UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Numerical Tools Final Project

Author:

Matheus Victor Do Prado Amaral

Professor:

Alex Ferrer Ferre

Index

Chapter 1

Introduction to Numerical Tools

1.1 Continuous optimization

References: Convex Optimization (Boyd) and Numerical Optimization (Nocedal).

Mathematical problem

$$\min_{\theta} f_0(\theta)$$

$$s.t. \quad \underset{i=1...m}{f_i} (\theta) \leq 0 \quad \leftarrow constraints$$

Similarly

$$\min_{\theta} f_0 \ (\theta)$$

$$\theta \in C$$

where

$$C = \{\theta \mid f_i(\theta) \leq 0 \ \forall i = 1...m\}$$

Names: Optimization, Mathematical programing (Operational research)

Convex optimization

We ask

$$f_0(\theta) \ to \ be \ convex$$

$$C \ to \ be \ convex$$

* Convex function

$f(t\theta_0 + (1-t)\theta_1) \leq tf(\theta_0) + (1-t)f(\theta_1) \ \forall \theta_0, \theta_1 \ and \ \forall t \in [0,1]$

Every value of $f(\theta)$ between θ_0 and θ_1 is lower or equal than every value of a straight line that connects the initial and final point.

For instance, linear functions are always convex.

Convex examples:

$$f(\theta) = \theta^2$$
$$f(\theta) = \theta$$
$$f(\theta) = |\theta|$$

Non-convex examples:

$$f(\theta) = -\theta^2$$

* Convex set

$t\theta_0 + (1-t)\theta_1 \in C \ \forall \theta_0, \theta_1 \ and \ \forall t \in [0,1]$

Convex examples:

$$A\theta \leq b \ (Linear \ constraint)$$
$$||\theta||^2 \leq r \ (Euclidian \ norm)$$
$$\theta_{lb} \leq \theta \leq \theta_{ub} \ (Bounds)$$

* Not trivial to always identifty convex sets

- Some operators preserve convexity
- There exists tricks to easily identify convex sets

* Why convexity is important?

- If the problem is convex → a local minimum is a global minimum
- Sometimes we only have local convexity (global optimum not garanteed)

First example: Linear Programming

$$\min_{\theta} C^T\theta$$
$$s.t. \ A\theta \leq b$$

Lots of applications.

Most of the times θ is really large.

Algorithms:

- Simplex: optimal solution is in a corner, so it travels along edges. Not very efficient.

- Interior Point Method: it goes inside the convex set. We apply a logarithm on the function we are optimizing.

1.2 Unconstrained Optimization

For every constraint there is an unknown value (λ) that appears as a reaction. On this section we are not dealing with constraints, so no reaction will appear.

$$min\ f(\theta)$$

f might be convex

$f\ \in C^1;\ \exists \nabla f(\theta)\ \forall \theta$

- In some applications, constraints are added as a penalization in the cost.
- Lots of examples in machine learning.

$$\begin{cases} \min\limits_{\theta}\ f(\theta) \\ A\theta = b \end{cases} \quad \rightarrow\ \min\limits_{\theta}\ f(\theta) + l(A\theta - b)$$

* First order optimality conditions

If f is convex and C^1 (differentiable), then $\forall \theta, \alpha \quad f(\theta) + \nabla f^T(\theta)(\alpha - \theta) \leq f(\alpha)$

* Taylor approximation of $f(\alpha)$ at θ

If the function is convex, local information like function and gradient at a point θ provides global information (underestimation). We know that every point of the function will be above the tangent line.

If $\exists \theta$ s.t. $\nabla f(\theta) = 0 \rightarrow f(\theta)\ \leq f(\alpha)\ \forall \alpha$ (θ is a global minimizer of f)

If f is not convex, but locally convex and C^2, a Taylor expansion gives

$$f(\alpha) = f(\theta) + \nabla f^T(\theta)(\alpha - \theta) + \frac{1}{2}(\alpha - \theta)\nabla^2 f(\theta)(\alpha - \theta) + ...$$

if $\exists \theta^+$ s.t. $\nabla f\left(\theta^+\right) = 0$ then $\forall \alpha$ s.t. $\left(\alpha - \theta^+\right)^T \nabla^2 f(\theta) \left(\alpha - \theta^+\right) \geq 0$

we have

$$f\left(\theta^+\right) = f(\alpha) - \left(\alpha - \theta^+\right)^T \nabla^2 f(\theta) \left(\alpha - \theta^+\right) \leq f(\alpha)$$

which means

$$\left(\alpha - \theta^+\right)^T \nabla^2 f(\theta) \left(\alpha - \theta^+\right) \geq 0$$

We have a minimum if the second derivative is positive.

Second example: Least Squared Problem

$$\begin{aligned} \min_{\theta} f(\theta) = \frac{1}{2}||A\theta - b||^2 = \ & \frac{1}{2}(A\theta - b)^T(A\theta - b) \\ = \ & \frac{1}{2}\theta^T A^T A\theta - \left(A^T b\right)^T \theta + \frac{1}{2}b^T b \end{aligned}$$

First order optimality condition:

$\nabla f(\theta) = A^T A\theta - A^T b = 0 \rightarrow \left(A^T A\right)\theta = A^T b$ (normal equations)

$A^T A$ is always square and symmetric

- Case A square $\rightarrow$ same as $A\theta = b$

$$A = \begin{bmatrix} & \\ & \end{bmatrix}$$

- Case A has vertical shape (overdetermined, more equations than unknowns)

$$A = \begin{bmatrix} & \\ & \\ & \\ & \end{bmatrix} \rightarrow \text{ minimize the error in the columns of A}$$

- Case A has horizontal shape (undetermined, more unkowns than equations)

$$A = \begin{bmatrix} & & \\ & & \end{bmatrix}$$

$A^T A$ has not full rank $\rightarrow$ ill-posed problem.

Plenty of solutions.

In machine learning it is an example of overfitting.

Gradient method

$\theta_{k+1} = \theta_k - \tau\nabla f(\theta_k)$ when τ is small enough

τ is called line search (learning rate in machine learning)

Descend directions:

$$f(\theta_{k+1}) = f(\theta_k) + \nabla f(\theta_k)^T(\theta_{k+1} - \theta_k) + \dots$$
$$= f(\theta_k) - \nabla f(\theta_k)^T[\tau \nabla f(\theta_k)] + \dots$$
$$= f(\theta_k) - \tau ||\nabla f(\theta_k)||^2 \leq f(\theta_k)$$

1.3 MATLAB Problem

1.3.1 Solve the normal equations

```
%% Problem 1
% Solve the normal equations

% n = m
A1 = [1 0 0
      0 1 0
      0 0 1];
B1 = [1;1;1];

Theta1 = (A1'*A1)\(A1'*B1);

% n < m
A2 = [1 0 0
      0 1 0];
B2 = [1;1];

% Theta2 = (A2'*A2)\(A2'*B2);

% n > m

A3 =[1 0 0
      0 1 0
      0 0 1
      1 1 1];

B3 = [1;1;1;1];

Theta3 = (A3'*A3)\(A3'*B3);
```

Listing 1.1: Problem 1 - Solve the normal equations

Case 2, for n < m, is not solvable directly.

1.3.2 Solve the optimization problem

```
%% Problem 2
% Solve the optimization problem with gradient descend and
% plot f(theta) and ||Gradf(theta)|| during iterations
tau = 0.1;
% n = m
theta_now = [0;0;0];
M = 50;
for i=1:M
    f_now           = 0.5*theta_now'*A1'*A1*theta_now-(A1'*B1)'*theta_now+0.5*B1'*
        B1;
    grad_now        = A1'*A1*theta_now-A1'*B1;
    f_values(i)     = f_now;
    grad_values(i) = norm(grad_now);
    theta_next      = theta_now-tau*grad_now;
    theta_now       = theta_next;
end

% Plot f_values and grad_values
figure;
plot(1:M, f_values, 'b-', 'LineWidth', 2, 'MarkerSize', 6);
hold on;
plot(1:M, grad_values, 'r-', 'LineWidth', 2, 'MarkerSize', 6);
hold off;

xlabel('Iteration');
ylabel('Value');
title('Convergence of f_{now} and ||grad_{now}||');
legend('f_{now}', '||grad_{now}||');
grid on;

% n < m
theta_now = [0;0;0];
M = 50;
for i=1:M
    f_now           = 0.5*theta_now'*(A2')*A2*theta_now-(A2'*B2)'*theta_now+0.5*(B2
        ')*B2;
    grad_now        = (A2')*A2*theta_now-(A2')*B2;
    f_values(i)     = f_now;
    grad_values(i) = norm(grad_now);
    theta_next      = theta_now-tau*grad_now;
    theta_now       = theta_next;
```

```
end

% Plot f_values and grad_values
figure;
plot(1:M, f_values, 'b-', 'LineWidth', 2, 'MarkerSize', 6);
hold on;
plot(1:M, grad_values, 'r-', 'LineWidth', 2, 'MarkerSize', 6);
hold off;

xlabel('Iteration');
ylabel('Value');
title('Convergence of f_{now} and ||grad_{now}||');
legend('f_{now}', '||grad_{now}||');
grid on;

% n > m
theta_now = [0;0;0];
M = 50;
for i=1:M
    f_now          = 0.5*theta_now'*A3'*A3*theta_now-(A3'*B3)'*theta_now+0.5*B3'*
        B3;
    grad_now       = A3'*A3*theta_now-A3'*B3;
    f_values(i)    = f_now;
    grad_values(i) = norm(grad_now);
    theta_next     = theta_now-tau*grad_now;
    theta_now      = theta_next;
end

% Plot f_values and grad_values
figure;
plot(1:M, f_values, 'b-', 'LineWidth', 2, 'MarkerSize', 6);
hold on;
plot(1:M, grad_values, 'r-', 'LineWidth', 2, 'MarkerSize', 6);
hold off;

xlabel('Iteration');
ylabel('Value');
title('Convergence of f_{now} and ||grad_{now}||');
legend('f_{now}', '||grad_{now}||');
grid on;
```

Listing 1.2: Problem 2 - Gradient descent and plotting

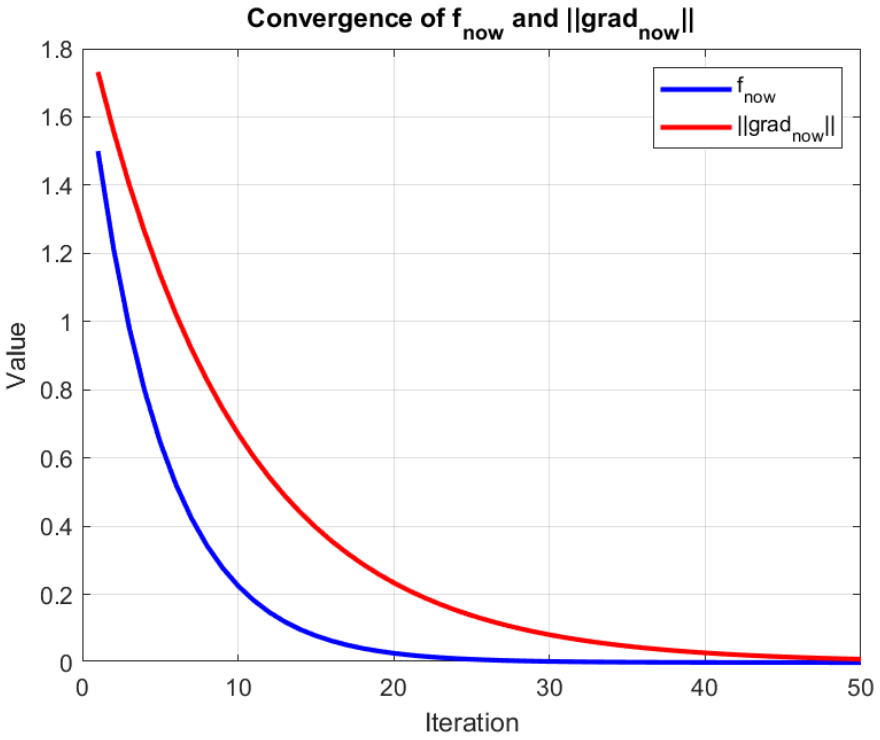


Figure 1.1: Convergence of f_{now} and $\|\nabla f_{\text{now}}\|$ for $n = m$

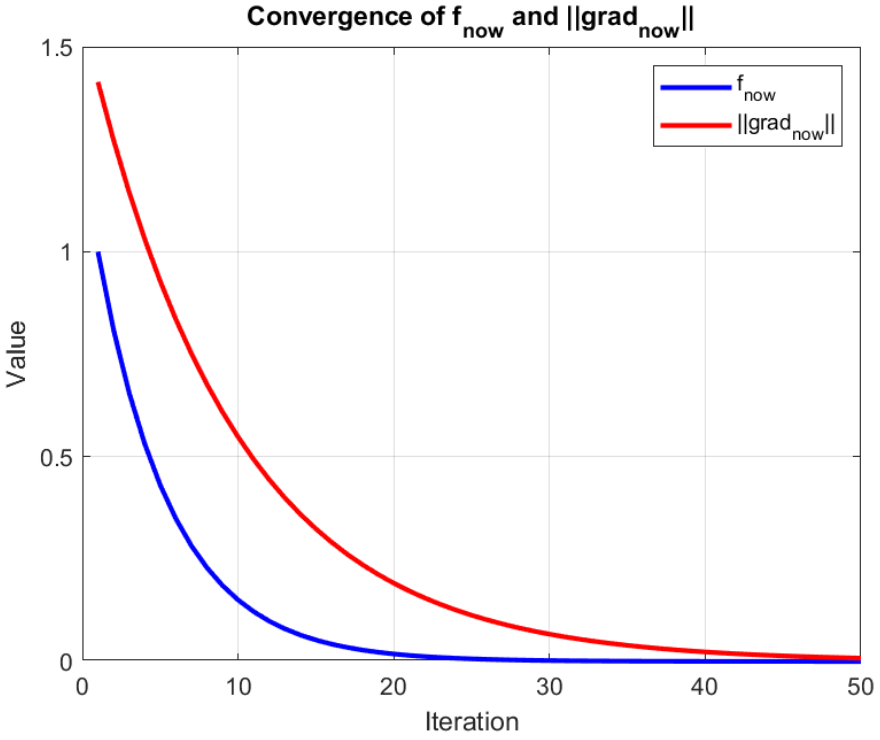


Figure 1.2: Convergence of f_{now} and $\|\nabla f_{\text{now}}\|$ for $n < m$

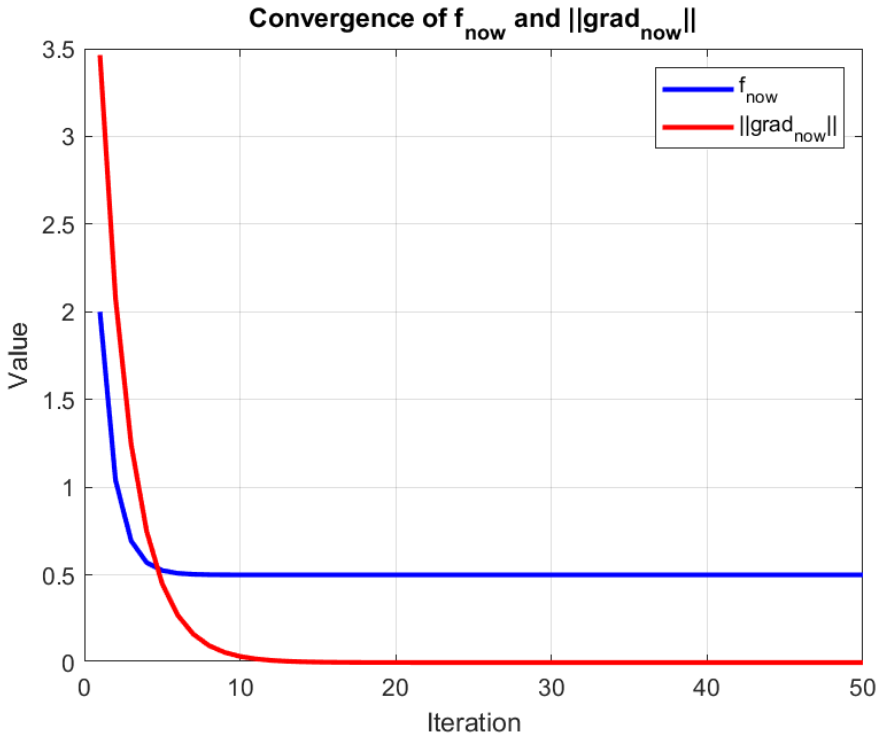


Figure 1.3: Convergence of f_{now} and $\|\nabla f_{\text{now}}\|$ for $n > m$

As it can be seen, the optimization problem is solved for each case and the gradient and the cost function lower for each iteration, which indicates that the value of the descent ratio used is appropriate.

Chapter 2

Machine Learning

2.1 Introduction

We classify:

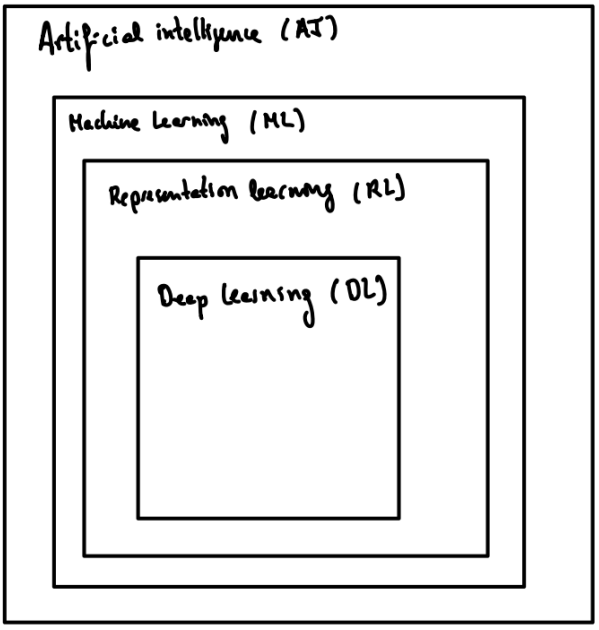


Figure 2.1: Classification of each of the Numerical Tool's areas

Where:

- **Artificial intelligence**: Knowledge based. *Ex: reasoning*
- **Machine learning**: Extracting patterns from data. *Ex: logistic regression*
- **Representation learning**: Finding combining ways of representing the data (features). *Ex: PCA, autoencoders*

- **Deep learning**: Representations expressed in terms of simple representations. *Ex: neural networks*

2.2 In ML: Supervised vs Unsupervised Learning

- In machine learning:
  - **Supervised learning**:

$$D = \{(x_i, y_i)\} \quad \sim \quad \begin{array}{l} x_i \rightarrow \text{features} \\ y_i \rightarrow \text{outputs} \end{array}$$

$$\left[\begin{array}{ccc|c} x_1 & x_2 & \dots & y \end{array}\right]$$

    Cases: regression, classification (e.g., linear, logistic)

  - **Unsupervised learning**:

$$D = \{x_i\}$$

    Cases:

    * Clustering $\Rightarrow$ x is important
    * Dimension reduction

    Ex clustering: separate the class into groups for each degree. We need sufficient features.

    Ex dimension reduction: projecting a very flat loss few DoFs to analyze a wing beam.

