AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

A Computational Study on Classification of Malignant and Benign Tissue

A Project Report

Submitted by:
Siju K S
CB.AI.R4CEN24003
ASAI

Submitted to:
Prof. (Dr.) Soman K.P.
Professor & Dean
ASAI

In partial fulfillment of the requirements for the course work of the P.hD. programme under Amrita School of Artificial Intelligence

NOVEMBER
2024

CERTIFICATE

This is to certify that the project report titled *"A Computational Study on Classification of Malignant and Benign Tissue"* is the original work of Mr. Siju K. S. and has been completed as part of the Ph.D. coursework at **Amrita School of Artificial Intelligence, Amrita Vishwa Vidyapeetham, Coimbatore**.

This project was conducted under my supervision and guidance, in alignment with the objectives of the doctoral program.

I confirm that this work is a bona fide effort by Mr. Siju K. S., carried out with diligence and in adherence to academic standards as part of his Ph.D. coursework.

Prof. Dr. Soman K. P.
Professor & Dean
Amrita School of Artificial Intelligence
Amrita Vishwa Vidyapeetham,
Coimbatore

DECLARATION

I, Mr. Siju K. S., hereby declare that the project report titled *“A Computational Study on Classification of Malignant and Benign Tissue”* submitted in partial fulfillment of the requirements for my Ph.D. coursework at the *Amrita School of Artificial Intelligence, Amrita Vishwa Vidyapeetham, Coimbatore*, is a record of my original work under the guidance of *Prof. Dr. Soman K. P., Professor & Dean, Amrita School of Artificial Intelligence*.
I further declare that this project has not been submitted, either in part or in full, for any degree, diploma, fellowship, or other similar titles or recognitions in any other institution or university.

Mr. Siju K. S.
Roll.No. CB.AI.R4CEN24003
Date: November 23, 2024

To all my Teachers

अज्ञानतिमिरान्धस्य ज्ञानाञ्जन शलाकया।
चक्षुरुन्मीलितं येन तस्मै श्रीगुरवे नमः॥

Acknowledgments

I would like to extend my sincere thanks to Prof. Dr. Soman K. P., Professor and Dean of Amrita School of Artificial Intelligence, for his unwavering support and encouragement as I begin my doctoral journey. His motivating lectures and the essential textbook, *Machine Learning with SVM and Other Kernel Methods*, have significantly deepened my comprehension of fundamental machine learning algorithms and their real-world applications.

I am especially grateful to my thesis advisor, Dr. Vipin V., whose guidance and support have been invaluable. His encouragement to explore new ideas and approaches in solving real-world problems has been a source of inspiration and growth throughout my research.

My sincere thanks go to my Doctoral Committee Members, Dr. Vinith R., Dr. Mithun Kumar Kar, and Dr. Unnikrishnan P., for their creative suggestions and continued encouragement, which have significantly shaped this work. I am also thankful to Dr. V. Soumya, Amrita School of Artificial Intelligence, for her support during my academic journey.

I extend my appreciation to Dr. Sudha T., Principal, Dr. M.D. Mathew, Dean Research, and Dr. Lekha Susan Jacob, Head of the Department of Mathematics, Saintgits College of Engineering (Autonomous), for their timely help and support, enabling me to pursue this coursework at Amrita Vishwa Vidyapeetham.

Lastly, I am grateful to my friends and colleagues at Amrita—Mr. Vipin Das, Mrs. Vandana Padmakumar, Mrs. Kavitha K., Mrs. Vrinda Alukkkal and all other research scholars at CEN—who provided camaraderie and support.

To everyone who has contributed to this project, thank you for your guidance and support in making this initial phase of my doctoral studies a rewarding experience.

Siju K S

Abstract

Breast cancer remains one of the leading causes of cancer-related deaths globally, with over 2.3 million new cases reported in 2020 alone. The urgency for timely and accurate diagnosis is critical, as early intervention significantly improves patient outcomes. This study enhances breast cancer prediction by integrating classification and clustering techniques using the UCI breast cancer dataset. Classification models—including linear regression, logistic regression, and support vector machines (SVM)—are initially applied to differentiate between benign and malignant cases. The SVM is formulated as a convex optimization problem and solved using MATLAB's CVX solvers.

Subsequently, K-means clustering is employed to reveal data patterns, comparing cluster assignments with actual diagnoses to identify misclassified cases and gain insights into diagnostic accuracy. This combination of approaches not only boosts predictive power but also enhances interpretability, offering a clearer understanding of patient classifications.

The findings highlight the synergy between machine learning and traditional diagnostic methods, paving the way for more informed and transparent healthcare solutions.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Traditional Global Breast Cancer Screening and Detection Methods . 1

1.3 Aims and Objectives 2

1.4 Various Stages in the Study 2

2 Literature Survey **5**

3 Methodology **9**

3.1 Dataset Summary 9

3.2 Basic Data Analysis 10

3.2.1 Class Distribution 10

3.2.2 Summary Statistics 11

3.2.3 Correlation Analysis 12

3.2.4 Multicollinearity of feature 14

3.2.5 Data Preprocessing 15

3.2.6 IQR-Based Approach for Outlier Detection 16

3.2.7 Data Visualization 17

3.3 Feature Selection 18

3.4 Machine Learning Algorithms 20

3.5 Terminologies Used in Machine Learning Model Development 21

3.6 Logistic Regression Classifier 23

3.7 Support Vector Machine (SVM) Classifier 29

3.7.1 Advantages of Convex Optimization in SVM 31

3.8 Decision Tree Classifier 35

3.8.1 Lagrangian Formulation of Decision Trees 38

3.9 Random Kitchen Sink (RKS) Kernel in Soft Margin SVM Classifier . . 39

3.10 Built-in Machine Learning Functions in MATLAB 40

3.10.1 Logistic Regression using `fitglm` 41

3.10.2 Support Vector Machines using `fitcsvm` 41

3.10.3 Decision Trees using `fitctree` 41

3.10.4 Comparative Evaluation and Tuning of Models 42

3.11 Towards an Explainable and Sustainable Tumor Classification Model 42

3.12 Tumor Classification Using K-Means Clustering 43

3.12.1 Classical K-Means Clustering Approach 43

3.12.2 Limitations of the Classical Approach 44

3.12.3 Transition to Lagrangian Multiplier Model 44
3.12.4 Optimization Model Solved with CVX 45
3.12.5 Evaluation of Clustering: Silhouette Score 46
3.13 Conclusion 46

4 Results and Discussions 49
4.1 Introduction 49
4.2 Skill of Logistic Regression Classifier 49
4.2.1 Performance Metrics Analysis 53
4.2.2 Overview of Performance Metrics 53
4.3 Skill of Support Vector Machines 56
4.3.1 Hard Margin Support Vector Machine with Linear Kernel . . . 56
4.3.2 Soft Margin Support Vector Machine with Linear Kernel . . . 57
4.3.3 Soft Margin Support Vector Machine with Non-linear Kernels 58
4.4 Skill of Decision Tree Algorithm 61
4.5 Skill of Soft Margin SVM with RKS Kernels 64
4.6 Model Comparison and Conclusion 66
4.6.1 Model Performance Analysis and Recommendations 66
4.7 Unsupervised Approach for Explainable Models 67
4.7.1 Experiment Interpretation 67
4.8 Conclusion 69
4.9 Future Directions and Sustainable Impact 70

5 Conclusions 71

List of Tables

2.1 Summary of Machine Learning Studies on Breast Cancer Dataset . . 8

3.1 Class distribution of the UCI Breast Cancer Wisconsin dataset. 10
3.2 Summary Statistics of Numerical Features 11
3.3 Correlation Coefficients of Features with Diagnosis 13
3.4 Variation Inflation Factor (VIF) Scores of Features 15
3.5 Outlier Status Before and After Square Root Transformation 17
3.6 Correlation Analysis of Features with Tumor Malignancy 19
3.7 Selected features with correlation coefficients and their target association 20

4.1 Performance Comparison of Linear and Logistic Regressors for Binary Classification 52
4.2 Performance metrics of logistic regression classifier across different feature subsets 53
4.3 Comparison of performance metrics of various regularized models and K-fold cross validation 54
4.4 comparison of model performance of logistic regression classifier. . . 55
4.5 Comparison of performance metrics of soft margin SVM with RBF kernel 59
4.6 Statistical summary of 5-fold cross validation results 60
4.7 61
4.8 Performance Metrics for Decision Tree Model using `fitctree()`. . . . 62
4.9 Performance Metrics for Decision Tree Model 62
4.10 Comparison of dominant features identified from two approaches. . . 63
4.11 Comparison of performance metrics of soft margin SVM with RKS kernel 65
4.12 Performance Metrics of Models 66
4.13 Distribution of Benign and Malignant cases over clusters. 69

List of Figures

2.1 Author keywords co-occurrence network of breast cancer prediction . 6

3.1 Correlation matrix of various features in the dataset 12
3.2 Distribution of top three highly correlated features 14
3.3 Box plot showing the distribution of features post-square root transformation, highlighting potential outliers. 18
3.4 Machine learning classification process 21

4.1 Distribution of dominant features across the target variable (malignant and benign) before preprocessing. 50
4.2 Distribution of dominant features after outlier removal and scaling. . 51
4.3 Separating planes and data distribution in linear and logistic regression 52
4.4 Skill of hard and soft margin linear SVM 57
4.5 Skill of soft margin linear SVM with RBF kernel 58
4.6 Post-pruned decision tree at level five. 63
4.7 Skill of soft margin SVM 65
4.8 Skill of K-means clustering model. 68

Chapter 1

Introduction

1.1 Background

Breast cancer remains a significant global health challenge, with millions of cases reported annually, underscoring the critical need for advanced diagnostic tools that facilitate early detection and timely treatment. Malignant BC cells spread to other areas of the body if they are not detected in time, and patients frequently have to undergo more involved, invasive therapies. Lately diagnosed patients frequently have worse survival rates and may pass away shortly after receiving their diagnosis.
By 2050, the landscape of breast cancer is projected to change significantly. The incidence rates for women are expected to rise to 59.63 cases per 100,000, marking a substantial 32.13% increase from 2019. For men, the incidence is anticipated to be much lower at 0.65 per 100,000, which is a modest increase of 1.74%.
In terms of mortality, forecasts indicate that the death rate for women will see a slight uptick, reaching 16.42 per 100,000, which represents a 4.69% increase since 2019. Conversely, the male breast cancer mortality rate is projected to decline significantly to 0.26 per 100,000, a drop of 19.84%.
These statistics suggest that by 2050, the global burden of breast cancer could result in 4.78 million diagnosed cases, leading to around 1.5 million deaths, predominantly affecting women. This highlights the urgent need for continued awareness, research, and support for those impacted by breast cancer. [1]. In this study Xu et.al underscores the pressing global need for comprehensive cancer prevention and control strategies, including reducing exposure, early screening, and improving treatment, to effectively decrease the global burden of breast cancer.

1.2 Traditional Global Breast Cancer Screening and Detection Methods

Depending on the resources available, different countries have different methods for screening for breast cancer (BC). While industrialized nations often rely on mammography, developing nations frequently employ clinical breast examinations (CBE) and breast self-examinations (BSE). The ”gold standard” for detecting breast

cancer is generally accepted to be the triple assessment test, which combines CBE, radiological imaging (mammography and/or ultrasonography), and pathology (FNAC or core needle biopsy). If at least one of these tests reveals malignancy, a BC diagnosis is verified. Despite these methods, breast cancer incidence and mortality remain significant. Conventional methods for diagnosing diseases such as breast cancer typically utilize a linear model known as regression [2, 3]. These regression techniques operate under the assumption of a direct linear correlation among various risk factors for breast cancer; however, such relationships are often non-linear. Consequently, reliance on regression could lead to inaccuracies in breast cancer detection.
Conversely, machine learning methods do not necessitate linear relationships and may provide a superior option for identifying breast cancer. Machine learning has introduced innovative techniques for analyzing complex datasets, offering promising avenues for enhancing classification tasks in the medical field.

1.3 Aims and Objectives

This project centers on the development of predictive and prescriptive models using the UCI Breast Cancer Wisconsin dataset, which includes a diverse set of features relevant to breast cancer diagnosis.

1.4 Various Stages in the Study

The primary objective of this study is to leverage both supervised and unsupervised learning methods to improve breast cancer diagnosis. To achieve this, the research process begins with comprehensive data preparation. The dataset is first structured as a matrix, enabling essential preprocessing steps, including data cleaning, outlier detection, and feature scaling. Outliers are detected using the Interquartile Range (IQR) method, and visualizations are employed to provide a detailed summary of the dataset's numerical features. This step ensures that the data is refined for robust model development and accurate analysis.
Following preprocessing, the study advances to the implementation of supervised classification models aimed at predicting the likelihood of a sample being benign or malignant. The analysis commences with linear and logistic regression models, utilizing matrix operations and the sigmoid function for binary classification. The methodology then extends to Support Vector Machines (SVM), with the model formulated as a convex optimization problem solved via eigenvalue decomposition of the dual Lagrangian function, and further optimized using MATLAB's CVX solvers. These models are evaluated based on accuracy, sensitivity, specificity, and F1-score, identifying the model best suited for medical diagnosis.
After establishing reliable classification models, the research explores K-means clustering as an unsupervised approach to uncover additional data patterns. Clustering serves as a complementary tool to the classification models by examining the structure of the dataset, comparing cluster assignments to actual diagnoses.

This analysis provides insight into potential misclassifications and helps interpret the distribution of cases across clusters, offering a more nuanced understanding of breast cancer diagnoses. The clustering results supplement the classification models, making the predictions more interpretable and enabling a prescriptive approach that may aid clinicians in determining the severity and treatment prioritization for patients.
This project aims to not only enhance diagnostic accuracy but also contribute to explainable machine learning in medical diagnostics. By adopting a methodical approach grounded in linear algebra and optimization combining classification and clustering techniques, this research seeks to explore the potential for improved diagnostic accuracy in breast cancer detection, ultimately contributing to advancements in medical analytics and patient care.

Chapter 2

Literature Survey

The application of machine learning (ML) in breast cancer prediction has grown significantly, especially with studies employing the Wisconsin Breast Cancer Diagnostic (WBCD) dataset since early 2020. A key study by Street et al. used interactive image processing and machine learning to diagnose breast cancer, implementing a technique known as “snakes,” or deformable splines, which identified cell nuclei in fine needle aspirate (FNA) images. Through this method, ten nuclear features, including radius, area, and compactness, were extracted and used in a classifier based on the Multi-surface Method (MSM). This approach achieved an accuracy of 97% for distinguishing between benign and malignant tumors using key features like worst area and mean texture [4]. This study demonstrated the feasibility of nuclear feature extraction combined with ML for high-accuracy breast cancer diagnosis. Another approach discussed is the Rule Induction Algorithm based on approximate classification, achieving an accuracy of 94.99% [5]. This method generates a set of rules from the training data that can be used to classify new instances. The rules are typically expressed in the form of “if-then” statements. Combining Linear Discriminant Analysis (LDA) with Neural Networks (NN) is yet another method explored in the source. This combined approach reached an impressive accuracy of 96.8% [6]. LDA seeks to find a linear combination of features that maximizes the separation between classes, while neural networks are powerful models inspired by the structure of the human brain that can learn complex non-linear relationships in the data.

Zaki’s comprehensive bibliometric analysis (shown in Table 2.1)highlights this trend, mapping the diverse ML algorithms applied in breast cancer research, including Bayesian networks, Radial Basis Function (RBF) networks, Support Vector Machines (SVM), and Decision Trees (DT) [7]. Early work primarily explored these algorithms individually to assess predictive capabilities, providing a foundation for the current use of ensemble and hybrid models that yield improved accuracy. The research keyword analysis is represented as a network diagram as shown in Figure 2.1.

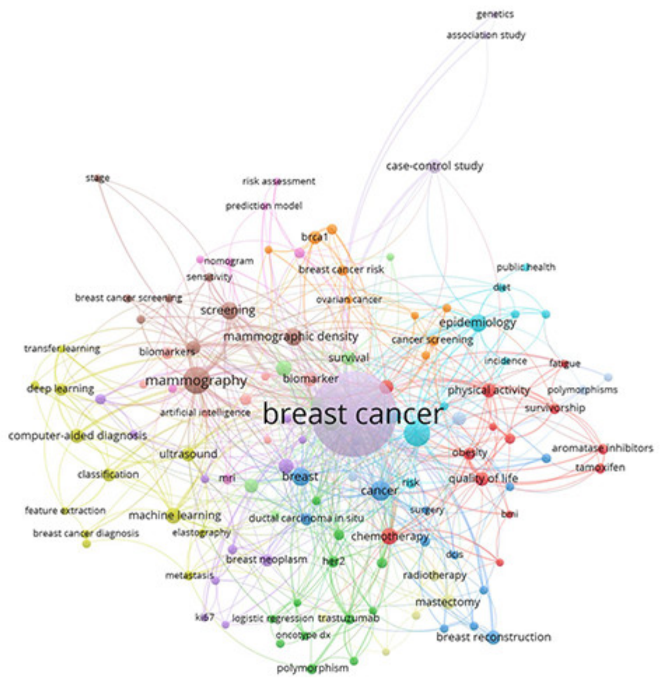


Figure 2.1: Author keywords co-occurrence network of breast cancer prediction using machine learning-related publications from 2015 to 2019.

Since 2019, researchers have expanded their approaches beyond the WBCD dataset, incorporating various datasets like the Digital Database for Screening Mammography (DDSM), the Federal Fluminense University Hospital Mammography Dataset, and data from Zhejiang Cancer Hospital. This variety has facilitated the testing of a broad range of models—from classical algorithms like Logistic Regression (LR) and k-Nearest Neighbors (KNN) to advanced Convolutional Neural Networks (CNNs) and multilayer perceptron (MLP) networks. SVM has emerged as a highly effective approach, achieving accuracies up to 97.2% on the WBCD dataset, underscoring its robustness in cancer classification tasks [8] [9].
Khairunnahar et.al mentions the use of Decision Tree methods, specifically the C4.5 algorithm, which attained an accuracy of 94.74% [10]. The latest studies have introduced innovative techniques to improve breast cancer prediction accuracy and model transparency. For instance, the BOAALO hybrid feature selection method significantly enhances diagnostic reliability, while two-phase deep learning models incorporating random forests boost prognosis prediction, achieving a 5.1% increase in survival sensitivity estimation [11]. Some studies also emphasize early detection via polygenic risk scores (PRSs) tailored for Asian populations, demonstrating the importance of population-specific calibrations in ML models [12].
Alongside predictive models, recent research emphasizes Explainable AI (XAI) frameworks to enhance interpretability and facilitate clinical acceptance. A comprehensive review of XAI literature confirms the growing priority for transparency and model explainability, which are crucial for real-world applicability [13]. Many studies apply classification matrices across key factors relevant to breast cancer, benchmarking model effectiveness and robustness. Additionally,

recent efforts to incorporate stacking-based ensemble learning and rigorous hyperparameter tuning have improved both predictive accuracy and computational speed, demonstrating potential for deployment in clinical settings [14]. A recent 2022 study developed a digital breast tomosynthesis (DBT) risk model specifically aimed at predicting late-stage and interval breast cancers that may arise after an initial negative screening. This model shows promise in supporting clinical decision-making by enhancing early identification and prioritizing high-risk cases, which could improve patient outcomes through timely intervention [15].

In addition to prediction, the proposed study contributes by proposing a prescriptive ML model capable of assessing breast cancer severity, which enables prioritizing cases for immediate treatment and guiding follow-up diagnostic procedures. This model extends beyond simple classification, offering a multi-level assessment to support clinicians in determining optimal treatment pathways based on cancer severity. This prioritization capability addresses a critical gap in current research, moving ML applications from mere diagnostic aids toward active clinical decision-making tools that integrate prediction and treatment guidance. By offering both predictive and prescriptive insights, this study aims to enhance the utility of ML in breast cancer care and facilitate more efficient, patient-centered outcomes on a global scale.

Table 2.1: Summary of Machine Learning Studies on Breast Cancer Dataset

Reference	**Dataset**	**Country***	**Sampling strategy**	**ML Algorithm**	**Summary measure (in %)**
Hernández-Julio et al.[16]	BCCD	Colombia	10-fold CV	Clusters + pivot table	95.90 (Accuracy)
Singh(2019)[17]	BCCD	India	67-33 training-testing	K-NN	92.11 (Accuracy)
Polat and Senturk (2018)[18]	BCCD	Turkey	10-fold CV	AdaBoost	91.37 (Accuracy)
Akben (2019) [19]	BCCD	Turkey	10-fold CV	DT	90.52 (Accuracy)
Islam and Poly (2019) [20]	BCCD	Taiwan (China)	10-fold CV	K-NN	86.00 (Accuracy)
Araújo et al. (2019) [21]	BCCD	Brazil	70-30 training-testing, 10-fold CV	NN	80.67 (Accuracy)
Aslan et al. (2018) [22]	BCCD	Turkey	80-20 training-testing	ELM	80.00 (Accuracy)
Livieris (2018) [23]	BCCD	Greece	10-fold CV	K-NN	62.00 (Accuracy)
Patrício et al.(2018) [24]	BCCD	Portugal	MCCV	SVM	87.00, 91.00 (95% CI for AUC)
Li and Chen[25]	BCCD	United Kingdom	70-30 training-testing	RF	78.50 (AUC)
Hung et al.(2018) [26]	BCCD	Vietnam	80-20 training-testing	DT	82.00 (F1 score)
Abdar and Makarenkov (2019) [27]	WBCD	Canada	50-50 training-testing	CWV-BANN-SVM	100.00 (Accuracy)
Elgedawy (2017) [28]	WBCD	Saudi Arabia	75-25 training-testing	RF	99.42 (Accuracy)
Hernández-Julio et al. (2019) [16]	WBCD	Colombia	10-fold CV	Clusters + pivot table	99.40 (Accuracy)
Chaurasia et al. (2017) [29]	WBCD	India	Stratified 10-fold CV	NB	97.36 (Accuracy)
Asri et al.(2016) [30]	WBCD	Morocco	10-fold CV	SVM	97.13 (Accuracy)
Alzubaidi et al. (2016) [31]	WBCD	United Kingdom	LOOCV	SVM (quadratic-linear kernel), K-NN	97.00 (Accuracy), 97.00 (Accuracy)
Islam et al. (2017) [32]	WBCD	Bangladesh	10-fold CV	SVM	97.00 (Accuracy)
Chaurasia and Pal (2018) [33]	WBCD	India	10-fold CV	SMO (SVM)	96.20 (Accuracy)
Bazazeh and Shubair[34]	WBCD	United Arab Emirates	10-fold CV	RF	99.90 (AUC)
Li and Chen[35]	WBCD	United Kingdom	70-30 training-testing	RF	98.90 (AUC)
Anastraj et al. (2021) [36]	WBCD	India	80-20 training-testing	BN,RBF	97.42(Accuracy)

Chapter 3

Methodology

This chapter delineates the methodological framework employed in this project, focusing on the UCI Breast Cancer Wisconsin dataset as the basis for model development. Initially, an exploration of the dataset was conducted to understand its structure and key characteristics. Essential statistical analyses were performed to summarize the data, including identifying outliers and assessing feature relationships. Following this, data preprocessing techniques were applied to ensure the dataset's quality, including normalization and outlier detection. Subsequently, a series of classification models—namely linear regression, logistic regression, and support vector machines—were implemented using matrix operations grounded in linear algebra and optimization principles. This systematic approach not only facilitated the development of robust predictive models but also enhanced the overall understanding of the data's patterns and distributions.

3.1 Dataset Summary

The UCI Breast Cancer Wisconsin dataset, sourced from the University of California, Irvine (UCI) Machine Learning Repository, is a widely used dataset for breast cancer diagnosis research. It comprises 569 instances, each representing a unique patient, along with 32 attributes that provide critical information regarding tumor characteristics. The dataset is structured as follows:

- **ID**: A unique identifier for each patient.
- **Diagnosis**: A categorical variable indicating the tumor classification, with two possible values:
  - `M`: Malignant (cancerous)
  - `B`: Benign (non-cancerous)
- **Features**: The dataset contains 30 continuous numerical attributes, derived from digitized images of fine needle aspirate (FNA)[1] of breast mass. These features represent various measurements related to the tumors, including:

[1]Fine Needle Aspiration (FNA) is a minimally invasive technique for collecting cytological samples

  - `radius_mean`, `texture_mean`, `perimeter_mean`, `area_mean`, `smoothness_mean`, and others.

  Each feature is calculated based on the mean, standard error, or worst (largest) value, offering a comprehensive view of the tumor's characteristics.

- **Data Format**: The dataset is provided in CSV format, making it easily accessible for data analysis and modeling tasks.

The primary objective of this dataset is to aid in the development of machine learning models capable of accurately predicting the diagnosis of breast cancer based on the provided features. This dataset has become a benchmark for evaluating various classification algorithms and serves as an essential resource for researchers and practitioners in the field of medical diagnostics.

3.2 Basic Data Analysis

This section outlines the primary analyses conducted on the UCI Breast Cancer Wisconsin dataset to understand its characteristics and prepare for model development.

3.2.1 Class Distribution

The class distribution of the 'diagnosis' variable was examined, categorizing instances as either malignant (M) or benign (B).

Class	Count
Benign (B)	357
Malignant (M)	212

Table 3.1: Class distribution of the UCI Breast Cancer Wisconsin dataset.

The class distribution table shows 357 instances classified as benign (B) and 212 as malignant (M) in the UCI Breast Cancer Wisconsin dataset. This results in an approximate distribution of 62.7% benign and 37.3% malignant instances.
The observed class imbalance may influence the performance of classification algorithms, potentially leading to a bias towards the benign class. Therefore, it is essential to implement strategies during model development to ensure reliable identification of malignant cases. The lower representation of malignant instances necessitates the use of effective feature extraction and validation techniques to enhance predictive accuracy.

from suspicious breast lesions using a thin needle under imaging guidance. This procedure provides rapid diagnoses and high-quality samples, facilitating the extraction of critical features for differentiating between benign and malignant cells, thereby enhancing cancer prediction models.

3.2.2 Summary Statistics

Summary statistics, including mean, median, standard deviation, minimum, maximum, and interquartile range (IQR), were calculated for each numerical feature. These statistics provide insights into the dataset's central tendencies and variations.

Table 3.2: Summary Statistics of Numerical Features

Feature	Mean	Median	Q1	Q2	Q3	IQR	Min	Max	SD
radius_mean	14.13	13.37	11.70	13.37	15.80	4.10	6.98	28.11	3.52
texture_mean	19.29	18.84	16.17	18.84	21.80	5.63	9.71	39.28	4.30
perimeter_mean	91.97	86.24	75.13	86.24	104.15	29.02	43.79	188.50	24.30
area_mean	654.89	551.10	420.18	551.10	784.15	363.98	143.50	2501.00	351.91
smoothness_mean	0.10	0.10	0.09	0.10	0.11	0.02	0.05	0.16	0.01
compactness_mean	0.10	0.09	0.06	0.09	0.13	0.07	0.02	0.35	0.05
concavity_mean	0.09	0.06	0.03	0.06	0.13	0.10	0.00	0.43	0.08
concave_points_mean	0.05	0.03	0.02	0.03	0.07	0.05	0.00	0.20	0.04
symmetry_mean	0.18	0.18	0.16	0.18	0.20	0.03	0.11	0.30	0.03
fractal_dimension_mean	0.06	0.06	0.06	0.06	0.07	0.01	0.05	0.10	0.01
radius_se	0.41	0.32	0.23	0.32	0.48	0.25	0.11	2.87	0.28
texture_se	1.22	1.11	0.83	1.11	1.47	0.64	0.36	4.88	0.55
perimeter_se	2.87	2.29	1.61	2.29	3.36	1.76	0.76	21.98	2.02
area_se	40.34	24.53	17.85	24.53	45.24	27.39	6.80	542.20	45.49
smoothness_se	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.03	0.00
compactness_se	0.03	0.02	0.01	0.02	0.03	0.02	0.00	0.14	0.02
concavity_se	0.03	0.03	0.02	0.03	0.04	0.03	0.00	0.40	0.03
concave_points_se	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.05	0.01
symmetry_se	0.02	0.02	0.02	0.02	0.02	0.01	0.01	0.08	0.01
fractal_dimension_se	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00
radius_worst	16.27	14.97	13.01	14.97	18.79	5.78	7.93	36.04	4.83
texture_worst	25.68	25.41	21.07	25.41	29.76	8.68	12.02	49.54	6.15
perimeter_worst	107.26	97.66	84.10	97.66	125.53	41.42	50.41	251.20	33.60
area_worst	880.58	686.50	514.97	686.50	1085.00	570.03	185.20	4254.00	569.36
smoothness_worst	0.13	0.13	0.12	0.13	0.15	0.03	0.07	0.22	0.02
compactness_worst	0.25	0.21	0.15	0.21	0.34	0.19	0.03	1.06	0.16
concavity_worst	0.27	0.23	0.11	0.23	0.38	0.27	0.00	1.25	0.21
concave_points_worst	0.11	0.10	0.06	0.10	0.16	0.10	0.00	0.29	0.07
symmetry_worst	0.29	0.28	0.25	0.28	0.32	0.07	0.16	0.66	0.06
fractal_dimension_worst	0.08	0.08	0.07	0.08	0.09	0.02	0.06	0.21	0.02

The summary statistics table provides an overview of the numerical features extracted from the dataset.
- **Mean** values indicate the average size and characteristics of the breast tumors, with the highest average observed for the *area* feature at **654.89**. The *perimeter* and *radius* also show considerable averages of **91.97** and **14.13**, respectively, reflecting the size and dimensions of the tumors.
- **Median** values closely follow the means, confirming the general distribution without extreme outliers. For instance, the median radius is **13.37**, suggesting that half the tumors have a radius smaller than this value.
- **Interquartile range (IQR)** highlights the variability, with the *area* feature exhibiting the highest IQR of **363.98**, indicating substantial differences in tumor sizes.
- The **minimum** and **maximum** values indicate the range of each feature. For instance, the *area* varies from **143.50** to **2501.00**, demonstrating significant size diversity among the tumors.
- The **standard deviation (SD)** values reflect the spread of the data points around

the mean, with features such as *perimeter* (SD = **24.30**) and *area* (SD = **351.91**) exhibiting higher variability compared to features like *smoothness_mean* (SD = **0.01**), which is more consistent across observations.
These statistics are crucial for understanding the dataset's characteristics and inform the subsequent steps in feature selection and model training.

3.2.3 Correlation Analysis

The Pearson correlation coefficient was computed for pairs of numerical variables to identify strong correlations. A correlation matrix was created to visualize relationships between features, guiding feature selection for classification. Figure 3.1 illustrate the correlation between the 30 features in the dataset

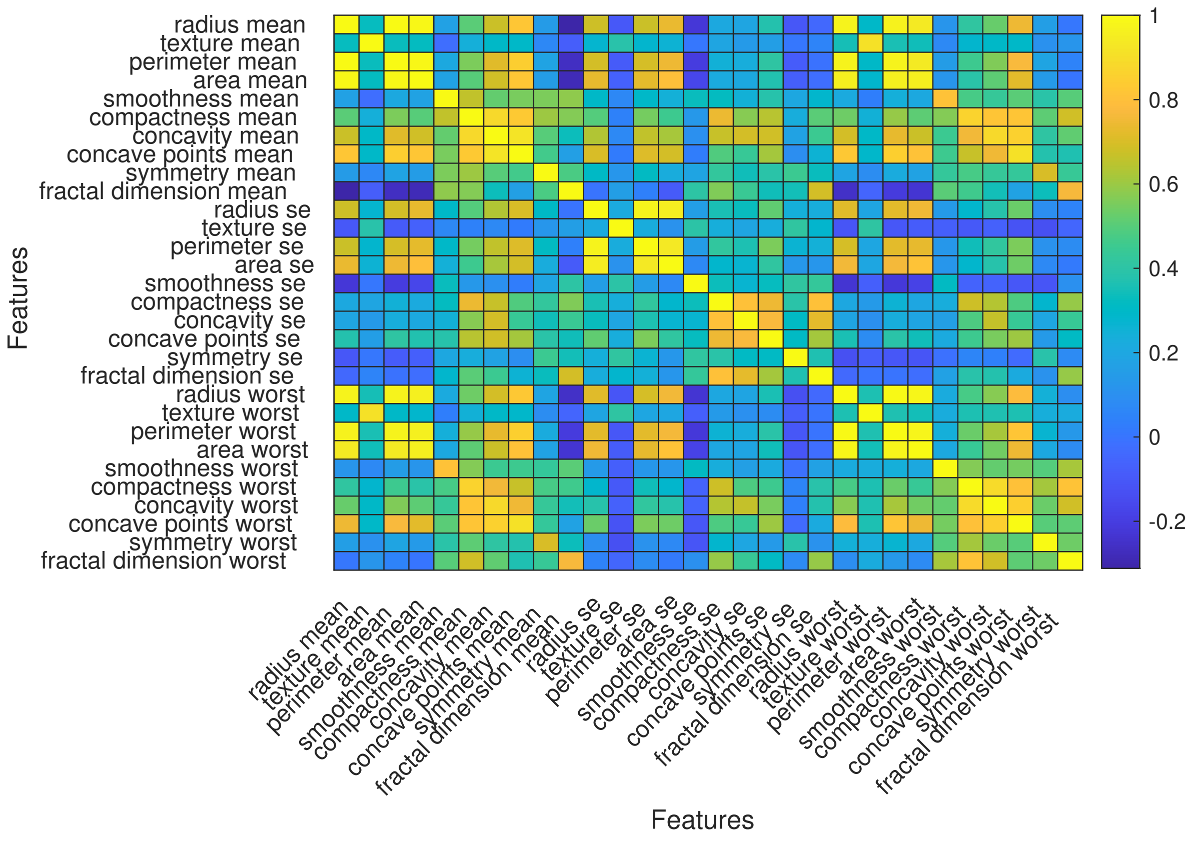


Figure 3.1: Correlation matrix of various features in the dataset

In order to identify the most significant features contributing to breast cancer diagnosis, a correlation analysis was conducted between each feature and the target variable (diagnosis). This statistical method allows us to quantify the strength of the linear relationship between the input features and the target class, which in this case helps to identify the features most strongly associated with distinguishing benign from malignant tumors.
The table below presents the correlation coefficients for each feature:

Table 3.3: Correlation Coefficients of Features with Diagnosis

Feature	Correlation
radius mean	0.73
texture mean	0.42
perimeter mean	0.74
area mean	0.71
smoothness mean	0.36
compactness mean	0.60
concavity mean	0.70
concave points mean	0.78
symmetry mean	0.33
fractal dimension mean	-0.01
radius se	0.57
texture se	-0.01
perimeter se	0.56
area se	0.55
smoothness se	-0.07
compactness se	0.29
concavity se	0.25
concave points se	0.41
symmetry se	-0.01
fractal dimension se	0.08
radius worst	0.78
texture worst	0.46
perimeter worst	0.78
area worst	0.73
smoothness worst	0.42
compactness worst	0.59
concavity worst	0.66
concave points worst	0.79
symmetry worst	0.42
fractal dimension worst	0.32

Based on the correlation analysis, the three features most strongly correlated with the target variable (diagnosis) are **'concave points worst'**, **'perimeter worst'**, and **'concave points mean'**. These features have correlation values of 0.79, 0.78, and 0.78, respectively. The high correlation values indicate a strong positive relationship with the diagnosis, suggesting that larger values of these features are typically associated with malignant tumors.

