Optimisation : Assignment 2

Yashaswi Sood

December 9, 2024

1 Problem statement

In this assignment, we are tasked with applying the Support Vector Machine (SVM), a widely-used technique in automatic pattern classification, to a real-world medical dataset provided by the University of Wisconsin. The data consists of measurements related to tumor cells, classified as either malignant or benign. The goal is to develop a classification model that can be used to predict whether a new tumor is benign or malignant based on various features such as radius, area, smoothness, and others.

2 Background

In pattern classification, the objective is to create a rule that can classify objects into one of two categories based on their features. In this case, we aim to classify tumor cells as either benign or malignant. We are provided with a set of m data points, each consisting of n features representing measurements of the tumor cells. These features are used to classify each data point into one of two categories.

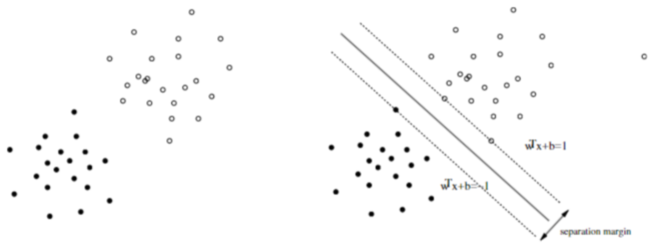

Figure 1: a) Two clusters of data points. b) Clusters and seperating plane

To solve this classification problem, we use the Support Vector Machine (SVM), a supervised machine learning algorithm that constructs a decision boundary, or hyperplane, to separate the two classes. The SVM algorithm finds the optimal hyperplane that maximizes the separation margin between the two classes. The points that are closest to the hyperplane are called support vectors, and these are the key elements in determining the optimal hyperplane. The SVM works by defining a linear decision boundary in an n-dimensional feature space. The hyperplane is represented by the equation:

$$\mathbf{w}^T\mathbf{x}_i + b = 0 \tag{1}$$

Where w is the weight vector, x is a feature vector, and b is the bias term. The objective is to find w and b such that the data points from one class lie on one side of the hyperplane, and the points from the other class lie on the opposite side. For an ideal case where the data is perfectly separable, the conditions for all data points can be written as:

$$w^T x_i + b = \begin{cases} 1, & \text{benign} \\ -1, & \text{malignant} \end{cases} \tag{2}$$

These conditions ensure that the benign class points are on one side of the hyperplane and the malignant class points are on the other. However, in real-world applications, data is often not perfectly separable, leading to violations of these conditions.

In situations where the data is not perfectly separable, we introduce slack variables ξ to allow some points to violate the separation conditions. This results in a new set of constraints:

$$\mathbf{w}^T x_i + b \geq 1 - i, \quad \text{benign} \tag{3}$$

$$w^T x_i + b \leq -1 + i, \quad \text{malignant} \tag{4}$$

$$\xi_i \geq 0, \quad \text{for all } i \tag{5}$$

The optimization problem is then reformulated as:

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m} \xi_i \tag{6}$$

Where C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing classification errors. The term $C\sum_{i=1}^{m} \xi_i$ penalizes violations of the margin constraints, while $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ ensures that the margin is maximized.

3 Tasks

3.1 Task 1

The problem of finding the optimal hyperplane for Support Vector Machines (SVMs) can be expressed as a quadratic programming (QP) problem in standard form. The decision variables are defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{w} \\ b \\ \boldsymbol{\xi} \end{bmatrix},$$

Where:

- $\mathbf{w}$ is the weight vector ($n \times 1$),
- b is the bias term (scalar),
- $\boldsymbol{\xi}$ is the vector of slack variables ($m \times 1$).

Thus, $\mathbf{x}$ has dimension $n + 1 + m$. The objective function for SVM is:

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m} \xi_i.$$

This can be written in standard QP form as:

$$\text{minimize} \quad \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{f}^T\mathbf{x},$$

Where:

- $\mathbf{H}$ is a block diagonal matrix:

  $$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

  Where $\mathbf{I}$ is the $n \times n$ identity matrix.

- $\mathbf{f}$ is:

  $$\mathbf{f} = \begin{bmatrix} \mathbf{0}_n \\ 0 \\ C\mathbf{1}_m \end{bmatrix},$$

  Where $\mathbf{0}_n$ is a zero vector of size n, and $\mathbf{1}_m$ is a vector of ones of size m.

The constraints are derived from the SVM conditions:

- $y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \forall i = 1, \dots, m,$
- $\xi_i \geq 0, \quad \forall i = 1, \dots, m.$

These can be rewritten in matrix form as:

$$\mathbf{A}\mathbf{x} \leq \mathbf{b},$$

Where:

- $\mathbf{A}$ is:

$$\mathbf{A} = \begin{bmatrix} \text{diag}(\mathbf{y}) \cdot \mathbf{X} & \mathbf{y} & -\mathbf{I}_m \\ \mathbf{0} & \mathbf{0} & -\mathbf{I}_m \end{bmatrix},$$

  Where:

  - diag($\mathbf{y}$) is the diagonal matrix of class labels y_i,
  - $\mathbf{X}$ is the $m \times n$ feature matrix,
  - $\mathbf{I}_m$ is the $m \times m$ identity matrix.

- $\mathbf{b}$ is:

$$\mathbf{b} = \begin{bmatrix} \mathbf{1}_m \\ \mathbf{0}_m \end{bmatrix},$$

  Where $\mathbf{1}_m$ and $\mathbf{0}_m$ are m-dimensional vectors of ones and zeros, respectively.