2.3 Predictions: Representation Matters

$$y = f_\theta(x)$$

- **Linear prediction**:

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- **Polynomial prediction**:

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 \quad \Rightarrow \quad \text{ML}$$

- **Representation learning**:

$$f_\theta(x) = \theta_0 \cdot f_\phi(x)$$

- **Deep learning**:

$$f_\theta(x) = \theta_1 f_1(\theta_2 f_2(\theta_3 f_3(x)))$$

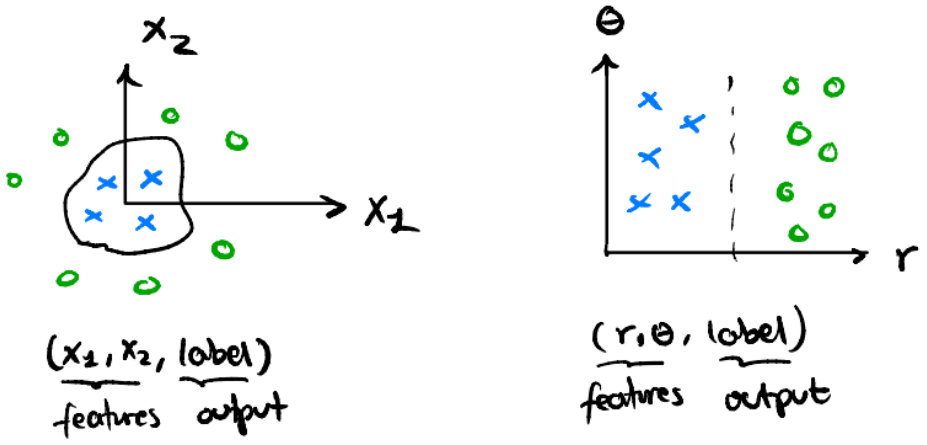


Figure 2.2: Ways of representing the data

2.4 Examples

2.4.1 First Example: Linear Regression in ℓ_2 Norm

$$y = X\theta \quad \text{with } X = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

$$D = \{(x_i, y_i)\}, \quad \min_\theta f(\theta) = \|y - X\theta\|_2^2 = (y - X\theta)^T (y - X\theta)$$

$$\text{Convex, differentiable,} \quad \nabla f(\theta) = -2X^T(y - X\theta)$$

$$\nabla f(\theta) = 0 \Rightarrow X^T X\theta = X^T y \Rightarrow \theta = (X^T X)^{-1} X^T y$$

Geometric intuition: projection in the linear space generated by columns of X

2.4.2 Second Example: Linear Regression in ℓ_1 Norm

$$\min_\theta f(\theta) = \|y - X\theta\|_1 = \sum_{i=1}^{N} |y_i - x_i^T \theta|$$

Convex, but not differentiable (adding outlier worsens result)

Canonical form:

$$\min t \quad \text{such that} \begin{cases} -t \leq y - x^T\theta \leq t \end{cases}$$

$$\text{or} \quad -t \leq y - X\theta \leq t$$

Then, it transforms to:

$$\min_{\mathbf{v}} \quad \mathbf{c}^T\mathbf{v}$$

$$\text{subject to} \quad A\mathbf{v} \leq \mathbf{b}$$

$$\text{with } \mathbf{v} = \begin{bmatrix} \theta \\ t \end{bmatrix}, \quad A = \begin{bmatrix} -x & -1 \\ x & -1 \end{bmatrix}, \quad \mathbf{c}^T = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} y \\ -y \end{bmatrix}$$

2.4.3 Third Example: Linear Regression in ℓ_∞ Norm (Chebyshev Approximation)

$$\min_\theta f(\theta) = \|y - X\theta\|_\infty = \max_i |y_i - x_i\theta|$$

Convex, not differentiable (adding outlier worsens), minimize the max

$$\min_{\theta,t} \quad t \quad \text{subject to: } -t \leq y - X\theta \leq t$$

Canonical form:

$$\min c^T u, \quad \text{where } u = \begin{bmatrix} \theta \\ t \end{bmatrix}, \quad c^T = [0\ 0\ 1], \quad A = \begin{bmatrix} X & -\mathbf{1} \\ -X & -\mathbf{1} \end{bmatrix}, \quad b = \begin{bmatrix} y \\ -y \end{bmatrix}$$

Norms and Interpretation

Define: $e = y - X\theta$

ℓ_0 norm: $\|e\|_0$ (non-convex)

ℓ_1 norm: $\|e\|_1$

ℓ_2 norm: $\|e\|_2$

ℓ_∞ norm: $\|e\|_\infty$ (minimize max test $\Rightarrow$ maximum error smaller)

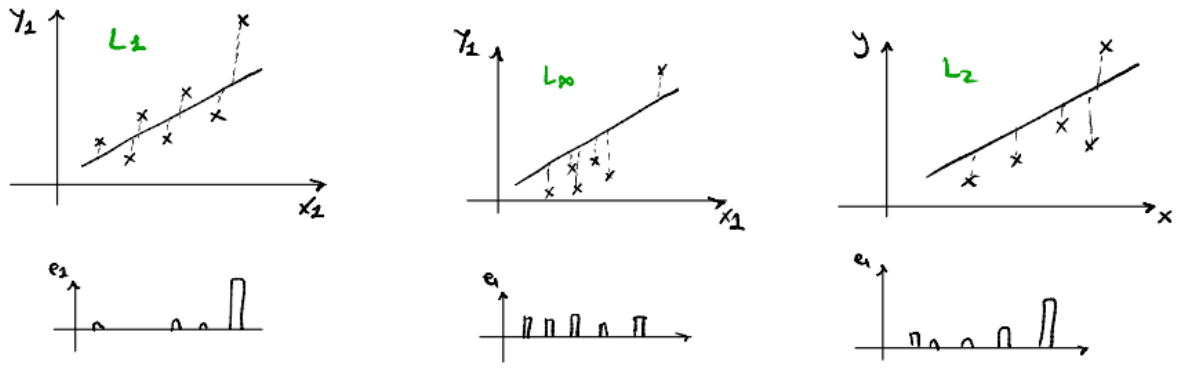


Figure 2.3: Norms and errors

2.5 MATLAB Problem

2.5.1 L2 Norm

```
%% L2 norm

% Generate Data
x = (linspace(0,10,30))';
A = [ones(size(x,1),1), x]; % Construct matrix A
X = A;
theta = [1;2];
for i=1:size(x,1)
    noise(i,1) = rand(1)*x(i)*3;
end
y = A*theta + noise;

% Gradient Descent Parameters
tau = 0.0001;
tol = 1e-4;
theta_now = theta;
grad_now = 10;
M = 50; % Number of iterations

% Store values
f_values = zeros(M,1);
grad_values = zeros(M,1);

% Gradient Descent Loop
for i = 1:M
```

```
    f_now = 0.5 * theta_now' * A' * A * theta_now - (A' * y)' * theta_now + 0.5 *
        y' * y;
    grad_now = A' * A * theta_now - A' * y;
    f_values(i) = f_now;
    grad_values(i) = norm(grad_now);
    theta_now = theta_now - tau * grad_now;
end

% Plot data and L2 regression line
figure;
hold on;
scatter(x, y, 50, 'b', 'o', 'filled'); % Plot noisy data points
plot(x, A * theta_now, 'r-', 'LineWidth', 2); % L2 regression line
xlabel('x');
ylabel('y');
title('L2 Norm Regression: Original Data vs. Fit');
legend('Noisy Data', 'L2 Regression Line', 'Location', 'Best');
grid on;
set(gca, 'FontSize', 12);
hold off;

% Plot f_values and grad_values
figure;
yyaxis left;
plot(1:M, f_values, 'b-', 'LineWidth', 2);
ylabel('f_{now}');
hold on;

yyaxis right;
plot(1:M, grad_values, 'r-', 'LineWidth', 2);
ylabel('||grad_{now}||');

xlabel('Iteration');
title('Convergence of f_{now} and ||grad_{now}||');
legend('f_{now}', '||grad_{now}||');
grid on;
hold off;

% Plot error L2
figure;
bar(x,abs(y-A*theta_now),'b')
xlabel('x');
```

```
67 ylabel('Error Magnitude');
68 title('Error magnitude for L2 norm');
69 grid on;
```

Listing 2.1: L2 norm regression with gradient descent in MATLAB

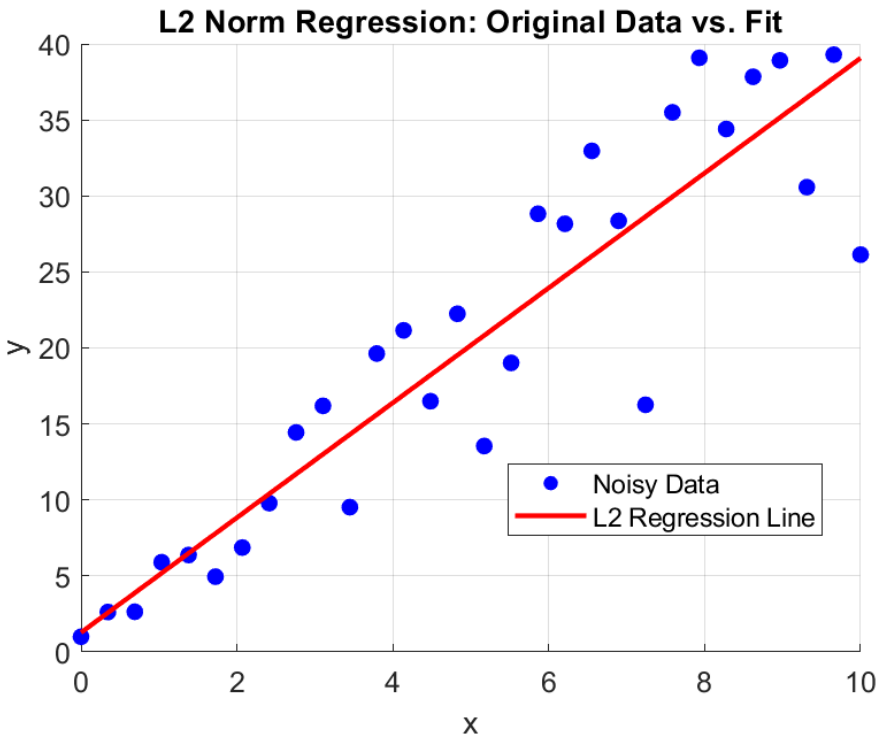


Figure 2.4: L2 Norm Regression

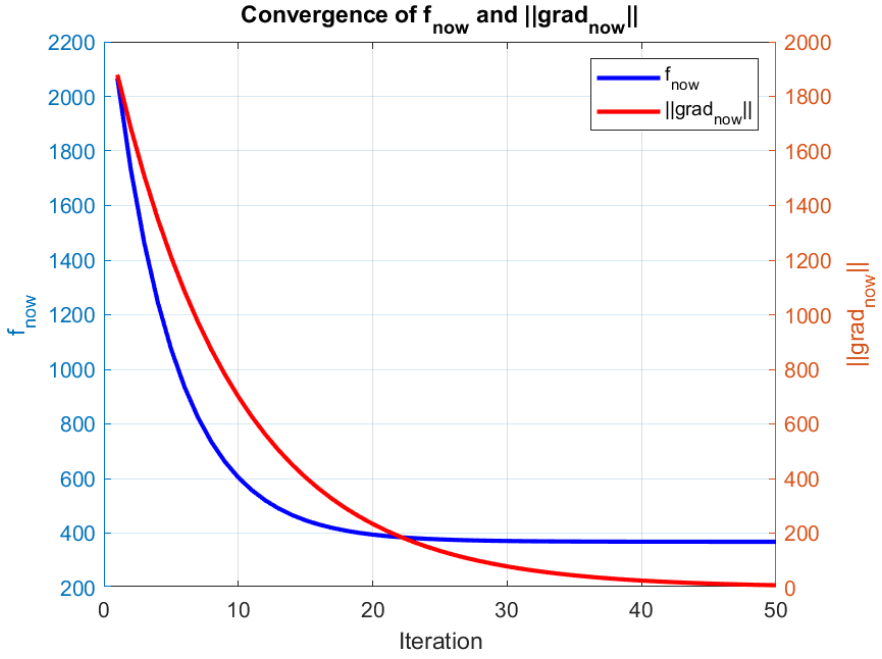


Figure 2.5: L2 norm and gradient convergence

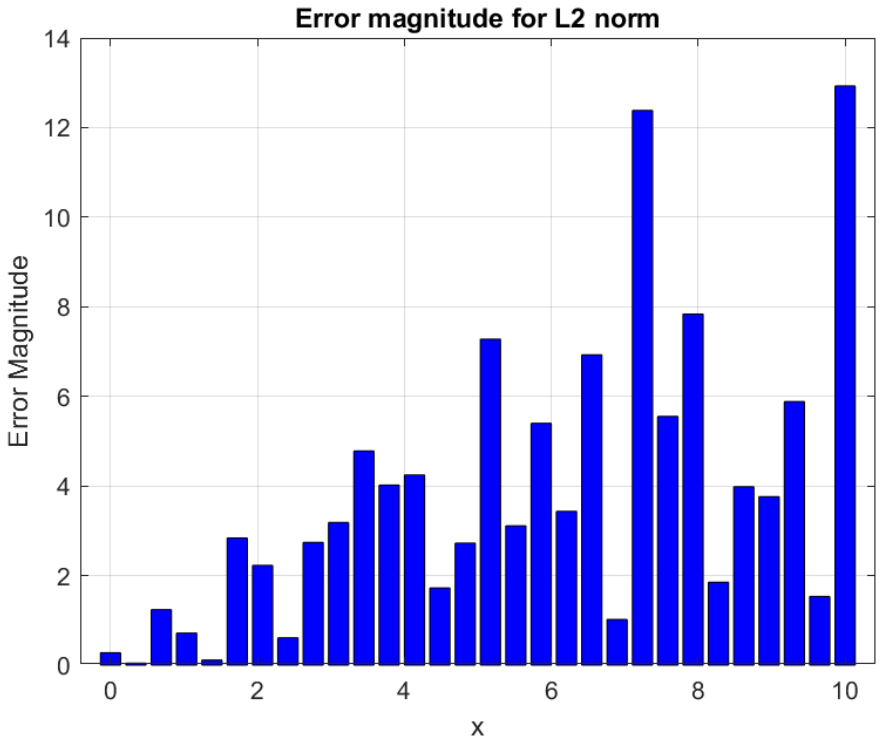


Figure 2.6: L2 norm error magnitude

2.5.2 L1 Norm - Linprog

```
%% L1 norm - linprog

C = [0 0 ones(1,size(x,1))];
b = [y;-y];
A = [X -eye(size(x,1)); -X -eye(size(x,1))];

sol = linprog(C,A,b);
thetaL1 = sol(1:2,1);
errL1 = abs(y-X*thetaL1);

% Plot error L1
figure;
bar(x,errL1,'b')
xlabel('x');
ylabel('Error Magnitude');
title('Error magnitude for L1 norm');
grid on;
```

Listing 2.2: L1 norm regression using linprog in MATLAB

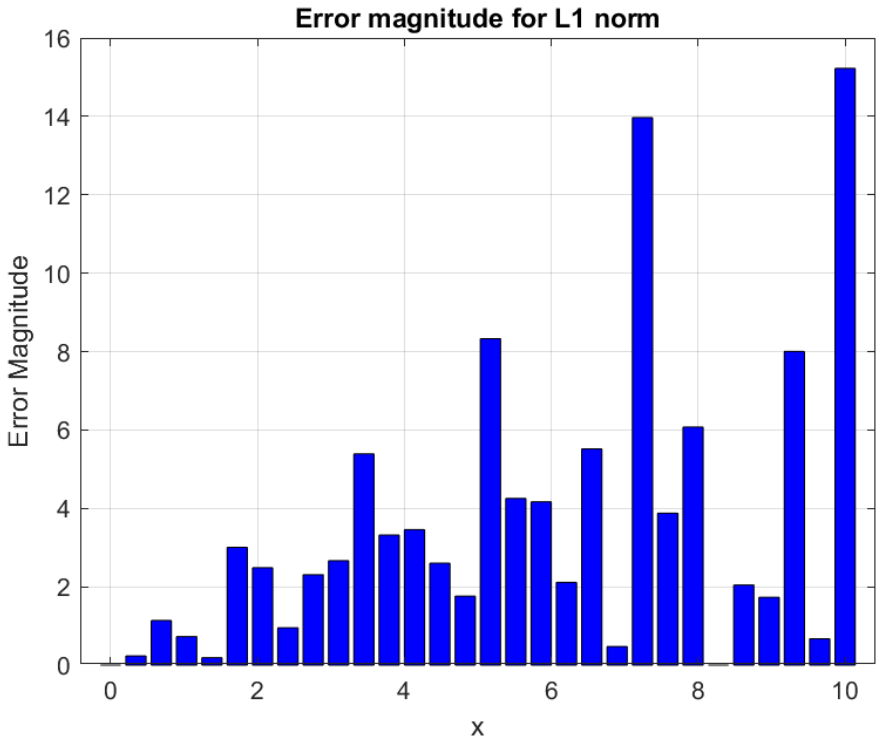


Figure 2.7: L1 norm error magnitude

2.5.3 Linf Norm - Linprog

```
%% Linf norm - linprog

C = [0 0 1];
b = [y;-y];
A = [X -ones(size(x,1),1); -X -ones(size(x,1),1)];

sol = linprog(C,A,b);
thetaLinf = sol(1:2,1);
errLinf = abs(y-X*thetaLinf);

% Plot error Linf
figure;
bar(x,errLinf,'b')
xlabel('x');
ylabel('Error Magnitude');
title('Error magnitude for L_{inf} norm');
grid on;
```

Listing 2.3: L_∞ norm regression using linprog in MATLAB

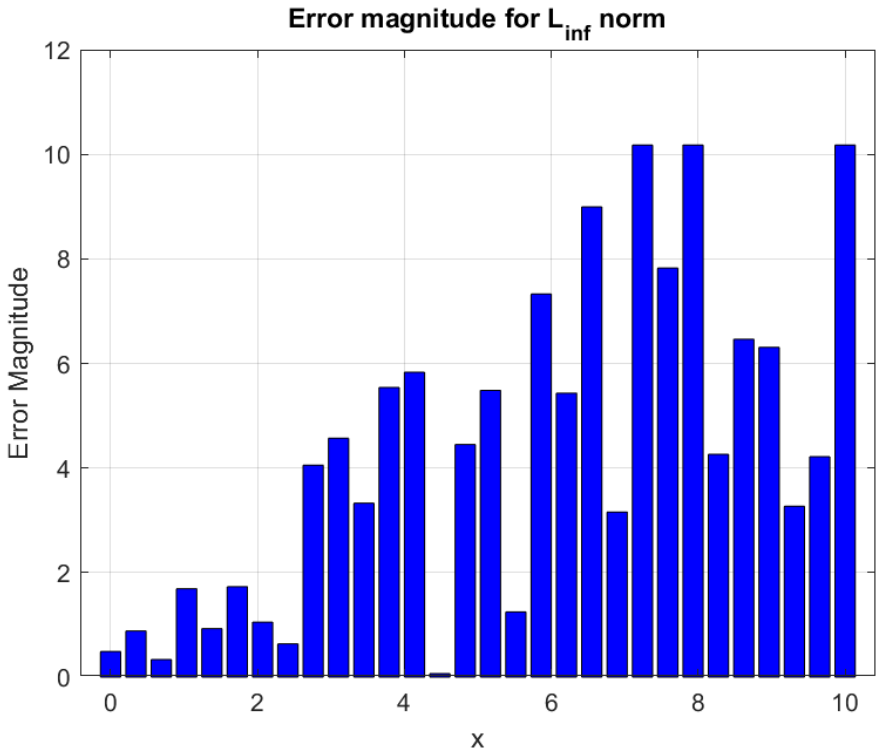


Figure 2.8: L_∞ norm error magnitude

As it can be seen, the results obtained for the error magnitudes of each of the norms correspond to the theory.

Chapter 3

Hypothesis Space, Capacity and Overfitting

Up to now:

$$|f(\theta) - y| \Rightarrow \text{find } \theta \text{ with good fit, define errors (norms)}$$

3.1 Hypothesis space

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \dots$$

But which dimension do we choose? Is it always useful to increase dimension?