Statistical interpretations of these results highlight that shape-related features, particularly those associated with concave points and perimeter, are critical in differentiating between benign and malignant tumors. These findings provide strong justification for focusing on these features in predictive modeling efforts. Visualizing the distribution of these features across benign and malignant classes further demonstrates their significance in separating the two groups. Distribution of

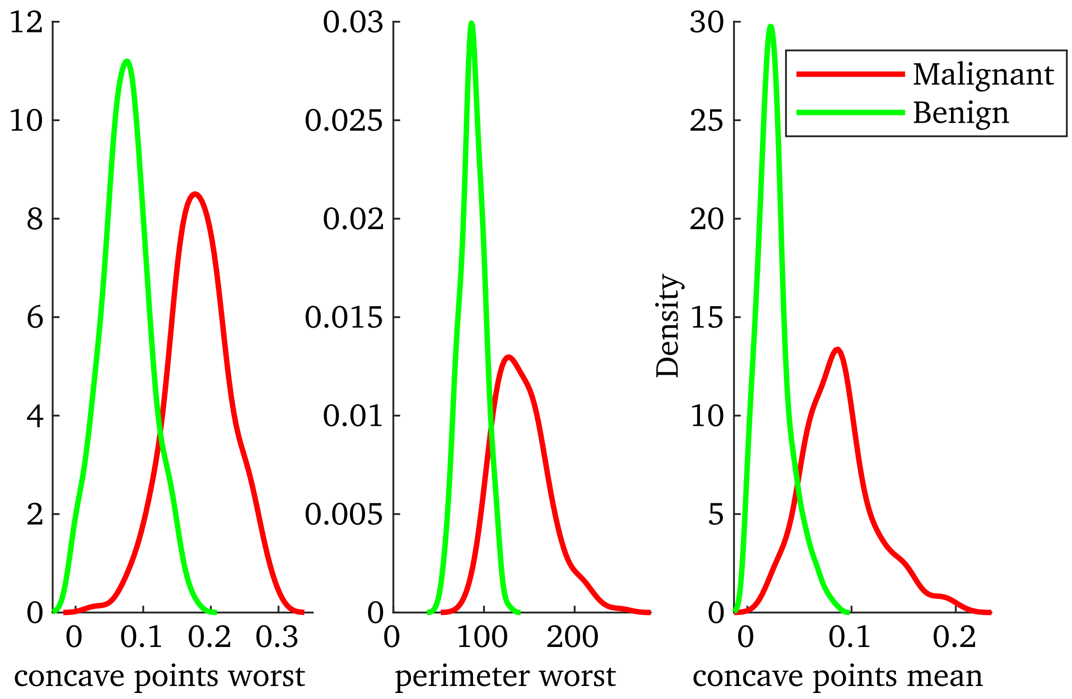


Figure 3.2: Distribution of top three highly correlated features

these three top features are shown in Figure 3.2.
Varition Inflation Factor score of the feature set is shown in Table 3.4.

3.2.4 Multicollinearity of feature

The Variation Inflation Factor (VIF) is a critical metric for evaluating multicollinearity among predictor variables, particularly in the context of linear regression and its extensions to classification models. A VIF score quantifies how much the variance of a regression coefficient is inflated due to linear dependence among the predictors. In general, a VIF score above 5 or 10 indicates problematic multicollinearity, warranting attention and potential remedial action.
Table 3.4 of VIF scores for various features reveals significant disparities, with several features exhibiting alarmingly high values. For instance, the VIF score for `radius mean` (3806.12) and `perimeter mean` (3786.40) suggests extreme multicollinearity, indicating that these variables are heavily interrelated. Such high VIF values raise concerns about the stability of coefficient estimates and their interpretability in predictive modeling.
Conversely, features like `symmetry mean` (4.22) and `fractal dimension mean` (15.76) present a mixed picture. While `symmetry mean` indicates acceptable multicollinearity levels, the higher score of `fractal dimension mean` underscores a need for careful consideration in model design. Features with high VIF scores can obscure the unique contributions of individual predictors, potentially leading to overfitting and unreliable model performance.
Given these observations, reliance on linear models— especially logistic

regression— may be inappropriate due to the instability caused by multicollinearity. To enhance model robustness, practitioners should consider techniques such as feature selection to mitigate multicollinearity's adverse effects or explore regularization methods to achieve more reliable predictive outcomes. Thus, addressing the issues highlighted by the VIF scores will be crucial in constructing an effective and interpretable classification model.

Table 3.4: Variation Inflation Factor (VIF) Scores of Features

Sl.No.	Feature	VIF Score
1	radius mean	3806.12
2	texture mean	11.88
3	perimeter mean	3786.40
4	area mean	347.88
5	smoothness mean	8.19
6	compactness mean	50.51
7	concavity mean	70.77
8	concave points mean	60.04
9	symmetry mean	4.22
10	fractal dimension mean	15.76
11	radius se	75.46
12	texture se	4.21
13	perimeter se	70.36
14	area se	41.16
15	smoothness se	4.03
16	compactness se	15.37
17	concavity se	15.69
18	concave points se	11.52
19	symmetry se	5.18
20	fractal dimension se	9.72
21	radius worst	799.11
22	texture worst	18.57
23	perimeter worst	405.02
24	area worst	337.22
25	smoothness worst	10.92
26	compactness worst	36.98
27	concavity worst	31.97
28	concave points worst	36.76
29	symmetry worst	9.52
30	fractal dimension worst	18.86

3.2.5 Data Preprocessing

Based on the analysis findings, preprocessing steps were taken to prepare the dataset for modeling. This included addressing missing values, normalizing features, and handling outliers to enhance data quality.

3.2.6 IQR-Based Approach for Outlier Detection

The Interquartile Range (IQR) is a measure of statistical dispersion, which is the spread of the middle 50% of a dataset. It is calculated as the difference between the third quartile ($Q3$) and the first quartile ($Q1$), where:

$$\text{IQR} = Q3 - Q1$$

The first quartile ($Q1$) represents the 25th percentile of the data, while the third quartile ($Q3$) represents the 75th percentile. The IQR is particularly useful for identifying outliers in datasets, as it is resistant to extreme values (unlike the standard deviation). Outliers are typically defined as data points that fall below $Q1 - 1.5 \times \text{IQR}$ or above $Q3 + 1.5 \times \text{IQR}$. These thresholds, often referred to as ”fences,” capture most of the central data, and any points outside this range are considered outliers.

The IQR-based approach is widely used in datasets where the distribution is skewed or does not follow a normal distribution, making it more robust compared to the z-score method. The use of 1.5 times the IQR to determine outliers is a common rule of thumb, although this factor can be adjusted based on the specific characteristics of the data. By identifying and potentially removing or investigating these outliers, it is possible to improve the accuracy and performance of statistical models and reduce bias introduced by extreme values.

- Lower Bound: $Q1 - 1.5 \times \text{IQR}$

- Upper Bound: $Q3 + 1.5 \times \text{IQR}$

Data points outside this range are flagged as potential outliers.

There are many outliers in almost all the features in the dataset. A square root transformation is used on features with terrible number of outliers (area se, perimeter se, radius se,area mean, area worst, fractal dimension worst). Result of the outlier removal is shown in Table 3.5.

Table 3.5: Outlier Status Before and After Square Root Transformation

Feature	Outliers Before	Outliers After
Radius Mean	13	0
Texture Mean	7	0
Perimeter Mean	13	0
Area Mean	23	12
Smoothness Mean	6	0
Compactness Mean	15	0
Concavity Mean	17	0
Concave Points Mean	9	0
Symmetry Mean	15	0
Fractal Dimension Mean	15	0
Radius SE	36	19
Texture SE	18	0
Perimeter SE	36	21
Area SE	61	30
Smoothness SE	28	0
Compactness SE	26	0
Concavity SE	22	0
Concave Points SE	18	0
Symmetry SE	27	0
Fractal Dimension SE	26	0
Radius Worst	16	0
Texture Worst	4	0
Perimeter Worst	13	0
Area Worst	31	15
Smoothness Worst	4	0
Compactness Worst	16	0
Concavity Worst	11	0
Concave Points Worst	0	0
Symmetry Worst	21	0
Fractal Dimension Worst	21	14

3.2.7 Data Visualization

To analyze the impact of the square root transformation and identify potential outliers, a box plot was created for the selected features. The box plot (Figure 3.3) illustrates the distribution of features, including area_se, perimeter_se, radius_se, area_mean, area_worst, and fractal_dimension_worst.
In the box plot, the central box represents the interquartile range (IQR), with the line indicating the median. Whiskers extend to the smallest and largest values within 1.5 times the IQR, while points outside this range are marked as potential outliers.
The analysis reveals notable outliers in features such as area_se, perimeter_se, and radius_se, necessitating further investigation. This visualization aids in understanding the effects of the square root transformation and guides subsequent

data preprocessing steps.

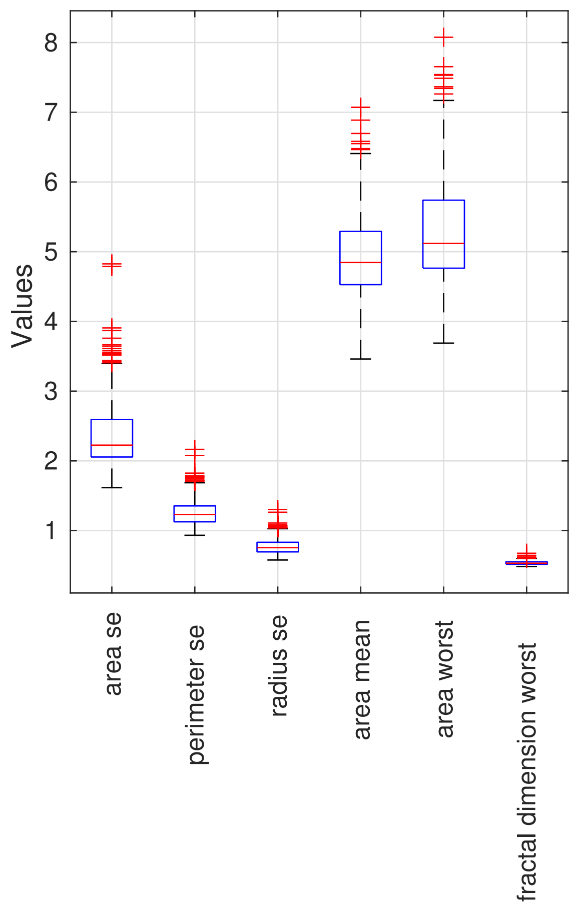


Figure 3.3: Box plot showing the distribution of features post-square root transformation, highlighting potential outliers.

3.3 Feature Selection

The correlation analysis reveals several features with significant positive correlations to tumor malignancy, particularly those exceeding a correlation coefficient of 0.75. Notable features include *Concave Points Worst* (0.79), *Area Worst* (0.78), and *Perimeter Worst* (0.78), all of which exhibit strong relationships with malignancy. These features, primarily related to tumor size and shape, are crucial in distinguishing between malignant and benign tumors. The low p-values (e.g., 1.97×10^{-124} for *Concave Points Worst*) further affirm their statistical significance.

Selecting these high-correlation features for model development enhances predictive accuracy in classifying tumors. Their clinical relevance, derived from strong statistical associations, supports their role in diagnostic processes. Thus, incorporating these features into logistic regression models can potentially improve classification outcomes, aiding in timely and accurate patient treatment decisions.

Table 3.6: Correlation Analysis of Features with Tumor Malignancy

Feature	ρ	P-Value
Concave Points Worst	0.79	1.97×10^{-124}
Area Worst	0.78	1.23×10^{-119}
Perimeter Worst	0.78	5.77×10^{-119}
Concave Points Mean	0.78	7.10×10^{-116}
Radius Worst	0.78	8.48×10^{-116}
Perimeter Mean	0.74	8.44×10^{-101}
Area Mean	0.73	3.45×10^{-97}
Radius Mean	0.73	8.47×10^{-96}
Area SE	0.71	9.52×10^{-89}
Concavity Mean	0.70	9.97×10^{-84}
Concavity Worst	0.66	2.46×10^{-72}
Perimeter SE	0.63	3.20×10^{-64}
Radius SE	0.63	1.77×10^{-63}
Compactness Mean	0.60	3.94×10^{-56}
Compactness Worst	0.59	7.07×10^{-55}
Texture Worst	0.46	1.08×10^{-30}
Smoothness Worst	0.42	6.58×10^{-26}
Symmetry Worst	0.42	2.95×10^{-25}
Texture Mean	0.42	4.06×10^{-25}
Concave Points SE	0.41	3.07×10^{-24}
Smoothness Mean	0.36	1.05×10^{-18}
Symmetry Mean	0.33	5.73×10^{-16}
Fractal Dimension Worst	0.32	2.47×10^{-15}
Compactness SE	0.29	9.98×10^{-13}
Concavity SE	0.25	8.26×10^{-10}
Fractal Dimension SE	0.08	0.063
Smoothness SE	-0.07	0.110
Fractal Dimension Mean	-0.01	0.760
Texture SE	-0.01	0.843
Symmetry SE	-0.01	0.877

Chi-square tests are commonly used for evaluating the independence between categorical variables or for assessing goodness of fit between observed and expected distributions. In particular, testing independence can help determine whether two or more variables are dependent across populations, allowing one to estimate the other. However, when applying Chi-square tests to this dataset, the results were inconclusive, likely due to the continuous nature of the transformed variables and the limitations of the Chi-square method in handling such data. These experiments consistently produced unreliable test statistics and p-values, highlighting the inadequacy of Chi-square for this feature selection task.

As a more suitable alternative, Pearson correlation analysis was employed to measure the linear relationship between the numerical features and the binary target variable. This method identified features with the strongest associations, where top features such as ”Concave Points Worst,” ”Area Worst,” ”Perimeter Worst”, ”Concave Points Mean” and ”Radius Worst” showed high correlation values (above 0.75). These features are derived from basic measurements like radius, area, and perimeter but encapsulate more complex geometric properties of the tumors. Thus, the correlation analysis not only simplifies the feature selection process but also highlights the significance of derived features, making it an effective choice for this dataset. Selected features for developing machine learning models for the breast cancer data set is shown in Table 3.7.

Table 3.7: Selected features with correlation coefficients and their target association

Feature	Correlation	P-value
Concave Points Worst	0.79	0
Area Worst	0.78	0
Perimeter Worst	0.78	0
Concave Points Mean	0.78	0
Radius Worst	0.78	0

3.4 Machine Learning Algorithms

Machine Learning (ML) algorithms, particularly supervised learning methods, are widely applied in predictive modeling. Among them, **Logistic Regression**, **Support Vector Machines (SVMs)**, **Decision Trees**, and **Random Forests** are commonly used for classification tasks.

Logistic Regression is a statistical method that models the probability of a binary outcome based on one or more input features. It is effective for problems where the relationship between features and the target variable is approximately linear. Its simplicity, interpretability, and ability to provide probabilistic outputs make it a popular choice for binary classification.

Support Vector Machines (SVMs) work by finding a hyperplane that best separates the data into different classes. SVMs can handle both linear and non-linear classification tasks, using kernel functions to transform the data when needed. They are particularly useful when the data is not linearly separable and perform well in high-dimensional spaces.

Decision Trees are flowchart-like structures where decisions are made at each node, based on feature values. They are easy to interpret and can handle both categorical and numerical data. However, decision trees can be prone to overfitting, especially when deep trees are built, which capture noise rather than underlying patterns.

Random Forests improve on decision trees by using an ensemble approach. Multiple decision trees are built on random subsets of the data, and the final prediction is made based on the majority vote of these trees. This method reduces overfitting,

improves generalization, and typically yields better accuracy than single decision trees.
In this project, these algorithms can be used to model the relationship between selected features and the target variable, offering robust performance across a range of classification problems.
A general structure of a Machine Learning Classification process is shown in Figure 3.4.

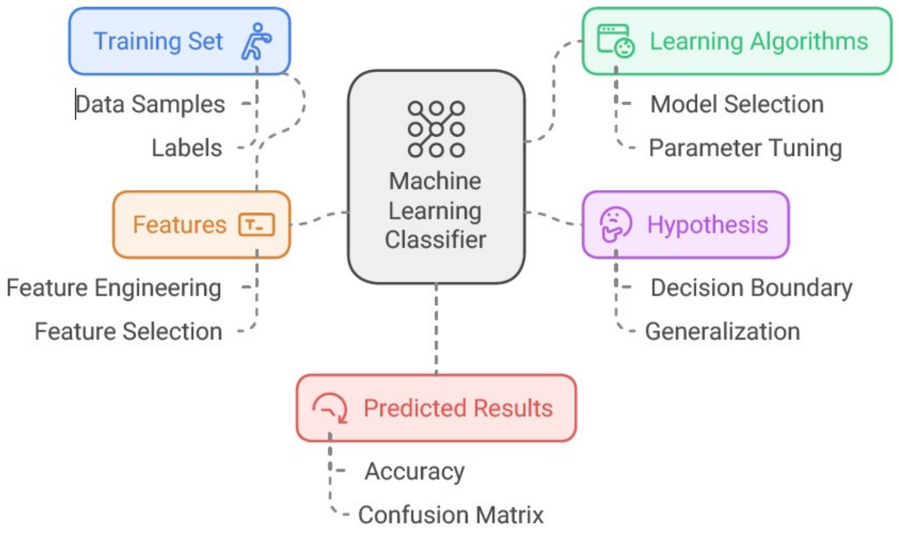


Figure 3.4: Machine learning classification process

The selection of these algorithms for the current project is based on their ability to handle both linear and non-linear relationships within the dataset. Logistic Regression offers a simple yet powerful baseline, while SVMs can efficiently manage more complex patterns. Decision Trees provide interpretability, allowing for easier understanding of feature importance, and Random Forests enhance model performance through ensemble learning, reducing the risk of overfitting. These algorithms collectively offer a robust toolkit for accurately classifying the data and handling the nuances of feature interaction in the project.

3.5 Terminologies Used in Machine Learning Model Development

Dataset

A dataset is a collection of data that contains features (input variables) and labels (target variable). In supervised learning, the dataset is used to train the model, with features representing the input data and labels indicating the desired output.

Train-Test Split

Train-test split is a technique used to evaluate the performance of a machine learning model. The dataset is divided into two subsets: the training set, which is used to train the model, and the test set, which is used to assess the model's performance on unseen data. A common split ratio is 80:20, where 80% of the data is used for training and 20% for testing.

Weights (w) and Bias (b)

Weights are coefficients assigned to each feature in the model, determining the influence of each feature on the prediction. The bias term is a constant added to the output of the model to adjust the prediction independently of the input features. Together, weights and bias form the parameters of the logistic regression model.

Learning Rate (α)

The learning rate is a hyperparameter that controls how much to change the model parameters during each iteration of gradient descent. A smaller learning rate may lead to slower convergence, while a larger learning rate can result in overshooting the optimal solution.

Iterations (T)

Iterations refer to the number of times the gradient descent algorithm updates the weights and bias. More iterations can improve model performance, but excessively high values may lead to overfitting or unnecessary computation.

Gradient Descent

Gradient descent is an optimization algorithm used to minimize the cost function by iteratively adjusting the model parameters (weights and bias) in the direction of the negative gradient of the cost function. This process continues until convergence is achieved or a predetermined number of iterations is reached.

K-Fold Cross-Validation

K-fold cross-validation is a technique used to assess the performance of a model by splitting the training data into k subsets (folds). The model is trained on $k - 1$ folds and validated on the remaining fold. This process is repeated k times, with each fold used as the validation set once. The results are then averaged to provide a more reliable estimate of model performance.

Validation Error

Validation error is the measure of how well a machine learning model performs on unseen data during the validation phase. It provides insight into the model's generalization capability and is crucial for detecting overfitting.

Performance Metrics

Performance metrics are quantitative measures used to evaluate the effectiveness of a machine learning model. Common metrics for classification tasks include accuracy, precision, recall, and F1-score, which provide insights into the model's predictive capabilities.
In assessing the skill of a logistic regression classifier, several performance measures are crucial for a comprehensive evaluation. **Accuracy** reflects the overall correctness of the model's predictions, but it can be misleading in imbalanced datasets. **Sensitivity** (or Recall) measures the model's ability to correctly identify positive instances, making it essential in situations where missing positive cases is costly (e.g., detecting diseases). **Specificity** assesses the ability to correctly classify negative instances, which is important in avoiding false positives. The **Area Under the ROC Curve (AUC-ROC)** provides a more holistic measure by summarizing the trade-off between sensitivity and specificity across different thresholds. A higher AUC indicates that the model performs well in distinguishing between positive and negative classes. Together, these metrics provide insights into the model's strengths and weaknesses, helping assess how well it generalizes and handles different types of classification errors.

3.6 Logistic Regression Classifier

The **binary logistic regression model** is employed to predict a binary response based on one or more predictor variables (features). Logistic regression assesses the relationship between a categorical dependent variable and one or more independent variables by estimating probabilities through a logistic function, which represents the cumulative logistic distribution. The term ”regression” signifies that we are fitting a linear model to the feature space, which can consist of both categorical and continuous variables. Logistic regression adopts a probabilistic approach to classification, providing a means to model the likelihood of the outcome being one of the two categories.

Linear Regression Model as a Starting Point

Logistic regression extends the principles of linear regression, where the objective is to predict a continuous outcome y as a linear combination of input features X:

$$y = X\beta + \epsilon$$

Here, $X \in \mathbb{R}^{n \times p}$ is the matrix of feature vectors (with n samples and p features), $\beta \in \mathbb{R}^p$ is the vector of model parameters (coefficients), and ϵ denotes the error term. For a new observation x_i, the predicted output is:

$$\hat{y}_i = x_i^\top \beta$$

However, in binary classification, predicting a continuous value is not suitable. Instead, we need to transform the output into a range between 0 and 1 to represent probabilities.

From Probability to Odds and Log-Odds

In logistic regression, we model the probability that the output y_i equals 1 as follows:

$$P(y_i = 1|x_i) = \sigma(x_i^\top \beta)$$

where $\sigma(z)$ is defined as the sigmoid function:

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

This function ensures that the output is constrained between 0 and 1, reflecting the probability of the outcome being 1.
The odds of an event occurring is defined as the ratio of the probability of the event to the probability of the event not occurring:

$$\text{Odds}(y_i = 1|x_i) = \frac{P(y_i = 1|x_i)}{1 - P(y_i = 1|x_i)} = \frac{\sigma(x_i^\top \beta)}{1 - \sigma(x_i^\top \beta)}$$

To derive this, we first express $1 - P(y_i = 1|x_i)$:

$$1 - P(y_i = 1|x_i) = 1 - \sigma(x_i^\top \beta) = 1 - \frac{1}{1 + e^{-x_i^\top \beta}} = \frac{e^{-x_i^\top \beta}}{1 + e^{-x_i^\top \beta}}$$

Thus, the odds become:

$$\text{Odds}(y_i = 1|x_i) = \frac{\sigma(x_i^\top \beta)}{1 - \sigma(x_i^\top \beta)} = \frac{\frac{1}{1+e^{-x_i^\top \beta}}}{\frac{e^{-x_i^\top \beta}}{1+e^{-x_i^\top \beta}}} = e^{x_i^\top \beta}$$

Taking the natural logarithm of the odds yields the log-odds or logit:

$$\log\left(\frac{P(y_i = 1|x_i)}{1 - P(y_i = 1|x_i)}\right) = x_i^\top \beta$$

Thus, logistic regression models the log-odds of the probability of a binary outcome as a linear function of the input features.

Logistic Regression Model

For all n observations, the model can be expressed in matrix form:

$$\hat{y} = \sigma(X\beta)$$

where $X \in \mathbb{R}^{n \times p}$ is the matrix of feature vectors, $\beta \in \mathbb{R}^p$ is the parameter vector, and $\hat{y} \in [0, 1]^n$ represents the predicted probabilities.

Loss Function: Maximum Likelihood Estimation (MLE)

To estimate the parameters β, logistic regression utilizes **Maximum Likelihood Estimation (MLE)**. The likelihood function, derived from the probabilities, is defined as:

$$L(\beta) = \prod_{i=1}^{n} P(y_i | x_i; \beta)$$

The log-likelihood function is easier to optimize and is given by:

$$\ell(\beta) = \sum_{i=1}^{n} \left[y_i \log(\sigma(x_i^\top \beta)) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

In matrix form, the log-likelihood function can be represented as:

$$\ell(\beta) = y^\top \log(\sigma(X\beta)) + (1 - y)^\top \log(1 - \sigma(X\beta))$$

Negative Log-Likelihood (Loss Function)

To convert the maximization problem of the log-likelihood into a minimization problem, we consider the negative log-likelihood:

$$\mathcal{L}(\beta) = -\ell(\beta) = -\sum_{i=1}^{n} \left[y_i \log(\sigma(x_i^\top \beta)) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

Optimization Problem

The optimization problem can be stated as:

$$\beta^* = \arg\min_{\beta} \mathcal{L}(\beta)$$

Matrix Formulation

In matrix form, if y is the vector of outcomes and X is the design matrix of features, the negative log-likelihood can be expressed as:

$$\mathcal{L}(\beta) = -\left(y^\top \log(\sigma(X\beta)) + (1 - y)^\top \log(1 - \sigma(X\beta)) \right)$$

Gradient and Optimization

To maximize the log-likelihood function, optimization techniques such as **gradient descent** are applied, given the non-linearity of the function. The gradient of the log-likelihood with respect to β is computed as follows:

$$\nabla_\beta \ell(\beta) = X^\top (y - \sigma(X\beta))$$

This gradient is utilized in iterative algorithms to update the parameter vector β.

Closed-form Solution and Iterative Methods

Unlike linear regression, logistic regression lacks a closed-form solution due to the non-linearity introduced by the sigmoid function. Therefore, iterative methods such as gradient descent, stochastic gradient descent, or the Newton-Raphson method (known as **Iteratively Reweighted Least Squares (IRLS)** in logistic regression) are employed for parameter estimation.
1. **Gradient Descent** updates the parameters using:

$$\beta_{t+1} = \beta_t + \alpha \nabla_\beta \ell(\beta)$$

where α is the learning rate.
2. **Newton-Raphson** employs the Hessian matrix of second derivatives for parameter updates:

$$\beta_{t+1} = \beta_t - H^{-1} \nabla_\beta \ell(\beta)$$

where H represents the Hessian matrix, reflecting the curvature of the log-likelihood function. The sigmoid function $\sigma(z)$ is vital in logistic regression. It converts the output of the linear model, $x_i^\top \beta$, into a probability within the range of [0, 1]. This transformation allows logistic regression to effectively predict binary outcomes. Additionally, the derivative of the sigmoid function, $\sigma(z)(1 - \sigma(z))$, guarantees that the log-likelihood is a concave function, facilitating efficient optimization through gradient-based methods.

Definition of the Separating Plane

The separating plane in logistic regression is a hyperplane that distinguishes between two classes in a feature space. In a binary classification problem, this hyperplane is determined based on the estimated probabilities of the logistic function, which maps linear combinations of the input features to values between 0 and 1.

Mathematical Representation

Logistic regression models the probability that the dependent variable y equals 1 (the positive class) given a set of independent variables x. The model can be expressed as:

$$P(y_i = 1 \mid x_i) = \sigma(w^T x_i + b)$$

where:

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

is the logistic (sigmoid) function.

- w is the vector of weights (coefficients) for the features.
- b is the bias (intercept) term.
- x_i is the feature vector for the i-th observation.

Separating Hyperplane

The decision boundary, or separating plane, is where the probability is exactly 0.5. Therefore, we set the probability equal to 0.5:

$$\sigma(w^T x + b) = 0.5$$

To find this boundary, we can simplify this equation:
The logistic function equals 0.5 when its argument is zero:

$$w^T x + b = 0$$

Rearranging gives us the equation of the hyperplane:

$$w^T x = -b$$

Interpretation of the separating plane

Classification: For any observation x:

- If $w^T x + b > 0$, the predicted class is 1 (positive class).
- If $w^T x + b < 0$, the predicted class is 0 (negative class).

In a two-dimensional feature space, this separating plane is simply a line, and in three dimensions, it becomes a plane. In higher dimensions, it remains a hyperplane. Algorithm for *Logistic Regression* is given in Algorithm 1.

Algorithm 1 Logistic Regression with Train-Test Split and K-Fold Cross Validation

```
 1: **Input:** Dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, learning rate $\alpha$, number of iterations $T$,
    number of folds $k$
 2: **Output:** Trained model parameters $w$, $b$
 3: **Step 1:** Train-Test Split
 4: Split the dataset $D$ into training set $D_{train}$ and test set $D_{test}$ with ratio 80:20.
 5: Let $X_{train}, y_{train}$ be the training features and labels.
 6: Let $X_{test}, y_{test}$ be the testing features and labels.
 7: **Step 2:** Initialize weights $w = 0$ and bias $b = 0$
 8: **Step 3:** Gradient Descent on Logistic Regression
 9: **for** each iteration $t = 1, 2, \ldots, T$ **do**
10:     Compute the linear combination: $z^{(i)} = w^T x^{(i)} + b$
11:     Apply the sigmoid function: $h_\theta(x^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$
12:     Calculate gradients:
            $$\frac{\partial J(w,b)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$
            $$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$
13:     Update the parameters:
            $$w_j = w_j - \alpha \cdot \frac{\partial J(w,b)}{\partial w_j}, \quad b = b - \alpha \cdot \frac{\partial J(w,b)}{\partial b}$$
14: **end for**
15: **Step 4:** K-Fold Cross-Validation
16: Split the training data $D_{train}$ into $k$ folds.
17: **for** each fold $i = 1, 2, \ldots, k$ **do**
18:     Use the $i$-th fold as the validation set and the rest as the training set.
19:     Train the logistic regression model using Gradient Descent on the training
    set.
20:     Compute validation error and store it.
21: **end for**
22: Average the validation errors across $k$ folds to estimate the model's performance.
23: **Step 5:** Evaluate on Test Set
24: Compute predictions on the test set $X_{test}$ using the final model parameters $w$ and
    $b$.
25: Calculate the accuracy or other performance metrics on $y_{test}$.
26: **Return:** Trained model parameters $w$ and $b$.
```

Detailed discussion on the implementation, findings and results of Logistic regression classifier on the selected dataset will be given in the results and discussion chapter (Chapter 4).

3.7 Support Vector Machine (SVM) Classifier

Support Vector Machines (SVM) are widely used for classification tasks in machine learning. The SVM classifier aims to find the optimal separating hyperplane between two classes, which maximizes the margin between them. This problem can be formulated as a convex optimization problem using Lagrange multipliers and solved using both primal and dual methods [37].

Primal Problem Formulation

The primal optimization problem for an SVM is defined as follows:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to:

$$y_i(w^T x_i + b) \geq 1, \quad \forall i = 1, 2, \dots, n$$

where $x_i \in \mathbb{R}^d$ are the input vectors, $y_i \in \{-1, 1\}$ are the class labels, $w \in \mathbb{R}^d$ is the weight vector, and $b \in \mathbb{R}$ is the bias.
This optimization problem can also be written in matrix form:

$$\min_{w,b} \frac{1}{2} w^T w$$

subject to:

$$Y(Xw + \mathbf{b}) \geq \mathbf{1}$$

where $X \in \mathbb{R}^{n \times d}$ is the matrix of input vectors, $Y \in \mathbb{R}^{n \times n}$ is the diagonal matrix of labels, and $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones.

Lagrangian for the Primal Problem

The primal problem can be solved using the Lagrangian method. Define the Lagrangian function:

$$L(w, b, \lambda) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \lambda_i \left[y_i(w^T x_i + b) - 1\right]$$

where $\lambda_i \geq 0$ are the Lagrange multipliers.
The optimal solution satisfies the following KKT conditions:

- $\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \lambda_i y_i x_i = 0$
- $\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \lambda_i y_i = 0$
- $\lambda_i \left[y_i(w^T x_i + b) - 1\right] = 0, \quad \lambda_i \geq 0$

From the first condition, we derive:

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

In matrix form, this becomes:

$$w = X^T \Lambda y$$

where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ is the diagonal matrix of Lagrange multipliers.

Dual Problem Formulation

Substituting the expression for w into the Lagrangian, we eliminate w and b, leading to the dual problem:

$$\min_{\lambda} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \lambda_i$$

subject to the constraints:

$$\sum_{i=1}^{n} \lambda_i y_i = 0, \quad \lambda_i \geq 0$$

In matrix form, the dual problem is written as:

$$\min \lambda \frac{1}{2} \lambda^T (Y X X^T Y) \lambda - \mathbf{I}^T \lambda$$

where $Y \in \mathbb{R}^{n \times n}$ is the diagonal matrix of labels and $\lambda \in \mathbb{R}^n$ is the vector of Lagrange multipliers.

Solving the SVM Dual Problem

The dual problem is a quadratic programming (QP) problem, which can be solved using numerical methods. Once the optimal λ is obtained, the weight vector w is computed as:

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

The bias term b is computed from the support vectors (instances for which $\lambda_i > 0$).

Decision Function

The decision function for a new input x is given by:

$$f(x) = w^T x + b = \sum_{i=1}^{n} \lambda_i y_i x_i^T x + b$$

The predicted class label $\hat{y}$ is:

$$\hat{y} = \text{sign}(f(x))$$

Kernel Trick for Nonlinear SVM

In the case of non-linear decision boundaries, the kernel trick can be used to map the input data into a higher-dimensional space. The dual problem becomes:

$$\min \lambda \frac{1}{2}\lambda^T(YKY)\lambda - \mathbf{I}^T\lambda$$

where $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel function.
The SVM classifier can be formulated as a convex optimization problem, which can be solved using the Lagrange multipliers method. Both the primal and dual problems have well-defined solutions, and the dual formulation provides insights into the role of support vectors and the kernel trick for handling non-linear classification tasks.

3.7.1 Advantages of Convex Optimization in SVM

Convex optimization offers several significant advantages when applied to SVM classification:

- **Global Optimum**: Convex problems guarantee that any local minimum is the global minimum. This is crucial in SVM, ensuring that the optimal separating hyperplane is found without getting trapped in local minima.
- **Efficiency with Quadratic Programs**: The optimization problems in SVM (both primal and dual) are quadratic programs, which are convex by nature. Efficient algorithms such as interior-point methods and active-set methods can be used to solve these quadratic convex optimization problems.
- **Handling of Constraints**: Convex optimization frameworks like `CVX` can easily handle constraints (both equality and inequality), which are naturally imposed in SVM formulations. These include the non-negativity of the Lagrange multipliers and margin constraints.
- **Kernel Methods**: The dual formulation of SVM allows the use of kernel functions, enabling classification in high-dimensional spaces without explicitly computing the coordinates. Convex optimization helps solve these non-linear problems efficiently.

Convex Optimization and the Lagrange Function in SVM

The SVM optimization problem naturally leads to a convex optimization framework. In the primal form, the goal is to maximize the margin between two classes, which results in a convex optimization problem due to the quadratic nature of the margin constraint.
The *Lagrange function* is used to incorporate these constraints into the objective function. By transforming the problem into its dual form, we solve for the Lagrange multipliers, which are constrained to be non-negative. The convexity of the dual problem ensures it can be solved efficiently using convex optimization techniques.

Benefits of Solving the Dual Formulation of SVM

The dual formulation of SVM provides several advantages over the primal formulation:

- **Identification of Support Vectors**: The solution to the dual problem provides the Lagrange multipliers, λ_i, associated with each data point. Only the data points with non-zero values of λ_i lie on the margin, and these are called *support vectors*.
- **Efficiency with Kernels**: The dual formulation allows the use of kernel functions, which enable SVM to handle non-linearly separable data by implicitly mapping data points to higher-dimensional spaces.
- **Regularization**: In the dual form of soft margin SVM, the regularization parameter C is naturally incorporated as an upper bound on the Lagrange multipliers. This helps balance the trade-off between maximizing the margin and minimizing the classification error.

CVX Syntax for Solving Dual SVM Problems in MATLAB

The `CVX` toolbox in MATLAB can be used to solve both the hard margin and soft margin SVM dual problems using convex optimization. Below is the CVX syntax for each case.

