w0, b0
        previous_w, previous_b = np.zeros(num_features), 0
        max_iterations, iteration = 500, 0

        while iteration < max_iterations:
            v_w, v_b = direction(w_k, X, y, M)
            d_w, d_b = v_w - w_k, v_b - b_k
            tau_k = step_size(w_k, d_w)

            w_k, b_k = w_k + tau_k * d_w, b_k + tau_k * d_b
            norm_diff = np.linalg.norm(w_k - previous_w) + abs(b_k -
previous_b)
            if norm_diff < epsilon:
                break

            previous_w, previous_b = w_k, b_k
            iteration += 1

        # Compute the objective function value
        objective_value = 0.5 * np.linalg.norm(w_k)**2

        # Check constraints
        satisfied, violations = check_constraints(X, y, w_k, b_k)

        # Compute accuracy score
        accuracy_score = accuracy(X, y, w_k, b_k)

        # Add everything to results df
        results.append([M, epsilon, objective_value, w_k.tolist(), b_k,
tau_k, w0.tolist(), b0, iteration, satisfied, violations, accuracy_score])

# Store results in a DataFrame
results_df = pd.DataFrame(results, columns=['M', 'Epsilon', 'Objective
Value', 'w_opt', 'b_opt', 'tau_k', 'w0', 'b0', 'Iterations', 'Constraints
Satisfied', 'Violations', 'Accuracy'])
print(results_df)

# Takes around 3 minutes to run
```

<u>Output Table:</u>

| Index | M | Epsilon | bjective Valu | w_opt | b_opt | tau_k | w0 | b0 | Iterations | traints Sati | Violations | Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.01 | 0.767641 | [0.08938546682740516, 0.4581005830415231, -0.9832402535163012, -0.5921787071587912] | 1 | 0.122882 | [-0.07843137254901959, 1.0, -1.0, -1.0] | 1 | 2 | False | 2 | 1 |
| 1 | 1 | 0.0001 | 0.767641 | [0.08938546682740516, 0.4581005830415231, -0.9832402535163012, -0.5921787071587912] | 1 | 0.122882 | [-0.07843137254901959, 1.0, -1.0, -1.0] | 1 | 2 | False | 2 | 1 |
| 2 | 1 | 1e-06 | 0.767641 | [0.08938546682740516, 0.4581005830415231, -0.9832402535163012, -0.5921787071587912] | 1 | 0.122882 | [-0.07843137254901959, 1.0, -1.0, -1.0] | 1 | 2 | False | 2 | 1 |
| 3 | 1 | 1e-08 | 0.767641 | [0.08938546781443328, 0.45810058004560256, -0.9832402498236542, -0.5921787149969572] | 1 | 0.122873 | [-0.07843137254901959, 1.0, -1.0, -1.0] | 1 | 3 | False | 3 | 1 |
| 4 | 10 | 0.01 | 0.749182 | [-0.04713431720178595, 0.5204641293188066, -1.003617787296277, -0.4669171610611163] | 1.46331 | 0.00386101 | [0.9803921568627452, 10.0, -10.0, -10.0] | 10 | 131 | True | 0 | 1 |
| 5 | 10 | 0.0001 | 0.748348 | [-0.0462099634111393, 0.5213428906457951, -1.0033668459427247, -0.4647762699695963] | 1.45357 | 0.00142865 | [0.9803921568627452, 10.0, -10.0, -10.0] | 10 | 500 | True | 0 | 1 |
| 6 | 10 | 1e-06 | 0.748348 | [-0.0462099634111393, 0.5213428906457951, -1.0033668459427247, -0.4647762699695963] | 1.45357 | 0.00142865 | [0.9803921568627452, 10.0, -10.0, -10.0] | 10 | 500 | True | 0 | 1 |
| 7 | 10 | 1e-08 | 0.748348 | [-0.0462099634111393, 0.5213428906457951, -1.0033668459427247, -0.4647762699695963] | 1.45357 | 0.00142865 | [0.9803921568627452, 10.0, -10.0, -10.0] | 10 | 500 | True | 0 | 1 |
| 8 | 50 | 0.01 | 0.749106 | [-0.04719548507534549, 0.520421436818202, -1.0036121122533814, -0.466807013491628] | 1.46378 | 0.00355991 | [5.686274509803922, 50.0, -50.0, -50.0] | 50 | 140 | True | 0 | 1 |
| 9 | 50 | 0.0001 | 0.748348 | [-0.04686476221934687, 0.5215997672696969, -1.0033861431315252, -0.4643801509993115] | 1.45594 | 0.0015546 | [5.686274509803922, 50.0, -50.0, -50.0] | 50 | 500 | True | 0 | 1 |
| 10 | 50 | 1e-06 | 0.748348 | [-0.04686476221934687, 0.5215997672696969, -1.0033861431315252, -0.4643801509993115] | 1.45594 | 0.0015546 | [5.686274509803922, 50.0, -50.0, -50.0] | 50 | 500 | True | 0 | 1 |
| 11 | 50 | 1e-08 | 0.748348 | [-0.04686476221934687, 0.5215997672696969, -1.0033861431315252, -0.4643801509993115] | 1.45594 | 0.0015546 | [5.686274509803922, 50.0, -50.0, -50.0] | 50 | 500 | True | 0 | 1 |
| 12 | 100 | 0.01 | 0.749105 | [-0.047187216100561355, 0.5203412915641247, -1.003670267662792, -0.4667690539903383] | 1.4641 | 0.00359116 | [11.568627450980394, 100.0, -100.0, -100.0] | 100 | 140 | True | 0 | 1 |
| 13 | 100 | 0.0001 | 0.748348 | [-0.04708617118711686, 0.521681581296269, -1.0034477457072468, -0.46413306803817567] | 1.45678 | 0.0028173 | [11.568627450980394, 100.0, -100.0, -100.0] | 100 | 500 | True | 0 | 1 |
| 14 | 100 | 1e-06 | 0.748348 | [-0.04708617118711686, 0.521681581296269, -1.0034477457072468, -0.46413306803817567] | 1.45678 | 0.0028173 | [11.568627450980394, 100.0, -100.0, -100.0] | 100 | 500 | True | 0 | 1 |
| 15 | 100 | 1e-08 | 0.748348 | [-0.04708617118711686, 0.521681581296269, -1.0034477457072468, -0.46413306803817567] | 1.45678 | 0.0028173 | [11.568627450980394, 100.0, -100.0, -100.0] | 100 | 500 | True | 0 | 1 |