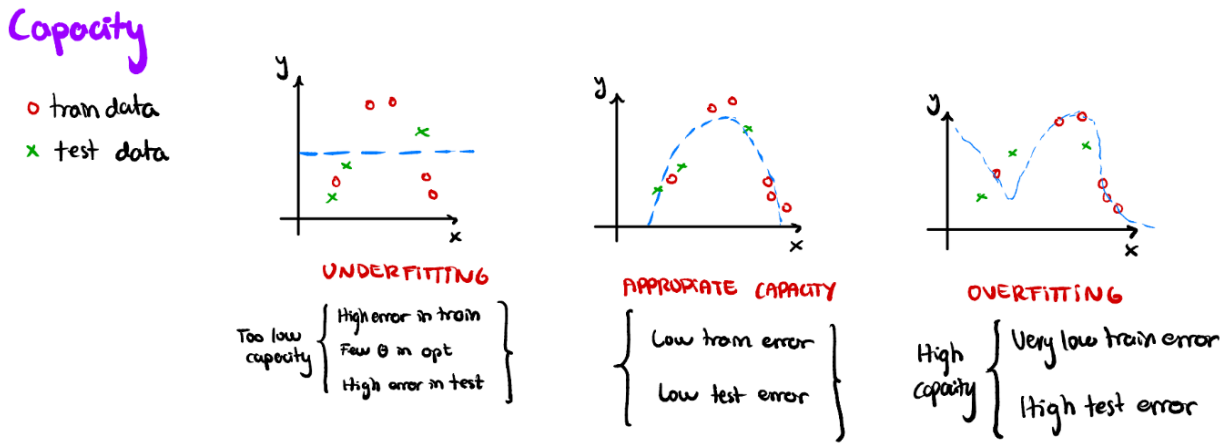
We divide the data:

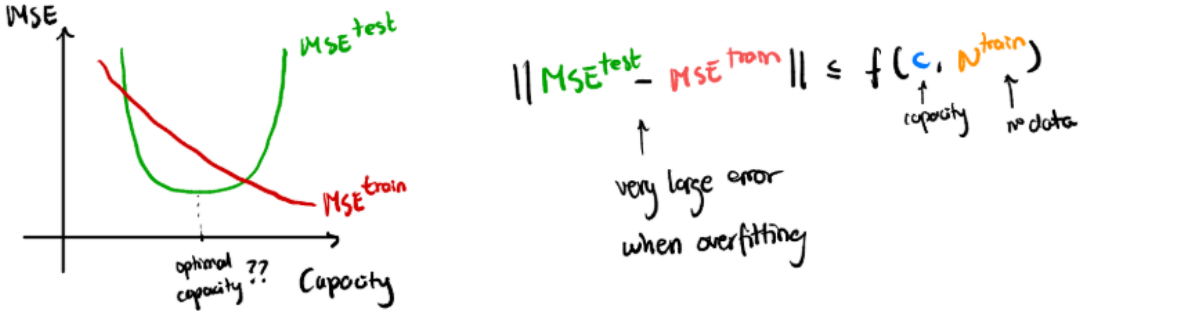
- **TRAIN** $\sim 80\%$: minimize MSE and obtain θ ($\text{MSE}^{\text{train}}$)
- **TEST** $\sim 20\%$: validate (MSE^{test})

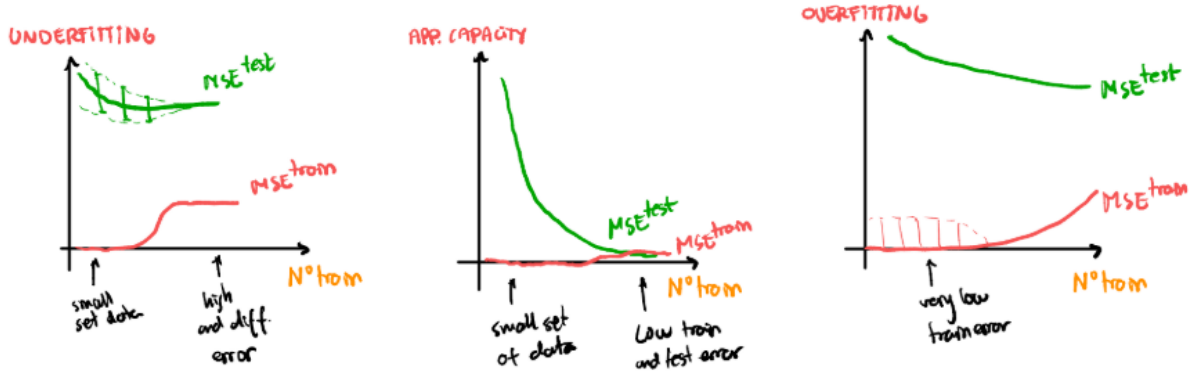
Optimization vs ML:

- **Optimization:** Mathematical, finds the best minimizers $\rightarrow$ $\text{MSE}^{\text{train}}$ goal
- **ML:** Predict using new data $\rightarrow$ small error $\rightarrow$ MSE^{test} goal

3.2 Capacity







3.3 Underfitting and Overfitting

Linear regression:

We minimize the L2 norm:

$$\min_{\theta} \|\mathbf{y} - X\theta\|^2$$

- **Underfitting:**

$$X = \begin{bmatrix} \\ \\ \end{bmatrix}, \quad X^\top X = \begin{bmatrix} \\ \\ \end{bmatrix} \Rightarrow \theta \text{ is too small in comparison to data}$$

- **Overfitting:**

$$X = \begin{bmatrix} \\ \\ \end{bmatrix}, \quad X^\top X = \begin{bmatrix} \text{Not} \\ \text{full} \\ \text{rank} \end{bmatrix} \Rightarrow \theta \text{ is too large in comparison to data}$$

3.4 Optimal Capacity and Regularization

We minimize the following objective:

$$\min_\theta \|\mathbf{y} - X\theta\|_2^2 + \lambda\|\theta\|_2^2$$

- $\lambda\|\theta\|_2^2$: regularization term
- $\lambda > 0$: Prior knowledge of θ, penalizes large values of θ
- Useful when X is sparse

This leads to the modified normal equations:

$$(X^\top X + \lambda I)\theta = X^\top y$$

- Makes the matrix full rank
- Controls the norm of θ to be lower

Other Regularization Terms

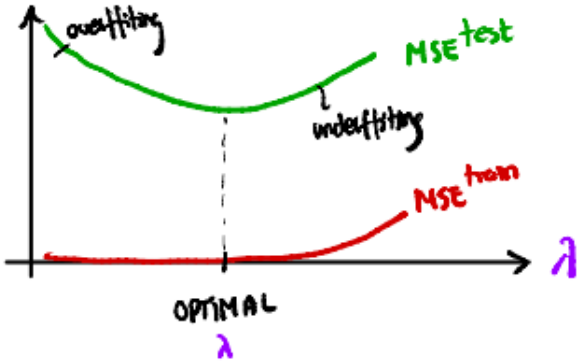
- L_1 regularization: promotes sparsity

$$\|\theta\|_1 \Rightarrow \text{more sparse } \theta$$

- $\|\nabla\theta\|^2$: smoothness of function
- Total variation: $\|\nabla\theta\|_1$, encourages sparse gradient of θ

Effects of Regularizing

- It is possible to use large capacity models with appropriate λ



3.5 MATLAB Problem

3.5.1 Synthetic Quadratic Data Generation

```
N = 100;
x = linspace(-3, 3, N)';
theta_true = [2; -1; 0.5];
noise = randn(N, 1) * 0.5;
y = theta_true(1) + theta_true(2)*x + theta_true(3)*x.^2 + noise;

rng(1);
idx = randperm(N);
n_train = round(0.8 * N);
idx_train = idx(1:n_train);
idx_test = idx(n_train+1:end);

x_train = x(idx_train);
y_train = y(idx_train);
x_test = x(idx_test);
y_test = y(idx_test);

fig = figure;
scatter(x_train, y_train, 40, 'b', 'filled'); hold on;
scatter(x_test, y_test, 40, 'r', 'filled');
title('Synthetic Quadratic Data: Train vs Test');
xlabel('x'); ylabel('y');
legend('Train Data', 'Test Data');
grid on;
set(fig, 'Color', 'w');
```

```
exportgraphics(fig, 'NT3Fig4.png', 'BackgroundColor', 'white');
```

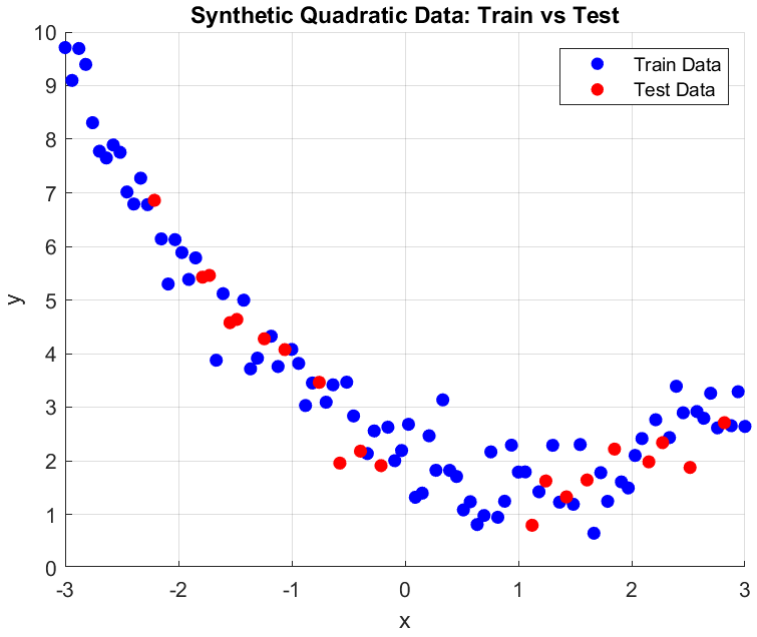


Figure 3.1: Train vs Test data on synthetic quadratic signal

3.5.2 Small Capacity Model and MSE Analysis

```
Theta_small_init = randn(2,1) * 0.1;  % reinit every time
MSE_test_small = zeros(size(x_train,1),1);
MSE_train_small = zeros(size(x_train,1),1);

for i = 2:size(x_train,1)  % start from 2 to avoid singularities

    Theta_small = Theta_small_init;
    eps = 1e-4;
    diff = [1000;1000];
    Theta_old = Theta_small;

    % Prepare training data
    x_small_train = [ones(i,1), x_train(1:i)];
    y_small = y_train(1:i);

    % Fixed test data
    x_test_small = [ones(length(x_test),1), x_test];

    tau = 0.001;

```

```
    % Gradient descent loop
    while (norm(diff) > eps)
        gradient = x_small_train' * x_small_train * Theta_small -
            x_small_train' * y_small;
        Theta_small = Theta_old - tau * gradient;
        diff = Theta_small - Theta_old;
        Theta_old = Theta_small;
    end

    % Compute predictions
    y_est_train = x_small_train * Theta_small;
    y_est_test = x_test_small * Theta_small;

    % Compute MSEs
    MSE_train_small(i,1) = mean((y_small - y_est_train).^2);
    MSE_test_small(i,1) = mean((y_test - y_est_test).^2);

end

% Plot both MSE curves
fig = figure;
plot(2:size(x_train,1), MSE_train_small(2:end), 'b-', 'LineWidth', 2);
   hold on;
plot(2:size(x_train,1), MSE_test_small(2:end), 'r--', 'LineWidth', 2);
xlabel('Number of Training Points');
ylabel('Mean Squared Error');
legend('Train MSE','Test MSE');
title('Train vs Test MSE (Small Capacity Model)');
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT3Fig5.png', 'BackgroundColor', 'white');
```

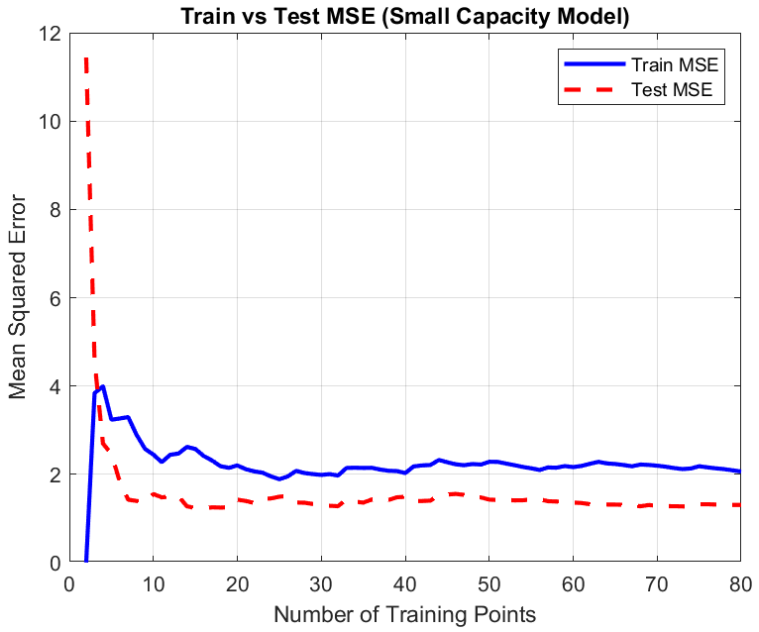


Figure 3.2: Train vs Test MSE for Small Capacity Model

3.5.3 Larger Capacity Model and MSE

```
max_degree = 2;
MSE_train_large = zeros(max_degree, 1);
N = length(x_train);

for d = 1:max_degree
    X_poly_train = ones(N, d + 1);  % [1 x x^2 ... x^d]
    for k = 1:d
        X_poly_train(:, k+1) = x_train.^k;
    end

    % Gradient descent for Theta of size (d+1)
    Theta_large = randn(d+1, 1) * 0.1;
    Theta_old = Theta_large;
    eps = 1e-4;
    diff = ones(d+1,1) * 1000;
    tau = 0.001;

    % Gradient descent loop
    while norm(diff) > eps
        gradient = X_poly_train' * X_poly_train * Theta_large -
           X_poly_train' * y_train;
        Theta_large = Theta_old - tau * gradient;
```

```
        diff = Theta_large - Theta_old;
        Theta_old = Theta_large;
    end

    % Estimate output
    y_est_train = X_poly_train * Theta_large;

    % Compute MSE for training data
    MSE_train_large(d) = mean((y_train - y_est_train).^2);
end

% Plot MSE vs Capacity
fig = figure;
plot(1:max_degree, MSE_train_large, 'bo-', 'LineWidth', 2, 'MarkerSize',
    8);
xlabel('Model Capacity (Polynomial Degree)');
ylabel('Mean Squared Error (Train)');
title('Training MSE vs Model Capacity');
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT3Fig6.png', 'BackgroundColor', 'white');
```

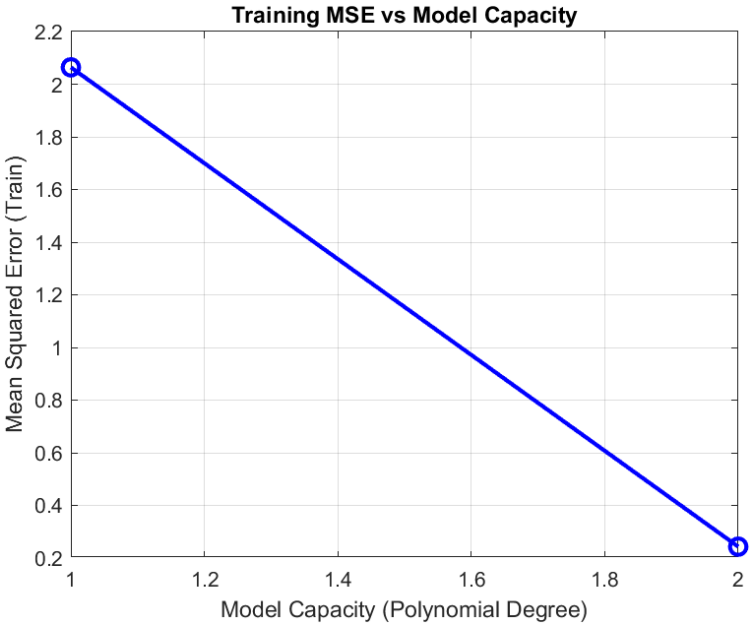


Figure 3.3: Training MSE vs Model Capacity

3.5.4 Cross-Validation on Polynomial Regression

```
clear; close all; clc;

% Synthetic quadratic data: y =       +    x    +     x     + noise

N = 100;
x = linspace(-3, 3, N)';
theta_true = [2; -1; 0.5];
noise = randn(N, 1) * 0.5;
y = theta_true(1) + theta_true(2)*x + theta_true(3)*x.^2 + noise;

% Cross-validation setup (5-fold)
K = 5;
cv = cvpartition(N, 'KFold', K);

% Model capacities to test
max_degree = 10;
MSE_test_all = zeros(K, max_degree+1);

% Cross-validation loop
for d = 0:max_degree
    for k = 1:K
        idx_train = training(cv, k);
        idx_test = test(cv, k);

        x_train = x(idx_train);
        y_train = y(idx_train);
        x_test = x(idx_test);
        y_test = y(idx_test);

        % Design matrix for train/test
        Phi_train = zeros(length(x_train), d+1);
        Phi_test = zeros(length(x_test), d+1);
        for j = 0:d
            Phi_train(:,j+1) = x_train.^j;
            Phi_test(:,j+1) = x_test.^j;
        end

        % Fit model (least squares)
```

```
        theta = Phi_train \ y_train;

        % Predict and compute MSE on test set
        y_pred = Phi_test * theta;
        MSE_test_all(k, d+1) = mean((y_test - y_pred).^2);
    end
end

% Compute mean and std of MSE
mu = mean(MSE_test_all);
sigma = std(MSE_test_all);

% Plot       1.96
degrees = 0:max_degree;
fig = figure;
hold on;
errorbar(degrees, mu, 1.96 * sigma, 'o-', 'LineWidth', 2, 'DisplayName', '
   Cross-validation');
xlabel('Model Capacity (Degree)');
ylabel('Test MSE');
title('Cross-Validation: Mean    1.96 Std of Test MSE');
grid on;
legend;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT3Fig7.png', 'BackgroundColor', 'white');
```

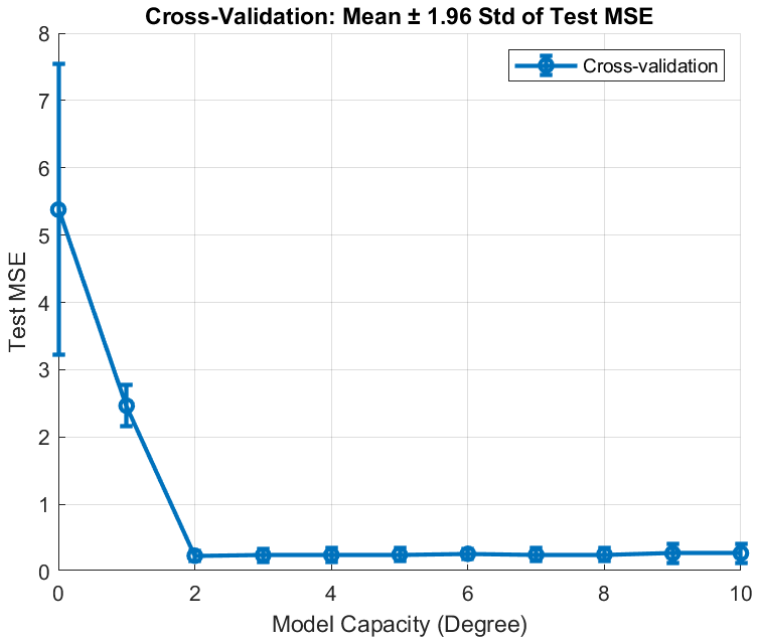


Figure 3.4: Cross-validation: Test MSE vs Model Capacity

3.5.5 Ridge Regression with High Capacity

```
lambda = 1; % Ridge regularization parameter
d_ridge = 10;

% Full data
Phi_ridge = zeros(N, d_ridge+1);
for j = 0:d_ridge
    Phi_ridge(:, j+1) = x.^j;
end

% Ridge solution
theta_ridge = (Phi_ridge' * Phi_ridge + lambda * eye(d_ridge+1)) \ (
   Phi_ridge' * y);

% Predict
x_fit = linspace(-3.2, 3.2, 200)';
Phi_fit = zeros(length(x_fit), d_ridge+1);
for j = 0:d_ridge
    Phi_fit(:, j+1) = x_fit.^j;
end
y_fit = Phi_fit * theta_ridge;

% Plot result
```

```
fig = figure;
scatter(x, y, 25, 'filled'); hold on;
plot(x_fit, y_fit, 'r-', 'LineWidth', 2);
title(sprintf('Ridge Regression (degree = %d, \\lambda = %.2f)', d_ridge,
    lambda));
xlabel('x'); ylabel('y');
legend('Data', 'Ridge fit');
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT3Fig8.png', 'BackgroundColor', 'white');
```