Hard Margin SVM in MATLAB

The following code solves the dual problem for a hard margin SVM using `CVX`:

```
% Inputs:
% K: Kernel matrix (n x n) where K(i,j) = K(x_i, x_j)
% y: Labels vector (n x 1), y_i in {-1, +1}
% n: Number of data points

cvx_begin
    variable lambda(n)
    minimize( 0.5 * quad_form(lambda .* y, K) - sum(lambda) )
    subject to
        sum(lambda .* y) == 0
        lambda >= 0
cvx_end
```

Soft Margin SVM in MATLAB

The following code solves the dual problem for a soft margin SVM using `CVX`:

```
% Inputs:
% K: Kernel matrix (n x n)
% y: Labels vector (n x 1)
% C: Regularization parameter

cvx_begin
    variable lambda(n)
    minimize( 0.5 * quad_form(lambda .* y, K) - sum(lambda) )
    subject to
        sum(lambda .* y) == 0
        0 <= lambda <= C
cvx_end
```

After solving the dual problem, the optimal Lagrange multipliers λ can be interpreted as follows:

- **Support Vectors**: Data points corresponding to non-zero λ_i values are the support vectors.

The weight vector w can be computed as:

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

The bias term b can be computed using any support vector with $0 < \lambda_i < C$ as:

$$b = y_i - \sum_{j=1}^{n} \lambda_j y_j K(x_j, x_i)$$

Convex optimization plays a crucial role in solving SVM classification problems. By transforming the primal problem into its dual form, we can solve for the Lagrange multipliers, identify support vectors, and efficiently handle non-linear classification problems using kernel methods. The `CVX` package in MATLAB provides a straightforward way to solve both hard and soft margin SVM problems. Algorithm to implement the SVM classifier using the CVX solver is given in Algorithm 2.

Algorithm 2 Solving SVM Classification via Convex Optimization

```
1: **Input:** Training data $\{(x_i, y_i)\}_{i=1}^{n}$, regularization parameter $C$, kernel function
   $K(x_i, x_j)$ (optional)
2: **Output:** Optimal weight vector $w$, bias term $b$, decision function $f(x)$
3: **Step 1: Initialize Parameters**
4: Initialize Lagrange multipliers $\lambda_i \leftarrow 0$ for $i = 1, \ldots, n$
5: Set convergence criteria $\epsilon$
6: Initialize weight vector $w \leftarrow 0$, bias $b \leftarrow 0$
7: **Step 2: Formulate the Dual Problem**
8: Formulate the dual objective:
   $$\min_{\lambda} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \lambda_i$$
9: Subject to constraints:
   $$\sum_{i=1}^{n} \lambda_i y_i = 0, \quad 0 \leq \lambda_i \leq C$$
10: **Step 3: Solve the Dual Problem**
11: Solve the quadratic optimization problem using a suitable QP solver such as SMO
    or any other algorithm.
12: **Step 4: Compute the Weight Vector**
13: Once $\lambda$ is obtained, compute:
    $$w \leftarrow \sum_{i=1}^{n} \lambda_i y_i x_i$$
    (For a kernel SVM, use the implicit kernel representation for $w$.)
14: **Step 5: Compute the Bias Term**
15: Choose any support vector $x_k$ where $0 < \lambda_k < C$ and compute the bias:
    $$b \leftarrow y_k - \sum_{i=1}^{n} \lambda_i y_i K(x_i, x_k)$$
16: **Step 6: Construct the Decision Function**
17: Define the decision function as: $f(x) \leftarrow \sum_{i=1}^{n} \lambda_i y_i K(x_i, x) + b$
18: Predict the class label for a new input $x$ as: $\hat{y} \leftarrow \text{sign}(f(x))$
19: **Step 7: Convergence Check**
20: Check the convergence of the optimization process. If the solution has not
    converged within the specified tolerance $\epsilon$, repeat the optimization process.
21: **Step 8: Output the Classifier**
22: Return the weight vector $w$, bias $b$, and decision function $f(x)$.
```

3.8 Decision Tree Classifier

Decision trees are a popular and interpretable model used for classification and regression tasks in machine learning. The algorithm builds a tree-like model of decisions and their possible consequences, effectively partitioning the feature space into distinct regions. The main objective is to create a model that predicts the target variable by learning simple decision rules inferred from the data features.

Tree Structure Representation

A decision tree is represented as a hierarchical structure composed of nodes and branches. Each internal node represents a decision based on a feature, each branch represents the outcome of the decision, and each leaf node represents a class label (for classification tasks) or a continuous value (for regression tasks).
Let D be the dataset with n instances, where each instance is represented as (x_i, y_i) for $i = 1, 2, \ldots, n$, with $x_i \in \mathbb{R}^d$ as the feature vector and y_i as the target variable.

Splitting Criteria

The core of building a decision tree lies in selecting the best feature to split the data at each node. The goal is to maximize the information gain or minimize the impurity after the split.

Information Gain

Information Gain (IG) measures the reduction in entropy after a dataset is split on an attribute. The entropy $H(D)$ of a dataset D is defined as:

$$H(D) = -\sum_{c} P(c|D) \log_2 P(c|D)$$

where $P(c|D)$ is the proportion of instances in class c.
When splitting the dataset D on feature A, the entropy of the resulting subsets D_v for each value v of A is computed as follows:

$$H(D|A) = \sum_{v} P(v|D) H(D_v)$$

The Information Gain for the attribute A is given by:

$$IG(D, A) = H(D) - H(D|A)$$

The attribute that yields the highest Information Gain is selected for the split.

Gini Impurity

Alternatively, Gini Impurity can be used as a splitting criterion. The Gini Impurity $G(D)$ of a dataset D is defined as:

$$G(D) = 1 - \sum_c P(c|D)^2$$

For a split on feature A, the Gini Impurity after the split is given by:

$$G(D|A) = \sum_v P(v|D)G(D_v)$$

The feature that minimizes the Gini Impurity is chosen for the split.

Recursive Partitioning

The process of constructing a decision tree involves recursively partitioning the data based on the selected features until a stopping criterion is met. Common stopping criteria include:

- Maximum tree depth
- Minimum number of samples in a node
- No further information gain from splits

At each leaf node, a prediction is made based on the majority class (for classification) or the average value (for regression) of the instances in that node.

Overfitting and Pruning

One challenge in building decision trees is overfitting, where the model becomes too complex and captures noise in the data. Pruning is a technique used to address this issue by removing branches that provide little predictive power.
Two common pruning strategies are:

- **Pre-pruning:** Stop growing the tree when further splits do not significantly improve the model (e.g., based on a threshold for Information Gain).
- **Post-pruning:** Grow the full tree and then remove nodes that do not improve model performance on a validation set.

Decision Rule

Once the decision tree is constructed, the decision rule for predicting a new instance x can be formulated as follows:

- Start at the root node and evaluate the feature x_j corresponding to the decision.

- Traverse the tree by following the branches based on the values of x_j until a leaf node is reached.
- The predicted class label $\hat{y}$ is the label associated with the leaf node.

Mathematically, the prediction can be represented as:

$$\hat{y} = f(x) = \text{label of the leaf node reached by } x$$

Convex Optimization Model

Although decision trees are not typically framed as convex optimization problems, we can discuss the optimization perspective in terms of minimizing the overall impurity across the tree. The goal is to minimize the weighted impurity of the nodes:

$$\min_{\theta} \sum_{v} P(v|D)G(D_v)$$

where θ represents the parameters defining the splits of the tree. While individual node splits may not yield a convex loss function, the overall objective can be viewed through an optimization lens.
The convex nature arises in ensemble methods built upon decision trees, such as Gradient Boosting, where loss functions can be designed to be convex. The optimization framework often involves:

$$\min_{\theta} L(y, f(x;\theta))$$

where L is a convex loss function, y is the target variable, and $f(x;\theta)$ represents the prediction from the ensemble of trees.

Advantages of Decision Trees

Decision trees offer several advantages as a machine learning model:

- **Interpretability:** Decision trees provide a clear and interpretable model that can be visualized easily.
- **Non-parametric:** They do not assume any underlying distribution for the data.
- **Handling Mixed Data Types:** Decision trees can handle both numerical and categorical data.
- **Feature Importance:** Decision trees naturally provide insights into the importance of different features.

Overall, decision trees are a versatile and powerful tool in machine learning, providing a foundation for more complex ensemble methods such as Random Forests and Gradient Boosting Machines.

Algorithm 3 Decision Tree Algorithm

```
1: Input: Dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, stopping criteria
2: Output: Decision tree model
3: Step 1: If all instances in $D$ belong to the same class, return a leaf node with that class label
4: Step 2: If stopping criteria are met, return a leaf node with the majority class label in $D$
5: Step 3: For each feature $A_j$, calculate Information Gain or Gini Impurity
6: Step 4: Select the feature $A_k$ with the highest Information Gain or lowest Gini Impurity for the split
7: Step 5: Split the dataset $D$ into subsets $D_v$ based on the values of feature $A_k$
8: for each subset $D_v$ do
9:     Step 6: Recursively call the decision tree algorithm on subset $D_v$
10: end for
11: Step 7: Combine the results to form the decision tree
```

3.8.1 Lagrangian Formulation of Decision Trees

Problem Statement

Given a dataset (X, y), where $X \in \mathbb{R}^{n\times m}$ is the feature matrix with n samples and m features, and $y \in \mathbb{R}^n$ is the target vector, we aim to construct a decision tree that minimizes a loss function while adhering to certain constraints.

Objective Function

The objective is to minimize the total loss, which can be defined as the sum of the loss at each node of the tree. A common choice for the loss function is the mean squared error (MSE) for regression tasks or cross-entropy for classification tasks. The overall optimization problem can be formulated as:

$$\min_T \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \lambda \cdot R(T) \tag{3.1}$$

Where:

- $L(y_i, \hat{y}_i)$ is the loss at node i.
- $R(T)$ is a regularization term (e.g., tree depth, number of leaves).
- λ is a hyperparameter controlling the trade-off between the loss and the regularization.

Lagrangian Function

To include constraints, we define a Lagrangian function. For a decision tree, you might want to include constraints on the maximum depth of the tree and the minimum number of samples at each leaf node.

The Lagrangian L can be defined as:

$$L(T, \alpha, \beta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \lambda \cdot R(T) + \alpha(D - D_{\text{max}}) + \beta(N_{\text{min}} - N) \tag{3.2}$$

Where:

- D is the depth of the tree.
- D_{max} is the maximum allowed depth.
- N_{min} is the minimum required samples at a leaf node.
- N is the number of samples in the node.

Interpretation of Lagrange Multipliers

The Lagrange multipliers λ_j play a critical role in balancing the trade-off between minimizing the loss function and satisfying the constraints. A positive λ_j indicates that the corresponding constraint is active, suggesting that the optimization process will prioritize satisfying this constraint. As the optimization progresses, the values of λ_j adjust, reflecting the importance of each constraint relative to the loss function.

3.9 Random Kitchen Sink (RKS) Kernel in Soft Margin SVM Classifier

The Random Kitchen Sink (RKS) method provides a novel approach for approximating kernel functions, particularly advantageous in the context of soft margin Support Vector Machine (SVM) classifiers. The RKS kernel allows for efficient computation of kernel evaluations by employing random feature mappings. Given a kernel function $K(x, y)$, the RKS method approximates it through a feature mapping $z : \mathbb{R}^n \to \mathbb{R}^d$, where the features are defined as:

$$z(x) = \sqrt{\frac{2}{d}} \begin{bmatrix} \cos(\omega_1^T x + b_1) \\ \cos(\omega_2^T x + b_2) \\ \vdots \\ \cos(\omega_d^T x + b_d) \end{bmatrix}$$

In this formulation, the random vectors ω_i are drawn from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$, while b_i are uniformly sampled from the interval $[0, 2\pi]$. The RKS kernel approximates the original kernel as:

$$K(x, y) \approx \frac{1}{d} z(x)^T z(y)$$

This representation capitalizes on the Fourier basis, allowing for an efficient estimation of kernels like the Radial Basis Function (RBF) kernel, which is defined as:

$$K(x,y) = \exp\left(-\frac{||x-y||^2}{2\sigma^2}\right)$$

The Fourier transform of the RBF kernel reveals its spectral properties, allowing RKS to efficiently sample features that capture the essential relationships in the data while avoiding the computational burden of direct kernel evaluations.
When incorporated into the soft margin SVM framework, the RKS kernel significantly enhances computational efficiency. The SVM seeks to minimize the following objective function:

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

where ξ_i are slack variables that allow for misclassifications, and C is a regularization parameter controlling the trade-off between margin maximization and classification errors. The optimization is subject to the constraints:

$$y_i\left(\mathbf{w}^T z(x_i) + b\right) \geq 1 - \xi_i, \quad \forall i$$

The dual formulation of the soft margin SVM can be represented as:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i,x_j) - \sum_{i=1}^{n}\alpha_i$$

subject to the constraints:

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C.$$

By substituting the RKS kernel into this formulation, the kernel evaluations $K(x_i, x_j)$ can be approximated using the random features derived from $z(x_i)$ and $z(x_j)$.
The advantages of using the RKS kernel are notable, especially regarding computational scalability and flexibility in modeling complex non-linear relationships. The accuracy of the RKS approximation improves as the number of random features d increases, enabling the kernel to capture a more accurate representation of the underlying data distribution. Additionally, as the sample size grows, the benefits of the RKS kernel become more pronounced, reinforcing its suitability for large datasets where traditional kernel methods may falter.

3.10 Built-in Machine Learning Functions in MATLAB

To supplement custom-built models with standardized methods, several built-in machine learning functions are available in MATLAB. These functions, such as `fitglm`, `fitcsvm`, and `fitctree`, allow for efficient model fitting, fine-tuning, and evaluation, making them useful for both preliminary analysis and performance benchmarking against hand-crafted models.

3.10.1 Logistic Regression using `fitglm`

Logistic regression is a widely used technique for binary classification tasks, and MATLAB provides the `fitglm` function for this purpose. The `fitglm` function fits a generalized linear model to the data, which in the context of binary classification utilizes a binomial distribution to model the probability of a given class. The basic syntax for fitting a logistic regression model is:

```
model = fitglm(X, y, 'Distribution', 'binomial');
```

Here, `X` is the matrix containing the predictor variables, and `y` is the response variable, which must be a binary outcome. The function fits the model by employing maximum likelihood estimation under the assumption of a binomial distribution, thus creating a probability-based classification. The trained model can be used for predicting new outcomes using the `predict` method, and the confidence intervals of the coefficients can be obtained via the `coefCI` method. This function provides flexibility for logistic regression problems and supports customization in terms of link functions and interaction terms, making it robust for exploring the relationship between predictor variables and binary outcomes.

3.10.2 Support Vector Machines using `fitcsvm`

Support Vector Machines (SVMs) are a powerful classification technique, particularly useful for high-dimensional spaces and cases where the classes are separable by a hyperplane. In MATLAB, the `fitcsvm` function implements SVMs for binary classification. The syntax to train a basic linear SVM is as follows:

```
model = fitcsvm(X, y, 'KernelFunction', 'linear');
```

In this case, `X` represents the set of features, and `y` contains the class labels. The `KernelFunction` argument specifies the type of kernel to use, with common options including 'linear', 'polynomial', and 'rbf' (radial basis function). SVMs are designed to find the hyperplane that maximally separates the two classes in the feature space. The choice of kernel function allows the SVM to handle non-linearly separable data by transforming it into a higher-dimensional space where a linear separation is possible.
Once the model is trained, predictions can be made using the `predict` method. Additionally, the model's performance can be evaluated using cross-validation by applying the `crossval` method, which divides the dataset into training and testing sets to estimate the generalization error. This method is crucial for assessing the model's reliability on unseen data, ensuring that the chosen hyperparameters (such as the regularization parameter or kernel type) are appropriate for the problem.

3.10.3 Decision Trees using `fitctree`

Decision trees are a non-parametric, interpretable method for classification, which recursively splits the data based on feature values to create a tree-like structure.

MATLAB's `fitctree` function provides a straightforward way to build decision trees for classification tasks. The basic syntax is:

```
model = fitctree(X, y);
```

The matrix `X` contains the predictor variables, while `y` holds the corresponding class labels. Decision trees work by recursively partitioning the feature space into regions that maximize the separation between different classes. The split criterion is often based on measures such as Gini impurity or information gain. The resulting tree can be visualized using the `view` function, which produces a graphical representation of the splits and class assignments at each node.
This function allows for further customization, such as setting the minimum leaf size to control overfitting, and the depth of the tree can be constrained to ensure that the model generalizes well to new data. Predictions can be made using the `predict` method, and the model can also undergo cross-validation to estimate its performance using the `crossval` method.

3.10.4 Comparative Evaluation and Tuning of Models

Once models are trained using the built-in functions, their performance can be compared against the custom-developed models by evaluating standard classification metrics such as accuracy, precision, recall, and the F1 score. These metrics can be computed using MATLAB's `confusionmat` function, which creates a confusion matrix by comparing true class labels with predicted labels. This allows for the calculation of sensitivity (true positive rate), specificity (true negative rate), and overall accuracy.
For a more detailed evaluation of the model's discriminative ability, the receiver operating characteristic (ROC) curve can be plotted, and the area under the curve (AUC) can be computed to measure the trade-off between sensitivity and specificity at different classification thresholds.
Additionally, the built-in models offer hyperparameter tuning options. For example, the regularization parameter in SVMs can be adjusted to control the trade-off between margin size and classification error on the training data. This fine-tuning process helps to improve the model's generalization performance, ensuring that it performs well not only on the training data but also on unseen data.

3.11 Towards an Explainable and Sustainable Tumor Classification Model

In the complex domain of tumor classification, the limitations of traditional supervised learning models reveal the need for a more nuanced approach, one that emulates the diagnostic expertise of seasoned physicians. Experienced doctors often rely on pattern recognition developed over years of practice to classify cancer stages or assess severity, sometimes bypassing extensive invasive tests. By

interpreting symptoms, test results, and clinical indicators in context, doctors provide a valuable, cost-effective, and sustainable approach to patient care.
Adopting an unsupervised machine learning method, such as clustering, aims to replicate this diagnostic intuition by discovering intrinsic groupings within the data without relying on predefined labels. Such an approach is crucial in tumor data analysis, where outliers may indicate unique or severe cases, potentially revealing new tumor subtypes or stages. By retaining outliers, clustering preserves diagnostic granularity, allowing the model to identify rare but critical cases that may inform tailored treatment plans. This methodology aligns with the healthcare goal of sustainability by minimizing the need for excessive confirmatory testing.
Furthermore, a model capable of generating explainable classifications from unsupervised learning can adapt as new data arises, providing medical practitioners with flexible insights into disease progression. The challenge of maintaining model interpretability is addressed through techniques that prioritize clinical relevance in the latent features, ensuring that the model's insights remain transparent and actionable. This direction holds promise for developing robust machine learning tools that support precision medicine by aiding in accurate diagnosis, enabling early intervention, and optimizing patient care outcomes.

3.12 Tumor Classification Using K-Means Clustering

Using the unsupervised methods such as clustering, the model can uncover hidden patterns in the data that align with medical science. This approach allows for the creation of explainable tumor classes based on the identified patterns, offering insights into potential new sub types or stages of disease progression. In this context feasibility of a clustering model is investigates as explained in this section.

3.12.1 Classical K-Means Clustering Approach

The K-means clustering algorithm is a widely used unsupervised machine learning technique for partitioning data into distinct clusters. It aims to group data points such that the variance within each cluster is minimized. The algorithm proceeds through the following steps:

1. **Initialization**: Randomly select k initial centroids μ_j for the clusters.
2. **Assignment Step**: Assign each data point x_i to the nearest centroid based on Euclidean distance:
$$z_{ij} = \begin{cases} 1 & \text{if } j = \arg\min_{j'} \|x_i - \mu_{j'}\|^2 \\ 0 & \text{otherwise} \end{cases}$$
3. **Update Step**: Recalculate the centroids μ_j as the mean of all points assigned to cluster j:
$$\mu_j = \frac{1}{\sum_{i=1}^{N} z_{ij}} \sum_{i=1}^{N} z_{ij} x_i$$

4. **Convergence Check**: Repeat the assignment and update steps until the centroids do not change significantly or a specified number of iterations is reached.

The loss function to minimize during this process is the within-cluster sum of squares:

$$\min_{z,\mu} \sum_{j=1}^{k} \sum_{i=1}^{N} z_{ij} \|x_i - \mu_j\|^2$$

3.12.2 Limitations of the Classical Approach

While traditional K-means clustering effectively groups data, it often fails to capture the complex patterns inherent in medical datasets, such as tumor classification. Misclassification can occur, especially in cases where the data contains outliers or when the underlying patterns do not conform to a strict binary classification. These limitations necessitate a more nuanced approach that can handle variability in the data, akin to the diagnostic skills of experienced medical practitioners.

3.12.3 Transition to Lagrangian Multiplier Model

To address these limitations, we reformulate the K-means clustering problem using a Lagrangian multiplier model. This approach allows us to incorporate constraints directly into the optimization framework, making it suitable for more complex and realistic scenarios.

Problem Setup

Let:

- $X = \{x_1, x_2, \ldots, x_N\}$ be the dataset with N data points, where each $x_i \in \mathbb{R}^d$.
- k be the number of clusters.
- $\mu_j \in \mathbb{R}^d$ represent the centroid of cluster j.
- $z_{ij} \in \{0, 1\}$ be a binary variable, where $z_{ij} = 1$ if x_i is assigned to cluster j, and $z_{ij} = 0$ otherwise.

Objective Function with Constraints

We seek to minimize the within-cluster variance while enforcing assignment constraints:

$$\min_{z,\mu} \sum_{j=1}^{k} \sum_{i=1}^{N} z_{ij} \|x_i - \mu_j\|^2$$

subject to the constraints:

1. **Assignment Constraint**: Each point must be assigned to exactly one cluster:

$$\sum_{j=1}^{k} z_{ij} = 1, \quad \forall i = 1, \dots, N$$

2. **Binary Constraint**: The variable z_{ij} must be binary:

$$z_{ij} \in \{0, 1\}, \quad \forall i = 1, \dots, N, \quad j = 1, \dots, k$$

Lagrangian Formulation

To incorporate the constraints into the optimization problem, we introduce a Lagrange multiplier λ_i for each point x_i to enforce the assignment constraint. The Lagrangian for this problem can be expressed as:

$$\mathcal{L}(z, \mu, \lambda) = \sum_{j=1}^{k} \sum_{i=1}^{N} z_{ij} \|x_i - \mu_j\|^2 + \sum_{i=1}^{N} \lambda_i \left(1 - \sum_{j=1}^{k} z_{ij}\right)$$

This formulation allows us to minimize the within-cluster variance while ensuring that each point is assigned to exactly one cluster.

3.12.4 Optimization Model Solved with CVX

Using CVX, we can solve this constrained optimization problem iteratively:

1. **Step 1**: Fix μ and solve for z with CVX, enforcing the assignment constraints.
2. **Step 2**: Fix z and update μ as the mean of assigned points.

MATLAB CVX Code Example

```
% Load your data in X and specify the number of clusters
X = ...; % (NxD matrix, where N is number of points and D is dimensions)
k = ...; % Number of clusters
[N, D] = size(X);

% Initialize centroids randomly
mu = X(randperm(N, k), :);

for iter = 1:100  % Set max iterations or use a convergence criterion

    % Step 1: Solve for z with fixed mu using CVX
    cvx_begin
        variable z(N, k)
        % Objective function: Minimize within-cluster variance
        minimize(sum(sum(z .* repmat(sum((X - mu').^2, 2), 1, k))))
        % Subject to assignment constraints
        subject to
            sum(z, 2) == 1;    % Each point assigned to one cluster
            z >= 0;            % Relaxed binary constraint for convex optimization
    cvx_end

    % Step 2: Update centroids based on current z
    for j = 1:k
        mu(j, :) = sum(repmat(z(:, j), 1, D) .* X, 1) / sum(z(:, j));
    end

    % Check for convergence if needed
end
```

3.12.5 Evaluation of Clustering: Silhouette Score

After obtaining clustering assignments, evaluate model performance using the silhouette score. For each data point x_i, calculate the mean intra-cluster distance $a(i)$ and the mean nearest-cluster distance $b(i)$:

1. **Mean Intra-Cluster Distance** $a(i)$:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{x_j \in C_i} \|x_i - x_j\|^2$$

   where C_i is the cluster containing x_i.

2. **Mean Nearest-Cluster Distance** $b(i)$:

$$b(i) = \min_{j \neq i} \frac{1}{|C_j|} \sum_{x_k \in C_j} \|x_i - x_k\|^2$$

3. **Silhouette Score** $s(i)$:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

The overall silhouette score provides insight into clustering quality and helps compare models with different k values.

3.13 Conclusion

In this study, we revisited the mathematical models of classical machine learning algorithms, specifically focusing on Logistic Regression, Support Vector Machines (both hard and soft margin), and Decision Trees. A significant aspect of our approach involved employing convex optimization techniques to formulate and solve these models. The convex optimization framework allowed for effective minimization of

the loss functions associated with each algorithm, ensuring global optima in the training process.
These custom models were utilized to predict the likelihood of breast cancer using the Breast Cancer dataset from the University of Wisconsin at Madison. By applying these models, we gained insights into the underlying relationships between the features and the target variable, which is crucial for understanding the predictive capabilities of each algorithm in the context of breast cancer diagnosis.
Subsequently, we leveraged built-in MATLAB functions such as `fitglm`, `fitcsvm`, and `fitctree` to perform the same predictive task.

Chapter 4

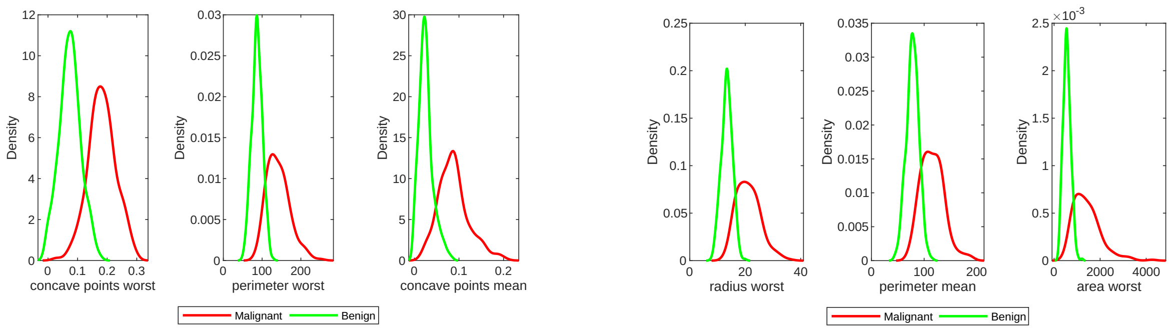
Results and Discussions

4.1 Introduction

This chapter presents the results obtained from implementing various classification algorithms to predict breast cancer using the Breast Cancer dataset from the University of Wisconsin at Madison. The algorithms evaluated in this study include custom implementations of Logistic Regression, Support Vector Machines (SVM) with both hard and soft margins, and Decision Trees, all formulated through convex optimization techniques. Additionally, the performance of built-in MATLAB functions—`fitglm`, `fitcsvm`, and `fitctree`—is also assessed for comparison.
The primary objective of this chapter is to provide a detailed analysis of the classification performance of each algorithm based on several metrics, including accuracy, sensitivity, specificity, F1 score, Receiver Operating Characteristic (ROC) curve, and Area Under the Curve (AUC). By systematically evaluating the results, we aim to uncover insights into the strengths and weaknesses of each approach in predicting breast cancer.
Furthermore, the discussions will address the implications of the findings, emphasizing the significance of mathematical modeling and optimization in machine learning applications. The interplay between the custom models and MATLAB's built-in functions will be explored to highlight how both approaches contribute to achieving robust predictive performance in the medical domain. Through this examination, we aim to provide a comprehensive understanding of the effectiveness of different classification techniques in the context of breast cancer diagnosis.

4.2 Skill of Logistic Regression Classifier

In preparation for logistic regression to predict malignant cells, we conducted a correlation analysis to identify the most dominant features associated with the target variable (malignancy). The analysis revealed six features with a correlation coefficient greater than 0.75, indicating a strong relationship with the diagnosis outcome. The distribution of these six features across the target classes (malignant and benign) is visualized in the figure, providing insights into how these features differ between the two classes. This analysis serves as a foundational step in feature selection for the logistic regression model. Our aim is to find the optimal number of features which produce better performance metrics in logistic regression. Distribution of top six dominant feature in classification into benign or malignant is shown in Figure 4.1.



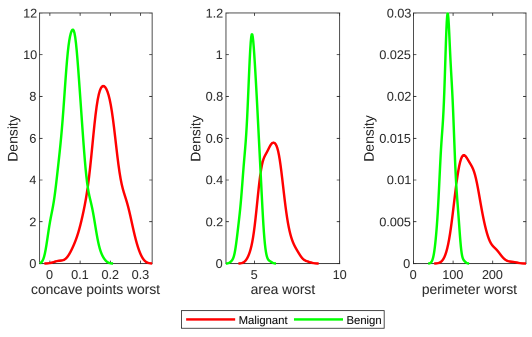
(a) Top three dominant features **(b)** Top four to six dominant features

Figure 4.1: Distribution of dominant features across the target variable (malignant and benign) before preprocessing.

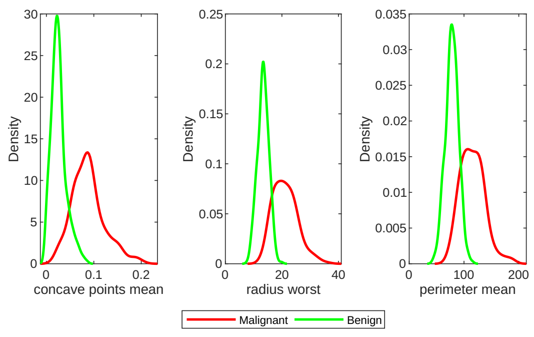
In logistic regression, outlier removal and feature scaling are critical preprocessing steps that significantly improve model performance. Outliers can distort the decision boundary by disproportionately affecting the estimated coefficients, leading to poor generalization and inaccurate predictions. Logistic regression assumes a linear relationship between the features and the log-odds of the target class. Outliers, especially in features with large values, can skew this relationship, resulting in an overfitted model with reduced interpretability.

Feature scaling is equally important because logistic regression is sensitive to the relative magnitudes of feature values. Features with larger scales may dominate the learning process, leading to biased coefficient estimates. Scaling ensures that all features contribute equally to the optimization of the cost function during gradient descent. This improves convergence speed and stability of the optimization process.

After applying these preprocessing steps, a correlation analysis was performed to identify the six most correlated features with the target variable (malignant or benign). The top six features were selected based on their correlation coefficient (greater than 0.74), and their distribution across the target variable is shown in Figure 4.2. These features provide the most predictive power in distinguishing between malignant and benign cells, enhancing the logistic regression classifier's ability to make accurate predictions.



(a) Top three dominant features



(b) Top four to six dominant features

Figure 4.2: Distribution of dominant features after outlier removal and scaling.

Incorporating these steps ensures a robust and reliable logistic regression model, capable of accurately predicting the likelihood of breast cancer based on the most informative features.

It is observed that after scaling and outlier removal, the top six highly correlated features with the target variable remained the same, though their order of correlation changed slightly.

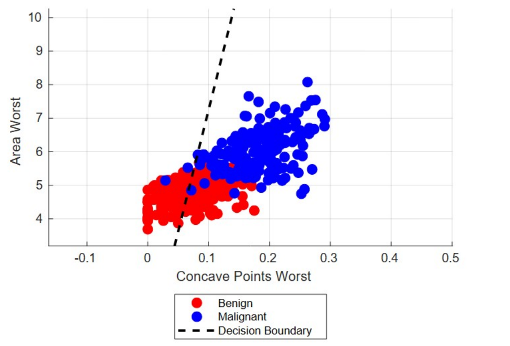
Here, we explored a mathematical approach to develop a binary classifier from a linear regressor. Using a linear model, the output is predicted as $W^T x$, where W represents the weight vector and x the input features. For binary classification, the separating plane is defined by the equation $W^T x = 0$, which divides the input space into two regions corresponding to the two classes.

The key objective is to optimize the weight vector W using a kernel regressor designed for linear regression. By employing kernel regression, we aimed to capture nonlinear relationships between features and improve the performance of the model. The linear regressor's predictions are then transformed into binary outputs based on the separating plane, where values greater than zero were classified into one class, and those less than zero into the other.

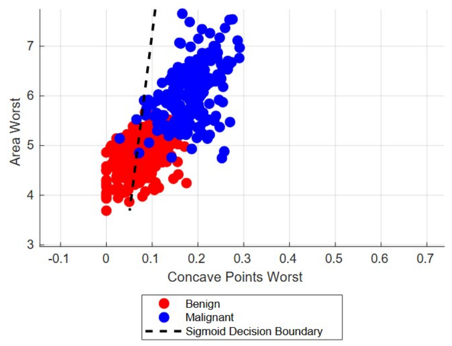
This approach bridges the gap between regression and classification by transferring the continuous output of a regressor to define a decision boundary for classification. The results of this methodology demonstrated that the binary classifier derived from the optimized linear regressor performs effectively in separating the two target classes. This method is particularly useful when there is a linear or nearly linear relationship between the input features and the output classes.

Models with 30 cleaned and scaled features shows almost similar performance metrics scores in logistic regression.

Figure 4.3 shows the separating planes and data points plotted over the top two correlated features in both linear separating boundary (Figure 4.3a) and sigmoid boundary (Figure 4.3b).



(a) Separating boundary $W^T X = 0$ of linear regression



(b) Separating boundary $\sigma(W^T X) = 0.5$ of sigmoid regression

Figure 4.3: Separating planes and data distribution in linear and logistic regression

A comparison of performance of both the generated linear regressor and logistic regressor is shown in Table 4.1.

Table 4.1: Performance Comparison of Linear and Logistic Regressors for Binary Classification

Metric	Linear Regressor	Logistic Regressor
Accuracy	0.56	0.97
Sensitivity	1.00	0.96
Specificity	0.30	0.97
F1 Score	0.471	0.96
AUC	0.20	0.99

From Table 4.1, it is clear that logistic regressor is a clear winner.

In this study, we aimed to determine the optimal number of features that enhance the classification performance of our model. To achieve this, we employed 5-fold cross-validation, which helps stabilize performance metrics by mitigating the variability that may arise from different training and testing splits. We applied a linear regression classifier across various feature subsets, using the correlation with the target variable as a benchmark for feature selection. This approach enabled us to systematically evaluate the impact of feature count on classification skill and identify the most effective subset for improved predictive accuracy. Table 4.2 presents the performance metrics associated with different numbers of features corresponding to the selected correlation level.

Table 4.2: Performance metrics of logistic regression classifier across different feature subsets

#Features	$\rho \geq$	Acc-K-fold	Acc	Sen	Spe	Er.rate	AUC	F1-Score
5	0.75	0.95	0.95	0.92	0.96	0.045	0.990	0.9396
9	0.70	0.95	0.96	0.94	0.97	0.030	0.993	0.9548
13	0.60	0.93	0.97	0.95	0.97	0.029	0.994	0.9599
15	0.50	0.95	0.96	0.93	0.98	0.030	0.990	0.9543
20	0.40	0.96	0.98	0.96	0.98	0.019	0.990	0.9699
23	0.30	0.95	0.97	0.96	0.98	0.020	0.990	0.9699
25	0.20	0.95	0.98	0.97	0.98	0.017	0.998	0.9750

4.2.1 Performance Metrics Analysis

The analysis of the performance metrics presented in Table 4.2 emphasizes the goal of developing a robust and consistent model with an optimal number of features for binary classification in medical diagnosis.