3.2 Task 4 and 5

The optimization problem was solved for different values of the regularization parameter C. Below are the performance metrics obtained for $C = 1000$:

- **Accuracy**: 0.9469
- **Sensitivity**: 1.0000
- **Specificity**: 0.9310

For $C = 1000$, we observe the following confusion matrix outcomes:

- **False Positives (FP)**: 6
- **False Negatives (FN)**: 0
- **True Negatives (TN)**: 81
- **True Positives (TP)**: 26

These results suggest that the classifier achieves perfect sensitivity (no false negatives) and high specificity with only a small number of false positives.

3.3 Task 6

The optimization problem was solved for different values of the regularization parameter C, where:

$$C = [100, 200, 300, 500, 700, 900, 1000, 1200]$$

The desired traits of the classifier are **high accuracy**, **high specificity**, and **high sensitivity**. Based on the graph, $\mathbf{C = 1200}$ gives the best value for all three of these parameters.

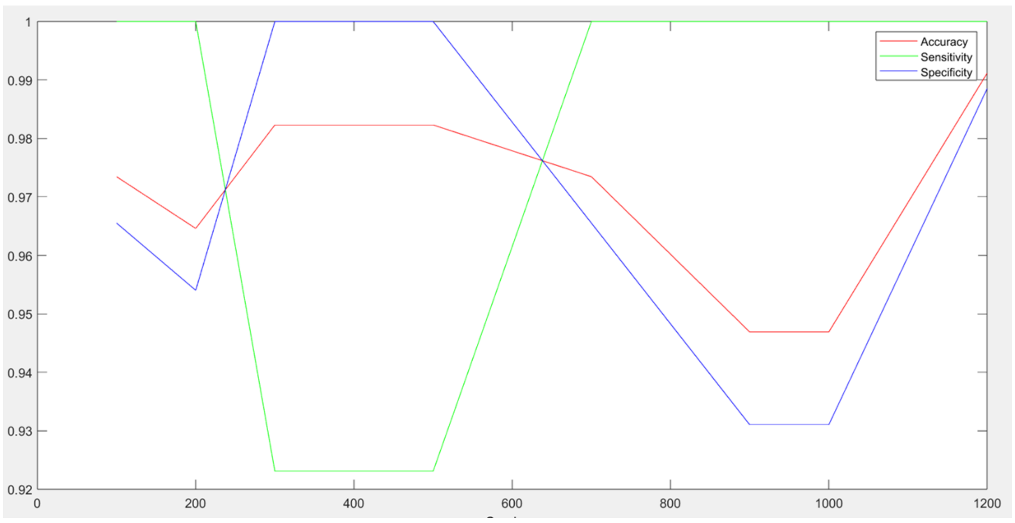

Figure 2: Accuracy, Sensitivity and Specificity vs C value

4 Appendix

4.1 Code

```
%Loading Data - Data Wrangling is done in CSV format
data = readtable("C:\Users\Tejaswi S\Documents\yvs\period2\optimization\ass2\svm\bcwd\wdbc_modifiedlat.cs
%Extracting labels
labels = data{:, 2};
%Extracting relevant features (Dropping patient ID)
features = data{:, 3:end};

%Train/Test Split : 80/20
train_features = features(1:456, :);
train_labels = labels(1:456);
y = train_labels;
X = train_features;
test_features = features(457:end, :);
test_labels = labels(457:end);
%C value
C = 1000;
[m, n] = size(train_features);

%Building the required matrices
%H
H = [eye(n), zeros(n, m + 1);
     zeros(m + 1, n + m + 1)];
%Margin constraints
A_margin = [-y .* X, -y, -eye(m)];
%Slack variables constraint matrix
A_slack = [zeros(m, n + 1), eye(m)];
%Combining both A matrices
A = [A_margin; A_slack];
%Same for b
b_margin = ones(m, 1);
b_slack = zeros(m, 1);
b = [b_margin; b_slack];

%Data storage
acc = []; sens = []; spec = [];
tp = []; fn = []; tn = []; fp = [];
%Testing for various C values
```

```
C_values = [100, 200, 300, 500, 700, 900, 1000, 1200];
for i = 1:length(C_values)
     C = C_values(i);
     %Setting f vector
     f = [zeros(n, 1); 0; C * ones(m, 1)];

     %Solution
     options = optimoptions('quadprog', 'Display', 'off');
     [sol, fval] = quadprog(H, f, A, b);

     %Resultant weights and biases
     w = sol(1:n);
     bias = sol(n+1);
     slack = sol(n+2:end);
     disp(size(w))
     disp(size(slack));
     disp(size(bias));

     %Test
     predictions = sign(test_features * w + bias);
     accuracy = mean(predictions == test_labels);
     disp(accuracy)

     % Evaluate predictions
     true_positives = sum((predictions == 1) & (test_labels == 1));
     false_positives = sum((predictions == 1) & (test_labels == -1));
     true_negatives = sum((predictions == -1) & (test_labels == -1));
     false_negatives = sum((predictions == -1) & (test_labels == 1));

     tp = [tp, (true_positives)];
     tn = [tn, (true_negatives)];
     fp = [fp, (false_positives)];
     fn = [fn, (false_negatives)];

     accuracy1 = (true_positives + true_negatives) / length(test_labels);
     sensitivity = true_positives / (true_positives + false_negatives);
     specificity = true_negatives / (true_negatives + false_positives);

     % Display results
     acc = [acc,accuracy1];
     sens = [sens, sensitivity];
     spec = [spec, specificity];
end
```