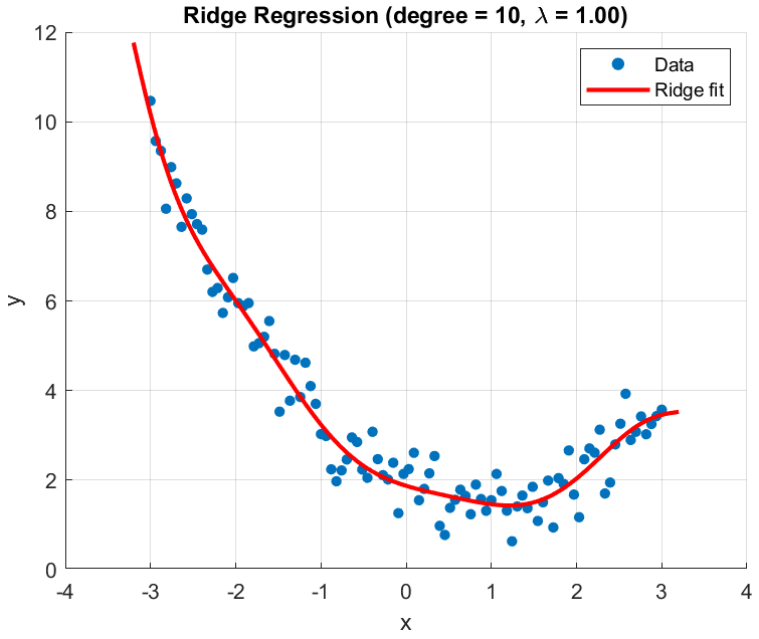


Figure 3.5: Ridge Regression fit for high capacity model

Chapter 4

Supervised Learning I

- **Supervised:** $\mathcal{D} = \{(x^i, y^i)\}$

  Example:

  - x_1: day
  - x_2: place
  - y: temperature

Linear Regression (Polynomial Regression):

We minimize the mean squared error:

$$\min_{\theta} f(\theta) = \text{MSE}(\theta) = \|y - X\theta\|_2^2$$

We can apply different norms; the output $y \in \mathbb{R}^n$

Where the input matrix X can be:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots \end{bmatrix} \quad \text{(with one feature)}$$

or

$$X = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & \dots \end{bmatrix} \quad \text{(with more features)}$$

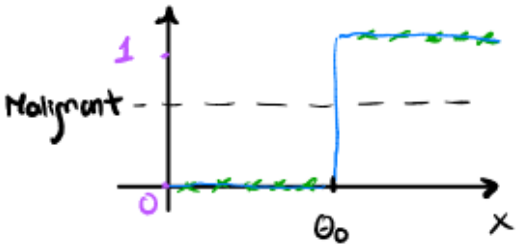
Regularization:

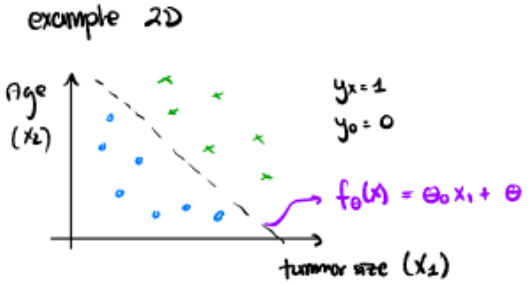
We add a penalty term to control overfitting:

$$\min_{\theta} \|y - X\theta\|^2 + \lambda\|\theta\|^2 \Rightarrow (X^\top X + \lambda I)\theta = X^\top y$$

4.1 Logistic Regression

We classify the output to be 0,1:



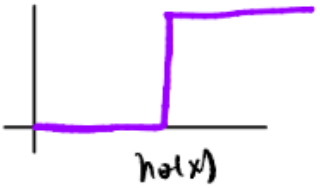


- **Hypothesis space / Hypothesis function**

$$h_\theta(x) = x\theta$$

- **Thresholding (Heaviside function)**

$$g(h_\theta(x)) = \begin{cases} 0 & \text{if } h_\theta(x) < 0 \\ 1 & \text{if } h_\theta(x) \geq 0 \end{cases}$$



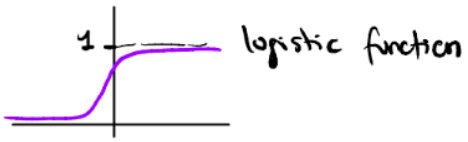
We want it to be smooth.

- **Sigmoid function (regularized Heaviside)**

$$g(z) = \frac{1}{1 + e^{-z}} \quad \text{(logistic function)}$$

$$g(z - \theta_0) = \frac{1}{1 + e^{-(z-\theta_0)}}$$

$$g'(z) = g(z)(1 - g(z))$$



- **Interpretation of** $g(h_\theta(x))$:

  Estimated probability that $y = 1$ on input x. For example, given a x_1, if $g(h_\theta(x_1)) = 0.7$, then there's a 70% change that the tumor is malignant.

- **Decision boundary**

$$g(\theta^\top x) = \begin{cases} 1 & \text{if } \theta^\top x \geq 0 \\ 0 & \text{if } \theta^\top x < 0 \end{cases}$$

  (lineal)

- **Cost function: Negative log-likelihood / Cross-entropy**

$$f_\theta(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^i \log(g(h_\theta(x^i))) - (1 - y^i) \log(1 - g(h_\theta(x^i))) \right]$$

  - If $y = 1$, $g \to 1$: $-\log(g) \to 0$ (we want y to be similar to g)
  - If $y = 1$, $g \to 0$: $-\log(g) \to \infty$
  - If $y = 0$, $g \to 1$: $-\log(1 - g) \to \infty$
  - If $y = 0$, $g \to 0$: $-\log(1 - g) \to 0$

- **Gradient**

$$\frac{\partial f}{\partial \theta} = \frac{1}{m} \sum_{i=1}^{m} \left[\left(g(h_\theta(x^i)) - y^i \right) x^i \right] = -\frac{1}{m} \sum_{i=1}^{m} \left[y^i - g(h_\theta(x^i)) \right] x^i$$

- **Remark:** Linear regression:

$$\min_\theta \|X\theta - y\|^2 \Rightarrow \frac{\partial f}{\partial \theta} = 2X^\top X\theta - 2X^\top y \Rightarrow X^\top (X\theta - y) = 0$$

Remarks:

- We can use **gradient method**

$$\Theta_{n+1} = \Theta_n - \tau \nabla f(\Theta_n)$$

Stochastic gradient randomly selects l values of the data

$$\nabla f_\ell(\Theta) \approx \frac{1}{l} \sum_{i=1}^{\ell} \left[g(h_\Theta(x^i)) - y^i \right] x^i, \quad \ell \ll m$$

Typically we take hundreds of data to compute the s.g.; when we have used all the data, we say we spend one epoch.

- **Regularization is also important**

$$\min_{\Theta} \quad \square + \lambda \frac{\sigma}{2} \Theta$$

$$\frac{\partial f}{\partial \Theta} = \square + 2\lambda\Theta$$

Gradient expensive $\rightarrow$ stochastic

Stopping criteria $\|\nabla f\| \approx 0$ is expensive, so sometimes we finish with a number of iterations.

Computing the cost is expensive (we usually do not compute)

4.2 MATLAB Problem

1) Create data with a bit of noise.
Given θ_1, θ_2

$$y = \text{Heaviside}(h_\theta(x)), \quad h_\theta(x) = \theta_1 x_1 + \theta_2 x_2 + \text{noise}$$

$$D = \{x_1, x_2, y\}$$

2) Solve logistic regression and find θ_1, θ_2

$$h_\theta(x) = \theta_1 x_1 + \theta_2 x_2$$

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \\ \theta_2 \end{bmatrix}$$

Plot the objective function f_0 over iterations (iter).

3) Do the same as 1) and 2) but with polynomial representation:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$$

$$= \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \Rightarrow \text{we may need to regularize with } \lambda$$

4.2.1 Practical Part – Sections a) and b)

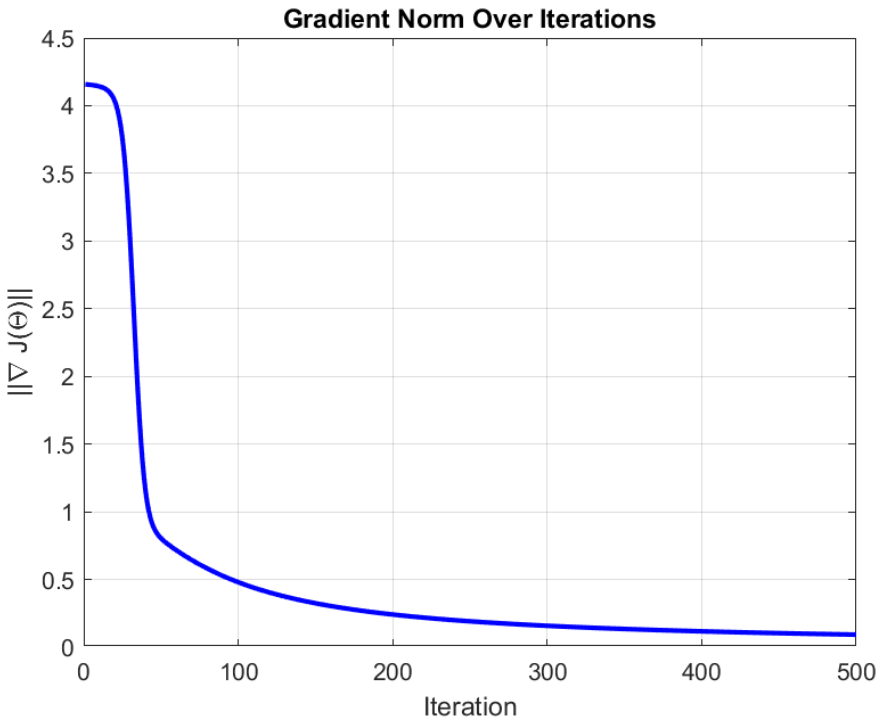
```
%% Initial data
clear all
close all

%% Data definition
N      = 15;
x1     = rand(N,1)*10;
x2     = rand(N,1)*10;
Theta  = [1;1];
noise  = 30 * randn(size(x2));
x      = [x1, x2];
h      = x*Theta+noise;
y      = zeros(N,1);
for i=1:N
    if h(i)<0
        y(i)=0;
    else
        y(i)=1;
    end
end
D = [x1, x2, y];

%% Separate the data between train (70% of points) and test (30% of points)
idx = randperm(N);
train_size = round(0.7 * N);
train_idx = idx(1:train_size);
test_idx = idx(train_size+1:end);

D_train = D(train_idx, :);
D_test = D(test_idx, :);
```

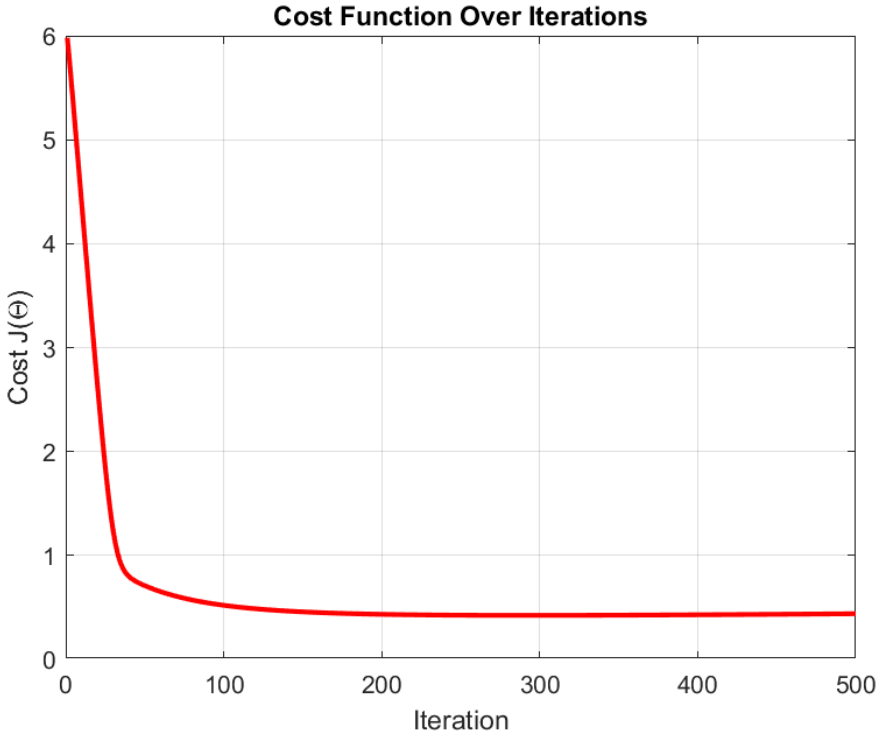
```
31
32 x1_train = D_train(:,1);
33 x2_train = D_train(:,2);
34 y_train = D_train(:,3);
35
36 x1_test = D_test(:,1);
37 x2_test = D_test(:,2);
38 y_test = D_test(:,3);
39
40 %% Solve logistic regression and find Theta1 and Theta2, decision line is x2 = -
       Theta1/Theta2 * X1
41 tau = 0.01;
42 iter = 500;
43 lambda = 0.1;
44 X_train = [x1_train, x2_train];
45 N_linear = size(X_train,1);
46 Cost = zeros(iter,1);
47 Grad_save = zeros(iter,1);
48
49 for k = 1:iter
50     h = X_train * Theta;
51     g = 1 ./ (1 + exp(-h));
52
53     Gradient = zeros(size(Theta));
54
55     for i = 1:N_linear
56         xi = X_train(i,:)';
57         Gradient = Gradient + (g(i) - y_train(i)) * xi;
58     end
59
60     Gradient = (-1 / N_linear) * Gradient;
61     Grad_save(k) = norm(Gradient);
62     Theta = Theta + tau * Gradient;
63
64     sum_cost = 0;
65     epsilon = 1e-8;
66
67     for i = 1:N_linear
68         sum_cost = sum_cost + (1 - y_train(i)) * (-log(1 - g(i) + epsilon)) ...
69             + y_train(i) * (-log(g(i) + epsilon));
70     end
71
```

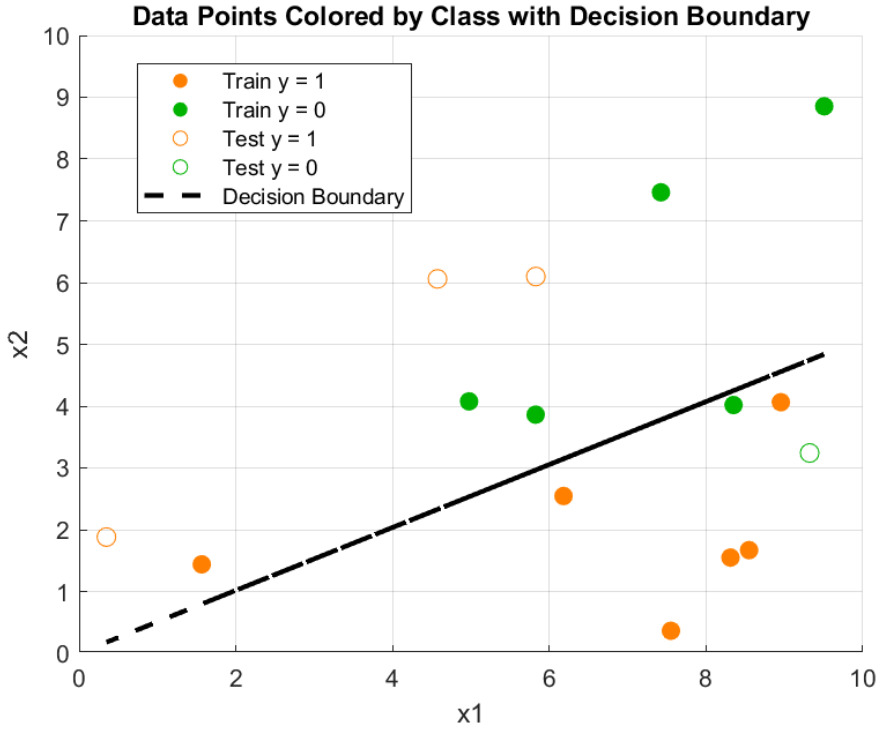
```
72      cost = (1 / N_linear) * sum_cost;
73      reg_term = lambda * (Theta' * Theta);
74      Cost(k,1) = cost + reg_term;
75  end
76
77  fig = figure;
78  plot(1:iter, Grad_save, 'b-', 'LineWidth', 2);
79  xlabel('Iteration');
80  ylabel('||\nabla J(\Theta)||');
81  title('Gradient Norm Over Iterations');
82  grid on;
83  set(fig, 'Color', 'w');
84  exportgraphics(fig, 'NT4Fig6.png', 'BackgroundColor', 'white');
85
86  fig = figure;
87  plot(1:iter, Cost, 'r-', 'LineWidth', 2);
88  xlabel('Iteration');
89  ylabel('Cost J(\Theta)');
90  title('Cost Function Over Iterations');
91  grid on;
92  set(fig, 'Color', 'w');
93  exportgraphics(fig, 'NT4Fig7.png', 'BackgroundColor', 'white');
94
95  decision_line = -(Theta(1)/Theta(2)) * x1;
96
97  fig = figure;
98
99  % === TRAINING DATA ===
100 % y = 1     orange, filled
101 idx_train_1 = y_train == 1;
102 scatter(x1_train(idx_train_1), x2_train(idx_train_1), 60, [1 0.5 0], 'filled');
        hold on;
103
104 % y = 0     green, filled
105 idx_train_0 = y_train == 0;
106 scatter(x1_train(idx_train_0), x2_train(idx_train_0), 60, [0 0.7 0], 'filled');
107
108 % === TEST DATA ===
109 % y = 1     orange, empty
110 idx_test_1 = y_test == 1;
111 scatter(x1_test(idx_test_1), x2_test(idx_test_1), 60, [1 0.5 0]);
112
```

```
% y = 0      green, empty
idx_test_0 = y_test == 0;
scatter(x1_test(idx_test_0), x2_test(idx_test_0), 60, [0 0.7 0]);

% === DECISION LINE ===
plot(x1, decision_line, 'k--', 'LineWidth', 2);

xlabel('x1');
ylabel('x2');
title('Data Points Colored by Class with Decision Boundary');

legend({'Train y = 1', 'Train y = 0', 'Test y = 1', 'Test y = 0', 'Decision
    Boundary'}, 'Location', 'best');
grid on;
axis([0 10 0 10]);
hold off;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT4Fig8.png', 'BackgroundColor', 'white');
```

Listing 4.1: Logistic Regression Training and Visualization Code







4.2.2 Practical Part – Section c

In this section, we extend logistic regression by using a polynomial feature representation. Specifically, we use a quadratic expansion of the input features:

- A column of ones (bias)
- x_1^2

- x_2^2
- $x_1 \cdot x_2$

The MATLAB code for this implementation is shown below:

```
%% Polynomial Logistic Regression

X_poly = [ones(size(x1_train)), ...
          x1_train.^2, ...
          x2_train.^2, ...
          x1_train .* x2_train];

Theta_poly = randn(size(X_poly, 2), 1);
Cost_poly = zeros(iter, 1);
Grad_save_poly = zeros(iter, 1);
tau = 0.001;
iter = 500;
lambda = 0.3;

for k = 1:iter
    h = X_poly * Theta_poly;
    g = 1 ./ (1 + exp(-h));
    Gradient = zeros(size(Theta_poly));

    for i = 1:N_linear
        xi = X_poly(i,:)';
        Gradient = Gradient + (g(i) - y_train(i)) * xi;
    end

    Gradient = (-1 / N_linear) * Gradient;
    Grad_save_poly(k) = norm(Gradient);
    Theta_poly = Theta_poly + tau * Gradient;

    sum_cost = 0;
    for i = 1:N_linear
        sum_cost = sum_cost + (1 - y_train(i)) * (-log(1 - g(i) + epsilon)) ...
            + y_train(i) * (-log(g(i) + epsilon));
    end

    cost = (1 / N_linear) * sum_cost;
    reg_term = lambda * (Theta_poly' * Theta_poly);
    Cost_poly(k) = cost + reg_term;
end
```