4.2.2 Overview of Performance Metrics

Number of Features and Correlation (ρ)

The correlation values (ρ) indicate a decline from 0.75 with the 5-feature subset to 0.20 with the 25-feature subset. This suggests that while fewer features exhibit stronger individual relationships with the target variable, the inclusion of more features does not necessarily correlate with a proportional improvement in model performance.

Accuracy (Acc)

The accuracy of the model shows minimal variation as the number of features increases, peaking at 0.98 with both the 20 and 25 feature subsets. This indicates that while the model maintains a high level of accuracy, the incremental benefit of adding more features is marginal. The results suggest a point of diminishing returns in terms of accuracy, emphasizing the importance of feature selection for model simplicity and interpretability.

K-fold Cross-Validation Accuracy (Acc-K-fold)

The K-fold cross-validation accuracy remains relatively consistent across all feature subsets, ranging from 0.93 to 0.96. This stability suggests that the model's performance is robust and not overly dependent on the number of features included. It reinforces the notion that a more parsimonious model could be equally effective, if not more so, in terms of generalization and interpretability.

Sensitivity

Sensitivity values, ranging from 0.92 to 0.97, indicate the model's strong ability to identify malignant cases consistently. However, the slight variations in sensitivity across different feature subsets suggest that increasing the feature count does not significantly enhance this critical aspect of performance.

Specificity

Specificity remains high, ranging from 0.96 to 0.98 across all subsets, indicating that the model effectively identifies non-malignant cases. This consistency in specificity further supports the potential for a simpler model without compromising the ability to accurately classify both malignant and non-malignant cases.

Error Rate

The error rate decreases from 0.045 (with 5 features) to 0.017 (with 25 features), suggesting a slight improvement in reliability. However, the reduction is modest compared to the increase in complexity associated with additional features, highlighting the necessity of considering feature selection carefully.

Area Under the Curve (AUC)

The AUC values consistently remain high (0.990 to 0.998), indicating the model's strong discriminatory power across all feature subsets. This stability in AUC suggests that a robust model can be developed without necessitating an excessive number of features.
As a final step, regularization techniques—L1, L2, and Elastic Net—are applied to the logistic regression model using gradient descent to minimize the mean square error in prediction. The analysis is conducted on two feature sets-**Full Feature Set (30 scaled features)** and **Dominant Subset (Top 5 correlated features)**.
Performance metrics, including accuracy, sensitivity, specificity, F1-score, and AUC, are computed for each model. Summary of these results along with the K-fold cross validation of L1 regularization model is shown in Table 4.3.

Table 4.3: Comparison of performance metrics of various regularized models and K-fold cross validation

Regularized Model	Performance Metrics							
	30 feature set				Top 5 correlated subset			
Metric $\longrightarrow$	Acc	Sen	Spec	F1	Acc	Sen	Sepc	F1
L2	0.968	0.961	0.971	0.960	0.947	0.930	0.960	0.930
L1	0.975	0.970	0.980	0.970	0.944	0.920	0.960	0.920
Elastic Net	0.975	0.970	0.980	0.970	0.944	0.920	0.960	0.920
L1 with K-fold CV	0.973	0.960	0.980	0.960	0.942	0.910	0.960	0.920

These findings during the training phase, highlight the effectiveness of **L1 regularization** for feature selection, model accuracy, and consistency, making it the most suitable technique for this classification task.
Each of these models applied on the test dataset. A summary of accuracy on the training and test data set is shown in Table 4.4.

Table 4.4: Comparison of performance metrics of various logistic regression models on train and test dataset.

Regularized Model	Performance Metrics			
	30 feature set		Top 5 correlated subset	
Metric $\longrightarrow$	Training Accuracy	Test Accuracy	Training Accuracy	Test Accuracy
L2	0.968	0.964	0.947	0.930
L1	0.975	0.964	0.944	0.942
Elastic Net	0.975	0.957	0.944	0.942

The results from both the training and test phases provide a clear indication of each regularized model's strengths and applicability for this classification task.

- **L1 Regularization** consistently achieved the highest test accuracy on both feature sets, with near-equal performance on training and test data, demonstrating its effectiveness in improving model interpretability without compromising generalization. This outcome aligns with L1's characteristic of promoting sparsity, effectively zeroing out less informative features, which is particularly advantageous in reducing dimensionality and potential overfitting. The AUC of 0.99 further validates its discriminative power, confirming its suitability for distinguishing between malignant and benign cases.

- **L2 Regularization** showed stable accuracy across the training and test sets, particularly in the full 30-feature set where it achieved similar test accuracy to L1 (0.964). This stability suggests that L2 is effective in managing multicollinearity and controlling model complexity, making it a solid choice for datasets where feature interpretability is less critical.

- **Elastic Net Regularization** provided high training accuracy but slightly lower test accuracy on the 30-feature set, indicating a small degree of overfitting. However, Elastic Net matched L1 in test accuracy on the top 5 feature subset (0.942), making it a balanced choice when both L1 and L2 penalties are desired to handle feature selection alongside multicollinearity.

In conclusion, **L1 Regularization emerges as the optimal method**, offering a balanced approach to accuracy, feature selection, and model consistency, especially on datasets with potentially redundant or irrelevant features. This approach provides strong generalization and interpretability, which is beneficial for robust classification in this domain.

This suggests that, for the goal of developing a better and more consistent model, it is essential to focus on identifying an optimal number of features that strike a balance between performance and model simplicity. The results advocate for feature selection based on correlation strength with the target variable while considering the overall performance stability to enhance interpretability and ensure effective medical diagnosis. Thus, an emphasis on parsimony may yield a model that is not only efficient but also easier to implement in practical scenarios.

4.3 Skill of Support Vector Machines

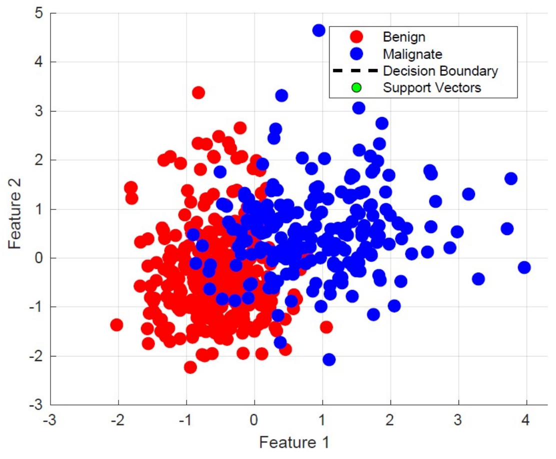
Despite the extensive preprocessing applied to the dataset—including feature scaling and outlier removal—five features still exhibited significant outliers during the logistic regression analysis (chapter 3, section 3.2.7, Figure 3.3). While logistic regression performed well on the data, theoretical insights suggest that in the presence of outliers, especially in high-dimensional spaces, Support Vector Machines (SVMs) offer superior performance due to their robustness in handling such irregularities.

SVMs are designed to find an optimal hyperplane that maximally separates classes by focusing on the most critical data points—referred to as support vectors—thus minimizing the influence of outliers. This property makes SVMs particularly suitable for scenarios where the data may not be perfectly linearly separable, as was observed in the earlier experiments.

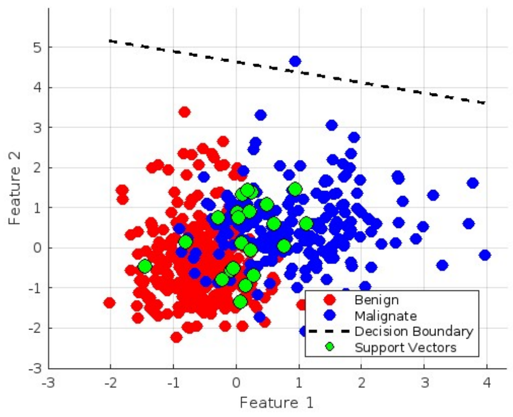
Given the residual outliers and the need for a more robust classification framework, SVMs are a natural next step to explore. The aim is to leverage their ability to manage outliers while improving the overall performance metrics of the model, particularly in terms of classification accuracy and consistency across folds. The following sections will discuss the application of SVMs to this dataset and provide a comparative analysis with logistic regression.

4.3.1 Hard Margin Support Vector Machine with Linear Kernel

A hard margin Support Vector Machine (SVM) with a linear kernel is applied to the dataset. The dual Lagrange multiplier formulation is solved using the CVX solver in MATLAB. The model achieved an accuracy of 62.7%. However, it was observed that the sensitivity (true positive rate) is 0, indicating that the model failed to correctly classify any malignant cases. Even with the top 5 correlated features, the same result is obtained. This outcome suggests that the hard margin SVM, which does not allow for any misclassification or margin violation, is overly rigid for the given data. The lack of flexibility in hard margin SVMs means that it prioritizes separating the majority class perfectly, leading to poor sensitivity, especially when the classes are not perfectly linearly separable. Figure 4.5a demonstrate this issue.



(a) Distribution of data points over first two features under hard margin svm



(b) Distribution of data points over first two features under soft margin svm

Figure 4.4: Distribution of data points over first two features under hard and soft margin linear SVM

This result underscores the need for a more flexible approach, such as soft margin SVMs, to allow some misclassification and better accommodate the complexity of the data. Using soft margin linear SVM on the 30 feature set did not produce even a support vector. After normalizing the feature set, a soft margin SVM with a linear kernel and regularization parameter $c = 1$ was applied. The model produced 22 support vectors and achieved an improved accuracy of 79.79%. The sensitivity and specificity were 55.66% and 94.12%, respectively.Distribution of data points over first two features and the separation boundary is shown in Figure 4.5b.
The introduction of a soft margin allowed the model to tolerate some misclassifications, enhancing its flexibility compared to the hard margin SVM. This improved the model's ability to correctly classify malignant cases, as reflected in the increased sensitivity. The high specificity suggests that the model maintained strong performance in identifying benign cases.

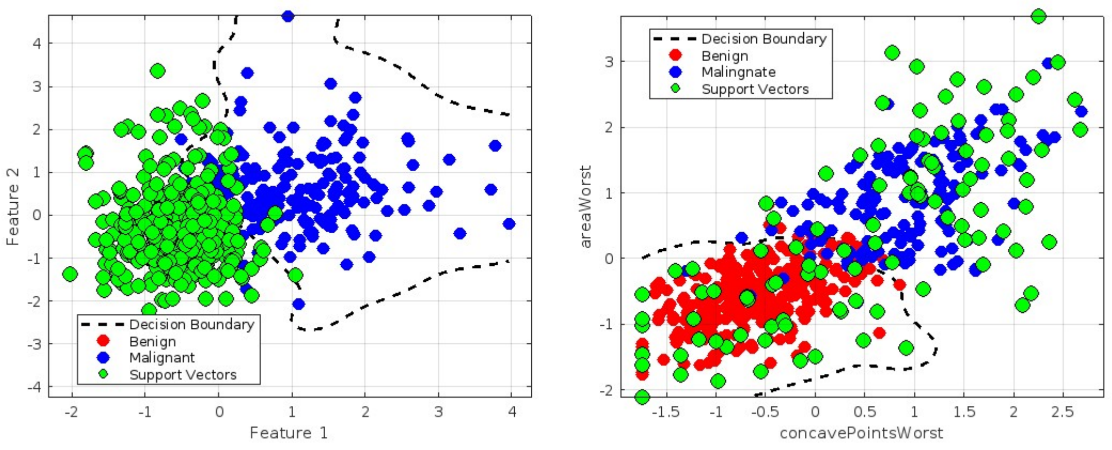
4.3.2 Soft Margin Support Vector Machine with Linear Kernel

Using the top 5 highly correlated features, a soft margin Support Vector Machine (SVM) with a linear kernel and regularization parameter $c = 10$ is applied. This model identified 5 support vectors and showed a significant improvement in performance, achieving an accuracy of 94.4%. The sensitivity (95.8%) and specificity (92.82%) indicate that the model effectively balances both positive (malignant) and negative (benign) class predictions.
The introduction of the soft margin allows for some misclassification, enabling the model to better handle the presence of noise and outliers in the dataset. This flexibility leads to improved sensitivity, which is critical in medical diagnosis, as it ensures that malignant cases are correctly identified. The higher specificity further confirms the model's capability to accurately classify benign cases, resulting in a more robust classification.

The trade-off between sensitivity and specificity indicates that while the model effectively reduces misclassifications, it still struggles with identifying all malignant cases, suggesting the need for further optimization or more sophisticated kernels.

4.3.3 Soft Margin Support Vector Machine with Non-linear Kernels

To further improve classification performance, the soft margin SVM was applied using the Radial Basis Function (RBF) kernel. This kernel allows the model to capture more complex decision boundaries, which are particularly useful in cases where the data is not linearly separable.
Experiments were conducted on both the full feature set and the top five correlated feature subset, with the regularization parameter C varied to control the trade-off between margin maximization and classification error. The accuracy of the model ranged from 47.98% to 96.15%, indicating that the choice of C plays a significant role in optimizing the classification skill.
The RBF kernel produced a more flexible and acceptable decision boundary compared to the linear kernel, especially in handling the non-linearity present in the data.
Distribution of data points and separation boundaries are shown in Figure 4.5.



(a) Distribution of data points of full dataset with $C = 1$.

(b) Distribution of data points of subset with $C = 2.5$.

Figure 4.5: Distribution of data points over first two features under soft margin SVM with RBF kernel

The higher accuracy in the upper range of C values suggests that the model was able to find an optimal balance between underfitting and overfitting, leading to improved classification performance across both feature sets.
Table 4.5 presents the performance metrics of the soft margin Support Vector Machine (SVM) with an RBF kernel applied to two different versions of the dataset: the full 30-feature set and a reduced subset comprising the top five highly

correlated features. The regularization parameter C is varied to explore the impact of the margin's flexibility on classification performance. The primary objective is to investigate whether the reduced feature subset offers competitive accuracy and other performance metrics while lowering computational costs.

Table 4.5: Comparison of performance metrics of full feature set and the five top correlated subset over the regularization parameter C.

Regularization parameter	Performance Metrics							
	30 feature set				Top 5 correlated subset			
C	Acc	Sen	Spec	F1	Acc	Sen	Sepc	F1
1.0	0.956	0.975	0.975	0.950	0.974	0.962	0.980	0.960
1.5	0.959	0.929	0.977	0.945	0.979	0.967	0.986	0.971
2.0	0.958	0.943	0.966	0.943	0.981	0.967	0.988	0.977
2.5	0.961	0.943	0.972	0.947	0.981	0.967	0.988	0.977
3.0	0.950	0.948	0.952	0.934	0.982	0.972	0.988	0.976
3.5	0.608	0.986	0.384	0.652	0.944	0.976	0.988	0.978
4.0	0.479	1.000	0.170	0.589	0.984	0.976	0.988	0.978
4.5	0.441	1.000	0.109	0.574	0.984	0.985	0.983	0.978
5.0	0.954	0.967	0.946	0.940	0.985	0.986	0.986	0.981

From the results, it is clear that the reduced feature subset consistently provides high accuracy, sensitivity, and specificity across various values of C, comparable to or even outperforming the full feature set in most cases. This suggests that the five most correlated features capture sufficient information for effective classification without requiring the entire feature set, significantly reducing model complexity.

As the regularization parameter C increases, the performance of the full feature set tends to degrade, especially for extreme values, as indicated by the sharp decline in accuracy and specificity. However, the reduced feature subset maintains stability even at higher values of C, with only minor fluctuations in performance metrics. This highlights the robustness of the reduced subset, particularly under varying regularization conditions.

A cross-validation is the right method to validate the model's performance and ensure that the reduced feature subset does not lead to performance inconsistencies across different data splits. It helps confirm that the reduced set is a viable and stable representation for classification. A 5-fold cross validation of the five top correlated subset is done for each value of the regularization parameter C. Result of this experiment is shown in Table 4.6.

Table 4.6: Statistical summary of 5-fold cross validation results

C	Performance Metrics							
	Accuracy		Sensitivity		Specificity		F1-score	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
1.0	0.933	0.015	0.924	0.035	0.938	0.021	0.915	0.022
1.5	0.933	0.015	0.938	0.036	0.930	0.017	0.913	0.022
2.0	0.933	0.009	0.943	0.039	0.927	0.018	0.913	0.014
2.5	0.928	0.007	0.948	0.030	0.916	0.010	0.907	0.010
3.0	0.927	0.007	0.948	0.030	0.915	0.010	0.907	0.010
3.5	0.926	0.007	0.948	0.030	0.913	0.018	0.905	0.010
4.0	0.923	0.009	0.948	0.030	0.907	0.021	0.901	0.011
4.5	0.923	0.007	0.948	0.030	0.907	0.021	0.901	0.008
5.0	0.926	0.008	0.953	0.023	0.916	0.016	0.905	0.009

The table summarizes the performance of the Support Vector Machine (SVM) model with a Radial Basis Function (RBF) kernel across various values of the regularization parameter C. Key metrics include accuracy, sensitivity, specificity, and F1-score, along with their corresponding standard deviations.

Key Observations

- **Accuracy**: The accuracy is quite stable across different values of C, ranging from 92.3% to 93.3%, with only slight variations.
- **Sensitivity**: Sensitivity, which measures the model's ability to correctly identify positive cases, peaks at $C = 5.0$ with a mean of 95.3%. Higher sensitivity is ideal in medical diagnoses, where false negatives are critical.
- **Specificity**: Specificity, which measures the model's ability to correctly identify negative cases, remains consistently around 91.5% to 93.8%. A slight dip is observed at $C = 4.0$, but this isn't drastic.
- **F1-Score**: The F1-score, which balances precision and recall, stays within a close range (90.1% to 91.5%), indicating a well-balanced model performance across all values of C.
- **Standard Deviations**: Standard deviations across all metrics are relatively low, implying stable and reliable performance across the cross-validation folds.

Recommendation

Given that the differences in performance across different values of C are minimal, the value $C = 5.0$ seems optimal due to its high sensitivity and balanced performance across other metrics.

Feature Selection: Top 5 Correlated Features vs. All 30 Features

- **Top 5 Correlated Features**: The performance with the top 5 correlated features is strong, as seen from the cross-validation results. The model is achieving good accuracy, sensitivity, and specificity, while being computationally less expensive compared to using all 30 features.
- **Using All 30 Features**: While using all 30 features may marginally improve the model's ability to capture more complex patterns, it risks overfitting, especially if many of the features are redundant or weakly correlated. Additionally, training with a larger feature set increases computational complexity.

Considering the consistently strong results with the top 5 correlated features and the risks associated with high-dimensional feature spaces, it is recommended to proceed with the top 5 correlated features. This simplifies the model without sacrificing performance, ensuring better generalization on unseen data.
A final decision can be made by comparing model accuracy on the test dataset for these two versions. Result of the model performance comparison on training and test data set is shown in Table 4.7.

Table 4.7

C=5	Performance Metrics			
	30 feature set		Top 5 correlated subset	
Metric $\longrightarrow$	Training Accuracy	Test Accuracy	Training Accuracy	Test Accuracy
SVM (RBF Kernel)	1.00	0.898	0.964	0.963
SVM (fitsvm)	0.976	0.965	0.949	0.946

4.4 Skill of Decision Tree Algorithm

The attempt to optimize the decision tree loss function through convex optimization using the CVX solver has produced subpar results, particularly with an accuracy of only 0.3726, specificity of 1.00, and a sensitivity of 0.0. These metrics reveal an imbalanced performance, where the model is essentially overfitting to the negative class and failing to detect any positive cases, a critical flaw in medical diagnostic applications. The CVX solver's warning about relying on successive approximation suggests challenges in achieving reliable convergence for the log-likelihood minimization, potentially due to complex non-linearities in the decision tree objective function. Given these limitations and the resulting low accuracy, it is recommended to adopt the classical CART (Classification and Regression Tree) algorithm, readily implemented in MATLAB, which has a well-established track record for stability and effectiveness in classification tasks.
The decision tree model implemented on the full dataset using MATLAB's `fitctree()` function demonstrates excellent predictive performance across

multiple evaluation metrics. With an overall accuracy of 0.99, the model achieves a sensitivity of 0.99, indicating a high rate of correct identification of positive cases, crucial for minimizing false negatives in a medical context. Specificity of 0.99 and precision of 0.98 further highlight the model's ability to discriminate accurately between classes and minimize false positives. An F1-score of 0.99 and an AUC of 0.99 underscore the balanced and robust performance, making this decision tree model highly suitable for medical diagnostic applications. Summary of the performance metrics for decision tree algorithm is shown in Table 4.8.

Table 4.8: Performance Metrics for Decision Tree Model using `fitctree()`.

Metric	Value
Accuracy	0.99
Sensitivity	0.99
Specificity	0.99
Precision	0.98
F1-score	0.99
AUC	0.99

Results in Table 4.8 indicates that there are still chances for over-fitting. A 5- fold cross validation is done to check the consistency of the model performance. Results of Decision tree classifier on the training and testing dataset is shown in Table 4.9.

Table 4.9: Performance Metrics for Decision Tree Model on training and testing dataset.

Metric	Training	Testing
Accuracy	0.928	0.908
Sensitivity	0.897	0.857
Specificity	0.940	0.931
Precision	0.876	0.83
F1-score	0.901	0.845
AUC	0.913	0.863

The tree is post-pruned at level five to identify the top five dominant features among the 30 used in the dataset. The pruned decision tree is shown in Figure 4.6.
The Decision Tree model applied to tumor prediction demonstrates high overall accuracy (90.8
While these metrics affirm the Decision Tree model as a suitable tool for tumor classification, there is room for improvement, especially in sensitivity to minimize missed tumor detections. Given its interpretability and ease of use, this model is valuable in clinical settings, where transparency is essential. However, incorporating ensemble methods like Random Forests may improve sensitivity without significantly impacting specificity, enhancing the model's reliability in a medical diagnostic context.
The initial model exhibited higher performance metrics, the cross-validation results provide a more robust and trustworthy evaluation, emphasizing the necessity of

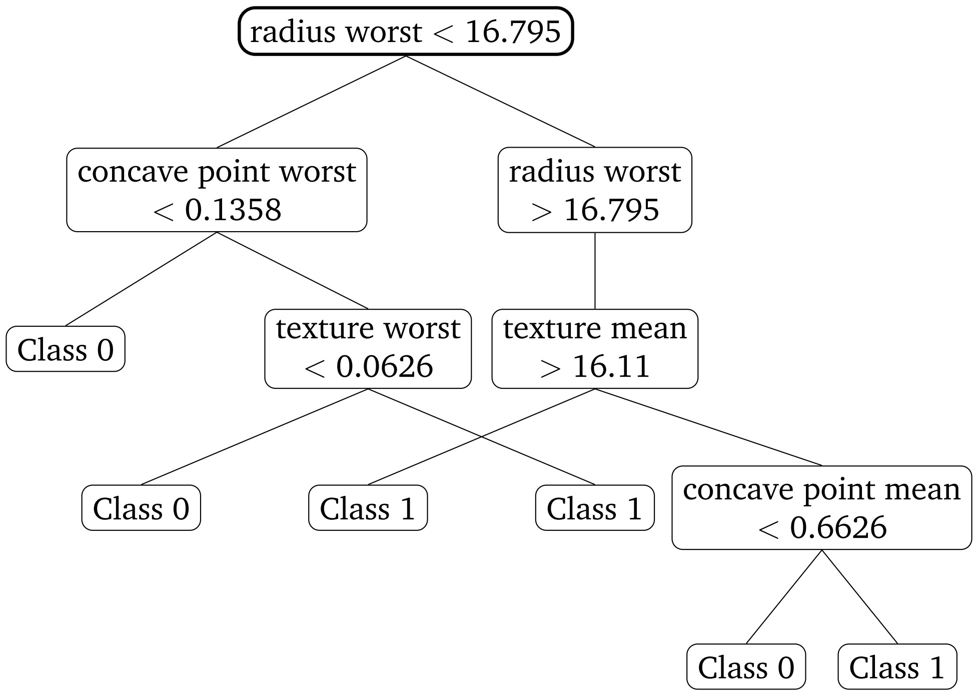


Figure 4.6: Post-pruned decision tree at level five.

employing rigorous validation methods to ensure the reliability of machine learning applications in medical diagnostics.
From the post-pruned decision tree shown in Figure 4.6, the top five dominant features are extracted and compared with those listed from the correlation analysis shown in Table 3.7. Result of this comparison is shown in Table 4.10.

Table 4.10: Comparison of dominant features identified from two approaches.

Rank	Decision Tree	Correlation Analysis
1	Radius worst	Concave point worst
2	Concave point worst	Area worst
3	Texture worst	Perimeter worst
4	Texture mean	Concave point mean
5	Concave point mean	Radius worst

The differences in feature rankings between the decision tree approach and feature-target correlation analysis stem from their distinct methodologies. Correlation analysis captures linear relationships, using metrics like Pearson's correlation coefficient to rank features based on how strongly they associate with the target variable. This method highlights features like *Concave point worst* that exhibit strong linear correlations. In contrast, decision trees employ information gain to identify features that significantly reduce uncertainty in classification, capturing complex, non-linear interactions. For instance, *Radius worst* ranks highest in the decision tree, indicating its critical role in distinguishing classes, which may not be evident through linear correlation alone.

These disparities emphasize the necessity of choosing analytical methods suited to the data's nature. While correlation analysis can provide initial insights into feature importance, it risks overlooking significant non-linear relationships essential for accurate classifications. By integrating both methods, researchers can enhance feature selection and model performance, ensuring a more comprehensive understanding of the factors influencing tumor classification. This approach ultimately leads to improved diagnostic accuracy in clinical settings, as it combines the strengths of both linear and non-linear analyses.

Both the decision tree and correlation analyses identify key features relevant to breast cancer classification. Radius worst and Concave point worst are significant in both rankings, with Radius worst leading in the decision tree analysis while Concave point worst tops the correlation analysis. Additionally, Concave point mean appears in both approaches, albeit in different ranks, highlighting its importance. Texture mean is also notable in the decision tree but is absent from the correlation analysis. These common features illustrate the complementary nature of the two methodologies in identifying influential variables for breast cancer prediction.

4.5 Skill of Soft Margin SVM with RKS Kernels

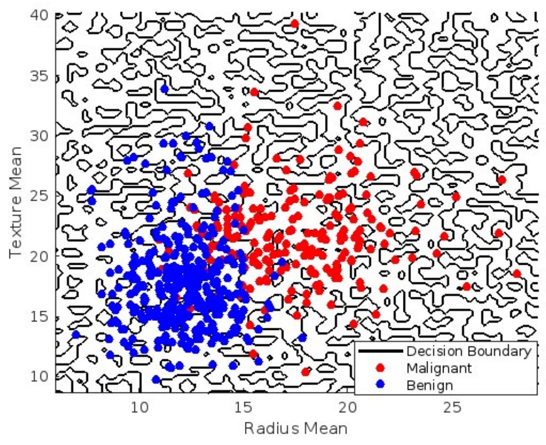
In this section, we present the outcomes of utilizing the Random Kitchen Sinks (RKS) kernel in conjunction with the soft margin Support Vector Machine (SVM) classifier. Building upon previous experiments that included logistic regression, its regularized variants, SVM with both hard and soft margins, and decision tree classifiers, our exploration of the RKS kernel revealed its remarkable capability to capture non-linear relationships in the data. The experiments demonstrated that the RKS kernel not only effectively handled non-linearly separable datasets but also significantly enhanced classification performance. The results indicated a substantial improvement in accuracy and robustness, underscoring the transformative potential of the RKS kernel for complex classification tasks.

From the experiments, it is found that the regularization parameter $C > 0$ has no significant effect outside $[0.5, 10]$. But the number of random features of RKS (dimension) has a potential impact in classification power of soft margin SVM. Table 4.11 shows this result.

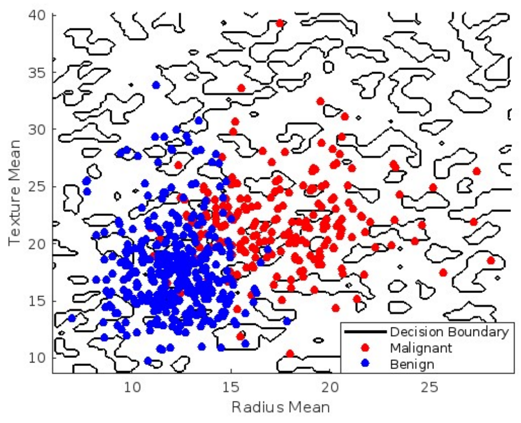
Table 4.11: Comparison of performance metrics of soft margin SVM with RKS kernel over the regularization parameter C.

Random Features	Performance Metrics							
	$C = 5$				$C = 10$			
d	Acc	Sen	Spec	F1	Acc	Sen	Sepc	F1
25	0.70	0.86	0.43	0.52	0.70	0.86	0.43	0.52
50	0.73	0.84	0.55	0.60	0.74	0.84	0.55	0.60
75	0.81	0.88	0.68	0.72	0.81	0.89	0.68	0.73
100	0.87	0.90	0.81	0.82	0.87	0.91	0.79	0.82
125	0.92	0.94	0.88	0.89	0.93	0.96	0.89	0.91
150	0.99	0.99	0.98	0.98	1.00	1.00	1.00	1.00
156	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

From Table 4.11, the soft margin SVM with RKS kernel is over-fitted. So consistency of this model is verified with a 5-fold cross validation. For various d values, the model shows very low performance measures. Even with $d = 900$ and $d = 1500$, the model produced the same performance measures. Separating boundaries of soft margin SVM with RKF kernel is shown in Figure 4.7.



(a) Distribution of data points over first two features under soft margin SVM (RKS, $d = 156$).



(b) Distribution of data points under cross validated soft margin SVM (RKS,$d = 1500$).

Figure 4.7: Distribution of data points over first two features under soft margin SVM with RKS kernel.

After many trials the maximum values for the average performance measures of the 5-fold cross validated soft margin SVM with RKS kernel is included in Table 4.12.

Thus, the RKS kernel allows for a practical and efficient approximation of the RBF kernel, particularly suitable for cases where $d \ll N$, bypassing the need for expensive Gram matrix operations while retaining high classification capability on non-linear datasets. This approach can, however, lead to under fitting if d is too small, as the random feature map may not adequately capture all the complex structures in the data, which could explain the observed decline in performance under 5-fold cross-validation with an insufficient basis size d.

4.6 Model Comparison and Conclusion

In this medical dataset, where diagnostic accuracy and sensitivity are paramount, several models were tested for their predictive performance. Table 4.12 summarizes the training results across different models, number of features used, and key performance metrics (accuracy, sensitivity, specificity, F1-score).

Table 4.12: Performance Metrics of Models

Algorithm	# Features	Accuracy	Sensitivity	Specificity	F1-score
Logistic	30	0.96	0.93	0.98	0.96
Logistic+L1	30	0.97	0.96	0.98	0.96
SVM (RBF)	30	1.00	1.00	1.00	1.00
SVM (RBF)	5	0.98	0.98	0.98	0.98
SVM (RBF, 5-fold)	5	0.93	0.92	0.94	0.91
SVM (RKS)	30	1.00	1.00	1.00	1.00
SVM (RKS,5-fold)	30	0.52	0.38	0.30	0.32
fitgm() + L1	30	0.97	0.96	0.98	0.96
fitsvm()	5	0.97	0.96	0.99	0.97
fitctree()	30	0.99	0.99	0.99	0.99
fitctree()+5-fold	30	0.92	0.89	0.94	0.90

4.6.1 Model Performance Analysis and Recommendations

Based on the performance metrics in Table 4.12, this section interprets each model's suitability for medical diagnosis by comparing their accuracy, sensitivity, specificity, and F1-score.

- **Superior Performance with SVM (RBF Kernel):** The *SVM with RBF kernel* achieves an accuracy, sensitivity, specificity, and F1-score of 1.00 across all metrics when using 30 features, suggesting exceptional capability in distinguishing classes effectively. When reduced to 5 features, this model still performs well (accuracy, sensitivity, specificity, and F1-score all at 0.98), demonstrating robustness in both feature-rich and reduced-feature contexts. However, under 5-fold cross-validation, its metrics drop to 0.93 accuracy, 0.92 sensitivity, 0.94 specificity, and 0.91 F1-score, indicating slight variability with increased complexity.

- **Instability in SVM with Random Kitchen Sink (RKS) Kernel:** Although the *SVM with RKS kernel* shows perfect scores (1.00 across all metrics) with 30 features in a non-cross-validated setting, it experiences a sharp performance decline under 5-fold cross-validation. The results fall to 0.52 accuracy, 0.38 sensitivity, 0.30 specificity, and 0.32 F1-score, reflecting substantial instability and limited generalization. While RKS can effectively capture non-linear separability in simple settings, the cross-validation results suggest it may be unreliable for clinical applications requiring consistent performance.

- **High Accuracy with Interpretability in fitctree Model:** The *fitctree model* demonstrates excellent performance (0.99 in accuracy, sensitivity, specificity, and F1-score) with all 30 features, indicating reliable classification ability. This decision tree model offers interpretability, as it ranks feature importance, a valuable trait for medical diagnostics where understanding feature influence is critical. While cross-validation reduces its metrics slightly (0.92 accuracy, 0.90 sensitivity, 0.94 specificity, and 0.90 F1-score), it remains robust, highlighting its value for transparent and interpretable diagnosis.
- **Feature-Efficient Alternatives:** Both *Logistic regression with L1 regularization* and the *fitsvm model* with 5 features demonstrate competitive results, with accuracies of 0.97 for Logistic+L1 and fitsvm, and F1-scores of 0.96 and 0.97, respectively. These models perform well with reduced feature sets, making them feasible options when computational efficiency or minimal feature use is essential.

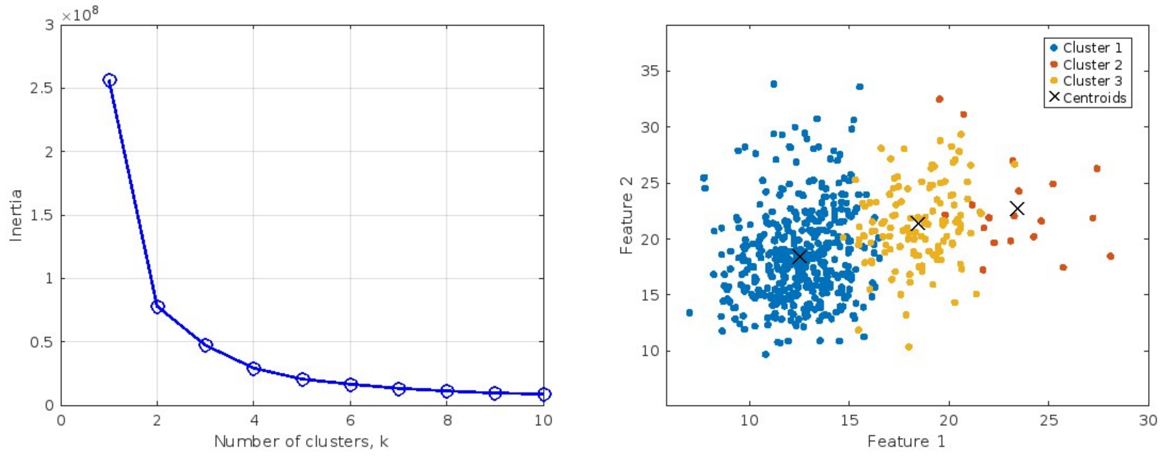
Recommendation: For reliable, interpretable medical diagnostics, the *fitctree model with all 30 features* is recommended for its high performance across all metrics and transparency in feature importance. If feature efficiency is prioritized, the *fitsvm model with 5 features* offers an effective trade-off between performance and simplicity.

4.7 Unsupervised Approach for Explainable Models

In medical diagnosis, especially with datasets involving various cancer indicators, binary classification might overly simplify the inherent patterns. The notion that cancers can vary in stages aligns with the understanding that medical conditions often manifest in gradients of severity rather than clear-cut classes. In this scenario, clustering could offer a richer segmentation of the data, possibly aligning with underlying stages or risk profiles of the condition, which binary classification may overlook.