The plots below show the convergence of the gradient norm and the cost function over iterations:

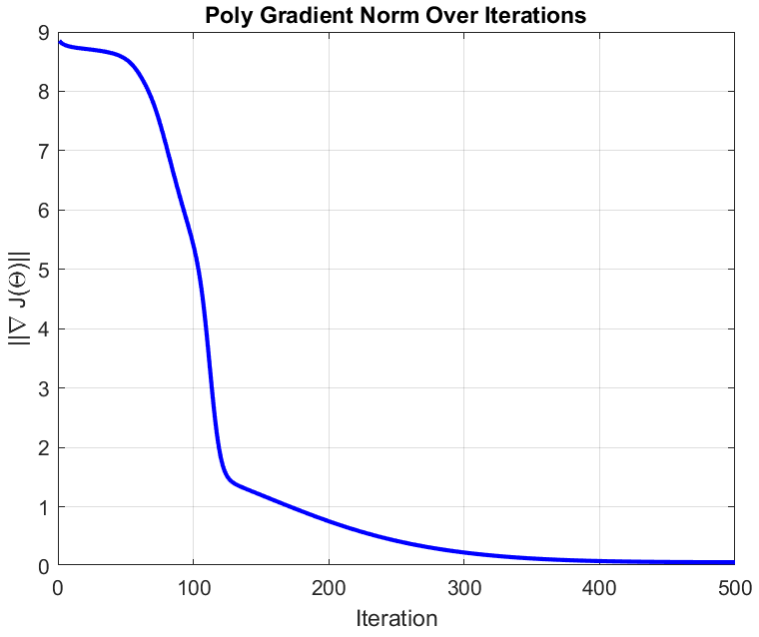


Figure 4.1: Polynomial Logistic Regression - Gradient Norm Over Iterations

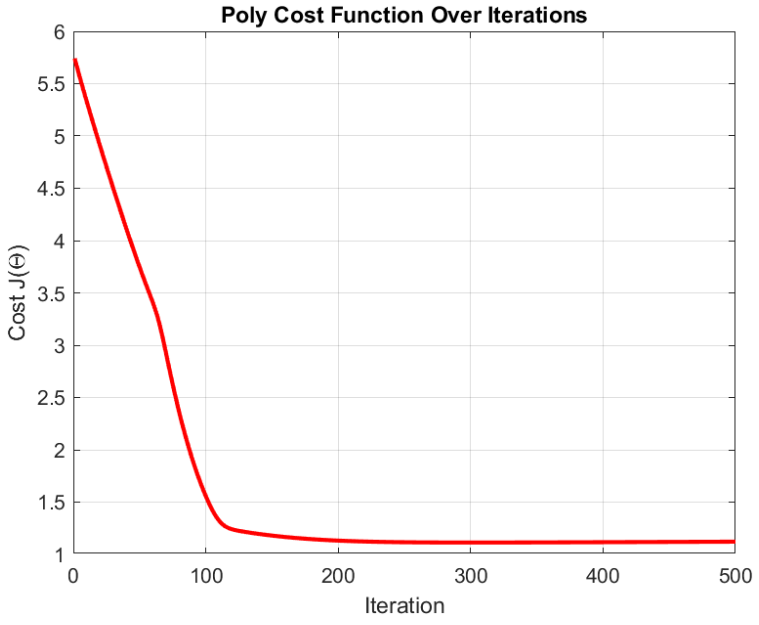


Figure 4.2: Polynomial Logistic Regression - Cost Function Over Iterations

To visualize the decision boundary in the polynomial feature space, we create a meshgrid and evaluate the logistic function. The boundary is then plotted using a contour at $g(x) = 0.5$:

```
[x1_grid, x2_grid] = meshgrid(linspace(0, 10, 300), linspace(0, 10, 300));
x1_sq = x1_grid.^2;
x2_sq = x2_grid.^2;
x1x2 = x1_grid .* x2_grid;
```

```
5
6  X_mesh = [ones(numel(x1_grid), 1), ...
7            x1_sq(:), ...
8            x2_sq(:), ...
9            x1x2(:)];
10
11 h_mesh = X_mesh * Theta_poly;
12 g_mesh = reshape(1 ./ (1 + exp(-h_mesh)), size(x1_grid));
13
14 contour(x1_grid, x2_grid, g_mesh, [0.5, 0.5], 'k--', 'LineWidth', 2); hold on;
15 scatter(x1_train(y_train==1), x2_train(y_train==1), 60, [1 0.5 0], 'filled');
16 scatter(x1_train(y_train==0), x2_train(y_train==0), 60, [0 0.7 0], 'filled');
17 scatter(x1_test(y_test==1), x2_test(y_test==1), 60, [1 0.5 0]);
18 scatter(x1_test(y_test==0), x2_test(y_test==0), 60, [0 0.7 0]);
```

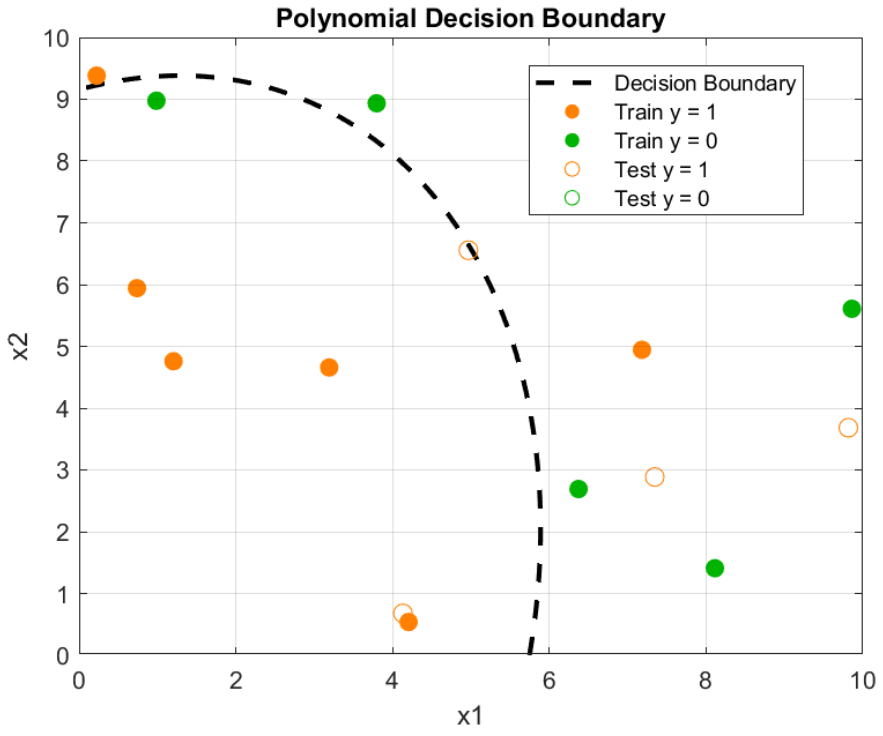


Figure 4.3: Polynomial Logistic Regression - Decision Boundary and Data Points

Chapter 5

Supervised Learning II

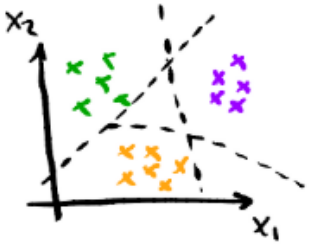
5.1 Multiclass Classification

Instead of outputs that are only 1 or 0, we want to have more outputs ⇒ not continuous.

Example:

$$y \in \{1, 2, 3\}$$

We obtain 3 decision lines.



Cost Function

We transform discrete data into binary:

$$y = \begin{bmatrix} 1 \\ 1 \\ 3 \\ 2 \\ \vdots \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

The cost function:

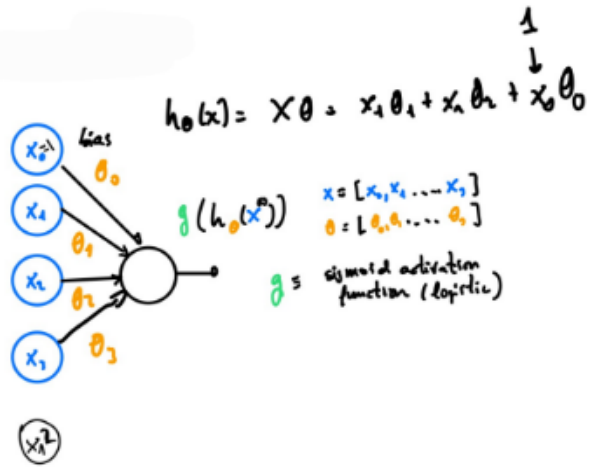
$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log(h_k(x^{(i)})) - (1-y_k^{(i)})\log(1-h_k(x^{(i)}))\right] + \lambda\|\Theta\|^2$$

$$h = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_K \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

5.2 Neural Model (Logistic Regression)

Linear Unit:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$



One-vs-All

The output will be a vector instead of a number:

$$\begin{bmatrix} g^{(1)}(x) \\ g^{(2)}(x) \\ g^{(3)}(x) \end{bmatrix} = \begin{bmatrix} g(h_\theta^{(1)}(x)) \\ g(h_\theta^{(2)}(x)) \\ g(h_\theta^{(3)}(x)) \end{bmatrix} \sim \begin{bmatrix} y \end{bmatrix}$$

If $y_i = \{1,0,0\}$ then $g^{(1)} = g(h_1)$, $g^{(2)} = g(h_2) = 0$, etc.

Regularized Cost for One-vs-All

$$J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[(1-y_k^{(i)})(-\log(1-g(h_k(\boldsymbol{x}^{(i)})))) + y_k^{(i)}(-\log(g(h_k(\boldsymbol{x}^{(i)}))))\right] + \lambda\|\boldsymbol{\Theta}\|^2$$

$$= J_1(\boldsymbol{\theta}_1) + J_2(\boldsymbol{\theta}_2) + J_3(\boldsymbol{\theta}_3) + \lambda\|\boldsymbol{\Theta}\|^2$$

Gradient expressions for each class:

$$\nabla_{\boldsymbol{\theta}_1} J = \frac{1}{m}\sum_{i=1}^{m}\left[g(h_{\theta_1}(\boldsymbol{x}^{(i)})) - y_1^{(i)}\right]\boldsymbol{x}^{(i)} + \lambda\boldsymbol{\theta}_1$$

$$\nabla_{\boldsymbol{\theta}_2} J = \frac{1}{m}\sum_{i=1}^{m}\left[g(h_{\theta_2}(\boldsymbol{x}^{(i)})) - y_2^{(i)}\right]\boldsymbol{x}^{(i)} + \lambda\boldsymbol{\theta}_2$$

$$\nabla_{\boldsymbol{\theta}_3} J = \frac{1}{m}\sum_{i=1}^{m}\left[g(h_{\theta_3}(\boldsymbol{x}^{(i)})) - y_3^{(i)}\right]\boldsymbol{x}^{(i)} + \lambda\boldsymbol{\theta}_3$$

Neural Network (Classification)

We are given a training set:

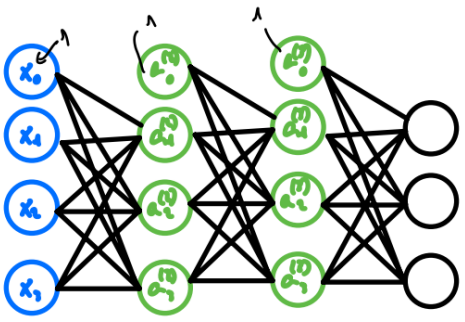
$$\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), (\boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$$

Let:

- L = total number of layers
- S_l = number of units in layer l

The output of the neural network is:

$$h_\Theta(\boldsymbol{x}) = a^{(L)}$$



Cost Function

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[(1-y_k^{(i)})\log(1-h_\Theta(\boldsymbol{x}^{(i)})) + y_k^{(i)}\log(h_\Theta(\boldsymbol{x}^{(i)}))\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{r=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(\Theta_{jr}^{(l)}\right)^2$$

Note: The regularization term excludes bias weights (i.e., terms where $r = 0$).

Forward Propagation

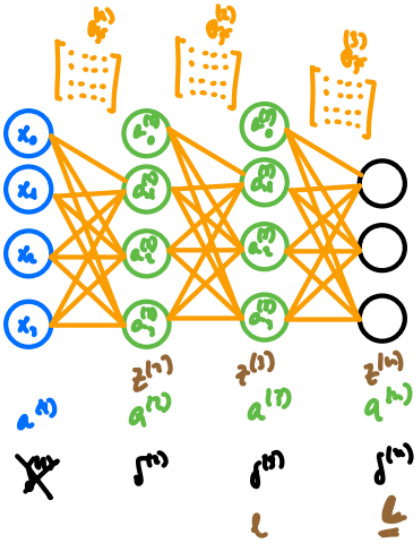
We perform forward propagation through the neural network layer by layer.

Let:

- $a^{(1)} = \boldsymbol{x}$ (input layer activations)
- $z^{(l)}$ = linear combination at layer l
- $a^{(l)}$ = activation at layer l

The propagation through layers is as follows:

$$\begin{aligned}
a^{(1)} &= \boldsymbol{x} \\
z^{(2)} &= \Theta^{(1)}a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \\
z^{(3)} &= \Theta^{(2)}a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \\
&\vdots \\
z^{(L)} &= \Theta^{(L-1)}a^{(L-1)} \\
a^{(L)} &= g(z^{(L)})
\end{aligned}$$



Remember:

$$g(z) = \frac{1}{1+e^{-z}}$$
$$g'(z) = g(z)(1-g(z))$$

Gradient Computation

Before: One-layer architecture

$$\begin{aligned}\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_k} &= \frac{1}{m}\sum_{i=1}^{m}\left[(1-y_k^{(i)})\cdot\left(-\frac{1}{1-g(h_k(\boldsymbol{x}^{(i)}))}\right)\left(-g(h_k(\boldsymbol{x}^{(i)}))(1-g(h_k(\boldsymbol{x}^{(i)})))\boldsymbol{x}^{(i)}\right)\right.\\ &\quad\left.+\,y_k^{(i)}\cdot\left(-\frac{1}{g(h_k(\boldsymbol{x}^{(i)}))}\right)g(h_k(\boldsymbol{x}^{(i)}))(1-g(h_k(\boldsymbol{x}^{(i)})))\boldsymbol{x}^{(i)}\right]\\ &= \frac{1}{m}\sum_{i=1}^{m}\left[g(h_k(\boldsymbol{x}^{(i)}))-y_k^{(i)}\right]\boldsymbol{x}^{(i)}\end{aligned}$$

Now: Several-layer architecture

For the last layer $(L-1)$***:***

$$\begin{aligned}\frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{jk}^{(L-1)}} &= \frac{1}{m}\sum_{i=1}^{m}\left[(1-y_j^{(i)})\cdot\left(-\frac{1}{1-a_j^{(L)(i)}}\right)(-g'(z_j^{(L)(i)}))a_k^{(L-1)(i)}\right.\\ &\quad \left. +y_j^{(i)}\cdot\left(-\frac{1}{a_j^{(L)(i)}}\right)g'(z_j^{(L)(i)})a_k^{(L-1)(i)}\right]\\ &= \frac{1}{m}\sum_{i=1}^{m}\left[\delta_j^{(L)(i)}\right]a_k^{(L-1)(i)}\\ &= \frac{1}{m}\sum_{i=1}^{m}\delta^{(L)}(a^{(L-1)})\end{aligned}$$

General formula:

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta^{(L-1)}} = \frac{1}{m}\delta^{(L)}(a^{(L-1)})$$

With:

$$\delta^{(L)} = g(z^{(L)}) - \boldsymbol{y}$$

For one before the last layers: $(L-2)$

$$\begin{aligned}\frac{\partial J(\mathbf{\Theta})}{\partial \Theta^{(L-2)}} &= \frac{1}{m}\sum_{i=1}^{m}\delta^{(L)}\frac{\partial z^{(L)}}{\partial \Theta^{(L-2)}}\\ &= \frac{1}{m}\sum_{i=1}^{m}\delta^{(L)}\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\cdot\frac{\partial a^{(L-1)}}{\partial \Theta^{(L-2)}}\end{aligned}$$

Given that:

$$a^{(L-1)} = g(z^{(L-1)}) \quad \text{and} \quad \frac{\partial a^{(L-1)}}{\partial \Theta^{(L-2)}} = g'(z^{(L-1)})\cdot a^{(L-2)}$$

Then:

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta^{(L-2)}} = \frac{1}{m}\sum_{i=1}^{m}\delta^{(L)}\cdot\Theta^{(L)}\cdot g'(z^{(L-1)})\cdot a^{(L-2)}$$

Let:

$$\delta^{(L-1)} = \delta^{(L)}\cdot\Theta^{(L)}\cdot g'(z^{(L-1)})$$

So finally:

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta^{(L-2)}} = \frac{1}{m}\sum_{i=1}^{m}\delta^{(L-1)}\cdot(a^{(L-2)})$$

Backpropagation

Forward propagation

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= g(z^{(4)})
\end{aligned}
$$

Backward propagation

Assume we are on layer 4:

$$
\begin{aligned}
\delta^{(4)} &= a^{(4)} - y \quad \text{(output layer error)} \\
\delta^{(3)} &= \Theta^{(3)\top} \delta^{(4)} \circ g'(z^{(3)}) \\
\delta^{(2)} &= \Theta^{(2)\top} \delta^{(3)} \circ g'(z^{(2)})
\end{aligned}
$$

Note: No $\delta^{(1)}$

We propagate the error backward.