4.7.1 Experiment Interpretation

In this experiment, the objective was to identify the optimal number of clusters for the breast cancer dataset using K-means clustering. A grid search cross validation approach is used to find optimal number of clusters. Figure 4.8a shows the distribution of inertia over number of clusters. Since the slope of the tangent of the inertia is not further decreased significantly after $k = 3$, optimal number of cluster is fixed to 3.



(a) Elbow plot demonstrating distribution of inertia over number of clusters.

(b) Distribution of data points under K-means clustering ($k = 3$).

Figure 4.8: Skill of K-means clustering model.

The analysis revealed that $k = 3$ is the most suitable number of clusters, as determined by the elbow method. This conclusion was supported by both the silhouette distance and the within-cluster sum of squares (WCSS) measures.
The minimum distance recorded during clustering is 4.726×10^{-7}, indicating a high level of compactness within the clusters. This suggests that the data points within each cluster are closely grouped together, reflecting the effectiveness of the clustering algorithm in delineating the data structure. Furthermore, the overall silhouette score of 0.6637 indicates a reasonable level of separation between the clusters. A silhouette score ranges from -1 to +1, where values closer to +1 denote well-defined clusters, values around 0 indicate overlapping clusters, and negative values suggest that data points may have been assigned to the wrong cluster. Therefore, a score of 0.6637 signifies that the clusters are relatively well-defined, although there is still room for improvement.
The support for the identified clusters showed a distribution of 425 points in Cluster 1, 123 points in Cluster 2, and 21 points in Cluster 3. This distribution indicates that the majority of the data points (approximately 74.69%) belong to Cluster 1, which may represent a predominant diagnosis category, potentially benign cases. Cluster 2 comprises 21.62% of the points, suggesting a secondary category, while Cluster 3, containing only 3.69% of the data, may represent an outlier group or a specific sub-type of malignant cases.
The subsequent phase of this analysis will involve examining the distribution of benign and malignant cases within the identified clusters. By evaluating how these diagnosis categories are represented in each cluster, we can gain insights into the relationship between clustering results and the underlying characteristics of the data. This analysis will be critical in determining the effectiveness of the clustering algorithm in distinguishing between benign and malignant cases and understanding the clinical implications of these findings.
Table 4.13 demonstrate the distribution of Benign and Malignant cases over the identified clusters through K-means clustering.

Table 4.13: Distribution of Benign and Malignant cases over clusters.

Cluster	Diagnosis				Total
	Benign		Malignant		
	Count	%	Count	%	
C1	355	83.53	70	16.47	425
C2	2	1.63	121	98.37	123
C3	0	0.00	21	100.0	21
Total cleaned samples					569

The clustering results shown in Table 4.13 reveal that Cluster 1 (C1) predominantly comprises benign cases, with 83.53% of the total 425 samples classified as benign. Conversely, Cluster 2 (C2) exhibits a significant majority of malignant cases, accounting for 98.37% of the 123 samples in that cluster. Notably, Cluster 3 (C3) consists entirely of malignant cases, indicating a critical subset of patients who may be at higher risk for severe breast cancer.

4.8 Conclusion

The Clustering approach enhances our understanding of patient categorization beyond traditional classification methods. By segmenting patients into distinct clusters based on their diagnosis, we can prioritize further treatment and intervention strategies for those identified within the higher-risk clusters. Specifically, patients in C2 and C3 should be closely monitored and provided with targeted treatment plans due to their higher likelihood of having malignant conditions.

In a public health context, these insights are invaluable for designing community treatment plans and executing an organized public health management system, particularly in highly populated countries where the incidence of breast cancer is rising. By identifying which patients and what fraction of patients face the threat of severe breast cancer, healthcare administrators can allocate resources more effectively, ensuring that high-risk populations receive timely and appropriate care.

In conclusion, the incorporation of clustering analysis provides a prescriptive add-on to classification models. It allows us to go beyond merely determining whether a person is a cancer patient or not; it aids in understanding the severity of their condition and facilitates a more nuanced approach to patient management. This will ultimately contribute to improved health outcomes through tailored public health initiatives.

4.9 Future Directions and Sustainable Impact of Explainable Machine Learning in Medical Diagnosis

Based on the performance metrics in Table 4.12, we can interpret each model's suitability for medical diagnosis by comparing their accuracy, sensitivity, specificity, and F1-score:

- **Superior Performance with SVM (RBF Kernel):** The *SVM with RBF kernel* achieves an accuracy, sensitivity, specificity, and F1-score of 1.00 across all metrics when using 30 features, suggesting exceptional capability in distinguishing classes effectively. When reduced to 5 features, this model still performs well (accuracy, sensitivity, specificity, and F1-score all at 0.98), demonstrating robustness in both feature-rich and reduced-feature contexts. However, under 5-fold cross-validation, its metrics drop to 0.93 accuracy, 0.92 sensitivity, 0.94 specificity, and 0.91 F1-score, indicating slight variability with increased complexity.

- **Instability in SVM with Random Kitchen Sink (RKS) Kernel:** Although the *SVM with RKS kernel* shows perfect scores (1.00 across all metrics) with 30 features in a non-cross-validated setting, it experiences a sharp performance decline under 5-fold cross-validation. The results fall to 0.52 accuracy, 0.38 sensitivity, 0.30 specificity, and 0.32 F1-score, reflecting substantial instability and limited generalization. While RKS can effectively capture non-linear separability in simple settings, the cross-validation results suggest it may be unreliable for clinical applications requiring consistent performance.

- **High Accuracy with Interpretability in fitctree Model:** The *fitctree model* demonstrates excellent performance (0.99 in accuracy, sensitivity, specificity, and F1-score) with all 30 features, indicating reliable classification ability. This decision tree model offers interpretability, as it ranks feature importance, a valuable trait for medical diagnostics where understanding feature influence is critical. While cross-validation reduces its metrics slightly (0.92 accuracy, 0.90 sensitivity, 0.94 specificity, and 0.90 F1-score), it remains robust, highlighting its value for transparent and interpretable diagnosis.

- **Feature-Efficient Alternatives:** Both *Logistic regression with L1 regularization* and the *fitsvm model* with 5 features demonstrate competitive results, with accuracies of 0.97 for Logistic+L1 and fitsvm, and F1-scores of 0.96 and 0.97, respectively. These models perform well with reduced feature sets, making them feasible options when computational efficiency or minimal feature use is essential.

Chapter 5

Conclusions

This project work has outlined a comprehensive, mathematically grounded framework for classifying breast tissue as benign or malignant using the UCI Breast Cancer Wisconsin dataset. Addressing a critical need for precise diagnostics in oncology, our research combined supervised and unsupervised machine learning models with careful data preprocessing and robust optimization techniques, resulting in both high accuracy and interpretability.

Our methodology commenced with an extensive preprocessing phase to enhance data quality and model reliability. Data imbalances, multicollinearity, and outliers were carefully managed using statistical techniques such as the Interquartile Range (IQR) method and square root transformations, which refined the feature space. This allowed us to construct models on a foundation of clean, normalized data, reducing the risk of model bias and enhancing the accuracy of predictive insights.

A diverse set of classification models was employed, including logistic regression, support vector machines (SVM), and decision trees. Logistic regression served as a robust baseline, effectively capturing linear relationships between features and the target variable. However, SVM proved particularly valuable due to its formulation as a convex optimization problem, enabling it to maximize the margin between classes while maintaining computational efficiency. This approach, solved using Lagrange multipliers, was especially effective in high-dimensional spaces where data separability is complex. The use of both linear and non-linear kernels in SVM further allowed the model to capture intricate patterns within the dataset, achieving significant diagnostic accuracy.

Incorporating decision trees added interpretability, making it easier to understand feature importance—an aspect critical to healthcare professionals who require transparency in model decision-making. By using ensemble methods, such as Random Forest, we further improved model stability, reducing overfitting risks and offering consistent classification results across different subsets of data. This blend of models underscored the importance of balancing complexity and interpretability, a trade-off that is essential in applications where the insights must be understandable and actionable.

Additionally, K-means clustering was introduced as an unsupervised learning approach, supplementing classification by revealing underlying patterns and group structures. Through clustering, we were able to compare cluster assignments with actual diagnoses, providing valuable insights into misclassified cases and highlighting clusters that exhibited similar tumor characteristics. This dual approach of classification and clustering augmented the overall interpretability and reliability of the model outcomes, allowing for a multi-layered diagnostic tool that could aid clinicians in detecting cancerous tissue with greater confidence.

The results demonstrated that SVM with non-linear kernels and regularization techniques delivered the highest accuracy, effectively classifying malignant versus benign cases. Meanwhile, logistic regression provided a more interpretable baseline for diagnosis, making it suitable as a complementary model for quick assessments. The clustering analysis supplemented these findings, enabling a more nuanced understanding of how different features influence classification and adding an extra dimension of diagnostic precision.

Recommendation: For reliable, interpretable medical diagnostics, the *fitctree model with all 30 features* is recommended for its high performance across all metrics and transparency in feature importance. If feature efficiency is prioritized, the *fitsvm model with 5 features* offers an effective trade-off between performance and simplicity.

Building on the insights derived from clustering and the aforementioned supervised learning methods, there lies a significant opportunity to explore Explainable Machine Learning models. These models can elucidate the reasoning behind patient classifications, detailing why a patient is considered affected, the stage of cancer they may be in, and the implications for their treatment. By providing interpretable and actionable insights, Explainable ML not only augments clinical decision-making but also empowers healthcare professionals to deliver personalized patient care.

In the context of public health management, the adoption of informed diagnostic tools based on Explainable ML can greatly enhance the treatment and management of health systems, particularly for underserved populations. It facilitates a comprehensive understanding of patient needs, enabling targeted interventions that are both efficient and equitable. This informed approach to diagnosis not only improves patient outcomes but also supports the sustainability of healthcare systems by optimizing resource allocation and reducing unnecessary procedures.

Moreover, the alignment of such research initiatives with the United Nations' Sustainable Development Goals (SDGs) is crucial. Specifically, this work contributes to SDG 3 (Good Health and Well-Being) by promoting health equity and ensuring that all individuals have access to essential health services. The integration of Explainable ML in medical education can foster a new generation of healthcare professionals who are adept at leveraging data-driven insights while maintaining a patient-centered approach. This transition is pivotal for achieving not only individual health improvements but also broader systemic changes that promote public health sustainability.

In conclusion, the incorporation of Explainable Machine Learning in medical diagnosis paves the way for innovative solutions that prioritize patient welfare and health system efficiency. By addressing the complexities of cancer diagnosis and treatment, we can make significant strides toward fulfilling the SDGs, particularly in the realms of health equity, education, and sustainable public health management.

Appendix

`Matlab` live script used for computational work of the project is included in this appendix section.

Implementation of Logistic Regression

```
% Load your dataset
data = readtable('transformed_data.csv'); % Replace with your actual dataset
path
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Define the target variable
target = categorical(data.diagnosis); % Assuming 'diagnosis' is your target
variable
targetNumeric = double(target) - 1; % Convert to numeric for correlation
analysis

% Define numerical variables (features)
numericalVars = data.Properties.VariableNames(3:end); % Exclude 'id' and
'diagnosis'
numFeatures = length(numericalVars);

% Initialize correlation stats
correlationStats = zeros(numFeatures, 1);
pValues = zeros(numFeatures, 1);

% Loop through each numerical variable to compute statistics
for i = 1:numFeatures
    [correlationStats(i), pValues(i)] = corr(data{:, numericalVars{i}},
targetNumeric, 'Type', 'Pearson');
end

% Create a result table for correlation results
resultTable = table(numericalVars', correlationStats, pValues, ...
    'VariableNames', {'Feature', 'Correlation', 'PValue'});

% Sort by p-value
resultTable = sortrows(resultTable, 'PValue');

% Display correlation results
disp(resultTable);
```

```
                Feature               Correlation      PValue   
    _________________________________    ___________    ___________

    {'concavePointsWorst'   }       0.79357      1.9691e-124
    {'areaWorst'            }       0.78426      1.2259e-119
    {'perimeterWorst'       }       0.78291      5.7714e-119
    {'concavePointsMean'    }       0.77661      7.1012e-116
    {'radiusWorst'          }       0.77645      8.4823e-116
    {'perimeterMean'        }       0.74264     8.4363e-101
    {'areaMean'             }        0.7336      3.4529e-97
    {'radiusMean'           }       0.73003      8.4659e-96
    {'areaSe'               }       0.71096      9.5164e-89
    {'concavityMean'        }       0.69636      9.9666e-84
    {'concavityWorst'       }       0.65961      2.4647e-72
    {'perimeterSe'          }       0.62984      3.1983e-64
    {'radiusSe'             }       0.62694      1.7717e-63
    {'compactnessMean'      }       0.59653      3.9383e-56
    {'compactnessWorst'     }         0.591      7.0698e-55
    {'textureWorst'         }        0.4569      1.0781e-30
    {'smoothnessWorst'      }       0.42146      6.5751e-26
    {'symmetryWorst'        }       0.41629      2.9511e-25
    {'textureMean'          }       0.41519      4.0586e-25
    {'concavePointsSe'      }       0.40804      3.0723e-24
    {'smoothnessMean'       }       0.35856      1.0519e-18
    {'symmetryMean'         }        0.3305      5.7334e-16
    {'fractalDimensionWorst'}       0.32356      2.4724e-15
    {'compactnessSe'        }         0.293       9.976e-13
    {'concavitySe'          }       0.25373      8.2602e-10
    {'fractalDimensionSe'   }      0.077972        0.063074
    {'smoothnessSe'         }     -0.067016          0.1103
    {'fractalDimensionMean' }     -0.012838         0.75994
    {'textureSe'            }    -0.0083033         0.84333
    {'symmetrySe'           }    -0.0065218         0.87664
```

```matlab
% Select features with correlation greater than 0.75
threshold = 0.2;
selectedFeatures = resultTable(abs(resultTable.Correlation) > threshold, :);

% Fit logistic regression using the selected features
if ~isempty(selectedFeatures)
    % Extracting the features to be used for fitting the model
    X = data{:, selectedFeatures.Feature}; % Selected features as matrix
    Y = target; % Target variable

    % Fit the logistic regression model
    mdl = fitglm(X, Y, 'Distribution', 'binomial', 'Link', 'logit');

    % Display the model summary
    disp(mdl);

    % Cross-validation to calculate classification accuracy
    k = 5; % Number of folds
    cv = cvpartition(Y, 'KFold', k);
    accuracy = zeros(k, 1);
    sensitivity = zeros(k, 1);
    specificity = zeros(k, 1);
    rocAUC = zeros(k, 1);

    for i = 1:cv.NumTestSets
        trainIdx = cv.training(i);
        testIdx = cv.test(i);
        % Train the model
        mdlTrain = fitglm(X(trainIdx, :), Y(trainIdx), 'Distribution',
'binomial', 'Link', 'logit');

        % Predict using the model
        probabilities = predict(mdlTrain, X(testIdx, :)); % Get predicted
probabilities
        predictions = probabilities > 0.5; % Classify using 0.5 threshold

        % Convert predictions to categorical for accuracy comparison
        predictionsCategorical = categorical(predictions, [0 1], {'0',
'1'}); % Adjust labels as needed

        % Calculate accuracy
        accuracy(i) = sum(predictionsCategorical == Y(testIdx)) /
length(Y(testIdx));

        % Confusion matrix
        cm = confusionmat(Y(testIdx), predictionsCategorical);

        % Calculate sensitivity and specificity
        TP = cm(2, 2); % True Positive
        TN = cm(1, 1); % True Negative
        FP = cm(1, 2); % False Positive
        FN = cm(2, 1); % False Negative

        sensitivity(i) = TP / (TP + FN); % Recall
        specificity(i) = TN / (TN + FP); % True Negative Rate

        % Calculate ROC AUC
        [~, ~, ~, rocAUC(i)] = perfcurve(Y(testIdx), probabilities, '1');
    end

% Display average metrics
   disp(['Average Accuracy: ', num2str(mean(accuracy))]);
   disp(['Average Sensitivity: ', num2str(mean(sensitivity))]);
   disp(['Average Specificity: ', num2str(mean(specificity))]);
   disp(['Average ROC AUC: ', num2str(mean(rocAUC))]);
   % Display the confusion matrix for the last fold
   disp('Confusion Matrix for the last fold:');
   disp(cm);
else
    disp('No features found with correlation greater than the threshold.');
end
```

```
Generalized linear regression model:
    logit(P(y='1')) ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13 + x14 + x15 +
    Distribution = Binomial

Estimated Coefficients:
                   Estimate      SE        tStat       pValue
                   _________    ________    _________    _________

    (Intercept)     97.279      204.71      0.47521      0.63464
    x1             -27.193      52.955     -0.51351       0.6076
    x2             -22.713      49.596     -0.45796      0.64698
    x3             0.04409     0.62112     0.070985      0.94341
    x4              122.27      121.45       1.0068      0.31405
    x5              6.7842      9.8969      0.68549      0.49303
    x6             0.76241      2.3512      0.32427      0.74574
    x7             -39.071      73.362     -0.53258      0.59432
    x8              -1.967      17.401     -0.11304         0.91
    x9              52.261       33.58       1.5563      0.11963
    x10             58.215      91.208      0.63827       0.5233
    x11             9.2485      21.036      0.43965      0.66019
    x12            -8.8841      28.963     -0.30674      0.75904
    x13            -112.26      112.83     -0.99495      0.31976
    x14            -173.37      118.67       -1.461      0.14401
    x15             44.664      39.956       1.1178      0.26364
    x16            0.38793     0.27789        1.396      0.16272
    x17             29.864      63.631      0.46933      0.63884
    x18             40.696      23.204       1.7538     0.079458
    x19            0.17113     0.38312      0.44667      0.65511
    x20             938.85      448.45       2.0935     0.036302
    x21             40.078      123.57      0.32433      0.74569
    x22            -49.046      40.693      -1.2053       0.2281
    x23             13.328      58.602      0.22744      0.82008
    x24            -310.97      167.52      -1.8563     0.063404
    x25            -83.731      78.106       -1.072      0.28371


569 observations, 543 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 709, p-value = 1.89e-133
Warning: Iteration limit reached.
Warning: The estimated coefficients perfectly separate failures from successes. This means the
theoretical best estimates are not finite. For the fitted linear combination XB of the predictors,
the sample proportions P of Y=N in the data satisfy:
   XB<-0.159441: P=0
   XB>-0.159441: P=1
To get proper estimates, fit with 'LikelihoodPenalty' set to 'jeffreys-prior'.
Warning: Iteration limit reached.
Warning: The estimated coefficients perfectly separate failures from successes. This means the
theoretical best estimates are not finite. For the fitted linear combination XB of the predictors,
the sample proportions P of Y=N in the data satisfy:
   XB<-0.201784: P=0
   XB>-0.201784: P=1
To get proper estimates, fit with 'LikelihoodPenalty' set to 'jeffreys-prior'.
Average Accuracy: 0.95606
Average Sensitivity: 0.94773
Average Specificity: 0.96092
Average ROC AUC: 0.97152
Confusion Matrix for the last fold:
    69     2
     0    43
```

Modularizing the code for Generalization

```
function [accuracy, sensitivity, specificity, rocAUC, errorRate, cm] =
evaluateLogisticRegression(X, Y)
```

```
    % This function evaluates a logistic regression model on the given
dataset.
    % Input:
    %   X - Feature matrix (numeric)
    %   Y - Target variable (categorical or numeric)
    % Output:
    %   accuracy - Classification accuracy
    %   sensitivity - True positive rate
    %   specificity - True negative rate
    %   rocAUC - Area under the ROC curve
    %   errorRate - Classification error rate
    %   cm - Confusion matrix

    % Train the logistic regression model on the entire dataset
    mdl = fitglm(X, Y, 'Distribution', 'binomial', 'Link', 'logit');

    % Predict probabilities for the entire dataset
    probabilities = predict(mdl, X); % Get predicted probabilities
    predictions = probabilities > 0.5; % Classify using 0.5 threshold

    % Convert predictions to categorical for accuracy comparison
    predictionsCategorical = categorical(predictions, [0 1], {'0', '1'}); %
Adjust labels as needed

    % Calculate accuracy
    accuracy = sum(predictionsCategorical == Y) / length(Y);

    % Confusion matrix
    cm = confusionmat(Y, predictionsCategorical);

    % Calculate sensitivity and specificity
    TP = cm(2, 2); % True Positive
    TN = cm(1, 1); % True Negative
    FP = cm(1, 2); % False Positive
    FN = cm(2, 1); % False Negative

    sensitivity = TP / (TP + FN); % Recall
    specificity = TN / (TN + FP); % True Negative Rate

    % Calculate ROC AUC
    [Xroc, Yroc, T, rocAUC] = perfcurve(Y, probabilities, '1');

    % Plot ROC curve
    figure;
    plot(Xroc, Yroc, 'LineWidth', 2);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title(['ROC Curve (AUC = ', num2str(rocAUC), ')']);
    grid on;
    axis square;
    % Calculate error rate
    errorRate = 1 - accuracy; % 1 - Accuracy
end
```

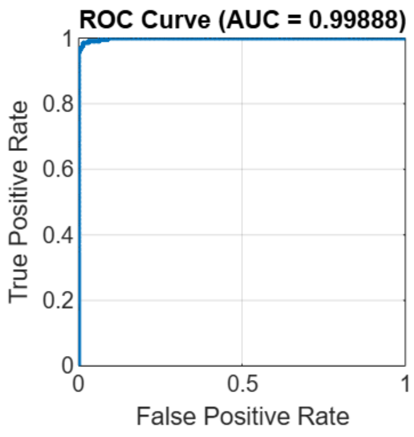
```
% calling function to resport classification perfomrnace measure
% Load transformed dataset
%data = readtable('transformed_data.csv');
data = readtable('transformed_data.csv'); % Replace with your actual dataset
path
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Define the target variable
target = categorical(data.diagnosis); % Assuming 'diagnosis' is your target
variable
targetNumeric = double(target) - 1; % Convert to numeric for correlation
analysis

% Define numerical variables (features)
numericalVars = data.Properties.VariableNames(3:end); % Exclude 'id' and
'diagnosis'
numFeatures = length(numericalVars);

% Initialize correlation stats
correlationStats = zeros(numFeatures, 1);
pValues = zeros(numFeatures, 1);

% Loop through each numerical variable to compute statistics
for i = 1:numFeatures
    [correlationStats(i), pValues(i)] = corr(data{:, numericalVars{i}},
targetNumeric, 'Type', 'Pearson');
end

% Create a result table for correlation results
resultTable = table(numericalVars', correlationStats, pValues, ...
    'VariableNames', {'Feature', 'Correlation', 'PValue'});

% Sort by p-value
resultTable = sortrows(resultTable, 'PValue');

% Display correlation results
disp(resultTable);
```

```
             Feature                Correlation      PValue
    _______________________      _____________   ____________
    {'concavePointsWorst'   }       0.79357     1.9691e-124
    {'areaWorst'            }       0.78426     1.2259e-119
    {'perimeterWorst'       }       0.78291     5.7714e-119
    {'concavePointsMean'    }       0.77661     7.1012e-116
    {'radiusWorst'          }       0.77645     8.4823e-116
    {'perimeterMean'        }       0.74264     8.4363e-101
    {'areaMean'             }        0.7336      3.4529e-97
    {'radiusMean'           }       0.73003      8.4659e-96
    {'areaSe'               }       0.71096      9.5164e-89
    {'concavityMean'        }       0.69636      9.9666e-84
    {'concavityWorst'       }       0.65961      2.4647e-72
    {'perimeterSe'          }       0.62984      3.1983e-64
    {'radiusSe'             }       0.62694      1.7717e-63
    {'compactnessMean'      }       0.59653      3.9383e-56
    {'compactnessWorst'     }         0.591      7.0698e-55
    {'textureWorst'         }        0.4569      1.0781e-30
    {'smoothnessWorst'      }       0.42146      6.5751e-26
    {'symmetryWorst'        }       0.41629      2.9511e-25
    {'textureMean'          }       0.41519      4.0586e-25
    {'concavePointsSe'      }       0.40804      3.0723e-24
    {'smoothnessMean'       }       0.35856      1.0519e-18
    {'symmetryMean'         }        0.3305      5.7334e-16
    {'fractalDimensionWorst'}       0.32356      2.4724e-15
    {'compactnessSe'        }         0.293       9.976e-13
    {'concavitySe'          }       0.25373      8.2602e-10
    {'fractalDimensionSe'   }      0.077972        0.063074
    {'smoothnessSe'         }     -0.067016          0.1103
    {'fractalDimensionMean' }     -0.012838         0.75994
    {'textureSe'            }    -0.0083033         0.84333
    {'symmetrySe'           }    -0.0065218         0.87664
```

```
% Select features with correlation greater than 0.75
threshold = 0.2;
selectedFeatures = resultTable(abs(resultTable.Correlation) > threshold, :);

% Fit logistic regression using the selected features
if ~isempty(selectedFeatures)
    % Extracting the features to be used for fitting the model
    X = data{:, selectedFeatures.Feature}; % Selected features as matrix
    Y = target; % Target variable
%X = data{:, 3:end}; % Assuming features are in all columns except the last
%Y = categorical(data.diagnosis); % Assuming 'diagnosis' is your target
variable

% Call the function
    [accuracy, sensitivity, specificity, rocAUC, errorRate, cm] =
evaluateLogisticRegression(X, Y);

    % Display results
    disp(['Accuracy: ', num2str(accuracy)]);
    disp(['Sensitivity: ', num2str(sensitivity)]);
    disp(['Specificity: ', num2str(specificity)]);
    disp(['ROC AUC: ', num2str(rocAUC)]);
    disp(['Error Rate: ', num2str(errorRate)]);
    disp('Confusion Matrix:');
    disp(cm);
else
    disp('No features found with correlation greater than the threshold.');
end
```

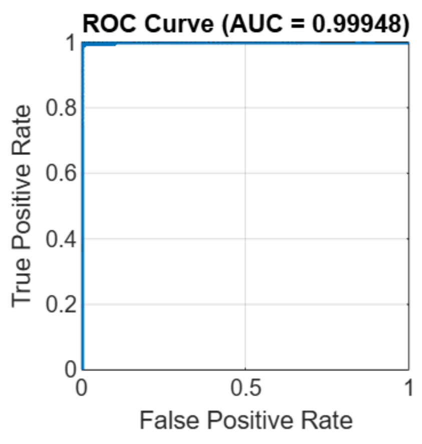


```
Accuracy: 0.98243
Sensitivity: 0.9717
Specificity: 0.9888
ROC AUC: 0.99888
Error Rate: 0.017575
Confusion Matrix:
   353     4
     6   206
```

```
% without using feature selection

% Load your dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, 3:end}; % Assuming features are in all columns except the last
Y = categorical(data.diagnosis); % Assuming 'diagnosis' is your target
variable

% Call the function
[accuracy, sensitivity, specificity, rocAUC, errorRate, cm] =
evaluateLogisticRegression(X, Y);
```



```
% Display results
disp(['Accuracy: ', num2str(accuracy)]);
```

```
Accuracy: 0.99649
```

```
disp(['Sensitivity: ', num2str(sensitivity)]);
```

```
Sensitivity: 0.99057
```

```
disp(['Specificity: ', num2str(specificity)]);
```

```
Specificity: 1
```

```
disp(['ROC AUC: ', num2str(rocAUC)]);
```

```
ROC AUC: 0.99948
```

```
disp(['Error Rate: ', num2str(errorRate)]);
```

```
Error Rate: 0.0035149
```

```
disp('Confusion Matrix:');
```

```
Confusion Matrix:
```

```
disp(cm);
```

```
   357     0
     2   210
```

Logistic Regression from the Scratch

Classifier with Constant treshold using linear regression

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
```

```
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Assume the last column is the target variable and the rest are features
X = data{:, 3:end}; % Features
y = data.diagnosis;     % Target variable (0s and 1s)

% Add a bias term (intercept) to the feature matrix
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the bias

% Calculate the weight vector W using the closed-form solution of linear
regression
W = (X' * X) \ (X' * y); % Weight vector calculation

% Make predictions using the logistic regression model
% Logistic function
logistic_function = @(z) 1 ./ (1 + exp(-z));

% Calculate probabilities
%probabilities = logistic_function(X * W);
probabilities = X * W;

% Classify samples based on the threshold of 0.75
threshold = 0.0;
predictions = probabilities >= threshold;

% Convert logical array to numeric class labels (0 or 1)
predicted_classes = double(predictions);

% Display results
%disp('Predicted Classes:');
%disp(predicted_classes);

% Calculate performance metrics
accuracy = sum(predicted_classes == y) / length(y);
sensitivity = sum(predicted_classes(y == 1) == 1) / sum(y == 1);
specificity = sum(predicted_classes(y == 0) == 0) / sum(y == 0);
AUC = trapz(sort(probabilities), cumsum(sort(predicted_classes)) /
sum(predicted_classes));

fprintf('Accuracy: %.2f%%\n', accuracy * 100);
```

```
Accuracy: 56.59%
```

```
fprintf('Sensitivity: %.2f%%\n', sensitivity * 100);
```

```
Sensitivity: 100.00%
```

```
fprintf('Specificity: %.2f%%\n', specificity * 100);
```

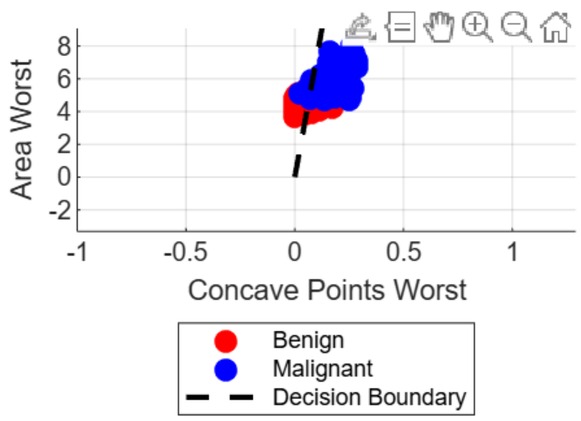
```
Specificity: 30.81%
```

```
fprintf('AUC: %.2f\n', AUC);
```

```
AUC: 0.98
```

Ploting the decision boundary over dominant features

```
% Extract features and target variable
X = data{:, {'concavePointsWorst', 'areaWorst'}};
y = data.diagnosis; % Assuming 'target' is the name of your target variable

% Calculate weights W using your previous method (this is just an example)
% Assuming W is calculated from your kernel regression or logistic regression
%W = [0.5; 0.3]; % Example weights; replace with your actual calculated
weights

% Create a figure for plotting
figure;
hold on;

% Plot the data points
gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled');

% Define the decision boundary
% W(1)*x1 + W(2)*x2 = 0  ->  x2 = - (W(1)/W(2)) * x1
x1_range = linspace(min(X(:,1)), max(X(:,1)), 100);
x2_boundary = - (W(29)/W(24)) * x1_range;

% Plot the decision boundary
plot(x1_range, x2_boundary, 'k--', 'LineWidth', 2);
% Set axes limits
xlim([min(X(:,1)) - 1, max(X(:,1)) + 1]); % Set limits for x-axis
ylim([min(X(:,2)) - 7, max(X(:,2)) + 1]); % Set limits for y-axis
% Add labels and legend
xlabel('Concave Points Worst');
ylabel('Area Worst');
%title('Sigmoid Decision Boundary vs Data Points');
legend('Benign', 'Malignant', 'Decision Boundary', 'Location',
'southoutside');
grid on;
hold off;
```



Decision Boundary with sigmoid

```
% Assume the last column is the target variable and the rest are features
X = data{:, 3:end}; % Features
y = data.diagnosis;     % Target variable (0s and 1s)

% Add a bias term (intercept) to the feature matrix
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the bias

% Calculate the weight vector W using the closed-form solution of linear
regression
W = (X' * X) \ (X' * y); % Weight vector calculation

% Make predictions using the logistic regression model
% Logistic function
logistic_function = @(z) 1 ./ (1 + exp(-z));

% Calculate probabilities
probabilities = logistic_function(X * W);
% Classify samples based on the threshold of 0.75
threshold = 0.5;
predictions = probabilities >= threshold;

% Convert logical array to numeric class labels (0 or 1)
predicted_classes = double(predictions);

% Display results
%disp('Predicted Classes:');
%disp(predicted_classes);

% Calculate performance metrics
accuracy = sum(predicted_classes == y) / length(y);
sensitivity = sum(predicted_classes(y == 1) == 1) / sum(y == 1);
specificity = sum(predicted_classes(y == 0) == 0) / sum(y == 0);
```

```
AUC = trapz(sort(probabilities), cumsum(sort(predicted_classes)) /
sum(predicted_classes));
% F1 Score
F1_score = 2 * (specificity * sensitivity) / (specificity + sensitivity);
fprintf('F1 Score: %.2f\n', F1_score);
```

```
F1 Score: 0.47
```

```
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
```

```
Accuracy: 56.59%
```

```
fprintf('Sensitivity: %.2f%%\n', sensitivity * 100);
```

```
Sensitivity: 100.00%
```

```
fprintf('Specificity: %.2f%%\n', specificity * 100);
```

```
Specificity: 30.81%
```

```
fprintf('F1 score: %.2f%%\n', F1_score * 100);
```

```
F1 score: 47.11%
```

```
fprintf('AUC: %.2f\n', AUC);
```

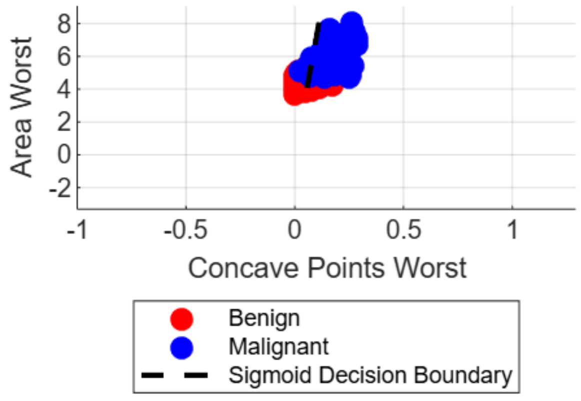
```
AUC: 0.20
```

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```


% Extract features and target variable
X = data{:, {'concavePointsWorst', 'areaWorst'}};
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Calculate weights W using your previous method (this is just an example)
% Assuming W is calculated from your kernel regression or logistic regression
% Replace this with your actual calculated weights
%W = [0.5; 0.3]; % Example weights; ensure to replace with your actual
calculated weights

% Create a figure for plotting
figure;
hold on;

% Plot the data points
```

```
gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled');

% Define a grid for plotting the sigmoid decision boundary
x1_range = linspace(min(X(:,1)), max(X(:,1)), 100);
x2_range = linspace(min(X(:,2)), max(X(:,2)), 100);
[X1, X2] = meshgrid(x1_range, x2_range);
Z = W(29) * X1 + W(24) * X2; % Calculate the linear combination W^T x
Z_sigmoid = 1 ./ (1 + exp(-Z)); % Apply the sigmoid function

% Plot the decision boundary where the sigmoid output is 0.5
contour(X1, X2, Z_sigmoid, [0.5 0.5], 'k--', 'LineWidth', 2);
%surf(X1, X2, Z_sigmoid, 'EdgeColor', 'none', 'FaceAlpha', 0.5);
% Set axes limits
xlim([min(X(:,1)) - 1, max(X(:,1)) + 1]); % Set limits for x-axis
ylim([min(X(:,2)) - 7, max(X(:,2)) + 1]); % Set limits for y-axis
% Add labels and legend
xlabel('Concave Points Worst');
ylabel('Area Worst');
%title('Sigmoid Decision Boundary vs Data Points');
legend('Benign', 'Malignant', 'Sigmoid Decision Boundary', 'Location',
'southoutside');
grid on;
hold off;
```