Gradient computation

$$
\frac{\partial J(\boldsymbol{\Theta})}{\partial \Theta^{(l)}} = \frac{1}{m} \sum_{i=1}^{m} \delta^{(l+1)} (a^{(l)})
$$

Remarks

- Now people use automatic differentiation
- Also use regularization

5.3 MATLAB Problem

5.3.1 Data Definition

```
Npoints_sector = 100;
N_sectors = 3;

x1_sector1 = rand(Npoints_sector,1)*10+25;
x2_sector1 = rand(Npoints_sector,1)*10+25;

x1_sector2 = rand(Npoints_sector,1)*10+15;
x2_sector2 = rand(Npoints_sector,1)*10+10;

x1_sector3 = rand(Npoints_sector,1)*10+10;
x2_sector3 = rand(Npoints_sector,1)*10+30;

y_sector1 = ones(Npoints_sector,1);
y_sector2 = ones(Npoints_sector,1)*2;
y_sector3 = ones(Npoints_sector,1)*3;

x1 = [x1_sector1; x1_sector2; x1_sector3];
x2 = [x2_sector1; x2_sector2; x2_sector3];
y  = [y_sector1; y_sector2; y_sector3];

D = [x1, x2, y];

colors = lines(3);
fig = figure;
hold on;
scatter(x1(y==1), x2(y==1), 50, colors(1,:), 'filled'); % Sector 1
scatter(x1(y==2), x2(y==2), 50, colors(2,:), 'filled'); % Sector 2
scatter(x1(y==3), x2(y==3), 50, colors(3,:), 'filled'); % Sector 3
hold off;
xlabel('x1');
ylabel('x2');
title('3 sector distribution');
legend({'Sector 1', 'Sector 2', 'Sector 3'}, 'Location', 'best');
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT5Fig5.png', 'BackgroundColor', 'white');
```

Listing 5.1: Generating synthetic sector data

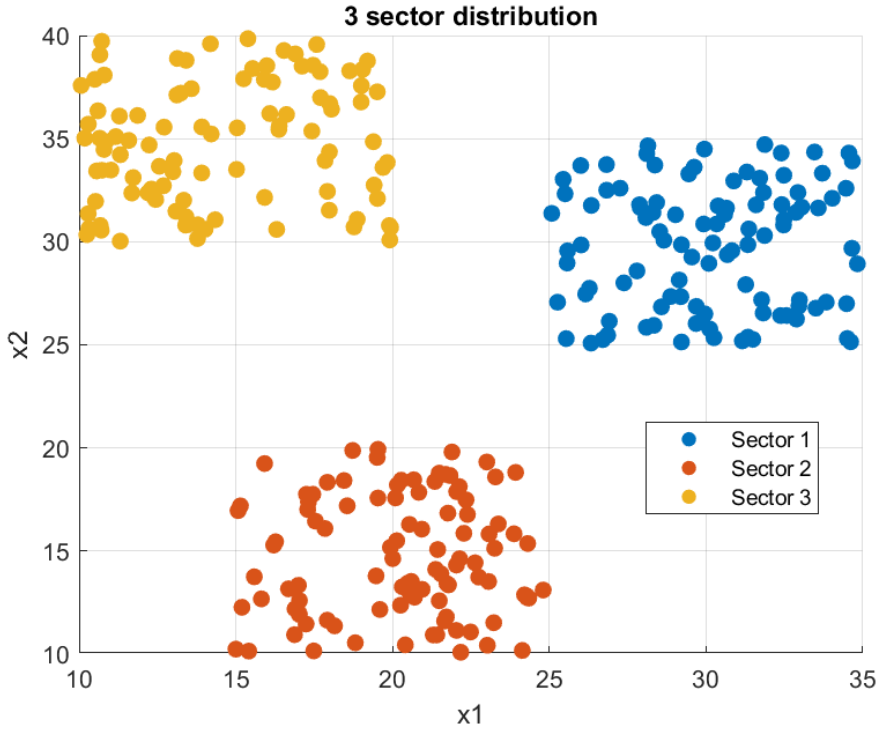


Figure 5.1: 3 Sector Distribution

5.3.2 Train-Test Split

```
Npoints_total = size(D, 1);

idx = randperm(Npoints_total);

train_size = round(0.7 * Npoints_total);

train_idx = idx(1:train_size);
test_idx = idx(train_size+1:end);

D_train = D(train_idx, :);
D_test = D(test_idx, :);

x1_train = D_train(:,1);
x2_train = D_train(:,2);
y_train = D_train(:,3);

x1_test = D_test(:,1);
x2_test = D_test(:,2);
y_test = D_test(:,3);
```

Listing 5.2: Splitting dataset into training and test sets

5.3.3 Linear Logistic Regression

```
N_train = size(x1_train,1);
N_dim = 3;

% Define matrix X
X_linear = [ones(N_train,1), x1_train, x2_train];

% Define matrix Y of 1's and 0's
Y = zeros(N_train,N_sectors);
for i=1:N_train
    Y(i,y_train(i))=1;
end

% Theta linear model x = [1,x1,x2]
Theta_linear = randn(N_dim,N_sectors) * 0.1;

% Heavyside function
noise = 30 * randn(N_train,N_sectors);
h_linear = X_linear*Theta_linear;

% Signoid function
g_linear = zeros(N_train,N_sectors);
for i=1:N_train
    for j=1:N_sectors
        g_linear(i,j) = 1/(1+exp(-h_linear(i,j)));
    end
end

% Gradient method
tau = 0.001;
iter = 200;
Lambda = 0.01;
Cost = zeros(iter,1);
Grad_save = zeros(iter, N_sectors);

for k=1:iter
    sum_grad = zeros(N_dim, N_sectors);

    % Compute gradient
    for i=1:N_train
        for j=1:N_sectors
```

```
            sum_grad(:,j) = sum_grad(:,j) + (g_linear(i,j) - Y(i,j)) * X_linear(i
                ,:)';
        end
    end

    Gradient = (1/N_train) * sum_grad;

    % Store gradient norms
    for j=1:N_sectors
        Grad_save(k,j) = norm(Gradient(:,j));
    end

    % Update Theta
    Theta_linear = Theta_linear - tau * Gradient;

    % Update heavyside and sigmoid
    h_linear = X_linear * Theta_linear;
    g_linear = 1 ./ (1 + exp(-h_linear));

    % Compute cost
    sum_cost = 0;
    for i=1:N_train
        for j=1:N_sectors
            sum_cost = sum_cost + (1 - Y(i,j)) * (-log(1 - g_linear(i,j))) + Y(i,j
                ) * (-log(g_linear(i,j)));
        end
    end

    reg_term = 0;
    for j=1:N_sectors
        reg_term = reg_term + Lambda * sum(Theta_linear(:,j).^2);
    end

    Cost(k,1) = (1/N_train) * sum_cost + reg_term;
end

% Cost function
fig = figure;
plot(1:iter, Cost, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Cost');
title('Cost evolution for linear model');
```

```
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT5Fig6.png', 'BackgroundColor', 'white');

% Gradient norm per class
fig = figure;
hold on;
for j = 1:N_sectors
    plot(1:iter, Grad_save(:,j), 'LineWidth', 2, 'DisplayName', ['Class ', num2str
        (j)]);
end
xlabel('Iteration');
ylabel('Gradient norm');
title('Gradient norm evolution per class for linear model');
legend('show');
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT5Fig7.png', 'BackgroundColor', 'white');

% Define the range of x1 for plotting the decision boundaries
x1_range = linspace(min(x1), max(x1), 100);

% Colors for each class
colors = lines(N_sectors);

% Plot all data points (filled)
fig = figure;
hold on;
scatter(x1(y==1), x2(y==1), 50, colors(1,:), 'filled');
scatter(x1(y==2), x2(y==2), 50, colors(2,:), 'filled');
scatter(x1(y==3), x2(y==3), 50, colors(3,:), 'filled');

% Overlay training points (empty)
scatter(x1_train, x2_train, 70, 'k', 'LineWidth', 1.5);

% Plot decision boundaries
for j = 1:N_sectors
    theta_j = Theta_linear(:,j);
    % Avoid division by zero
    if abs(theta_j(3)) > 1e-6
        x2_line = -(theta_j(1) + theta_j(2)*x1_range) / theta_j(3);
        plot(x1_range, x2_line, '--', 'Color', colors(j,:), 'LineWidth', 2, ...
```

```
122             'DisplayName', ['Boundary for class ', num2str(j)]);
123     end
124 end
125
126 xlabel('x1');
127 ylabel('x2');
128 title('Data points with decision boundaries and training set for linear model');
129 legend({'Sector 1', 'Sector 2', 'Sector 3', 'Training points'}, 'Location', 'best'
    );
130 grid on;
131 axis tight;
132 set(fig, 'Color', 'w');
133 exportgraphics(fig, 'NT5Fig8.png', 'BackgroundColor', 'white');
```

Listing 5.3: Solving linear logistic regression with regularization

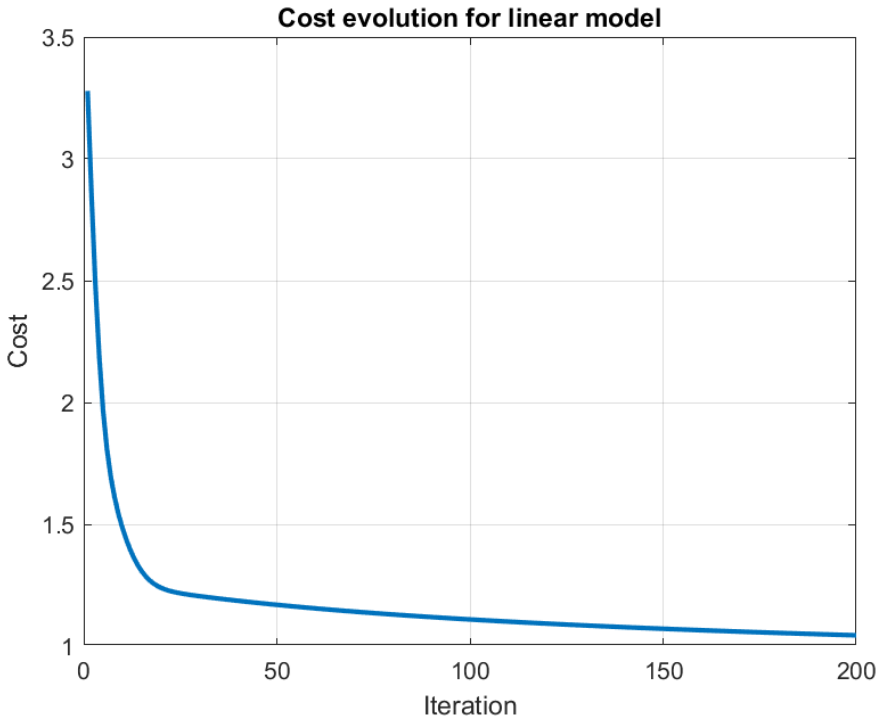


Figure 5.2: Cost evolution for linear model

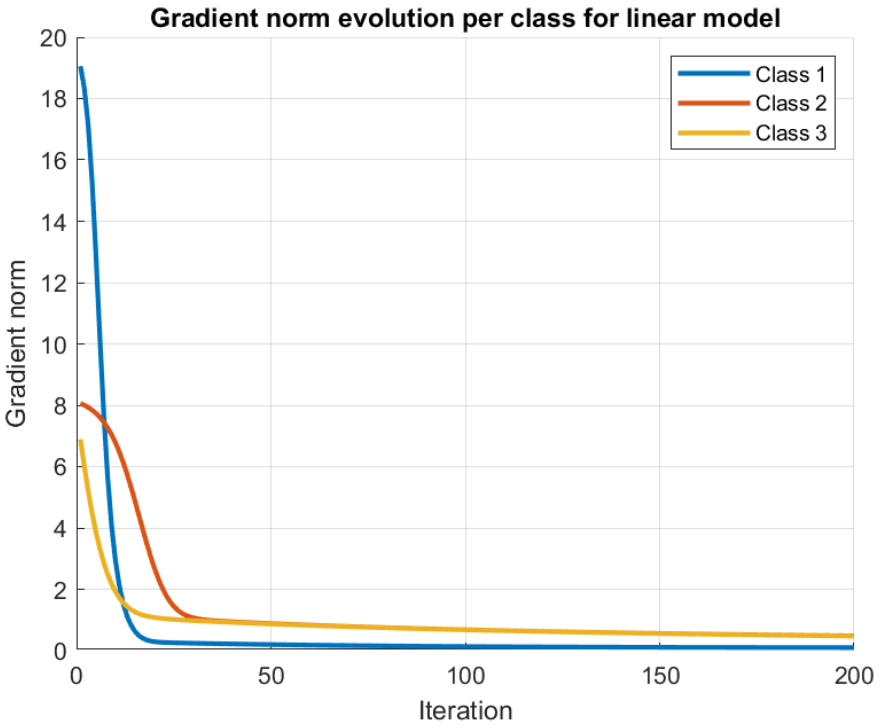


Figure 5.3: Gradient norm per class for linear model

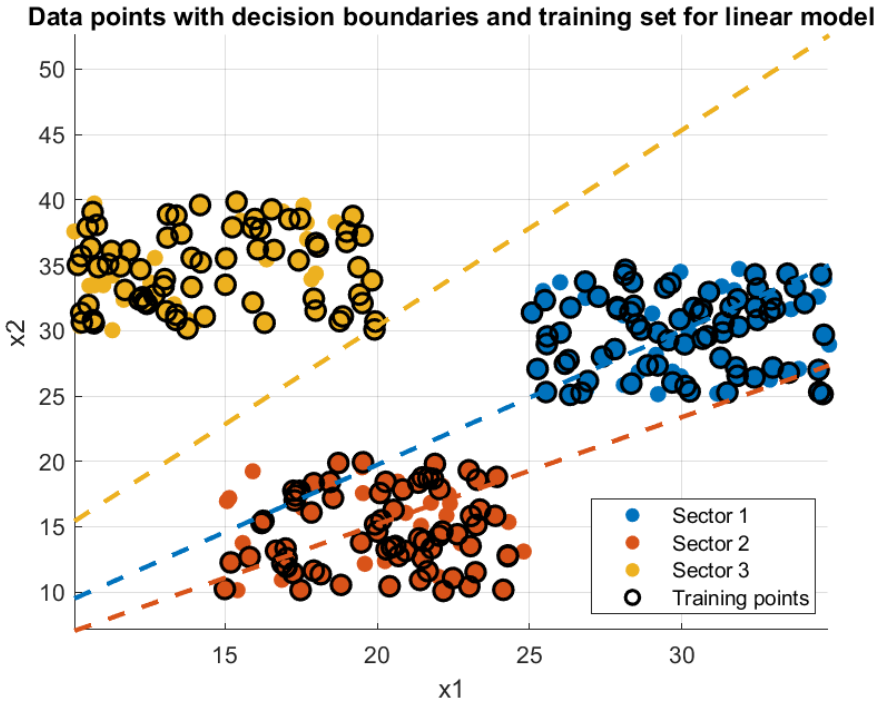


Figure 5.4: Decision boundaries for linear model

5.3.4 Quadratic Logistic Regression

```
1 %% Solve logistic regression considering quadratic model x = [1, x1, x2, x1^2, x1*
      x2, x2^2]
2
```

```
N_train = size(x1_train, 1);
N_dim = 6;

% Define matrix X
X_quadratic = [ones(N_train,1), x1_train, x2_train, x1_train.^2, x1_train.*
    x2_train, x2_train.^2];

% Target matrix Y
Y = zeros(N_train, N_sectors);
for i = 1:N_train
    Y(i, y_train(i)) = 1;
end

% Theta for quadratic model
Theta_quadratic = randn(N_dim, N_sectors) * 0.1;

% Gradient descent parameters
tau = 0.00001;
iter = 400;
Lambda = 0.01;
Cost_quad = zeros(iter,1);
Grad_save_quad = zeros(iter, N_sectors);

% Initial h and g
h_quad = X_quadratic * Theta_quadratic;
g_quad = 1 ./ (1 + exp(-h_quad));

for k = 1:iter
    sum_grad = zeros(N_dim, N_sectors);

    % Compute gradient
    for i = 1:N_train
        for j = 1:N_sectors
            sum_grad(:,j) = sum_grad(:,j) + (g_quad(i,j) - Y(i,j)) * X_quadratic(i
                ,:)';
        end
    end

    Gradient = (1/N_train) * sum_grad;

    % Store gradient norms
    for j = 1:N_sectors
```

```
        Grad_save_quad(k,j) = norm(Gradient(:,j));
    end

    % Update Theta
    Theta_quadratic = Theta_quadratic - tau * Gradient;

    % Recalculate h and g
    h_quad = X_quadratic * Theta_quadratic;
    g_quad = 1 ./ (1 + exp(-h_quad));

    % Compute cost
    sum_cost = 0;
    for i = 1:N_train
        for j = 1:N_sectors
            sum_cost = sum_cost + (1 - Y(i,j)) * (-log(1 - g_quad(i,j))) + Y(i,j)
                * (-log(g_quad(i,j)));
        end
    end

    reg_term = 0;
    for j=1:N_sectors
        reg_term = reg_term + Lambda * sum(Theta_quadratic(:,j).^2);
    end

    Cost_quad(k,1) = (1/N_train) * sum_cost + reg_term;
end

% Plot cost for quadratic model
fig = figure;
plot(1:iter, Cost_quad, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Cost');
title('Cost evolution for quadratic model');
grid on;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT5Fig9.png', 'BackgroundColor', 'white');

% Plot gradient norms for each sector
fig = figure;
hold on;
for j = 1:N_sectors
    plot(1:iter, Grad_save_quad(:,j), 'LineWidth', 2);
```

```
end
xlabel('Iteration');
ylabel('Gradient Norm');
title('Gradient norm evolution for quadratic model');
legend({'Sector 1', 'Sector 2', 'Sector 3'}, 'Location', 'best');
grid on;
hold off;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT5Fig10.png', 'BackgroundColor', 'white');
```

Listing 5.4: Solving quadratic logistic regression

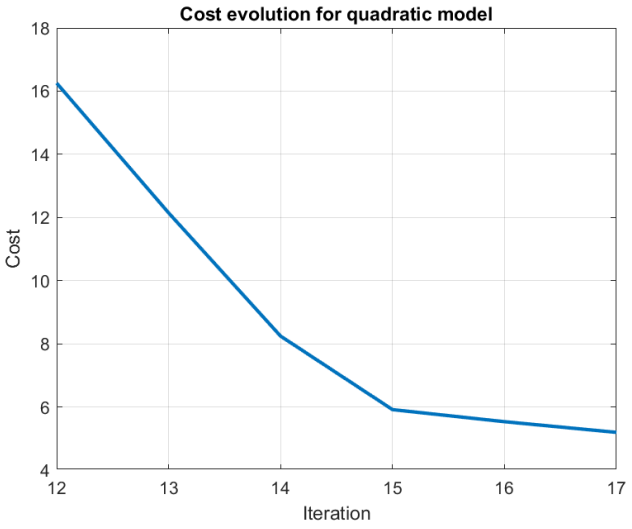


Figure 5.5: Cost evolution for quadratic model

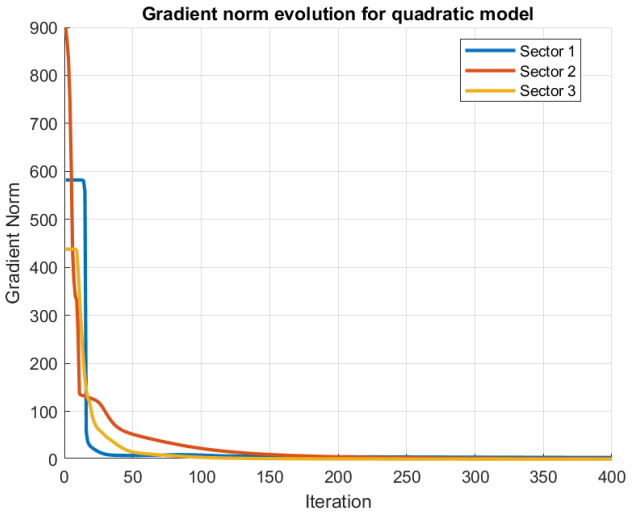


Figure 5.6: Gradient norm per class for quadratic model

5.3.5 Decision Boundaries for the Quadratic Model

```
% Decision boundaries for quadratic model
x1_range = linspace(min(x1), max(x1), 200);
x2_range = linspace(min(x2), max(x2), 200);
[X1_grid, X2_grid] = meshgrid(x1_range, x2_range);

% Feature expansion for each grid point
X_feat = [ones(numel(X1_grid),1), ...
          X1_grid(:), X2_grid(:), ...
          X1_grid(:).^2, X1_grid(:).*X2_grid(:), X2_grid(:).^2];

% Compute h(x) for each class
H_grid = X_feat * Theta_quadratic;

% Plot
fig = figure; hold on;

% Plot data points
scatter(x1(y==1), x2(y==1), 50, colors(1,:), 'filled');
scatter(x1(y==2), x2(y==2), 50, colors(2,:), 'filled');
scatter(x1(y==3), x2(y==3), 50, colors(3,:), 'filled');

% Overlay training points
scatter(x1_train, x2_train, 70, 'k', 'LineWidth', 1.5);

% Plot decision boundaries where h_j(x) = 0
contour_colors = {'b', 'r', 'g'};
for j = 1:N_sectors
    Hj = reshape(H_grid(:,j), size(X1_grid));
    contour(X1_grid, X2_grid, Hj, [0, 0], '--', 'LineWidth', 2, ...
        'LineColor', contour_colors{j});
end

xlabel('x1'); ylabel('x2');
title('Quadratic Model: Decision Boundaries and Data Points');
legend({'Sector 1','Sector 2','Sector 3','Training points'}, 'Location', 'best');
grid on; axis tight;
set(fig, 'Color', 'w');
exportgraphics(fig, 'NT5Fig12.png', 'BackgroundColor', 'white');
```

Listing 5.5: Quadratic decision boundaries using $h_j(x) = 0$

To visualize how the classifier separates the input space into different sectors, we use the decision functions

$h_j(x)$ from the quadratic model. For each class j, the boundary is defined by the implicit curve:

$$h_j(x) = \theta_{j0} + \theta_{j1}x_1 + \theta_{j2}x_2 + \theta_{j3}x_1^2 + \theta_{j4}x_1x_2 + \theta_{j5}x_2^2 = 0$$

These equations form second-degree (quadratic) curves in the (x_1, x_2) plane. We compute them over a grid and use MATLAB's `contour` function to draw the zero-level set of each $h_j(x)$, producing smooth polynomial decision boundaries for each sector.