Improving the logistic regression with Gradient Descent

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Extract features and target variable
```

```
%X = data{:, {'concavePointsWorst', 'areaWorst'}};
X = data{:, 3:end};
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Add a bias term to X
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Initialize parameters
learning_rate = 0.01;
num_iterations = 1000;
num_samples = size(X, 1);
W = zeros(size(X, 2), 1); % Initialize weights

% Gradient descent loop
for iter = 1:num_iterations
    % Calculate the predictions
    z = X * W; % Linear combination
    predictions = 1 ./ (1 + exp(-z)); % Sigmoid function

    % Compute the gradient using the derivative of the sigmoid
    error = predictions - y; % Difference between predictions and actual
    gradient = (X' * (error .* predictions .* (1 - predictions))) /
num_samples; % Correct gradient

    % Update weights
    W = W - learning_rate * gradient; % Gradient descent step
end

% Compute final probabilities and classify
final_probabilities = 1 ./ (1 + exp(-X * W)); % Get probabilities
final_predictions = final_probabilities >= 0.5; % Classification based on
threshold

% Calculate accuracy
accuracy = sum(final_predictions == y) / num_samples;
fprintf('Accuracy from scratch implementation: %.2f%%\n', accuracy * 100);
```

```
Accuracy from scratch implementation: 62.74%
```

```
% Calculate sensitivity, specificity, and F1 score
TP = sum((final_predictions == 1) & (y == 1)); % True Positives
TN = sum((final_predictions == 0) & (y == 0)); % True Negatives
FP = sum((final_predictions == 1) & (y == 0)); % False Positives
FN = sum((final_predictions == 0) & (y == 1)); % False Negatives

% Sensitivity (Recall)
sensitivity = TP / (TP + FN);
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 0.00
```

```
% Specificity
specificity = TN / (TN + FP);
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 1.00
```

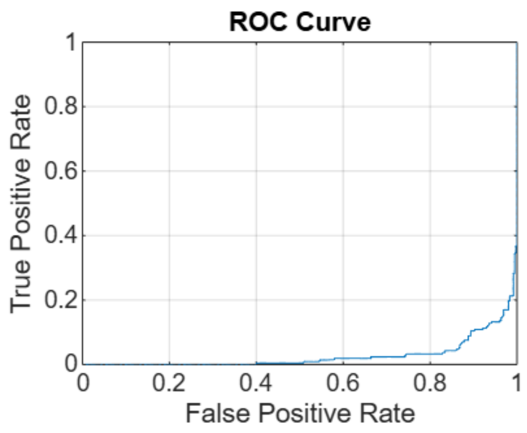
```
% Precision
precision = TP / (TP + FP);

% F1 Score
F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
fprintf('F1 Score: %.2f\n', F1_score);
```

```
F1 Score: NaN
```

```
% Calculate AUC
[X_ROC, Y_ROC, ~, AUC] = perfcurve(y, final_probabilities, 1); % 1 for
positive class
fprintf('AUC: %.2f\n', AUC);
```

```
AUC: 0.03
```

```
% Optionally plot ROC curve
figure;
plot(X_ROC, Y_ROC);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
grid on;
```



Logistic regression with normalized features

```
% Load the data
```

```
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, 3:end}; % Adjust the column indices as needed
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standard scaling (normalization)
mean_X = mean(X); % Compute mean of each feature
std_X = std(X);   % Compute standard deviation of each feature
X = (X - mean_X) ./ std_X; % Normalize features

% Add a bias term to X
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Initialize parameters
learning_rate = 0.01;
num_iterations = 1000;
num_samples = size(X, 1);
W = zeros(size(X, 2), 1); % Initialize weights

% Gradient descent loop
for iter = 1:num_iterations
    % Calculate the predictions
    z = X * W; % Linear combination
    predictions = 1 ./ (1 + exp(-z)); % Sigmoid function

    % Compute the gradient using the derivative of the sigmoid
    error = predictions - y; % Difference between predictions and actual
    gradient = (X' * (error .* predictions .* (1 - predictions))) /
num_samples; % Correct gradient

    % Update weights
    W = W - learning_rate * gradient; % Gradient descent step
end

% Compute final probabilities and classify
final_probabilities = 1 ./ (1 + exp(-X * W)); % Get probabilities
final_predictions = final_probabilities >= 0.5; % Classification based on
threshold

% Calculate accuracy
accuracy = sum(final_predictions == y) / num_samples;
fprintf('Accuracy from scratch implementation: %.2f%%\n', accuracy * 100);
```

```
Accuracy from scratch implementation: 96.84%
```

```
% Calculate sensitivity, specificity, and F1 score
TP = sum((final_predictions == 1) & (y == 1)); % True Positives
TN = sum((final_predictions == 0) & (y == 0)); % True Negatives
FP = sum((final_predictions == 1) & (y == 0)); % False Positives
FN = sum((final_predictions == 0) & (y == 1)); % False Negatives

% Sensitivity (Recall)
sensitivity = TP / (TP + FN);
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 0.96
```

```
% Specificity
specificity = TN / (TN + FP);
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 0.97
```

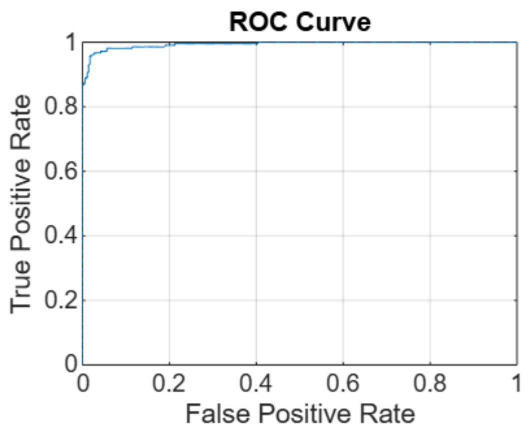
```
% Precision
precision = TP / (TP + FP);

% F1 Score
F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
fprintf('F1 Score: %.2f\n', F1_score);
```

```
F1 Score: 0.96
```

```
% Calculate AUC
[X_ROC, Y_ROC, ~, AUC] = perfcurve(y, final_probabilities, 1); % 1 for
positive class
fprintf('AUC: %.2f\n', AUC);
```

```
AUC: 0.99
```

```
% Optionally plot ROC curve
figure;
plot(X_ROC, Y_ROC);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
grid on;
```



Logistic regression with Train-test split

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, 3:end}; % Adjust the column indices as needed
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standard scaling (normalization)
mean_X = mean(X); % Compute mean of each feature
std_X = std(X);   % Compute standard deviation of each feature
X = (X - mean_X) ./ std_X; % Normalize features

% Add a bias term to X
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Split data into train and test sets (80-20 split)
cv = cvpartition(size(X, 1), 'HoldOut', 0.2);
X_train = X(training(cv), :);
y_train = y(training(cv), :);
X_test = X(test(cv), :);
y_test = y(test(cv), :);

% Parameters for logistic regression
learning_rate = 0.01;
num_iterations = 1000;
k = 5; % Number of folds for cross-validation
num_samples_train = size(X_train, 1);
```

```
W = zeros(size(X_train, 2), 1); % Initialize weights

% Perform k-fold cross-validation on training data
kfold_cv = cvpartition(num_samples_train, 'KFold', k);
fold_accuracies = zeros(k, 1);

for i = 1:k
    % Training and validation sets for current fold
    X_train_fold = X_train(training(kfold_cv, i), :);
    y_train_fold = y_train(training(kfold_cv, i), :);
    X_val_fold = X_train(test(kfold_cv, i), :);
    y_val_fold = y_train(test(kfold_cv, i), :);

    % Reset weights for each fold training
    W_fold = zeros(size(X_train_fold, 2), 1);

    % Gradient descent for logistic regression
    for iter = 1:num_iterations
        z = X_train_fold * W_fold;
        predictions = 1 ./ (1 + exp(-z));
        error = predictions - y_train_fold;
        gradient = (X_train_fold' * (error .* predictions .* (1 -
predictions))) / size(X_train_fold, 1);
        W_fold = W_fold - learning_rate * gradient;
    end

    % Evaluate on validation set for current fold
    val_probabilities = 1 ./ (1 + exp(-X_val_fold * W_fold));
    val_predictions = val_probabilities >= 0.5;
    fold_accuracies(i) = sum(val_predictions == y_val_fold) /
length(y_val_fold);
end

% Average accuracy from cross-validation
cv_accuracy = mean(fold_accuracies);
fprintf('Average Cross-Validation Accuracy: %.2f%%\n', cv_accuracy * 100);
```

```
Average Cross-Validation Accuracy: 96.71%
```

```

% Train final model on full training data using gradient descent
for iter = 1:num_iterations
    z = X_train * W;
    predictions = 1 ./ (1 + exp(-z));
    error = predictions - y_train;
    gradient = (X_train' * (error .* predictions .* (1 - predictions))) /
num_samples_train;
    W = W - learning_rate * gradient;
end

% Evaluate on test set
```

```
test_probabilities = 1 ./ (1 + exp(-X_test * W));
test_predictions = test_probabilities >= 0.5;
test_accuracy = sum(test_predictions == y_test) / length(y_test);
fprintf('Test Accuracy: %.2f%%\n', test_accuracy * 100);
```

```
Test Accuracy: 94.69%
```

```

% Calculate sensitivity, specificity, and F1 score for test set
TP = sum((test_predictions == 1) & (y_test == 1));
TN = sum((test_predictions == 0) & (y_test == 0));
FP = sum((test_predictions == 1) & (y_test == 0));
FN = sum((test_predictions == 0) & (y_test == 1));

sensitivity = TP / (TP + FN);
specificity = TN / (TN + FP);
precision = TP / (TP + FP);
F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

fprintf('Test Sensitivity: %.2f\n', sensitivity);
```

```
Test Sensitivity: 0.95
```

```
fprintf('Test Specificity: %.2f\n', specificity);
```

```
Test Specificity: 0.95
```

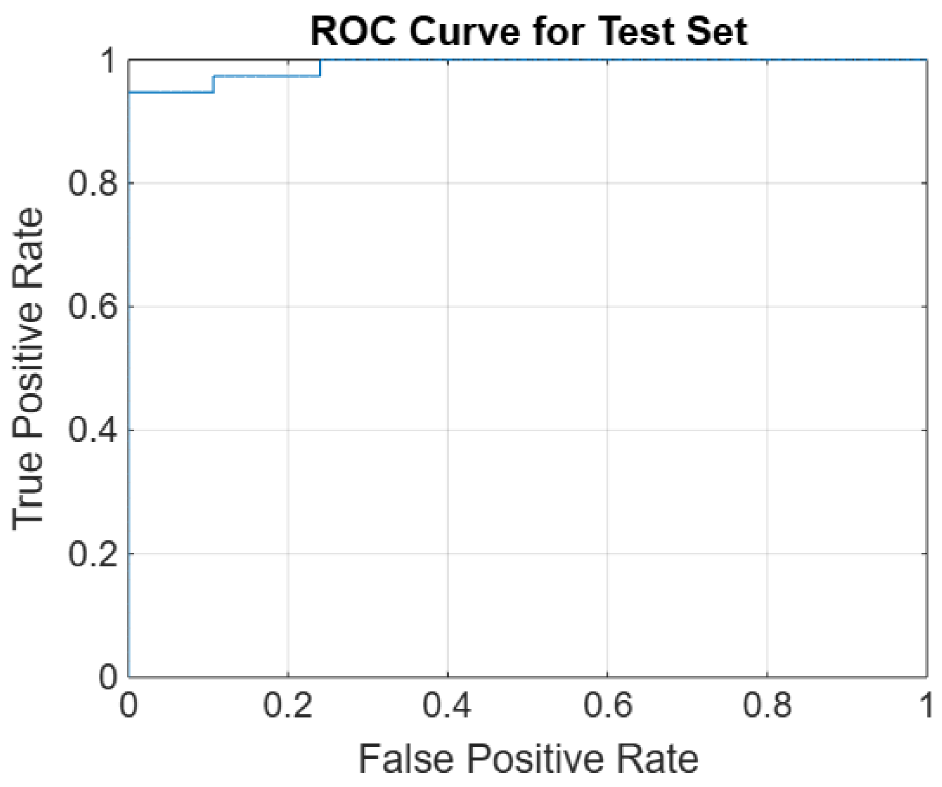
```
fprintf('Test F1 Score: %.2f\n', F1_score);
```

```
Test F1 Score: 0.92
```

```

% Calculate AUC for test set
[X_ROC, Y_ROC, ~, AUC] = perfcurve(y_test, test_probabilities, 1);
fprintf('Test AUC: %.2f\n', AUC);
```

```
Test AUC: 0.99
```

```

% Optionally plot ROC curve
figure;
plot(X_ROC, Y_ROC);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve for Test Set');
grid on;
```



Logistic Regression with regularization (L2)

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
%X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
% Extract features and target variable
X = data{:, 3:end}; % Adjust the column indices as needed
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standard scaling (normalization)
mean_X = mean(X); % Compute mean of each feature
std_X = std(X);   % Compute standard deviation of each feature
X = (X - mean_X) ./ std_X; % Normalize features

% Add a bias term to X
```

```
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Initialize parameters
learning_rate = 0.01;
num_iterations = 1000;
num_samples = size(X, 1);
W = zeros(size(X, 2), 1); % Initialize weights
lambda = 0.1; % Regularization strength

% Gradient descent loop
for iter = 1:num_iterations
    % Calculate the predictions
    z = X * W; % Linear combination
    predictions = 1 ./ (1 + exp(-z)); % Sigmoid function

    % Compute the gradient using the derivative of the sigmoid
    error = predictions - y; % Difference between predictions and actual
    gradient = (X' * (error .* predictions .* (1 - predictions))) /
num_samples; % Correct gradient

    % Add L2 regularization to the gradient
    gradient = gradient + (lambda / num_samples) * W; % Regularization term

    % Update weights
    W = W - learning_rate * gradient; % Gradient descent step
end

% Compute final probabilities and classify
final_probabilities = 1 ./ (1 + exp(-X * W)); % Get probabilities
final_predictions = final_probabilities >= 0.5; % Classification based on
threshold

% Calculate accuracy
accuracy = sum(final_predictions == y) / num_samples;
fprintf('Accuracy from scratch implementation with L2 regularization: %.2f%%
\n', accuracy * 100);
```

```
Accuracy from scratch implementation with L2 regularization: 94.73%
```

```
% Calculate sensitivity, specificity, and F1 score
TP = sum((final_predictions == 1) & (y == 1)); % True Positives
TN = sum((final_predictions == 0) & (y == 0)); % True Negatives
FP = sum((final_predictions == 1) & (y == 0)); % False Positives
FN = sum((final_predictions == 0) & (y == 1)); % False Negatives

% Sensitivity (Recall)
sensitivity = TP / (TP + FN);
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 0.93
```

```
% Specificity
specificity = TN / (TN + FP);
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 0.96
```

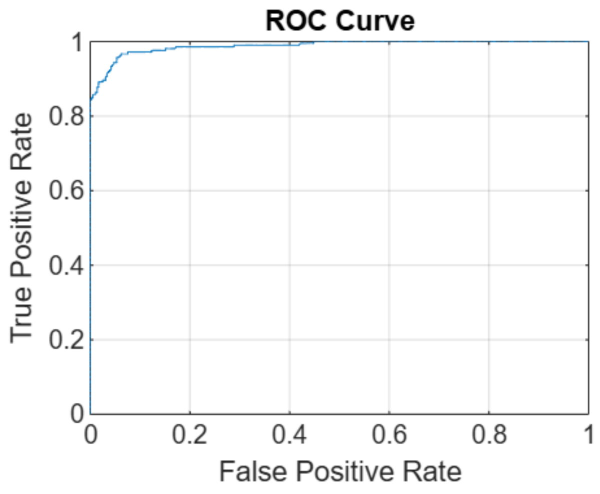
```
% Precision
precision = TP / (TP + FP);

% F1 Score
F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
fprintf('F1 Score: %.2f\n', F1_score);
```

```
F1 Score: 0.93
```

```
% Calculate AUC
[X_ROC, Y_ROC, ~, AUC] = perfcurve(y, final_probabilities, 1); % 1 for
positive class
fprintf('AUC: %.2f\n', AUC);
```

```
AUC: 0.99
```

```
% Optionally plot ROC curve
figure;
plot(X_ROC, Y_ROC);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
grid on;
```



Logistic Regression with L1 regularization

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
% Extract features and target variable
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
y = data.diagnosis; % Assuming 'target' is the name of your target variable
X = data{:, 3:end};
%y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standardize features
X = (X - mean(X)) ./ std(X);

% Add a bias term to X
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Initialize parameters
learning_rate = 0.01;
num_iterations = 1000;
num_samples = size(X, 1);
lambda = 0.1; % Regularization strength for L1 regularization
W = zeros(size(X, 2), 1); % Initialize weights

% Gradient descent loop
for iter = 1:num_iterations
    % Calculate the predictions
    z = X * W; % Linear combination
    predictions = 1 ./ (1 + exp(-z)); % Sigmoid function

    % Compute the gradient
    error = predictions - y; % Difference between predictions and actual
    gradient = (X' * error) / num_samples; % Base gradient

    % Add L1 regularization to the gradient
    % Use sign(W) to get the direction of the regularization
    l1_penalty = lambda * sign(W);
    gradient = gradient + l1_penalty / num_samples;

    % Update weights
    W = W - learning_rate * gradient; % Gradient descent step
end
```

```
% Compute final probabilities and classify
final_probabilities = 1 ./ (1 + exp(-X * W)); % Get probabilities
final_predictions = final_probabilities >= 0.5; % Classification based on
threshold

% Calculate accuracy
accuracy = sum(final_predictions == y) / num_samples;
fprintf('Accuracy from scratch implementation with L1 regularization: %.2f%%
\n', accuracy * 100);
```

```
Accuracy from scratch implementation with L1 regularization: 97.54%
```

```
% Calculate sensitivity, specificity, and F1 score
TP = sum((final_predictions == 1) & (y == 1)); % True Positives
TN = sum((final_predictions == 0) & (y == 0)); % True Negatives
FP = sum((final_predictions == 1) & (y == 0)); % False Positives
FN = sum((final_predictions == 0) & (y == 1)); % False Negatives

% Sensitivity (Recall)
sensitivity = TP / (TP + FN);
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 0.97
```

```
% Specificity
specificity = TN / (TN + FP);
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 0.98
```

```
% Precision
precision = TP / (TP + FP);

% F1 Score
F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
fprintf('F1 Score: %.2f\n', F1_score);
```

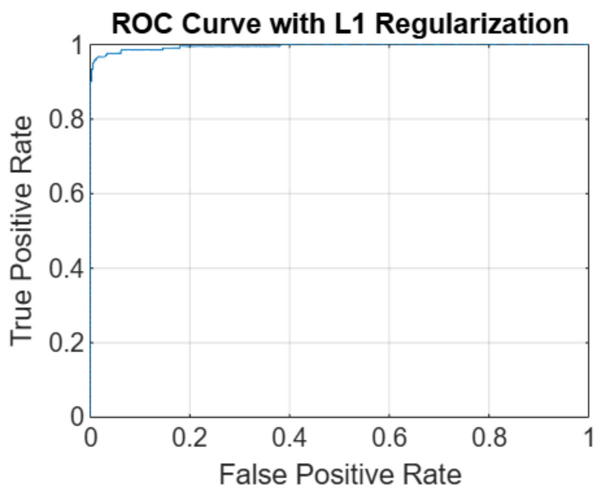
```
F1 Score: 0.97
```

```
% Calculate AUC
[X_ROC, Y_ROC, ~, AUC] = perfcurve(y, final_probabilities, 1); % 1 for
positive class
fprintf('AUC: %.2f\n', AUC);
```

```
AUC: 1.00
```

```
% Optionally plot ROC curve
```

```
figure;
plot(X_ROC, Y_ROC);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve with L1 Regularization');
grid on;
```



Logistic+L1 and L2 with test train split

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
%X = data{:, 3:end}; % Adjust the column indices as needed
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standard scaling (normalization)
X = (X - mean(X)) ./ std(X); % Normalize features

% Add a bias term to X
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Train-test split (80-20 split)
cv = cvpartition(size(X, 1), 'HoldOut', 0.2);
X_train = X(cv.training, :);
y_train = y(cv.training);
```

```
X_test = X(cv.test, :);
y_test = y(cv.test);

% Parameters
learning_rate = 0.01;
num_iterations = 1000;
lambda_l2 = 0.1; % L2 regularization strength
lambda_l1 = 0.1; % L1 regularization strength

% Initialize weights for L2 and L1 regularization
W_l2 = zeros(size(X_train, 2), 1);
W_l1 = zeros(size(X_train, 2), 1);

% K-fold cross-validation
K = 5;
cv_outer = cvpartition(size(X_train, 1), 'KFold', K);

% Perform K-fold cross-validation
cv_accuracy = zeros(K, 2); % Store accuracy for L2 and L1 regularization

for k = 1:K
    % Split data into training and validation sets
    X_train_cv = X_train(training(cv_outer, k), :);
    y_train_cv = y_train(training(cv_outer, k));
    X_val_cv = X_train(test(cv_outer, k), :);
    y_val_cv = y_train(test(cv_outer, k));

    % Train model with L2 regularization
    for iter = 1:num_iterations
        z = X_train_cv * W_l2;
        predictions = 1 ./ (1 + exp(-z));
        error = predictions - y_train_cv;
        gradient = (X_train_cv' * (error .* predictions .* (1 -
predictions))) / size(X_train_cv, 1);
        gradient = gradient + (lambda_l2 / size(X_train_cv, 1)) * W_l2; % L2
regularization
        W_l2 = W_l2 - learning_rate * gradient;
    end
    % Evaluate on validation set
    val_probs_l2 = 1 ./ (1 + exp(-X_val_cv * W_l2));
    val_preds_l2 = val_probs_l2 >= 0.5;
    cv_accuracy(k, 1) = sum(val_preds_l2 == y_val_cv) / length(y_val_cv);

    % Train model with L1 regularization
    for iter = 1:num_iterations
        z = X_train_cv * W_l1;
        predictions = 1 ./ (1 + exp(-z));
        error = predictions - y_train_cv;
        gradient = (X_train_cv' * error) / size(X_train_cv, 1);
        l1_penalty = lambda_l1 * sign(W_l1); % L1 regularization
        gradient = gradient + l1_penalty / size(X_train_cv, 1);
        W_l1 = W_l1 - learning_rate * gradient;
    end
    % Evaluate on validation set
    val_probs_l1 = 1 ./ (1 + exp(-X_val_cv * W_l1));
    val_preds_l1 = val_probs_l1 >= 0.5;
    cv_accuracy(k, 2) = sum(val_preds_l1 == y_val_cv) / length(y_val_cv);
end

% Print cross-validation accuracy
fprintf('Cross-validation accuracy (L2): %.2f%%\n', mean(cv_accuracy(:, 1))
* 100);
```

```
Cross-validation accuracy (L2): 94.74%
```

```
fprintf('Cross-validation accuracy (L1): %.2f%%\n', mean(cv_accuracy(:, 2))
* 100);
```

```
Cross-validation accuracy (L1): 94.74%
```

```

% Final evaluation on test set
% L2 regularized model on test set
test_probs_l2 = 1 ./ (1 + exp(-X_test * W_l2));
test_preds_l2 = test_probs_l2 >= 0.5;
accuracy_l2 = sum(test_preds_l2 == y_test) / length(y_test);

% L1 regularized model on test set
test_probs_l1 = 1 ./ (1 + exp(-X_test * W_l1));
test_preds_l1 = test_probs_l1 >= 0.5;
accuracy_l1 = sum(test_preds_l1 == y_test) / length(y_test);

% Print final test set accuracy
fprintf('Test accuracy (L2 regularization): %.2f%%\n', accuracy_l2 * 100);
```

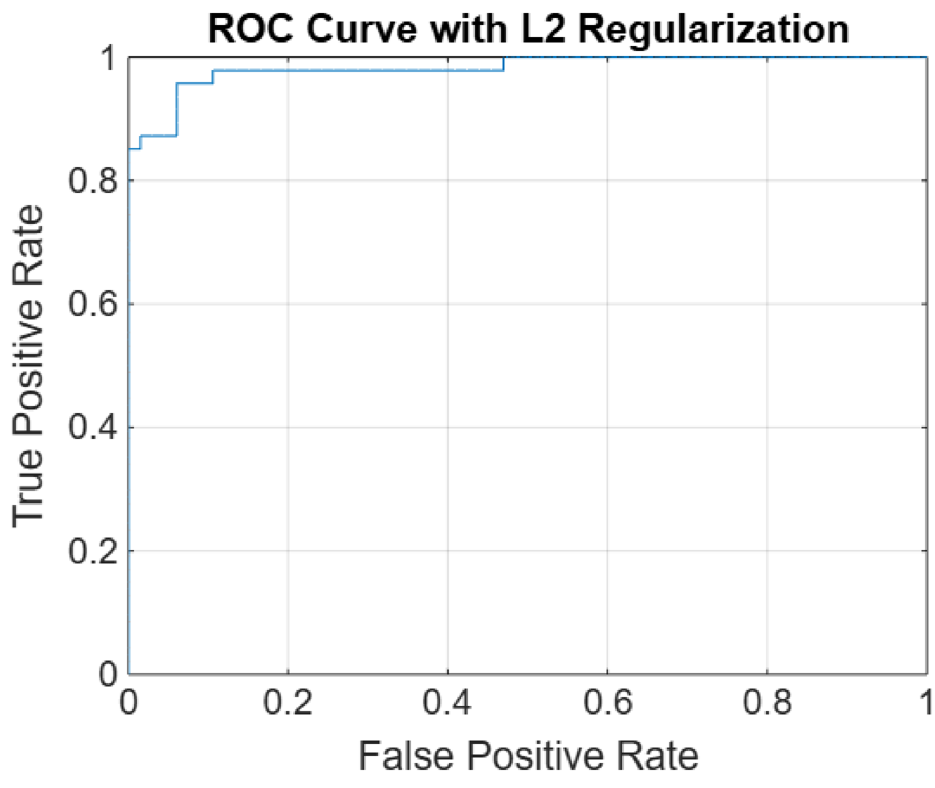
```
Test accuracy (L2 regularization): 92.04%
```

```
fprintf('Test accuracy (L1 regularization): %.2f%%\n', accuracy_l1 * 100);
```

```
Test accuracy (L1 regularization): 91.15%
```

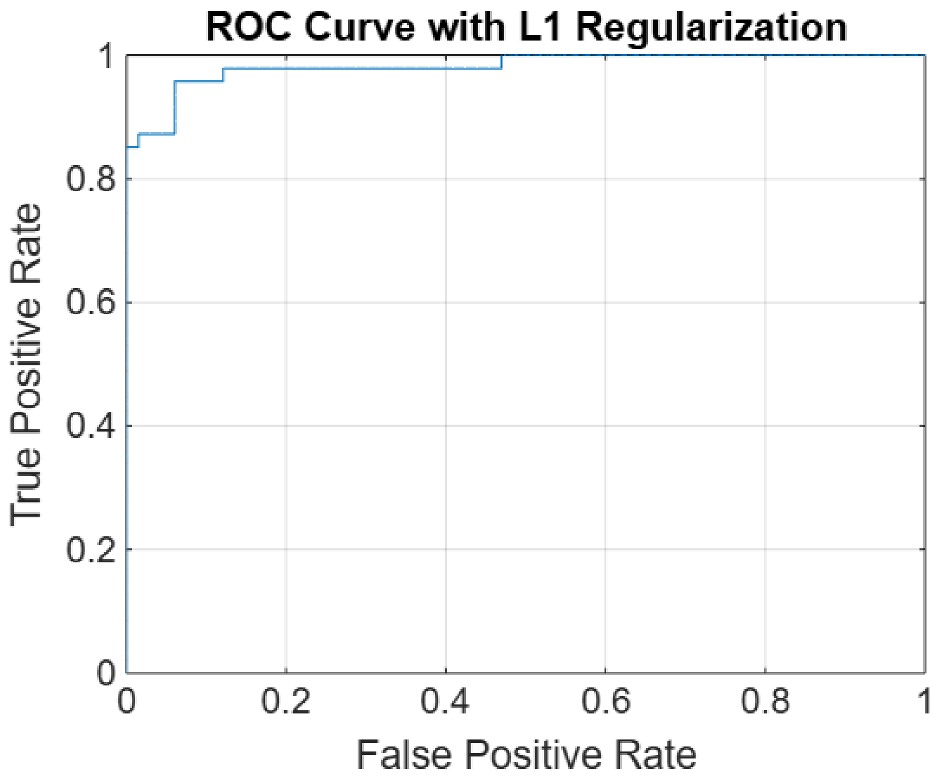
```

% Additional evaluation metrics for L2 and L1 regularization
for reg = ["L2", "L1"]
    if reg == "L2"
        final_predictions = test_preds_l2;
        final_probabilities = test_probs_l2;
    else
        final_predictions = test_preds_l1;
        final_probabilities = test_probs_l1;
    end

    % Calculate sensitivity, specificity, precision, and F1 score
    TP = sum((final_predictions == 1) & (y_test == 1));
    TN = sum((final_predictions == 0) & (y_test == 0));
    FP = sum((final_predictions == 1) & (y_test == 0));
    FN = sum((final_predictions == 0) & (y_test == 1));

    sensitivity = TP / (TP + FN);
    specificity = TN / (TN + FP);
    precision = TP / (TP + FP);
    F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

    fprintf('%s Regularization: Sensitivity = %.2f, Specificity = %.2f, F1
Score = %.2f\n', reg, sensitivity, specificity, F1_score);

    % ROC and AUC for test set
    [X_ROC, Y_ROC, ~, AUC] = perfcurve(y_test, final_probabilities, 1);
    fprintf('%s Regularization: AUC = %.2f\n', reg, AUC);

    % Plot ROC Curve
    figure;
    plot(X_ROC, Y_ROC);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title(sprintf('ROC Curve with %s Regularization', reg));
    grid on;
end
```

```
L2 Regularization: Sensitivity = 0.87, Specificity = 0.95, F1 Score = 0.90
L2 Regularization: AUC = 0.98
```



```
L1 Regularization: Sensitivity = 0.87, Specificity = 0.94, F1 Score = 0.89
```

```
L1 Regularization: AUC = 0.98
```



Logistic Regression with Elastic net

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
% Extract features and target variable
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
y = data.diagnosis; % Assuming 'target' is the name of your target variable
%X = data{:, 3:end};
%y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standardize features
X = (X - mean(X)) ./ std(X);

% Add a bias term to X
```

```
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Initialize parameters
learning_rate = 0.01;
num_iterations = 1000;
num_samples = size(X, 1);
lambda = 0.1; % Regularization strength for L1 regularization
W = zeros(size(X, 2), 1); % Initialize weights
lambda1=0.2;
% Gradient descent loop
for iter = 1:num_iterations
    % Calculate the predictions
    z = X * W; % Linear combination
    predictions = 1 ./ (1 + exp(-z)); % Sigmoid function

    % Compute the gradient
    error = predictions - y; % Difference between predictions and actual
    gradient = (X' * error) / num_samples; % Base gradient

    % Add L1 regularization to the gradient
    % Use sign(W) to get the direction of the regularization
    l1_penalty = lambda * sign(W);
    gradient = gradient +(lambda1 / num_samples) * W + l1_penalty /
num_samples;%elastic net gradient

    % Update weights
    W = W - learning_rate * gradient; % Gradient descent step
end

% Compute final probabilities and classify
final_probabilities = 1 ./ (1 + exp(-X * W)); % Get probabilities
final_predictions = final_probabilities >= 0.5; % Classification based on
threshold

% Calculate accuracy
accuracy = sum(final_predictions == y) / num_samples;
fprintf('Accuracy from scratch implementation with Elastic Net
regularization: %.2f%%\n', accuracy * 100);
```

```
Accuracy from scratch implementation with Elastic Net regularization: 94.38%
```

```

% Calculate sensitivity, specificity, and F1 score
TP = sum((final_predictions == 1) & (y == 1)); % True Positives
TN = sum((final_predictions == 0) & (y == 0)); % True Negatives
FP = sum((final_predictions == 1) & (y == 0)); % False Positives
FN = sum((final_predictions == 0) & (y == 1)); % False Negatives

% Sensitivity (Recall)
sensitivity = TP / (TP + FN);
```

```
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 0.92
```

```

% Specificity
specificity = TN / (TN + FP);
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 0.96
```

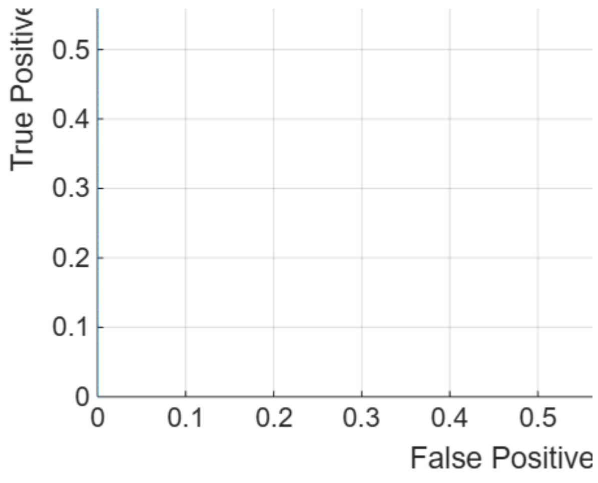
```

% Precision
precision = TP / (TP + FP);

% F1 Score
F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
fprintf('F1 Score: %.2f\n', F1_score);
```

```
F1 Score: 0.92
```

```

% Calculate AUC
[X_ROC, Y_ROC, ~, AUC] = perfcurve(y, final_probabilities, 1); % 1 for
positive class
fprintf('AUC: %.2f\n', AUC);
```

```
AUC: 0.99
```

```

% Optionally plot ROC curve
figure;
plot(X_ROC, Y_ROC);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve with L1 Regularization');
grid on;
```