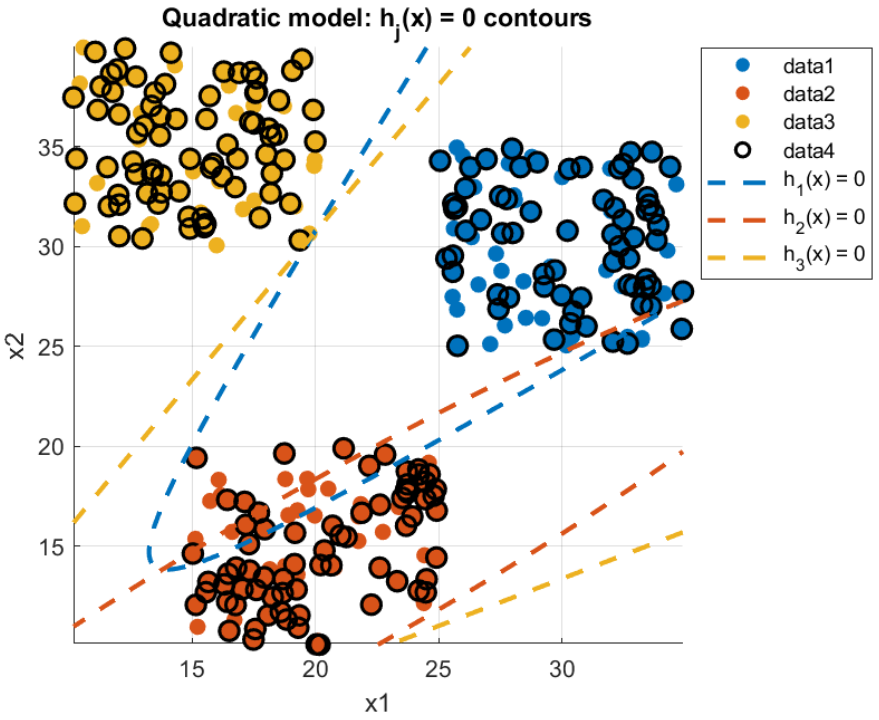


Figure 5.7: Quadratic model decision boundaries ($h_j(x) = 0$) for each sector, overlaid on the data points and training set.

Chapter 6

Unsupervised Learning

PCA is a reduction of features, while K-means is a reduction of data.

6.1 PCA - Principal Component Analysis

Dimensionality Reduction

$$\mathbf{x} \in \mathbb{R}^d \xrightarrow{\mathcal{F}} \mathbf{z} \in \mathbb{R}^l, \quad \text{with } l < d$$

where:

- $\mathbf{x}$: Input in ambient space
- $\mathbf{z}$: Output in latent space

Also known as:

- Discrete Karhunen-Loève Transform (KLT)
- Eckart–Young Theorem

Most widely used form of dimensionality reduction.

Main Idea

Find linear and orthogonal mapping $W \in \mathbb{R}^{d \times l}$ such that:

$$\mathbf{z} = W^T \mathbf{x}$$

is a good representation of $\mathbf{x} \in \mathbb{R}^d$, in the sense that decoding

$$\hat{\mathbf{x}} = W\mathbf{z} = WW^T\mathbf{x}$$

is close to the original data.

Optimization Objective (Reconstruction Error)

$$\min_W \mathcal{L}(W) = \frac{1}{N}\sum_{n=1}^{N} \left\|\mathbf{x}_n - WW^T\mathbf{x}_n\right\|^2 \quad \text{s.t.} \quad W^TW = I$$

Matrix Notation

$$X = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_N \end{bmatrix} \in \mathbb{R}^{d\times N}, \quad \hat{X} = WW^TX$$

Objective (Matrix Form)

$$\min_W \mathcal{L}(W) = \frac{1}{N}\left\|X - WW^TX\right\|_F^2$$

In Matrix Form

$$X \in \mathbb{R}^{d\times N}$$
$$Z = W^TX \in \mathbb{R}^{l\times N}$$
$$\hat{X} = WZ = WW^TX$$

Alternate Formulation

$$\min_W \mathcal{L}(W) = \frac{1}{N}\left\|X^T - X^TWW^T\right\|_F^2$$

Mapping as:

- **Parametric Model:** $\mathbf{z} = f(\mathbf{x};\theta)$
  - Representing step for deep learning algorithms
- **Non-parametric Model:** $\mathbf{z}_n = f(\mathbf{x}_n)$
  - Used in data visualization, manifold embedding, and simple linear adapters

Applications of PCA

PCA can be applied in various contexts. Below are two typical applications that illustrate the importance of identifying directions of maximal variance in data:

1. **Finding the direction of maximum variance**:
   - The first principal component identifies the direction along which the data varies the most.
   - Remaining directions (principal components) capture progressively less variance.
   - With just one principal direction, we can often capture the majority of the information in high-dimensional data.

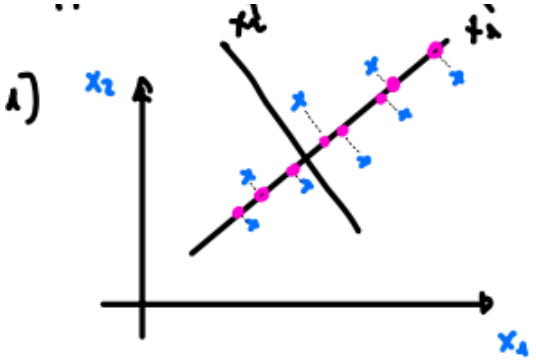


Figure 6.1: Illustration of PCA finding the direction of maximum variance (along $\hat{x}_1$) and projecting data onto it.

2. **Data visualization and pattern discovery**:
   - Projecting high-dimensional data (e.g., images) to two principal components (x_1, x_2) for visual inspection.
   - This can reveal interpretable structure, such as variations in image orientation or thickness.
   - Useful for understanding clusters, outliers, and latent features.

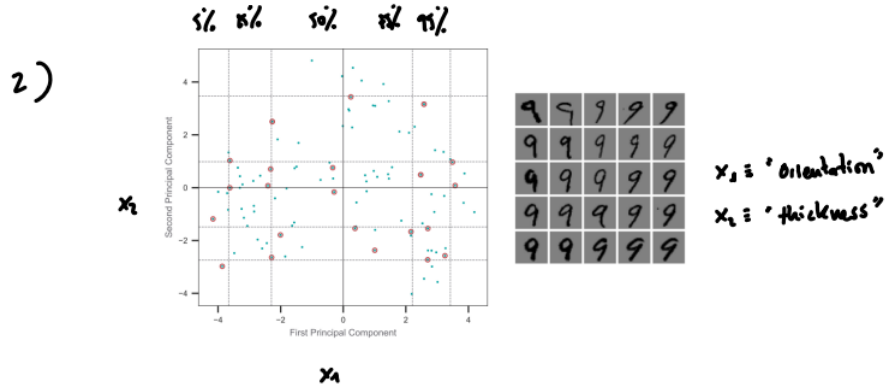


Figure 6.2: PCA applied to digit images, with x_1 encoding orientation and x_2 encoding thickness.

Canonical Form of the Data

In Principal Component Analysis (PCA), preprocessing the data into a canonical form is an essential step to ensure accurate and interpretable results. This preprocessing step involves standardizing each feature so that they have zero mean and unit variance.

Data Preprocessing

For each feature (column of the data matrix X):

- **Empirical mean:**

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$$

- **Standard deviation:**

$$s_j = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2}$$

Re-definition (Standardization)

We redefine the standardized data:

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{s_j}, \quad \hat{\mu}_j = \frac{1}{N}\sum_{i=1}^{N}\hat{x}_{ij} = 0$$

This ensures each feature has zero mean and unit variance.

Covariance Matrix

Let μ be the mean vector of all features. The empirical covariance matrix of X is then given by:

$$\Sigma_{uv} = \frac{1}{N}\sum_{i=1}^{N}(x_i^{(u)} - \mu_u)(x_i^{(v)} - \mu_v) = \frac{1}{N}(X - \rho\mathbf{1})^T(X - \rho\mathbf{1})$$

When the data is already centered:

$$\hat{\Sigma}_c = \frac{1}{N}\hat{X}^T\hat{X} \quad \Rightarrow \quad \hat{\Sigma} = \frac{1}{N}X^T X$$

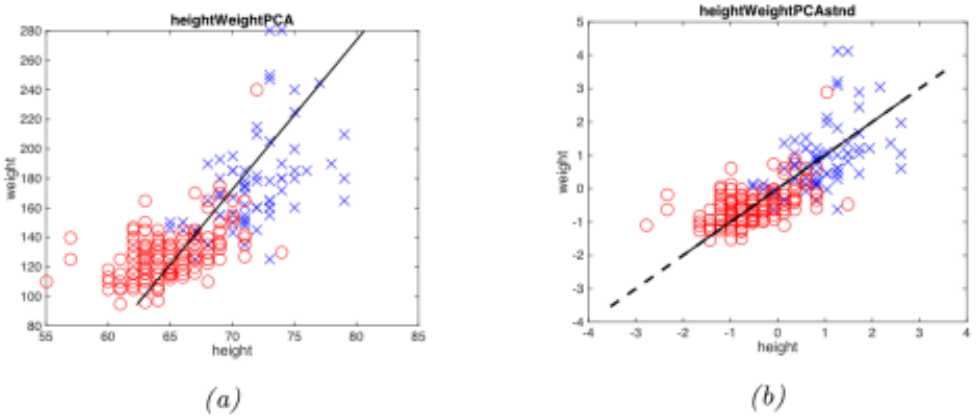


Figure 6.3: Effect of data standardization. (a) Raw data with original orientation. (b) Standardized data aligned with principal directions.

Computing PCA using SVD

PCA can be efficiently computed using the Singular Value Decomposition (SVD), which provides a numerically stable and robust alternative to eigenvalue decomposition of the covariance matrix.

PCA via Covariance Eigen-Decomposition

Let the covariance matrix be:

$$\Sigma = \frac{1}{N} X^\top X = W_\ell \Lambda_\ell W_\ell^\top$$

PCA via SVD of the Data Matrix

Let the data matrix $X \in \mathbb{R}^{N \times d}$ be decomposed using SVD:

$$X = U_\ell S_\ell V_\ell^\top$$

where:

- $U_\ell^\top U_\ell = I$, $V_\ell^\top V_\ell = I$
- S_ℓ is the diagonal matrix of the top ℓ singular values

Using this, we get the low-dimensional representation:

$$Z = XW_\ell = U_\ell S_\ell V_\ell^\top V_\ell = U_\ell S_\ell$$

and the reconstruction:

$$X \approx ZW_\ell^\top = U_\ell S_\ell V_\ell^\top$$

We also note:

$$\Sigma = \frac{1}{N} X^\top X = \frac{1}{N} V_\ell S_\ell^2 V_\ell^\top \quad \Rightarrow \quad \Lambda_\ell = \frac{S_\ell^2}{N}, \quad W_\ell = V_\ell$$

Advantages of SVD

- W_ℓ can be directly obtained from V_ℓ in the SVD.
- SVD is more efficient, stable, and precise in numerical computation.

Choosing ℓ

Selecting the number of components ℓ involves balancing between:

- Training error (decreases with ℓ)
- Test error (may increase after optimal ℓ due to overfitting)
- Singular values (drop rapidly; ℓ is chosen before the values become very small)

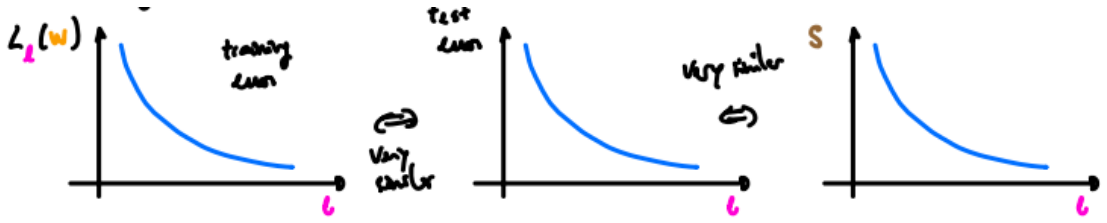


Figure 6.4: Computing PCA using SVD and choosing the optimal number of components ℓ.

Chapter 7

Final Project: Dimensionality Reduction and Visualization of Aircraft Characteristics using SVD

7.1 Introduction

The accelerated growth and investigation of aviation technology has led to the development of a wide variety and types of aircraft, each characterized by many different physical parameters. In order to understand patterns so we can compare these aircraft or identify clusters of similar designs, we want to explore the use of numerical tools and machine learning techniques to simplify the analysis of such complex data.

Dimensionality reduction is a key step in understanding these patterns. The technique used in this chapter is the Singular Value Decomposition (SVD), which stands out due to its mathematical rigor and practical effectiveness. We will apply the SVD technique to a dataset of aircraft characteristics to reduce the dimensionality and study the patterns in a two dimensional space (x_1, x_2). Each aircraft is represented as a point, with corresponding images for each point to make the visualization more clear and provide an intuitive understanding of their similarities.

7.2 Aim

The aim of this project is to project multidimensional commercial aircraft characteristics onto a two-dimensional plane using the SVD method and enhance interpretability through visual annotations using aircraft images.

Objectives:

- To collect and preprocess a dataset of aircraft specifications.
- To implement SVD for dimensionality reduction of high-dimensional aircraft data.
- To generate a 2D (x_1, x_2) representation of the aircraft in a visually meaningful way.
- To associate each data point with an image of the corresponding aircraft.
- To analyze the clustering or grouping behavior of aircraft based on their features.

7.3 Methodology

The following section describes the methodology used to process the aircraft dataset and prepare it for further numerical analysis, specifically dimensionality reduction through the SVD method.

7.3.1 Data Acquisition and Formatting

The dataset used in this study compiles physical and performance parameters for a variety of commercial aircraft. This data was collected from publicly available manufacturer websites and encyclopedic sources, and then manually curated into a CSV file named `aircraft_data_csv.csv`. The dataset includes both physical parameters (e.g., MTOW, wingspan, length) and performance parameters (e.g., cruise speed, range, takeoff distance).

7.3.2 Preprocessing in MATLAB

The preprocessing of the dataset was conducted in MATLAB using a dedicated script named `SVD.m`. This script performs the following steps:

- Reads the CSV file into a MATLAB table structure using the `readtable` function.
- Ensures all numerical entries are correctly formatted by replacing any commas with dots and re-saving the file as needed.
- Extracts the numerical variables from the table, excluding the aircraft names.
- Normalizes the data to have zero mean and unit standard deviation for each variable. This is done using the formula:

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

  where X is the original matrix, μ is the mean, and σ is the standard deviation of each column.

7.3.3 Dimensionality Reduction using SVD

Following normalization, the data matrix is ready for dimensionality reduction using SVD. SVD is a linear algebra technique that decomposes the normalized data matrix X_{norm} into three matrices:

$$X_{\text{norm}} = U\Sigma V^T$$

Where:

- U is an orthogonal matrix containing the left singular vectors,
- Σ is a diagonal matrix with singular values representing the magnitude of each principal direction,
- V^T is the transpose of an orthogonal matrix whose rows are the right singular vectors.

By selecting the first two dominant singular vectors, a reduced two-dimensional representation of the dataset can be obtained:

$$X_{2D} = X_{\text{norm}} V_2$$

where V_2 consists of the first two columns of V. This projection allows for visualization of the aircraft in a new coordinate space that captures the most variance in the data.

7.3.4 Objective of the Projection

The goal of applying SVD is to derive a two-dimensional representation of the aircraft dataset that preserves the structure and variance of the original high-dimensional space. This projection, also known as x_1–x_2 space, enables clearer insights into similarities and differences between aircraft types based on their physical and performance characteristics.

7.4 Background

As a case study, we focus on the specifications of commercial aircraft. These include technical parameters such as wingspan, maximum takeoff weight, cruising speed, engine thrust, range, and passenger capacity. Such features often exhibit correlations and redundancies, making them ideal candidates for dimensionality reduction techniques. By reducing the dimensionality of the data, we aim to obtain a lower-dimensional representation that captures the essential patterns and variations across different aircraft models.

7.4.1 Dataset Construction

To perform our analysis, we construct a dataset comprising various commercial aircraft, including both modern and classic models from major manufacturers such as Boeing, Airbus, Embraer, and Bombardier. The dataset is curated from publicly available technical specifications and industry databases.

Each aircraft is described by a set of quantitative features that represent its physical configuration, operational range, and performance capabilities. These features include:

- **Wingspan (m)** – the distance from one wingtip to the other.
- **Length (m)** – the total longitudinal dimension of the aircraft.
- **Height (m)** – the vertical dimension of the aircraft from ground to tail.
- **Maximum Takeoff Weight (MTOW) (kg)** – the maximum certified weight at which the aircraft can safely take off.
- **Cruising Speed (km/h)** – the average speed during long-distance travel at cruising altitude.
- **Maximum Range (km)** – the maximum distance the aircraft can travel without refueling.
- **Engine Thrust (kN)** – the combined thrust output of all engines under standard conditions.
- **Fuel Capacity (liters)** – the total volume of fuel the aircraft can carry.
- **Passenger Capacity** – the maximum number of passengers the aircraft is designed to accommodate.
- **Landing Distance (m)** – the required runway length for a safe landing under typical conditions.
- **Takeoff Distance (m)** – the required runway length for a safe takeoff.
- **Service Ceiling (m)** – the maximum operational altitude.

These attributes form a high-dimensional feature space that captures a wide range of design trade-offs and engineering choices. The objective of this project is to apply Singular Value Decomposition (SVD) to reduce this feature space to a two-dimensional latent representation. This latent space will then be visualized, with each data point associated with an image of the corresponding aircraft to enhance interpretability and provide visual context to the clustering patterns observed.

7.4.2 Aircraft Specification Table

The following tables present the compiled physical and performance specifications for a range of commercial aircraft, obtained from publicly available manufacturer data and encyclopedic sources [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

Aircraft	MTOW (kg)	Wingspan (m)	Length (m)	Height (m)	Engine Thrust (kN)	Fuel Capacity (L)	Passenger Capacity
Boeing 747-400	396890	64.9	70.6	19.4	276	216840	416
Boeing 777-300ER	299370	60.9	73.9	18.5	415	171176	368
Boeing 787-9 Dreamliner	254000	60.1	62.8	17	280	126206	296
Airbus A380-800	575000	79.8	72.7	24.1	356	320000	555
Airbus A350-900	280000	64.8	66.8	17.1	374	141000	350
McDonnell Douglas DC-10	259500	47.3	55.5	17.7	222	159000	380
Boeing 737-800	79000	35.8	39.5	12.5	121	26020	162
Boeing 737 MAX 8	82190	35.9	39.5	12.3	120	26020	178
Airbus A320 Neo	79000	35.8	37.6	11.8	120	26730	194
Airbus A321	93500	35.8	44.5	11.8	133	30030	220
McDonnell Douglas MD-80	67800	32.9	45.1	9	98	22100	155
Comac C919	72500	35.8	38.9	11.9	120	25000	168
Embraer E175	40370	26.0	31.7	9.8	82	11625	78
Embraer E195-E2	60700	35.1	41.5	10.9	104	12860	146
Bombardier CRJ900	38330	24.9	36.2	7.5	86	8000	90
ATR 72-600	23000	27.1	27.2	7.7	19.6	5000	70
MRJ90	42800	29.2	35.8	10.4	86	12100	92

Table 7.1: Physical parameters of selected commercial aircraft [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

Aircraft	Cruising Speed (km/h)	Landing Distance (m)	Takeoff Distance (m)	Service Ceiling (m)	Maximum Range (km)
Boeing 747-400	912	2179	3018	13700	12200
Boeing 777-300ER	950	1630	3050	13140	11120
Boeing 787-9 Dreamliner	903	1520	2900	13100	14010
Airbus A380-800	945	2900	2900	13100	15200
Airbus A350-900	903	2000	2600	13100	15000
McDonnell Douglas DC-10	908	2000	3100	12800	10600
Boeing 737-800	842	1780	2600	12500	5765
Boeing 737 MAX 8	842	1780	2600	12500	6570
Airbus A320 Neo	828	1500	2100	12100	6300
Airbus A321	828	1500	2100	12100	5950
McDonnell Douglas MD-80	873	1600	2000	11000	3900
Comac C919	834	1800	2000	12000	5555
Embraer E175	871	1280	1800	12500	3700
Embraer E195-E2	871	1500	2000	12500	4815
Bombardier CRJ900	829	1622	1944	12500	2900
ATR 72-600	510	1335	1315	7620	1528
MRJ90	828	1500	2000	12500	3704

Table 7.2: Performance parameters of selected commercial aircraft [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

7.4.3 Data Preprocessing

As mentioned previously, before applying the SVD method, it is needed to preprocess the dataset to ensure that all the data contribute the same to the analysis. Taking into account the different units and magnitudes of each of the features, there's a high chance some will contribute disproportionately and influence the results.

To address this, all features are standardized using **z-score normalization**, transforming each feature to have zero mean and unit variance:

$$z = \frac{x - \mu}{\sigma}$$

where x is the original feature value, μ is the mean, and σ is the standard deviation of the feature across all aircraft.

This standardization process ensures that each feature contributes equally to the principal components derived through SVD.

After preprocessing, the data is ready for dimensionality reduction using SVD, which will be discussed in the following sections.

Figure 7.1 illustrates the impact of normalization on two example features: Maximum Takeoff Weight (MTOW) and Cruising Speed. The left column displays the raw data, while the right one displays the normalized data. As it can be seen, when comparing the MTOW and the cruise speed using raw data, the difference is much higher than comparing them using the normalized scale.

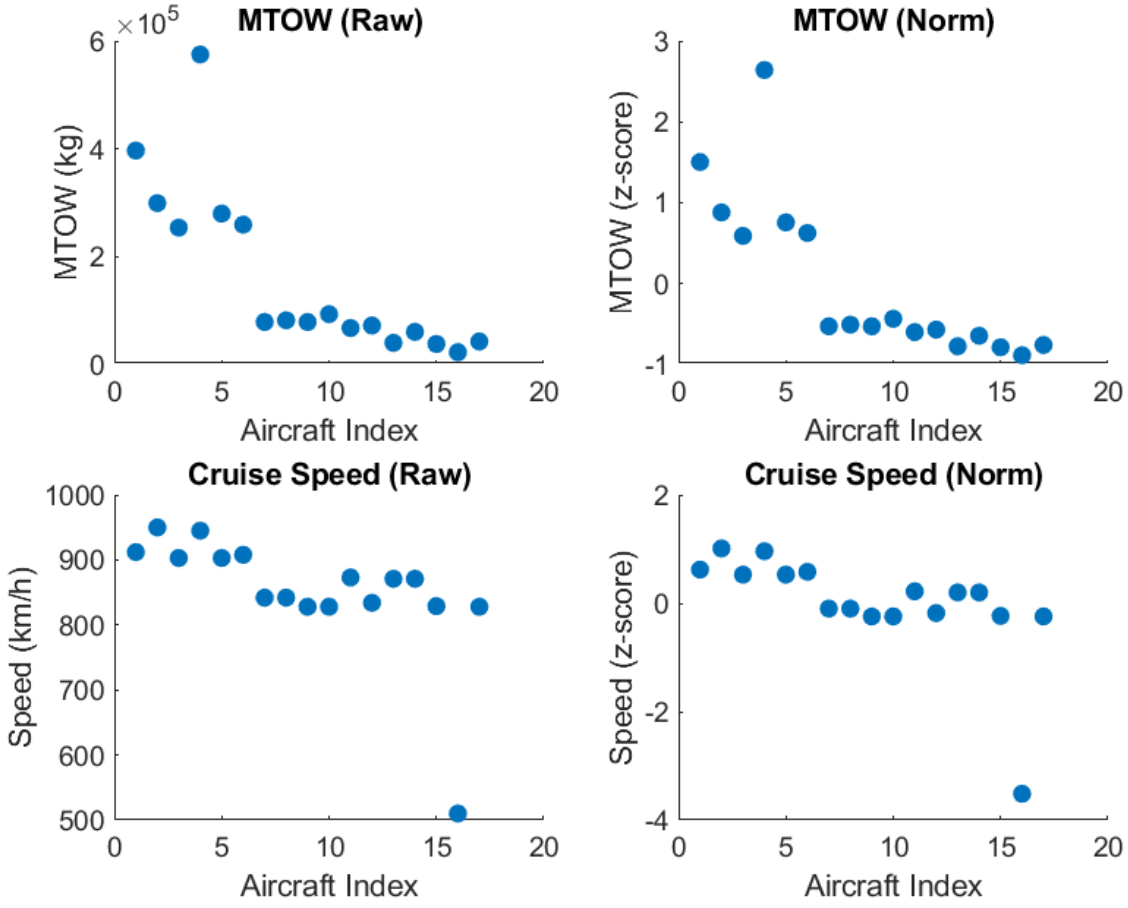


Figure 7.1: Comparison of raw and normalized data for MTOW and Cruising Speed.

7.5 Results and Analysis

The data was projected into a two-dimensional space using Singular Value Decomposition (SVD) to visualize and analyze the structure of the aircraft dataset. Each point in the resulting plot represents one aircraft, labeled by its corresponding index. This transformation helps identify patterns or clusters among aircraft based on their physical and performance characteristics.

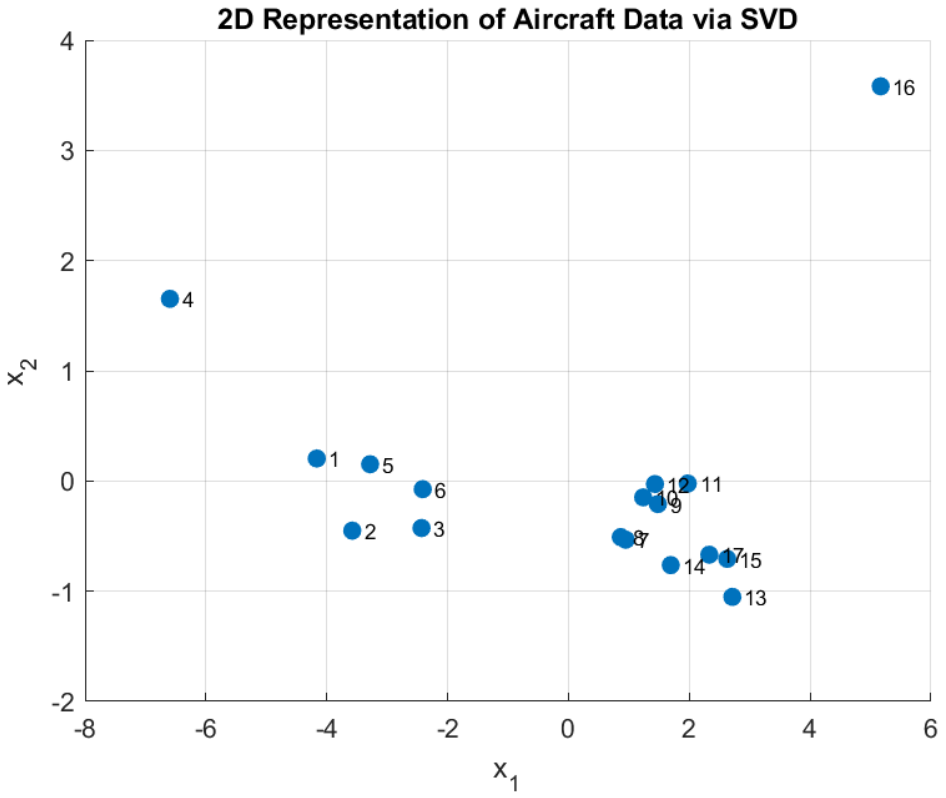


Figure 7.2: 2D projection of normalized aircraft data using SVD. Each point represents an aircraft, labeled by index from the dataset.

Several meaningful clusters can be observed in the 2D projection. Each cluster likely represents aircraft with similar physical and performance characteristics. Below, we analyze the different clusters in more detail.

7.5.1 Cluster 1: Large Long-Haul Aircraft

This cluster includes the following aircraft:

- 1 – Boeing 747-400
- 2 – Boeing 777-300ER
- 3 – Boeing 787-9 Dreamliner
- 5 – Airbus A350-900
- 6 – McDonnell Douglas DC-10

These aircraft share similar roles as long-haul wide-body jets with high MTOW, large passenger capacity, and substantial fuel reserves.

Figure 7.3: Cluster 1 aircraft: B747 [1], B777 [2], B787 [3], A350 [5], and DC-10 [6].

In table 7.3 the main specs of the cluster 1 aircraft are shown:

Aircraft	**MTOW** (kg)	**Wingspan** (m)	**Fuel Cap.** (L)	**Thrust** (kN)	**Passengers**
Boeing 747-400	396,890	64.4	216,840	276	416
Boeing 777-300ER	351,535	64.8	181,280	513	396
Boeing 787-9	254,000	60.1	126,200	320	296
Airbus A350-900	280,000	64.8	141,000	374	300
McDonnell Douglas DC-10	259,500	47.3	149,800	240	270

Table 7.3: Specifications of Cluster 1 Aircraft (Wide-body Long-haul)

7.5.2 Cluster 2: Narrow-Body and Regional Jets

This cluster includes:

- 7 – Boeing 737-800
- 8 – Boeing 737 MAX 8
- 9 – Airbus A320 Neo
- 10 – Airbus A321
- 11 – McDonnell Douglas MD-80
- 12 – Comac C919
- 13 – Embraer E175
- 14 – Embraer E195-E2

- 15 – Bombardier CRJ900
- 17 – MRJ90

These aircraft are generally used for short to medium haul routes and share characteristics such as moderate MTOW and smaller dimensions.

Figure 7.4: Cluster 2 aircraft: Narrow-body and regional jets including B737 [7], B737 MAX [8], A320NEO [9], A321 [10], MD-80 [11], C919 [12], E175 [13], CRJ900 [15], MRJ [17], and E195 [14].

Similarly as before, Table 7.4 shows the specifications for cluster 2 aircraft:

Aircraft	**MTOW** (kg)	**Wingspan** (m)	**Fuel Cap.** (L)	**Thrust** (kN)	**Passengers**
Boeing 737-800	79,010	35.8	26,000	121	189
Boeing 737 MAX 8	82,190	35.9	20,865	121	210
Airbus A320 Neo	79,000	35.8	26,730	120	195
Airbus A321	89,000	35.8	23,700	133	244
McDonnell Douglas MD-80	71,500	32.9	22,100	206	172
Comac C919	75,000	35.8	23,000	118	168
Embraer E175	37,200	26.0	12,971	76	88
Embraer E195-E2	61,500	35.1	13,300	103	132
Bombardier CRJ900	38,328	24.9	9,460	54	90
Mitsubishi MRJ90 (SpaceJet)	39,600	29.2	8,340	71	88

Table 7.4: Specifications of Cluster 2 Aircraft (Narrow-body and Regional Jets)

7.5.3 Cluster 3: Ultra-High Capacity Aircraft

- 4 – Airbus A380-800

The A380 is uniquely positioned due to its unparalleled passenger capacity, wingspan, and MTOW, making it stand out as a single-member cluster.

Figure 7.5: Cluster 3 aircraft: Airbus A380-800 [4].

The specs for this cluster are shown in Table 7.5:

Aircraft	**MTOW** (kg)	**Wingspan** (m)	**Fuel Cap.** (L)	**Thrust** (kN)	**Passengers**
Airbus A380-800	575,000	79.8	320,000	311 ×4	555

Table 7.5: Specifications of Cluster 3 Aircraft (Very Large Aircraft)

7.5.4 Cluster 4: Turboprop Regional Aircraft

- 16 – ATR 72-600

The ATR 72-600 is a turboprop aircraft with significantly lower MTOW and fuel capacity, explaining its separation from jet-powered aircraft.

Figure 7.6: Cluster 4 aircraft: ATR 72-600 [16].

Lastly, the specs for this cluster are shown in Table 7.6:

Aircraft	**MTOW** (kg)	**Wingspan** (m)	**Fuel Cap.** (L)	**Thrust** (kN)	**Passengers**
ATR 72-600	23,000	27.1	5,000	2 × 1,750 shp	78

Table 7.6: Specifications of Cluster 4 Aircraft (Turboprop Regional Aircraft)

7.6 Conclusions

In this assignment, a commercial aircraft dataset was examined through the lens of their physical attributes and performance metrics using Singular Value Decomposition (SVD), a form of dimensionality reduction technique to identify possible clusters. The obtained two-dimensional representation demonstrated four main clusters, which were later studied in the respect of specifications and appearance similarities.

The first cluster comprised of the wide body long haul aircraft: the Boeing 747, 777, 787, optioned by the Airbus A350 and the Mcdonnel Douglas DC-10. This group of aircraft supports long-distance, intercontinental travel using airways due to their extremely high maximum take-off weights (MTOW), large wingspans, high fuel capacity, and enormous number of passenger capacity.

Cluster 2 consisted of narrow body and regional jets such as members of the Boeing 737, Airbus A320, A321, and some Embraer and Bombardier models such as CRJ900. These aircraft are designed to serve short routes and regional connections, characterized by moderate MTOWs, lower fuel capacity, and reduced passenger volume.

And the last remaining cluster which includes the sole representative, and therefore outlier of the dataset the Airbus A380. Extremely large in every sense possible in terms of size, wingspan, and passenger capacity. The A380 is the only full-length double-decker in the set, categorizing it under ultra high capacity international airliners.

This fourth cluster contains the regional Turboprop also known as ATR 72-600 which distinguishes it.

The utility in the SVD as a technique to group aircraft taking into account its capacities and engineering purposes has been proven, making it a powerful tool when it comes to grouping a large number of aircraft and separating them into different categories. This is not only useful in the aerospace industry but in all sectors which may have different categories for a main product, making it easier to later develop new technologies following these clusters as a blank canvas.

7.7 MATLAB Code

The following MATLAB script was used for preprocessing the aircraft dataset, performing normalization, applying SVD for dimensionality reduction, and generating the resulting 2D projection and visualizations.

```
%% NT Final Project
% Matheus Victor Do Prado Amaral

close all
clear
clc

%% Data Preprocessing

opts = detectImportOptions('aircraft_data_csv.csv', 'VariableNamingRule', '
    preserve');
opts = setvartype(opts, 'Aircraft', 'string');  % Treat aircraft names as strings

data = readtable('aircraft_data_csv.csv', opts);

% Extract numerical data only (all columns except 'Aircraft')
X = table2array(data(:, 2:end));

% Normalize the data (zero mean, unit std)
X_norm = (X - mean(X)) ./ std(X);

% Extract MTOW and Cruise Speed
mtow_raw = X(:, 1);              % MTOW (column 1)
cruise_raw = X(:, 8);            % Cruise Speed (column 8)

mtow_norm = X_norm(:, 1);        % Normalized MTOW
cruise_norm = X_norm(:, 8);      % Normalized Cruise Speed

% Index for aircraft (1 to number of aircraft)
n = height(data);
```

```
idx = 1:n;

% Create figure
figure;

% Subplot 1: MTOW (raw)
subplot(2,2,1);
scatter(1:height(data), X(:,1), 'filled');
title('MTOW (Raw)');
xlabel('Aircraft Index');
ylabel('MTOW (kg)');

% Subplot 2: MTOW (normalized)
subplot(2,2,2);
scatter(1:height(data), X_norm(:,1), 'filled');
title('MTOW (Norm)');
xlabel('Aircraft Index');
ylabel('MTOW (z-score)');

% Subplot 3: Cruise Speed (raw)
subplot(2,2,3);
scatter(1:height(data), X(:,8), 'filled');
title('Cruise Speed (Raw)');
xlabel('Aircraft Index');
ylabel('Speed (km/h)');

% Subplot 4: Cruise Speed (normalized)
subplot(2,2,4);
scatter(1:height(data), X_norm(:,8), 'filled');
title('Cruise Speed (Norm)');
xlabel('Aircraft Index');
ylabel('Speed (z-score)');

set(gcf, 'Color', 'w');
exportgraphics(gcf, 'mtow_cruise_comparison.png', 'BackgroundColor', 'white', '
    ContentType', 'image');

%% SVD Method

% Compute SVD
[U, S, V] = svd(X_norm, 'econ');

```

```
% Choose number of components to keep (e.g., 2D projection)
k = 2;

% Project data onto the first k principal directions
Z = U(:, 1:k) * S(1:k, 1:k);  % Z = X_norm * V(:, 1:k);

% Plot the 2D representation
figure;
scatter(Z(:,1), Z(:,2), 50, 'filled');
title('2D Representation of Aircraft Data via SVD');
xlabel('x_1');
ylabel('x_2');
grid on;
hold on;

% Add numeric labels to each point
for i = 1:size(Z, 1)
    text(Z(i,1) + 0.2, Z(i,2), num2str(i), 'FontSize', 8);
end

% Save figure with white background
set(gcf, 'Color', 'w');
exportgraphics(gcf, 'svd_2d_projection.png', 'BackgroundColor', 'white', '
    ContentType', 'image');
```

Listing 7.1: NT Final Project MATLAB Code – Matheus Victor Do Prado Amaral

References

1. *Boeing 747-400*. 2024. Available also from: https://es.wikipedia.org/wiki/Boeing_747-400#Especificaciones.
2. *Boeing 777*. 2024. Available also from: https://es.wikipedia.org/wiki/Boeing_777#Especificaciones.
3. *Boeing 787*. 2024. Available also from: https://www.boeing.com/commercial/787/.
4. *Airbus A380-800*. 2024. Available also from: https://www.airbus.com/en/products-services/commercial-aircraft/passenger-aircraft/a380.
5. *Airbus A350-900*. 2024. Available also from: https://www.airbus.com/en/products-services/commercial-aircraft/passenger-aircraft/a350-family.
6. *McDonnell Douglas DC-10*. 2024. Available also from: https://en.wikipedia.org/wiki/McDonnell_Douglas_DC-10.
7. *Boeing 737 Next Generation*. 2024. Available also from: https://en.wikipedia.org/wiki/Boeing_737_Next_Generation.
8. *Boeing 737 MAX*. 2024. Available also from: https://en.wikipedia.org/wiki/Boeing_737_MAX.
9. *Airbus A320 Neo*. 2024. Available also from: https://aircraft.airbus.com/en/aircraft/a320-family/a320neo.
10. *Airbus A321*. 2024. Available also from: https://en.wikipedia.org/wiki/Airbus_A321.
11. *McDonnell Douglas MD-80*. 2024. Available also from: https://en.wikipedia.org/wiki/McDonnell_Douglas_MD-80.
12. *Comac C919*. 2024. Available also from: https://en.wikipedia.org/wiki/Comac_C919.
13. *Embraer E175*. 2024. Available also from: https://www.embraercommercialaviation.com/wp-content/uploads/2017/02/Embraer_spec_175_web.pdf.
14. *Embraer E195-E2*. 2024. Available also from: https://www.embraercommercialaviation.com/commercial-jets/e195-e2-commercial-jet/.
15. *Bombardier CRJ900*. 2024. Available also from: https://www.flyradius.com/bombardier-crj900/specifications.
16. *ATR 72*. 2024. Available also from: https://en.wikipedia.org/wiki/ATR_72.
17. *Mitsubishi SpaceJet*. 2024. Available also from: https://en.wikipedia.org/wiki/Mitsubishi_SpaceJet.