Logistic Regression with Elastic net- train test split

```matlab
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```matlab

% Extract features and target variable
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
y = data.diagnosis; % Assuming 'diagnosis' is the name of the target variable

% Standardize features
X = (X - mean(X)) ./ std(X);

% Train-test split (70-30 split)
cv = cvpartition(size(X, 1), 'HoldOut', 0.3);
trainIdx = cv.training;
testIdx = cv.test;

% Split the data
X_train = X(trainIdx, :);
y_train = y(trainIdx);
X_test = X(testIdx, :);
y_test = y(testIdx);

% Add a bias term to X_train and X_test
X_test = [ones(size(X_test, 1), 1), X_test];

% 5-Fold Cross-Validation setup on training data
kfolds = 5;
cv_train = cvpartition(y_train, 'KFold', kfolds);
train_accuracies = zeros(kfolds, 1);
train_sensitivities = zeros(kfolds, 1);
train_specificities = zeros(kfolds, 1);
train_f1_scores = zeros(kfolds, 1);

% Elastic Net parameters
learning_rate = 0.01;
num_iterations = 1000;
lambda1 = 0.1; % L2 regularization strength
lambda2 = 0.2; % L1 regularization strength

% Loop through each fold
```

```
for fold = 1:kfolds
    % Get training and validation indices for this fold
    fold_train_idx = cv_train.training(fold);
    fold_val_idx = cv_train.test(fold);

    % Split data for this fold
    X_fold_train = X_train(fold_train_idx, :);
    y_fold_train = y_train(fold_train_idx);
    X_fold_val = X_train(fold_val_idx, :);
    y_fold_val = y_train(fold_val_idx);

    % Add a bias term to training and validation sets
    X_fold_train = [ones(size(X_fold_train, 1), 1), X_fold_train];
    X_fold_val = [ones(size(X_fold_val, 1), 1), X_fold_val];

    % Initialize weights for this fold
    W = zeros(size(X_fold_train, 2), 1);

    % Gradient descent loop for training fold
    for iter = 1:num_iterations
        % Calculate predictions
        z = X_fold_train * W;
        predictions = 1 ./ (1 + exp(-z));

        % Compute gradient with Elastic Net regularization
        error = predictions - y_fold_train;
        gradient = (X_fold_train' * error) / numel(y_fold_train);
        l1_penalty = lambda2 * sign(W);
        l2_penalty = lambda1 * W;
        gradient = gradient + (l2_penalty + l1_penalty) /
numel(y_fold_train);

        % Update weights
        W = W - learning_rate * gradient;
    end

    % Evaluate on validation set
    val_probabilities = 1 ./ (1 + exp(-X_fold_val * W));
    val_predictions = val_probabilities >= 0.5;

    % Calculate metrics for this fold
    train_accuracies(fold) = sum(val_predictions == y_fold_val) /
numel(y_fold_val);

    TP = sum((val_predictions == 1) & (y_fold_val == 1));
    TN = sum((val_predictions == 0) & (y_fold_val == 0));
    FP = sum((val_predictions == 1) & (y_fold_val == 0));
    FN = sum((val_predictions == 0) & (y_fold_val == 1));

    % Sensitivity, Specificity, and F1 score
    train_sensitivities(fold) = TP / (TP + FN);
    train_specificities(fold) = TN / (TN + FP);
    precision = TP / (TP + FP);
    train_f1_scores(fold) = 2 * (precision * train_sensitivities(fold)) /
(precision + train_sensitivities(fold));
end

% Average metrics across folds
avg_train_accuracy = mean(train_accuracies);
avg_train_sensitivity = mean(train_sensitivities);
avg_train_specificity = mean(train_specificities);
avg_train_f1_score = mean(train_f1_scores);

% Print 5-fold CV training results
fprintf('5-Fold Cross-Validation Results on Training Data:\n');
```

```
5-Fold Cross-Validation Results on Training Data:
```

```
fprintf('Average Accuracy: %.2f%%\n', avg_train_accuracy * 100);
```

```
Average Accuracy: 94.73%
```

```
fprintf('Average Sensitivity: %.2f\n', avg_train_sensitivity);
```

```
Average Sensitivity: 0.93
```

```
fprintf('Average Specificity: %.2f\n', avg_train_specificity);
```

```
Average Specificity: 0.96
```

```
fprintf('Average F1 Score: %.2f\n', avg_train_f1_score);
```

```
Average F1 Score: 0.93
```

```

% Final model training on full training data (without validation split)
X_train = [ones(size(X_train, 1), 1), X_train];
W = zeros(size(X_train, 2), 1); % Re-initialize weights

for iter = 1:num_iterations
    % Train on entire training set
    z = X_train * W;
    predictions = 1 ./ (1 + exp(-z));
    error = predictions - y_train;
    gradient = (X_train' * error) / numel(y_train);
    l1_penalty = lambda2 * sign(W);
    l2_penalty = lambda1 * W;
    gradient = gradient + (l2_penalty + l1_penalty) / numel(y_train);
    W = W - learning_rate * gradient;
end

% Evaluate on test set
test_probabilities = 1 ./ (1 + exp(-X_test * W));
```

```
test_predictions = test_probabilities >= 0.5;

% Calculate metrics for test set
accuracy_test = sum(test_predictions == y_test) / numel(y_test);
TP_test = sum((test_predictions == 1) & (y_test == 1));
TN_test = sum((test_predictions == 0) & (y_test == 0));
FP_test = sum((test_predictions == 1) & (y_test == 0));
FN_test = sum((test_predictions == 0) & (y_test == 1));

sensitivity_test = TP_test / (TP_test + FN_test);
specificity_test = TN_test / (TN_test + FP_test);
precision_test = TP_test / (TP_test + FP_test);
f1_score_test = 2 * (precision_test * sensitivity_test) / (precision_test +
sensitivity_test);

% Print test set results
fprintf('Test Set Results:\n');
```

```
Test Set Results:
```

```
fprintf('Accuracy: %.2f%%\n', accuracy_test * 100);
```

```
Accuracy: 95.29%
```

```
fprintf('Sensitivity: %.2f\n', sensitivity_test);
```

```
Sensitivity: 0.93
```

```
fprintf('Specificity: %.2f\n', specificity_test);
```

```
Specificity: 0.96
```

```
fprintf('F1 Score: %.2f\n', f1_score_test);
```

```
F1 Score: 0.93
```

Logistic Regression with K-fold cross validation

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
%X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
X = data{:, 3:end};
```

```
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Standardize features
X = (X - mean(X)) ./ std(X);

% Add a bias term to X
X = [ones(size(X, 1), 1), X]; % Add a column of ones for the intercept

% Set parameters for k-fold cross-validation
k = 5; % Number of folds
num_samples = size(X, 1);
indices = crossvalind('Kfold', y, k); % Create indices for k-fold cross-
validation

% Initialize performance metrics
accuracy_list = zeros(k, 1);
sensitivity_list = zeros(k, 1);
specificity_list = zeros(k, 1);
F1_score_list = zeros(k, 1);
AUC_list = zeros(k, 1);
lambda = 0.1; % Regularization strength for L1 regularization
learning_rate = 0.01;
num_iterations = 1000;

for fold = 1:k
    % Split data into training and test sets
    test_indices = (indices == fold); % Logical indices for test set
    train_indices = ~test_indices; % Logical indices for training set

    X_train = X(train_indices, :);
    y_train = y(train_indices);
    X_test = X(test_indices, :);
    y_test = y(test_indices);

    % Initialize weights for the current fold
    W = zeros(size(X_train, 2), 1); % Initialize weights

    % Gradient descent loop for the current fold
    for iter = 1:num_iterations
        % Calculate the predictions
        z = X_train * W; % Linear combination
        predictions = 1 ./ (1 + exp(-z)); % Sigmoid function

        % Compute the gradient
        error = predictions - y_train; % Difference between predictions and
actual
        gradient = (X_train' * error) / size(X_train, 1); % Base gradient

        % Add L1 regularization to the gradient
        l1_penalty = lambda * sign(W);
        gradient = gradient + l1_penalty / size(X_train, 1);

        % Update weights
        W = W - learning_rate * gradient; % Gradient descent step
    end

    % Compute final probabilities and classify on the test set
    final_probabilities = 1 ./ (1 + exp(-X_test * W)); % Get probabilities
    final_predictions = final_probabilities >= 0.5; % Classification based
on threshold

    % Calculate accuracy for the current fold
    accuracy = sum(final_predictions == y_test) / sum(test_indices);
    accuracy_list(fold) = accuracy;

    % Calculate sensitivity, specificity, and F1 score
    TP = sum((final_predictions == 1) & (y_test == 1)); % True Positives
    TN = sum((final_predictions == 0) & (y_test == 0)); % True Negatives
    FP = sum((final_predictions == 1) & (y_test == 0)); % False Positives
    FN = sum((final_predictions == 0) & (y_test == 1)); % False Negatives

    % Sensitivity (Recall)
    sensitivity = TP / (TP + FN);
    sensitivity_list(fold) = sensitivity;

    % Specificity
    specificity = TN / (TN + FP);
    specificity_list(fold) = specificity;

    % Precision
    precision = TP / (TP + FP);

    % F1 Score
    F1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
    F1_score_list(fold) = F1_score;

    % Calculate AUC
    [X_ROC, Y_ROC, ~, AUC] = perfcurve(y_test, final_probabilities, 1); % 1
for positive class
    AUC_list(fold) = AUC;

    fprintf('Fold %d: Accuracy = %.2f%%, Sensitivity = %.2f, Specificity =
%.2f, F1 Score = %.2f, AUC = %.2f\n', ...
        fold, accuracy * 100, sensitivity, specificity, F1_score, AUC);
end
```

```
Fold 1: Accuracy = 99.12%, Sensitivity = 1.00, Specificity = 0.99, F1 Score = 0.99, AUC = 1.00
Fold 2: Accuracy = 96.46%, Sensitivity = 0.93, Specificity = 0.99, F1 Score = 0.95, AUC = 0.99
Fold 3: Accuracy = 96.52%, Sensitivity = 0.98, Specificity = 0.96, F1 Score = 0.95, AUC = 0.99
Fold 4: Accuracy = 95.58%, Sensitivity = 0.90, Specificity = 0.99, F1 Score = 0.94, AUC = 0.99
Fold 5: Accuracy = 99.13%, Sensitivity = 1.00, Specificity = 0.99, F1 Score = 0.99, AUC = 1.00
```

```
% Average performance metrics across all folds
mean_accuracy = mean(accuracy_list);
mean_sensitivity = mean(sensitivity_list);
mean_specificity = mean(specificity_list);
mean_F1_score = mean(F1_score_list);
mean_AUC = mean(AUC_list);

fprintf('Mean Accuracy across %d folds: %.2f%%\n', k, mean_accuracy * 100);
```

```
Mean Accuracy across 5 folds: 97.36%
```

```
fprintf('Mean Sensitivity across %d folds: %.2f\n', k, mean_sensitivity);
```

```
Mean Sensitivity across 5 folds: 0.96
```

```
fprintf('Mean Specificity across %d folds: %.2f\n', k, mean_specificity);
```

```
Mean Specificity across 5 folds: 0.98
```

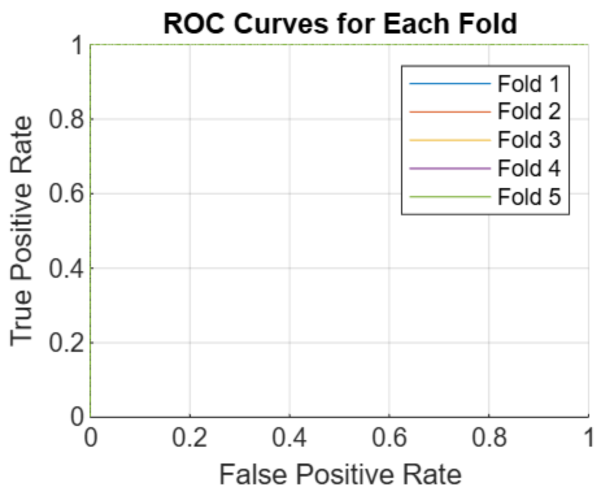
```
fprintf('Mean F1 Score across %d folds: %.2f\n', k, mean_F1_score);
```

```
Mean F1 Score across 5 folds: 0.96
```

```
fprintf('Mean AUC across %d folds: %.2f\n', k, mean_AUC);
```

```
Mean AUC across 5 folds: 0.99
```

```
% Optionally plot ROC curves for each fold
figure;
hold on;
for fold = 1:k
    if any(indices == fold) % Only plot if there are samples in this fold
        % Compute ROC for each fold and plot
        final_probabilities_fold = 1 ./ (1 + exp(-X_test * W)); % Get
probabilities for test fold
        [X_ROC, Y_ROC, ~] = perfcurve(y_test, final_probabilities_fold, 1);
% 1 for positive class
        plot(X_ROC, Y_ROC); % Plot each fold's ROC
    end
end
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curves for Each Fold');
grid on;
legend(arrayfun(@(x) sprintf('Fold %d', x), 1:k, 'UniformOutput', false));
hold off;
```


ROC Curves for Each Fold
True Positive Rate
False Positive Rate
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

Implementation of SVM Classifiers

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
%X = data{:, 3:end}; % Assuming the features start from the 3rd column
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable
% Convert y to -1 and 1
y(y == 0) = -1; % Assuming '0' is one class
y(y == 1) = 1;  % Assuming '1' is the other class

% Standardize features
X = (X - mean(X)) ./ std(X);

% Convert y to -1 and 1 for SVM
y(y == 0) = -1; % Convert class labels: assuming 0 is negative class
y(y == 1) = 1;  % Convert class labels: assuming 1 is positive class

% Get the number of samples and features
[n_samples, n_features] = size(X);

% Prepare the data for the CVX optimization
H = (y * y') .* (X * X'); % Kernel (linear) matrix for dual problem
f = ones(n_samples, 1); % Coefficients for the linear term

% Variable for Lagrange multipliers
cvx_begin quiet
    variable alpa(n_samples)
    minimize(0.5 * quad_form(alpa, H) - f' * alpa) % Dual problem objective
(minimization)
    subject to
        y' * alpa == 0; % Equality constraint
        alpa >= 0; % Lagrange multipliers must be non-negative
cvx_end

% Extract support vectors
support_vector_indices = find(alpa > 1e-5);
alpha_sv = alpa(support_vector_indices);
y_sv = y(support_vector_indices);
X_sv = X(support_vector_indices, :);
```

```
% Calculate weights (w) and bias (b)
w = sum(alpha_sv .* y_sv .* X_sv, 1)'; % Weight vector
b = mean(y_sv - (X_sv * w)); % Bias term (average margin)
% Calculate weights (w) and bias (b)
w = sum(alpha_sv .* y_sv .* X_sv, 1)'; % Weight vector
b = mean(y_sv - (X_sv * w)); % Bias term

% Prediction function (classify based on sign of decision function)
decision_function = X * w + b;
y_pred = sign(decision_function);

% Performance metrics
true_positive = sum((y_pred == 1) & (y == 1));
true_negative = sum((y_pred == -1) & (y == -1));
false_positive = sum((y_pred == 1) & (y == -1));
false_negative = sum((y_pred == -1) & (y == 1));

% Accuracy
accuracy = (true_positive + true_negative) / n_samples;

% Sensitivity (Recall or True Positive Rate)
sensitivity = true_positive / (true_positive + false_negative);

% Specificity (True Negative Rate)
specificity = true_negative / (true_negative + false_positive);

% Precision
precision = true_positive / (true_positive + false_positive);

% F1 Score
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

% ROC curve and AUC
[roc_x, roc_y, ~, auc] = perfcurve(y, decision_function, 1);

% Display the results
fprintf('Weights (w): \n');
```

```
Weights (w):
```

```
disp(w);
```

```
   1.0e-13 *

   -0.0469
   -0.5319
    0.2569
   -0.0697
    0.2276
```

```
fprintf('Bias (b): %.2f\n', b);
```

```
Bias (b): -0.25
```

```
fprintf('Accuracy: %.2f\n', accuracy * 100);
```

```
Accuracy: 62.74
```

```
fprintf('Sensitivity (Recall): %.2f\n', sensitivity * 100);
```

```
Sensitivity (Recall): 0.00
```

```
fprintf('Specificity: %.2f\n', specificity * 100);
```

```
Specificity: 100.00
```

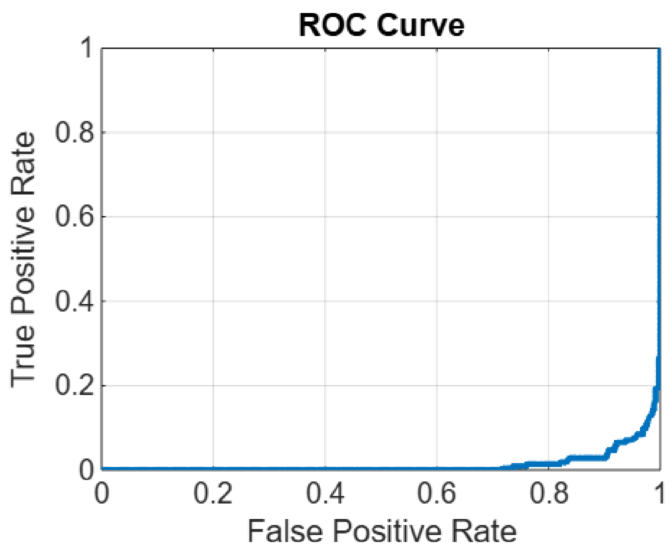
```
fprintf('F1 Score: %.2f\n', f1_score * 100);
```

```
F1 Score: NaN
```

```
fprintf('AUC: %.2f\n', auc);
```

```
AUC: 0.01
```

```

% Plot ROC curve
figure;
plot(roc_x, roc_y, 'LineWidth', 2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
grid on;
```



```
% Display the results
fprintf('Weights: \n');
```

```
Weights:
```

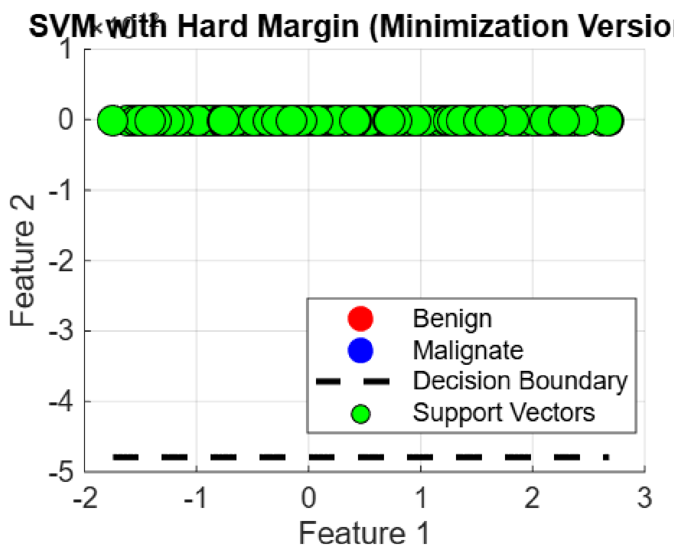
```
disp(w);
```

```
   1.0e-13 *
   -0.0469
   -0.5319
    0.2569
   -0.0697
    0.2276
```

```
fprintf('Bias: %.2f\n', b);
```

```
Bias: -0.25
```

```

% Optionally, plot the decision boundary
figure;
hold on;
gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled'); % Scatter plot of the
data points
x_range = linspace(min(X(:,1)), max(X(:,1)), 100); % Range for x-axis
decision_boundary = -(w(1) * x_range + b) / w(2); % Calculate decision
boundary
plot(x_range, decision_boundary, 'k--', 'LineWidth', 2); % Plot decision
boundary

% Plot support vectors
scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor', 'k'); %
Support vectors

xlabel('Feature 1');
ylabel('Feature 2');
title('SVM with Hard Margin (Minimization Version)');
legend('Benign', 'Malignate', 'Decision Boundary', 'Support Vectors',
'Location', 'Best');
grid on;
hold off;
```



Here the problem is not linearly seperable, so the dual problem is infeasible. Now try the SVM with soft margin

```
% Inputs: X (n_samples x n_features), y (n_samples x 1), C (regularization
parameter)
[n_samples, n_features] = size(X);
H = (y * y') .* (X * X'); % Gram matrix (Hessian) for dual formulation
f = -ones(n_samples, 1); % Objective linear term

% Soft margin SVM with dual formulation
C = 10; % Regularization parameter (tune as necessary)

cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * quad_form(alfa, H) - f' * alfa) % Dual objective
    subject to
        y' * alfa == 0; % Equality constraint
        0 <= alfa <= C; % Box constraints (soft margin)
cvx_end

% Extract support vectors
support_vector_indices = find(alfa > 1e-5 & alfa < C - 1e-5); % Support
vectors are those where 0 < alfa < C
alpha_sv = alfa(support_vector_indices);
y_sv = y(support_vector_indices);
X_sv = X(support_vector_indices, :);

% Calculate weights (w) and bias (b)
w = sum(alpha_sv .* y_sv .* X_sv, 1)'; % Weight vector
b = mean(y_sv - (X_sv * w)); % Bias term (average over support vectors)

% Display the results
fprintf('Weights (w): \n');
```

```
Weights (w):
```

```
disp(w);
```

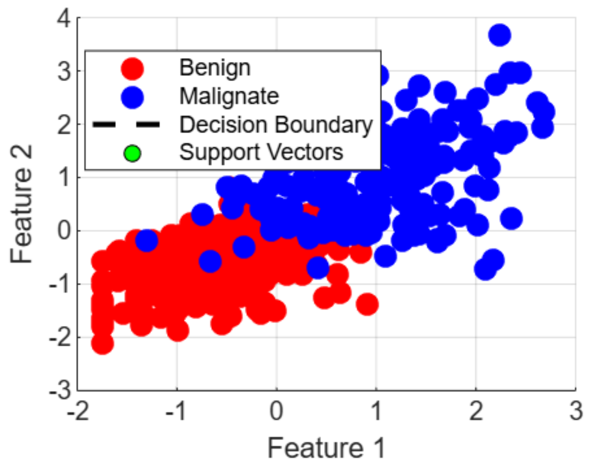
```
     0
     0
     0
     0
     0
```

```
fprintf('Bias (b): %.2f\n', b);
```

```
Bias (b): NaN
```

```


% Optionally, plot the decision boundary
figure;
hold on;
```

```
gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled'); % Scatter plot of the
data points
x_range = linspace(min(X(:,1)), max(X(:,1)), 100); % Range for x-axis
decision_boundary = -(w(1) * x_range + b) / w(2); % Decision boundary
equation
plot(x_range, decision_boundary, 'k--', 'LineWidth', 2); % Plot decision
boundary

% Plot support vectors
scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor', 'k'); %
Highlight support vectors

xlabel('Feature 1');
ylabel('Feature 2');
%title('Soft Margin SVM with CVX Solver');
legend('Benign', 'Malignate', 'Decision Boundary', 'Support Vectors',
'Location', 'Best');
grid on;
hold off;
```



```
% Inputs: X (n_samples x n_features), y (n_samples x 1), C (regularization
parameter)
[n_samples, n_features] = size(X);
H = (y * y') .* (X * X'); % Gram matrix (Hessian) for dual formulation
f = -ones(n_samples, 1); % Objective linear term

% Soft margin SVM with dual formulation
C = 1; % Regularization parameter (tune as necessary)

% Suppress CVX output
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * quad_form(alfa, H) - f' * alfa) % Dual objective
    subject to
        y' * alfa == 0; % Equality constraint
        0 <= alfa <= C; % Box constraints (soft margin)
cvx_end

% Extract support vectors (where alfa is greater than a small threshold but
less than C)
tolerance = 1e-5;
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);
alfa_sv = alfa(support_vector_indices);
y_sv = y(support_vector_indices);
X_sv = X(support_vector_indices, :);

% Calculate weights (w) and bias (b)
w = sum(alfa_sv .* y_sv .* X_sv, 1)'; % Weight vector
b = mean(y_sv - (X_sv * w)); % Bias term

% Prediction function (classify based on sign of decision function)
decision_function = X * w + b;
y_pred = sign(decision_function);

% Performance metrics
true_positive = sum((y_pred == 1) & (y == 1));
true_negative = sum((y_pred == -1) & (y == -1));
false_positive = sum((y_pred == 1) & (y == -1));
false_negative = sum((y_pred == -1) & (y == 1));

% Accuracy
accuracy = (true_positive + true_negative) / n_samples;

% Sensitivity (Recall or True Positive Rate)
sensitivity = true_positive / (true_positive + false_negative);

% Specificity (True Negative Rate)
specificity = true_negative / (true_negative + false_positive);

% Precision
precision = true_positive / (true_positive + false_positive);

% F1 Score
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

% ROC curve and AUC
[roc_x, roc_y, ~, auc] = perfcurve(y, decision_function, 1);

% Display the results
fprintf('Weights (w): \n');
```

```
Weights (w):
```

```
disp(w);
```

```
     0
     0
     0
     0
     0
```

```
fprintf('Bias (b): %.2f\n', b);
```

```
Bias (b): NaN
```

```
fprintf('Accuracy: %.2f\n', accuracy * 100);
```

```
Accuracy: 0.00
```

```
fprintf('Sensitivity (Recall): %.2f\n', sensitivity * 100);
```

```
Sensitivity (Recall): NaN
```

```
fprintf('Specificity: %.2f\n', specificity * 100);
```

```
Specificity: NaN
```

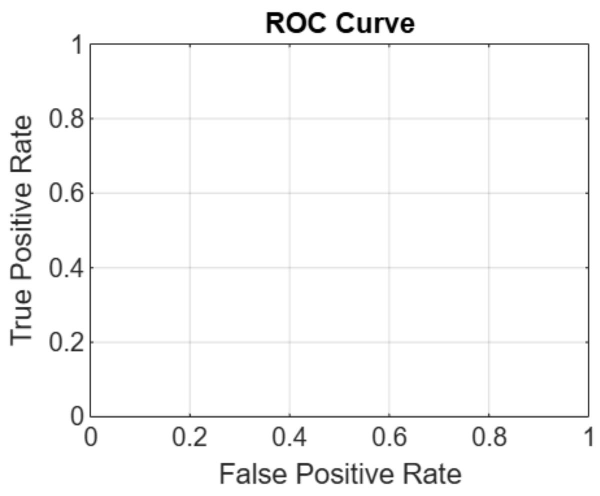
```
fprintf('F1 Score: %.2f\n', f1_score * 100);
```

```
F1 Score: NaN
```

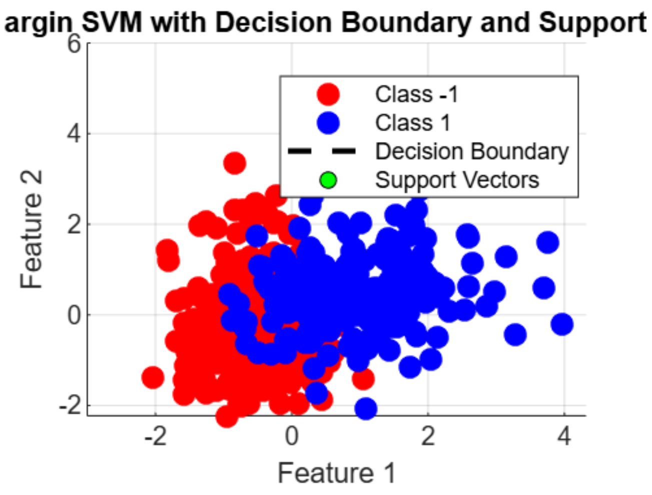
```
fprintf('AUC: %.2f\n', auc);
```

```
AUC: NaN
```

```

% Plot ROC curve
figure;
plot(roc_x, roc_y, 'LineWidth', 2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
grid on;
```



```
% Optionally, plot the decision boundary
figure;
hold on;
gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled'); % Scatter plot of data
x_range = linspace(min(X(:,1)), max(X(:,1)), 100); % Range for x-axis
decision_boundary = -(w(1) * x_range + b) / w(2); % Decision boundary
plot(x_range, decision_boundary, 'k--', 'LineWidth', 2); % Plot decision
boundary

% Plot support vectors
scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor', 'k'); %
Support vectors

xlabel('Feature 1');
ylabel('Feature 2');
title('Soft Margin SVM with Decision Boundary and Support Vectors');
legend('Class -1', 'Class 1', 'Decision Boundary', 'Support Vectors',
'Location', 'Best');
grid on;
hold off;
```



```
% trying support vectors check with higher C
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Extract features and target variable
X = data{:, 3:end}; % Assuming the features start from the 3rd column
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable
% Convert y to -1 and 1
```

```
y(y == 0) = -1; % Assuming '0' is one class
y(y == 1) = 1;  % Assuming '1' is the other class

% Normalize the features of the data (Z-score normalization)
X = zscore(X); % This will normalize each feature to have mean 0 and
standard deviation 1

% Inputs: X (n_samples x n_features), y (n_samples x 1), C (regularization
parameter)
[n_samples, n_features] = size(X);
H = (y * y') .* (X * X'); % Gram matrix (Hessian) for dual formulation
f = -ones(n_samples, 1); % Objective linear term

% Soft margin SVM with dual formulation
C = 100; % Regularization parameter (adjust as needed)

% Suppress CVX output
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * quad_form(alfa, H) - f' * alfa) % Dual objective
    subject to
        y' * alfa == 0; % Equality constraint
        0 <= alfa <= C; % Box constraints (soft margin)
cvx_end

% Extract support vectors (where alfa is greater than a small threshold but
less than C)
tolerance = 1e-5;
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);
alfa_sv = alfa(support_vector_indices);
y_sv = y(support_vector_indices);
X_sv = X(support_vector_indices, :);
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);
if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C or kernel.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);


    % Calculate weights (w) and bias (b)
    w = sum(alfa_sv .* y_sv .* X_sv, 1)'; % Weight vector
    b = mean(y_sv - (X_sv * w)); % Bias term

    % Prediction function (classify based on sign of decision function)
    decision_function = X * w + b;
    y_pred = sign(decision_function);

    % Performance metrics
    true_positive = sum((y_pred == 1) & (y == 1));
    true_negative = sum((y_pred == -1) & (y == -1));
    false_positive = sum((y_pred == 1) & (y == -1));
    false_negative = sum((y_pred == -1) & (y == 1));

    % Accuracy
    accuracy = (true_positive + true_negative) / n_samples;

    % Sensitivity (Recall or True Positive Rate)
    sensitivity = true_positive / (true_positive + false_negative);

    % Specificity (True Negative Rate)
    specificity = true_negative / (true_negative + false_positive);

    % Precision
    precision = true_positive / (true_positive + false_positive);

    % F1 Score
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

    % ROC curve and AUC
    [roc_x, roc_y, ~, auc] = perfcurve(y, decision_function, 1);

    % Display the results
    fprintf('Weights (w): \n');
    disp(w);
    fprintf('Bias (b): %.2f\n', b);
    fprintf('Accuracy: %.2f\n', accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f\n', sensitivity * 100);
    fprintf('Specificity: %.2f\n', specificity * 100);
    fprintf('F1 Score: %.2f\n', f1_score * 100);
    fprintf('AUC: %.2f\n', auc);

    % Plot ROC curve
    figure;
    plot(roc_x, roc_y, 'LineWidth', 2);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title('ROC Curve');
    grid on;

    % Optionally, plot the decision boundary
    figure;
    hold on;
    gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled'); % Scatter plot of
data
    x_range = linspace(min(X(:,1)), max(X(:,1)), 100); % Range for x-axis
    decision_boundary = -(w(1) * x_range + b) / w(2); % Decision boundary
    plot(x_range, decision_boundary, 'k--', 'LineWidth', 2); % Plot decision
boundary
```

```
% Plot support vectors
    scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor',
'k'); % Support vectors

    xlabel('Feature 1');
    ylabel('Feature 2');
    title('Soft Margin SVM with Decision Boundary and Support Vectors');
    legend('Class -1', 'Class 1', 'Decision Boundary', 'Support Vectors',
'Location', 'Best');
    grid on;
    hold off;
end
```

```
No support vectors found. Consider revising the regularization parameter C or kernel.
```

Using Linear kernal with some fine tuning

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
%X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
% Extract features and target variable
%X = data{:, {'concavePointsWorst', 'areaWorst'}};
X=data{:,3:end};
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X = zscore(X);

% Get the number of samples and features
[n_samples, n_features] = size(X);

% Linear Kernel
H = (y * y') .* (X * X');

% Define the linear term in the dual objective
f = -ones(n_samples, 1);
```

```
% Soft margin SVM dual formulation with linear kernel
C = 10; % Regularization parameter (adjust as needed)

% Set solver tolerance
tolerance = 1e-5;

% Solve the dual problem using CVX
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * alfa' * H * alfa + f' * alfa)
    subject to
        y' * alfa == 0;
        0 <= alfa <= C;
cvx_end

% Extract support vectors (0 < alfa < C)
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);

    % Extract support vectors
    alfa_sv = alfa(support_vector_indices);
    y_sv = y(support_vector_indices);
    X_sv = X(support_vector_indices, :);

    % Calculate weights (w) and bias (b)
    w = sum(alfa_sv .* y_sv .* X_sv, 1)';
    b = mean(y_sv - (X_sv * w));

    % Prediction function
    decision_function = X * w + b;
    y_pred = sign(decision_function);
    y_pred(y_pred == 0) = 1; % Handle zero predictions

    % Performance metrics
    true_positive = sum((y_pred == 1) & (y == 1));
    true_negative = sum((y_pred == -1) & (y == -1));
    false_positive = sum((y_pred == 1) & (y == -1));
    false_negative = sum((y_pred == -1) & (y == 1));

    % Accuracy
    accuracy = (true_positive + true_negative) / n_samples;

    % Sensitivity (Recall)
    sensitivity = true_positive / (true_positive + false_negative);
    % Specificity
    specificity = true_negative / (true_negative + false_positive);

    % Precision
    precision = true_positive / (true_positive + false_positive);

    % F1 Score
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

    % ROC and AUC
    [roc_x, roc_y, ~, auc] = perfcurve(y, decision_function, 1);

    % Display results
    fprintf('Weights (w): \n');
    disp(w);
    fprintf('Bias (b): %.4f\n', b);
    fprintf('Accuracy: %.2f%%\n', accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity * 100);
    fprintf('Specificity: %.2f%%\n', specificity * 100);
    fprintf('Precision: %.2f%%\n', precision * 100);
    fprintf('F1 Score: %.2f%%\n', f1_score * 100);
    fprintf('AUC: %.2f\n', auc);

    % Plot ROC curve
    figure;
    plot(roc_x, roc_y, 'LineWidth', 2);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title('ROC Curve');
    grid on;

    % Plot decision boundary and support vectors
    figure;
    hold on;
    gscatter(X(:,1), X(:,2), y, 'rb', 'o', 8, 'filled');
    x_range = linspace(min(X(:,1)), max(X(:,1)), 100);
    decision_boundary = -(w(1) * x_range + b) / w(2);
    plot(x_range, decision_boundary, 'k--', 'LineWidth', 2);
    scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor',
'k');

    xlabel('Feature 1');
    ylabel('Feature 2');
    %title('Soft Margin SVM with Decision Boundary and Support Vectors
(Linear Kernel)');
    legend('Benign', 'Malignate', 'Decision Boundary', 'Support Vectors',
'Location', 'Best');
    grid on;
    hold off;
```

```
end
```

```
Number of support vectors: 22
Weights (w):
    9.6563
   37.3238
   10.6097
   10.2268
    6.6977
   22.8915
   23.7582
   15.0714
  -11.1613
   -4.8500
    8.0185
   40.5651
   12.8249
   11.2341
   23.2933
   38.3906
   26.6141
   31.3056
   11.8510
   12.3878
    0.8953
   34.5471
    2.7923
    2.9035
   -1.0092
   16.8439
   20.5247
    6.5589
  -18.5627
   -5.9373
Bias (b): -172.8621
Accuracy: 79.79%
Sensitivity (Recall): 55.66%
Specificity: 94.12%
Precision: 84.89%
F1 Score: 67.24%
AUC: 0.90
```

Using non-linear kernel

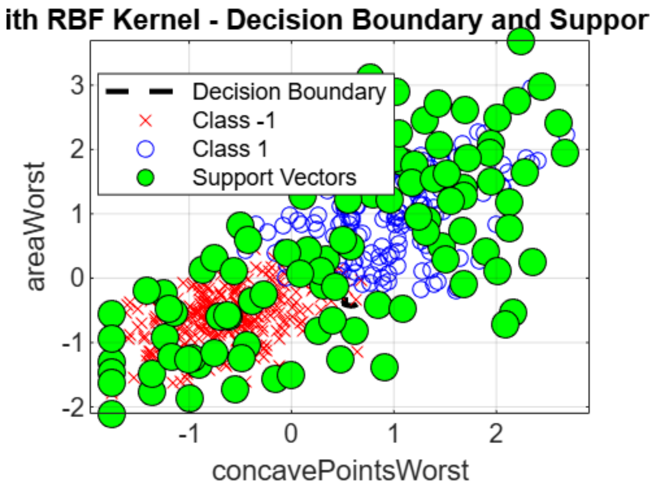
```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, {'concavePointsWorst', 'areaWorst'}};
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X = zscore(X);

% Parameters for RBF kernel
sigma = .30;  % Adjust sigma for the RBF kernel

% Get the number of samples and features
[n_samples, n_features] = size(X);

% RBF Kernel Calculation for Training Data
H = (y * y') .* exp(-pdist2(X, X).^2 / (2 * sigma^2));
```

```matlab
% Define the linear term in the dual objective
f = -ones(n_samples, 1);

% Soft margin SVM dual formulation with RBF kernel
C = 10; % Regularization parameter (adjust as needed)
tolerance = 1e-5;

% Solve the dual problem using CVX
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * alfa' * H * alfa + f' * alfa)
    subject to
        y' * alfa == 0;
        0 <= alfa <= C;
cvx_end

% Extract support vectors (0 < alfa < C)
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);

    % Extract support vectors
    alfa_sv = alfa(support_vector_indices);
    y_sv = y(support_vector_indices);
    X_sv = X(support_vector_indices, :);

    % Bias term (calculated from support vectors)
    b = mean(y_sv - sum(alfa_sv .* y_sv .* exp(-pdist2(X_sv, X_sv).^2 / (2 *
sigma^2)), 2));

    %% Decision boundary plotting for RBF kernel
    % Generate a grid for plotting decision boundary
    x_range = linspace(min(X(:,1)), max(X(:,1)), 100);
    y_range = linspace(min(X(:,2)), max(X(:,2)), 100);
    [xx, yy] = meshgrid(x_range, y_range);
    grid_points = [xx(:), yy(:)];  % Flatten the grid into a list of points

    % Compute RBF kernel between grid points and training data
    K_grid = exp(-pdist2(grid_points, X).^2 / (2 * sigma^2));

    % Compute decision function for each grid point
    %decision_function_grid = sum(alfa .* y .* K_grid, 2) + b;
    decision_function_grid = K_grid * (alfa .* y) + b;

    % Reshape the decision function to the grid shape for plotting
    decision_boundary_values = reshape(decision_function_grid, size(xx));

    % Plot decision boundary (where decision function = 0)
    figure;
    contour(xx, yy, decision_boundary_values, [0 0], 'k--', 'LineWidth', 2);
    hold on;

    % Plot training points and support vectors
    gscatter(X(:,1), X(:,2), y, 'rb', 'xo');
    scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor',
'k');

    xlabel('concavePointsWorst');
    ylabel('areaWorst');
    title('SVM with RBF Kernel - Decision Boundary and Support Vectors');
    legend('Decision Boundary', 'Class -1', 'Class 1', 'Support Vectors');
    grid on;
    hold off;
end
```

```
Number of support vectors: 99
```



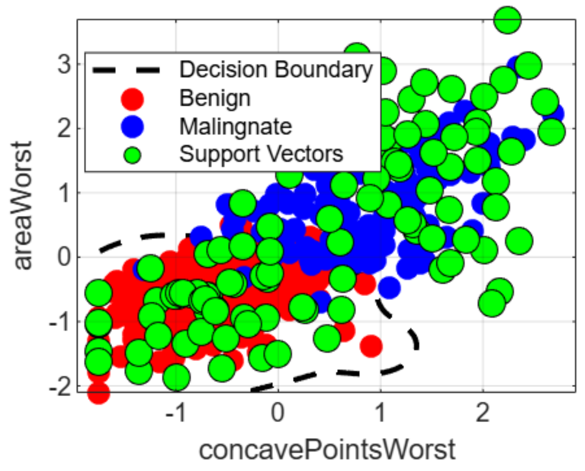
```
% trying kernel with performance matrices

% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

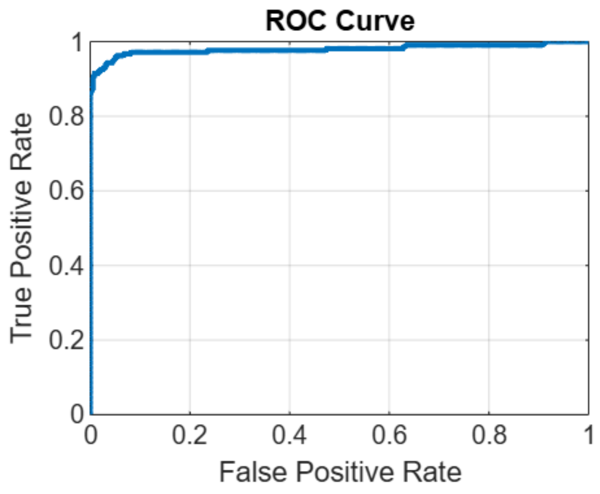
```
% Extract features and target variable
X = data{:, {'concavePointsWorst', 'areaWorst'}};
%X=data{:,3:end};
```

```
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X = zscore(X);

% Parameters for RBF kernel
sigma = 0.30;  % Adjust sigma for the RBF kernel

% Get the number of samples and features
[n_samples, n_features] = size(X);

% RBF Kernel Calculation for Training Data
H = (y * y') .* exp(-pdist2(X, X).^2 / (2 * sigma^2));

% Define the linear term in the dual objective
f = -ones(n_samples, 1);

% Soft margin SVM dual formulation with RBF kernel
C = 1; % Regularization parameter (adjust as needed)
tolerance = 1e-5;

% Solve the dual problem using CVX
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * alfa' * H * alfa + f' * alfa)
    subject to
        y' * alfa == 0;
        0 <= alfa <= C;
cvx_end

% Extract support vectors (0 < alfa < C)
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);

    % Extract support vectors
    alfa_sv = alfa(support_vector_indices);
    y_sv = y(support_vector_indices);
    X_sv = X(support_vector_indices, :);
    % Bias term (calculated from support vectors)
    b = mean(y_sv - sum(alfa_sv .* y_sv .* exp(-pdist2(X_sv, X_sv).^2 / (2 *
sigma^2)), 2));

    %% Decision boundary plotting for RBF kernel
    % Generate a grid for plotting decision boundary
    x_range = linspace(min(X(:,1)), max(X(:,1)), 100);
    y_range = linspace(min(X(:,2)), max(X(:,2)), 100);
    [xx, yy] = meshgrid(x_range, y_range);
    grid_points = [xx(:), yy(:)];  % Flatten the grid into a list of points

    % Compute RBF kernel between grid points and training data
    K_grid = exp(-pdist2(grid_points, X).^2 / (2 * sigma^2));

    % Compute decision function for each grid point
    decision_function_grid = K_grid * (alfa .* y) + b;

    % Reshape the decision function to the grid shape for plotting
    decision_boundary_values = reshape(decision_function_grid, size(xx));

    % Plot decision boundary (where decision function = 0)
    figure;
    contour(xx, yy, decision_boundary_values, [0 0], 'k--', 'LineWidth', 2);
    hold on;

    % Plot training points and support vectors
    gscatter(X(:,1), X(:,2), y, 'rb','o',8,'filled');
    scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled', 'MarkerEdgeColor',
'k');

    xlabel('concavePointsWorst');
    ylabel('areaWorst');
    %title('SVM with RBF Kernel - Decision Boundary and Support Vectors');
    legend('Decision Boundary', 'Benign', 'Malingnate', 'Support Vectors');
    grid on;
    hold off;

    %% Performance Metrics Calculation

    % Compute RBF kernel for predictions on training data
    K_train = exp(-pdist2(X, X).^2 / (2 * sigma^2));

    % Calculate decision function for training data
    decision_function_train = K_train * (alfa .* y) + b;

    % Predictions
    y_pred = sign(decision_function_train);
    y_pred(y_pred == 0) = 1; % Handle zero predictions

    % Performance metrics
    true_positive = sum((y_pred == 1) & (y == 1));
    true_negative = sum((y_pred == -1) & (y == -1));
    false_positive = sum((y_pred == 1) & (y == -1));
    false_negative = sum((y_pred == -1) & (y == 1));

    % Accuracy
    accuracy = (true_positive + true_negative) / n_samples;

    % Sensitivity (Recall)
    sensitivity = true_positive / (true_positive + false_negative);

    % Specificity
    specificity = true_negative / (true_negative + false_positive);

    % Precision
    precision = true_positive / (true_positive + false_positive);

    % F1 Score
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

    % ROC and AUC
    [roc_x, roc_y, ~, auc] = perfcurve(y, decision_function_train, 1);

    % Display results
    fprintf('Bias (b): %.4f\n', b);
    fprintf('Accuracy: %.2f%%\n', accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity * 100);
    fprintf('Specificity: %.2f%%\n', specificity * 100);
    fprintf('Precision: %.2f%%\n', precision * 100);
    fprintf('F1 Score: %.2f%%\n', f1_score * 100);
    fprintf('AUC: %.2f\n', auc);

    % Plot ROC curve
    figure;
    plot(roc_x, roc_y, 'LineWidth', 2);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title('ROC Curve');
    grid on;

end
```

```
Number of support vectors: 105
```



```
Bias (b): 0.3580
Accuracy: 95.61%
Sensitivity (Recall): 92.45%
Specificity: 97.48%
Precision: 95.61%
F1 Score: 94.00%
AUC: 0.98
```



Extending to all the 30 features

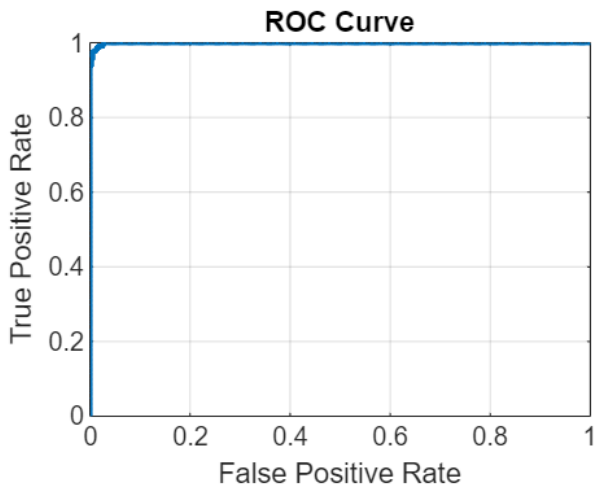
```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
% Extract features and target variable (all 30 features)
%X = data{:, 3:end}; % All features from column 3 onwards
```

```
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X = zscore(X);

% Parameters for RBF kernel
sigma = 0.30;  % Adjust sigma for the RBF kernel

% Get the number of samples and features
[n_samples, n_features] = size(X);

% RBF Kernel Calculation for Training Data
H = (y * y') .* exp(-pdist2(X, X).^2 / (2 * sigma^2));

% Define the linear term in the dual objective
f = -ones(n_samples, 1);

% Soft margin SVM dual formulation with RBF kernel
C = 5.0; % Regularization parameter (adjust as needed)
tolerance = 1e-5;

% Solve the dual problem using CVX
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * alfa' * H * alfa + f' * alfa)
    subject to
        y' * alfa == 0;
        0 <= alfa <= C;
cvx_end

% Extract support vectors (0 < alfa < C)
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);

    % Extract support vectors
    alfa_sv = alfa(support_vector_indices);
    y_sv = y(support_vector_indices);
    X_sv = X(support_vector_indices, :);
    % Bias term (calculated from support vectors)
    b = mean(y_sv - sum(alfa_sv .* y_sv .* exp(-pdist2(X_sv, X_sv).^2 / (2 *
sigma^2)), 2));

    %% Performance Metrics Calculation

    % Compute RBF kernel for predictions on training data
    K_train = exp(-pdist2(X, X).^2 / (2 * sigma^2));

    % Calculate decision function for training data
    decision_function_train = K_train * (alfa .* y) + b;

    % Predictions
    y_pred = sign(decision_function_train);
    y_pred(y_pred == 0) = 1; % Handle zero predictions

    % Performance metrics
    true_positive = sum((y_pred == 1) & (y == 1));
    true_negative = sum((y_pred == -1) & (y == -1));
    false_positive = sum((y_pred == 1) & (y == -1));
    false_negative = sum((y_pred == -1) & (y == 1));

    % Accuracy
    accuracy = (true_positive + true_negative) / n_samples;

    % Sensitivity (Recall)
    sensitivity = true_positive / (true_positive + false_negative);

    % Specificity
    specificity = true_negative / (true_negative + false_positive);

    % Precision
    precision = true_positive / (true_positive + false_positive);

    % F1 Score
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

    % ROC and AUC
    [roc_x, roc_y, ~, auc] = perfcurve(y, decision_function_train, 1);

    % Display results
    fprintf('Bias (b): %.4f\n', b);
    fprintf('Accuracy: %.2f%%\n', accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity * 100);
    fprintf('Specificity: %.2f%%\n', specificity * 100);
    fprintf('Precision: %.2f%%\n', precision * 100);
    fprintf('F1 Score: %.2f%%\n', f1_score * 100);
    fprintf('AUC: %.2f\n', auc);

    % Plot ROC curve
    figure;
    plot(roc_x, roc_y, 'LineWidth', 2);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title('ROC Curve');
    grid on;

end
```

```
Number of support vectors: 249
Bias (b): 0.7917
Accuracy: 98.59%
Sensitivity (Recall): 98.58%
Specificity: 98.60%
Precision: 97.66%
F1 Score: 98.12%
AUC: 1.00
```



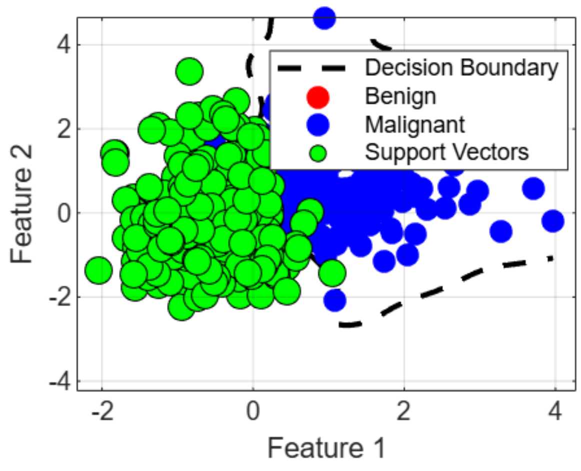
```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable (all 30 features)
X_full = data{:, 3:end}; % All features from column 3 onwards
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X_full = zscore(X_full);
```

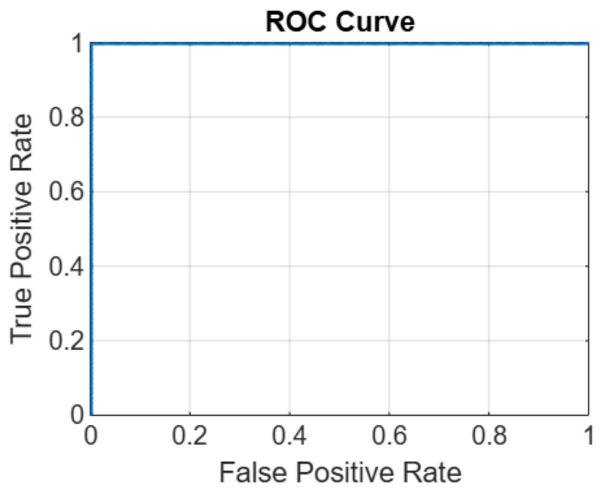
```
% Parameters for RBF kernel
sigma = 0.50;  % Adjust sigma for the RBF kernel

% Get the number of samples and features
[n_samples, n_features] = size(X_full);

% RBF Kernel Calculation for Training Data
H = (y * y') .* exp(-pdist2(X_full, X_full).^2 / (2 * sigma^2));

% Define the linear term in the dual objective
f = -ones(n_samples, 1);

% Soft margin SVM dual formulation with RBF kernel
C = 1; % Regularization parameter (adjust as needed)
tolerance = 1e-5;

% Solve the dual problem using CVX
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * alfa' * H * alfa + f' * alfa)
    subject to
        y' * alfa == 0;
        0 <= alfa <= C;
cvx_end

% Extract support vectors (0 < alfa < C)
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);

    % Extract support vectors
    alfa_sv = alfa(support_vector_indices);
    y_sv = y(support_vector_indices);
    X_sv = X_full(support_vector_indices, :);

    % Bias term (calculated from support vectors)
    b = mean(y_sv - sum(alfa_sv .* y_sv .* exp(-pdist2(X_sv, X_sv).^2 / (2 *
sigma^2)), 2));

    %% Decision boundary plotting for RBF kernel (using the first two
features)
    % Extract the first two features for plotting
    X = X_full(:, 1:2); % Use only the first two features for visualization

    % Generate a grid for plotting decision boundary
    x_range = linspace(min(X(:, 1)), max(X(:, 1)), 100);
    y_range = linspace(min(X(:, 2)-2), max(X(:, 2)), 100);
    [xx, yy] = meshgrid(x_range, y_range);
    grid_points = [xx(:), yy(:)];  % Flatten the grid into a list of points

    % Compute RBF kernel between grid points and training data
    K_grid = exp(-pdist2(grid_points, X).^2 / (2 * sigma^2));

    % Compute decision function for each grid point
    decision_function_grid = K_grid * (alfa .* y) + b;

    % Reshape the decision function to the grid shape for plotting
    decision_boundary_values = reshape(decision_function_grid, size(xx));

    % Plot decision boundary (where decision function = 0)
    figure;
    contour(xx, yy, decision_boundary_values, [0 0], 'k--', 'LineWidth', 2);
    hold on;

    % Plot training points and support vectors
    gscatter(X(:,1), X(:,2), y, 'rb','o',8,'filled');
    scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled','MarkerEdgeColor', 'k');

    xlabel('Feature 1');
    ylabel('Feature 2');
    %title('SVM with RBF Kernel - Decision Boundary and Support Vectors');
    legend('Decision Boundary', 'Benign', 'Malignant', 'Support Vectors');
    grid on;
    hold off;

    %% Performance Metrics Calculation

    % Compute RBF kernel for predictions on training data
    K_train = exp(-pdist2(X_full, X_full).^2 / (2 * sigma^2));

    % Calculate decision function for training data
    decision_function_train = K_train * (alfa .* y) + b;

    % Predictions
    y_pred = sign(decision_function_train);
    y_pred(y_pred == 0) = 1; % Handle zero predictions

    % Performance metrics
    true_positive = sum((y_pred == 1) & (y == 1));
    true_negative = sum((y_pred == -1) & (y == -1));
    false_positive = sum((y_pred == 1) & (y == -1));
    false_negative = sum((y_pred == -1) & (y == 1));

    % Accuracy
    accuracy = (true_positive + true_negative) / n_samples;
    % Sensitivity (Recall)
    sensitivity = true_positive / (true_positive + false_negative);

    % Specificity
    specificity = true_negative / (true_negative + false_positive);

    % Precision
    precision = true_positive / (true_positive + false_positive);

    % F1 Score
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

    % ROC and AUC
    [roc_x, roc_y, ~, auc] = perfcurve(y, decision_function_train, 1);

    % Display results
    fprintf('Bias (b): %.4f\n', b);
    fprintf('Accuracy: %.2f%%\n', accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity * 100);
    fprintf('Specificity: %.2f%%\n', specificity * 100);
    fprintf('Precision: %.2f%%\n', precision * 100);
    fprintf('F1 Score: %.2f%%\n', f1_score * 100);
    fprintf('AUC: %.2f\n', auc);

    % Plot ROC curve
    figure;
    plot(roc_x, roc_y, 'LineWidth', 2);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title('ROC Curve');
    grid on;
end
```

```
Number of support vectors: 357
```



```
Bias (b): -0.4005
Accuracy: 100.00%
```

```
Sensitivity (Recall): 100.00%
Specificity: 100.00%
Precision: 100.00%
F1 Score: 100.00%
AUC: 1.00
```



SVM on Train-test split

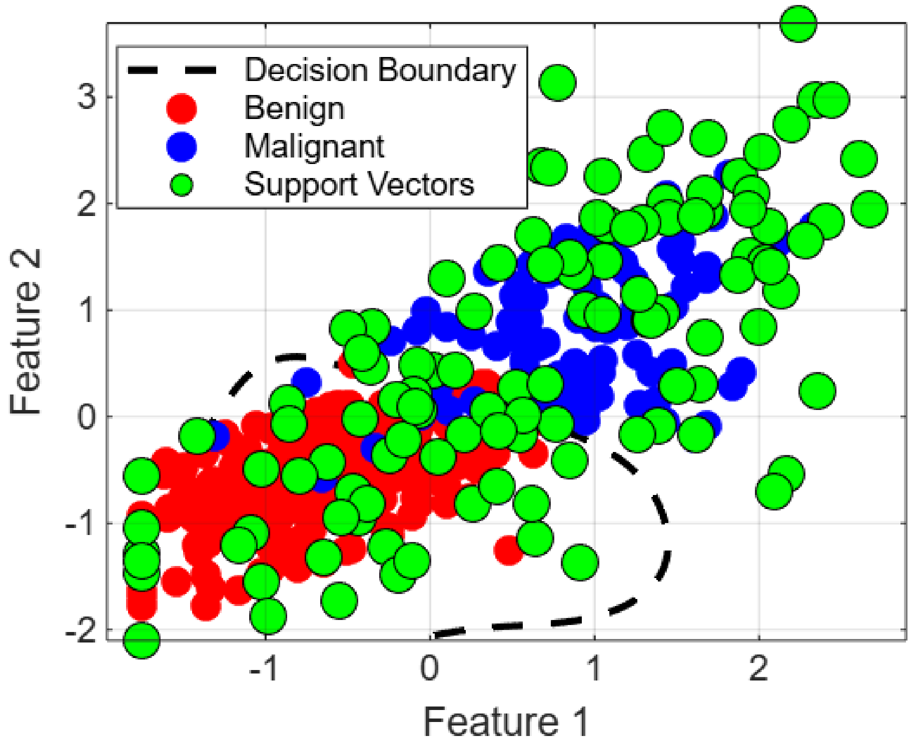
```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable (all 30 features)
X_full = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
%X_full = data{:, 3:end}; % All features from column 3 onwards
y = data.diagnosis; % Assuming 'diagnosis' is the name of your target
variable

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X_full = zscore(X_full);

% 80-20 Train-Test Split
cv = cvpartition(y, 'HoldOut', 0.2);
X_train = X_full(cv.training, :);
y_train = y(cv.training);
X_test = X_full(cv.test, :);
```

```
y_test = y(cv.test);

% Parameters for RBF kernel
sigma = 0.50;  % Adjust sigma for the RBF kernel

% Get the number of training samples and features
[n_samples, n_features] = size(X_train);

% RBF Kernel Calculation for Training Data
H = (y_train * y_train') .* exp(-pdist2(X_train, X_train).^2 / (2 *
sigma^2));

% Define the linear term in the dual objective
f = -ones(n_samples, 1);

% Soft margin SVM dual formulation with RBF kernel
C = 5; % Regularization parameter (adjust as needed)
tolerance = 1e-5;
%rng(42); % Random number seed
% Solve the dual problem using CVX
cvx_begin quiet
    variable alfa(n_samples)
    minimize(0.5 * alfa' * H * alfa + f' * alfa)
    subject to
        y_train' * alfa == 0;
        0 <= alfa <= C;
cvx_end

% Extract support vectors (0 < alfa < C)
support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

if isempty(support_vector_indices)
    disp('No support vectors found. Consider revising the regularization
parameter C.');
else
    disp(['Number of support vectors: ',
num2str(length(support_vector_indices))]);

    % Extract support vectors
    alfa_sv = alfa(support_vector_indices);
    y_sv = y_train(support_vector_indices);
    X_sv = X_train(support_vector_indices, :);

    % Bias term (calculated from support vectors)
    b = mean(y_sv - sum(alfa_sv .* y_sv .* exp(-pdist2(X_sv, X_sv).^2 / (2 *
sigma^2)), 2));

    %% Decision boundary plotting for RBF kernel (using the first two
features)
    % Extract the first two features for plotting
    X = X_train(:, 1:2); % Use only the first two features for visualization

    % Generate a grid for plotting decision boundary
    x_range = linspace(min(X(:, 1)), max(X(:, 1)), 100);
    y_range = linspace(min(X(:, 2)), max(X(:, 2)), 100);
    [xx, yy] = meshgrid(x_range, y_range);
    grid_points = [xx(:), yy(:)];  % Flatten the grid into a list of points

    % Compute RBF kernel between grid points and training data
    K_grid = exp(-pdist2(grid_points, X).^2 / (2 * sigma^2));

    % Compute decision function for each grid point
    decision_function_grid = K_grid * (alfa .* y_train) + b;

    % Reshape the decision function to the grid shape for plotting
    decision_boundary_values = reshape(decision_function_grid, size(xx));

    % Plot decision boundary (where decision function = 0)
    figure;
    contour(xx, yy, decision_boundary_values, [0 0], 'k--', 'LineWidth', 2);
    hold on;

    % Plot training points and support vectors
    gscatter(X(:,1), X(:,2), y_train, 'rb','o',8,'filled');
    scatter(X_sv(:,1), X_sv(:,2), 100, 'g', 'filled','MarkerEdgeColor', 'k');

    xlabel('Feature 1');
    ylabel('Feature 2');
    legend('Decision Boundary', 'Benign', 'Malignant', 'Support Vectors');
    grid on;
    hold off;

    %% Performance Metrics Calculation on Training Data

    % Compute RBF kernel for predictions on training data
    K_train = exp(-pdist2(X_train, X_train).^2 / (2 * sigma^2));

    % Calculate decision function for training data
    decision_function_train = K_train * (alfa .* y_train) + b;

    % Predictions for training data
    y_pred_train = sign(decision_function_train);
    y_pred_train(y_pred_train == 0) = 1; % Handle zero predictions

    % Performance metrics for training data
    [train_accuracy, train_sensitivity, train_specificity, train_precision,
train_f1, train_auc] = calculate_metrics(y_train, y_pred_train,
decision_function_train);

    % Display training results
    fprintf('Training Results:\n');
    fprintf('Bias (b): %.4f\n', b);
    fprintf('Accuracy: %.2f%%\n', train_accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f%%\n', train_sensitivity * 100);
    fprintf('Specificity: %.2f%%\n', train_specificity * 100);
    fprintf('Precision: %.2f%%\n', train_precision * 100);
    fprintf('F1 Score: %.2f%%\n', train_f1 * 100);
    fprintf('AUC: %.2f\n', train_auc);

    %% Performance Metrics Calculation on Test Data

    % Compute RBF kernel for predictions on test data
    K_test = exp(-pdist2(X_test, X_train).^2 / (2 * sigma^2)); % Use
training data for kernel

    % Calculate decision function for test data
    decision_function_test = K_test * (alfa .* y_train) + b;

    % Predictions for test data
    y_pred_test = sign(decision_function_test);
    y_pred_test(y_pred_test == 0) = 1; % Handle zero predictions

    % Performance metrics for test data
    [test_accuracy, test_sensitivity, test_specificity, test_precision,
test_f1, test_auc] = calculate_metrics(y_test, y_pred_test,
decision_function_test);

    % Display test results
    fprintf('\nTest Results:\n');
    fprintf('Accuracy: %.2f%%\n', test_accuracy * 100);
    fprintf('Sensitivity (Recall): %.2f%%\n', test_sensitivity * 100);
    fprintf('Specificity: %.2f%%\n', test_specificity * 100);
    fprintf('Precision: %.2f%%\n', test_precision * 100);
    fprintf('F1 Score: %.2f%%\n', test_f1 * 100);
    fprintf('AUC: %.2f\n', test_auc);
end
```

```
Number of support vectors: 113
```



```
Training Results:
Bias (b): 1.1365
Accuracy: 96.05%
Sensitivity (Recall): 97.06%
Specificity: 95.45%
Precision: 92.70%
F1 Score: 94.83%
AUC: 1.00
Test Results:
Accuracy: 96.46%
Sensitivity (Recall): 100.00%
Specificity: 94.37%
Precision: 91.30%
F1 Score: 95.45%
AUC: 1.00
```

```
function [accuracy, sensitivity, specificity, precision, f1, auc] =
calculate_metrics(y_true, y_pred, decision_function)
    % Performance metrics calculation
    true_positive = sum((y_pred == 1) & (y_true == 1));
    true_negative = sum((y_pred == -1) & (y_true == -1));
    false_positive = sum((y_pred == 1) & (y_true == -1));
    false_negative = sum((y_pred == -1) & (y_true == 1));

    % Accuracy
    accuracy = (true_positive + true_negative) / length(y_true);

    % Sensitivity (Recall)
    sensitivity = true_positive / (true_positive + false_negative);

    % Specificity
    specificity = true_negative / (true_negative + false_positive);

    % Precision
    precision = true_positive / (true_positive + false_positive);

    % F1 Score
    f1 = 2 * (precision * sensitivity) / (precision + sensitivity);

    % ROC and AUC
    [roc_x, roc_y, ~, auc] = perfcurve(y_true, decision_function, 1);
end
```

Nexted Cross Validation

```
% Load your dataset
data = readtable('breast-cancer-wisconsin-data.csv');

% Define the correct variable names
correctVarNames = {'id', 'diagnosis', 'radius mean', 'texture mean',
'perimeter mean', ...
                   'area mean', 'smoothness mean', 'compactness mean',
'concavity mean', ...
                   'concave points mean', 'symmetry mean', 'fractal
dimension mean', ...
                   'radius se', 'texture se', 'perimeter se', 'area se', ...
                   'smoothness se', 'compactness se', 'concavity se', ...
                   'concave points se', 'symmetry se', 'fractal dimension
se', ...
                   'radius worst', 'texture worst', 'perimeter worst', ...
                   'area worst', 'smoothness worst', 'compactness worst', ...
                   'concavity worst', 'concave points worst', 'symmetry
worst', ...
                   'fractal dimension worst'};

% Assign the correct variable names
data.Properties.VariableNames = correctVarNames;

% Convert diagnosis to numeric (1 for malignant, -1 for benign)
data.diagnosis = double(strcmp(data.diagnosis, 'M'));
y = data.diagnosis; % Target variable
X = data{:, 3:end}; % Extract features (assuming features start from the 3rd
column)

% Nested Cross-Validation parameters
outer_k = 5; % Outer fold for assessing generalization
inner_k = 5; % Inner fold for hyperparameter tuning

% Variable to store outer performance metrics
```

```
outer_accuracy = zeros(outer_k, 1);
outer_sensitivity = zeros(outer_k, 1);
outer_specificity = zeros(outer_k, 1);
outer_f1_scores = zeros(outer_k, 1);

% Outer cross-validation
outer_indices = crossvalind('Kfold', y, outer_k);

for outer_fold = 1:outer_k
    test_idx_outer = (outer_indices == outer_fold); % Index for outer test
fold
    train_idx_outer = ~test_idx_outer; % Index for outer train fold

    % Extract outer training and testing data
    X_train_outer = X(train_idx_outer, :);
    y_train_outer = y(train_idx_outer, :);
    X_test_outer = X(test_idx_outer, :);
    y_test_outer = y(test_idx_outer, :);

    % Inner Cross-Validation for hyperparameter tuning
    inner_indices = crossvalind('Kfold', y_train_outer, inner_k);

    best_c = 0; % Variable to store the best penalty parameter
    best_accuracy = 0; % Variable to store the best accuracy

    % Inner cross-validation
    for inner_fold = 1:inner_k
        test_idx_inner = (inner_indices == inner_fold); % Index for inner
test fold
        train_idx_inner = ~test_idx_inner; % Index for inner train fold

        % Extract inner training and testing data
        X_train_inner = X_train_outer(train_idx_inner, :);
        y_train_inner = y_train_outer(train_idx_inner, :);
        X_test_inner = X_train_outer(test_idx_inner, :);
        y_test_inner = y_train_outer(test_idx_inner, :);

        % Hyperparameter tuning - test various values of c
        c_values = [1, 10, 100]; % Possible values for the penalty parameter
        for c = c_values
            % Data mapping to high dimensions
            dim = 150; % Target dimensionality
            rng(2545); % Random number seed
            R = 2 * randn(size(X_train_inner, 2), dim); % Random projection
matrix

            % Map data to high-dimensional space
            M_data_train = X_train_inner * R; % Mapped training data
            data_RKS_train = [cos(M_data_train), sin(M_data_train)]; % RKS
mapped training data
```

```
            % Prepare for SVM classification using CVX
            n_mapped = size(data_RKS_train, 2); % Number of features after
mapping
            e = ones(size(data_RKS_train, 1), 1); % m-tuple one-vector

            % CVX optimization for SVM
            cvx_begin quiet
                variables w(n_mapped) g Psi(size(data_RKS_train, 1))
                minimize ((0.5 * w' * w) + (c * sum(Psi)))
                subject to
                    y_train_inner .* (data_RKS_train * w - g * e) + Psi - e
>= 0;
                    Psi >= 0;
            cvx_end

            % Classification on inner test data
            M_data_test_inner = X_test_inner * R; % Mapped test data
            data_RKS_test_inner = [cos(M_data_test_inner),
sin(M_data_test_inner)]; % RKS mapped test data

            z_test_inner = sign(data_RKS_test_inner * w - g); % Predictions
on inner test data

            % Compute accuracy for inner fold
            TP_inner = sum((z_test_inner == 1) & (y_test_inner == 1)); %
True Positives
            TN_inner = sum((z_test_inner == -1) & (y_test_inner == -1)); %
True Negatives
            accuracy_inner = (TP_inner + TN_inner) / length(y_test_inner);

            % Update best hyperparameter if current accuracy is better
            if accuracy_inner > best_accuracy
                best_accuracy = accuracy_inner;
                best_c = c; % Store the best penalty parameter
            end
        end
    end

    % Now train the model with the best_c on the outer training set
    % Use best_c to perform mapping and training on the outer training set
    dim = 150; % Target dimensionality
    rng(2545); % Random number seed
    R = 2 * randn(size(X_train_outer, 2), dim); % Random projection matrix
    M_data_train_outer = X_train_outer * R; % Mapped training data
    data_RKS_train_outer = [cos(M_data_train_outer),
sin(M_data_train_outer)]; % RKS mapped training data

    % Prepare for SVM classification using CVX
    n_mapped_outer = size(data_RKS_train_outer, 2); % Number of features
after mapping
    e_outer = ones(size(data_RKS_train_outer, 1), 1); % m-tuple one-vector

    % CVX optimization for SVM
    cvx_begin quiet
        variables w_outer(n_mapped_outer) g_outer
Psi_outer(size(data_RKS_train_outer, 1))
        minimize ((0.5 * w_outer' * w_outer) + (best_c * sum(Psi_outer)))
        subject to
            y_train_outer .* (data_RKS_train_outer * w_outer - g_outer *
e_outer) + Psi_outer - e_outer >= 0;
            Psi_outer >= 0;
    cvx_end

    % Classification on the outer test data
    M_data_test_outer = X_test_outer * R; % Mapped test data
    data_RKS_test_outer = [cos(M_data_test_outer), sin(M_data_test_outer)];
% RKS mapped test data

    z_test_outer = sign(data_RKS_test_outer * w_outer - g_outer); %
Predictions on outer test data

    % Compute performance metrics for outer test set
    TP_outer = sum((z_test_outer == 1) & (y_test_outer == 1)); % True
Positives
    TN_outer = sum((z_test_outer == -1) & (y_test_outer == -1)); % True
Negatives
    FP_outer = sum((z_test_outer == 1) & (y_test_outer == -1)); % False
Positives
    FN_outer = sum((z_test_outer == -1) & (y_test_outer == 1)); % False
Negatives

    % Store outer metrics
    outer_accuracy(outer_fold) = (TP_outer + TN_outer) /
length(y_test_outer);
    outer_sensitivity(outer_fold) = TP_outer / (TP_outer + FN_outer);
    outer_specificity(outer_fold) = TN_outer / (TN_outer + FP_outer);
    precision_outer = TP_outer / (TP_outer + FP_outer);
    recall_outer = TP_outer / (TP_outer + FN_outer);
    outer_f1_scores(outer_fold) = 2 * (precision_outer * recall_outer) /
(precision_outer + recall_outer);
end

% Average metrics across outer folds
mean_outer_accuracy = mean(outer_accuracy);
mean_outer_sensitivity = mean(outer_sensitivity);
mean_outer_specificity = mean(outer_specificity);
mean_outer_f1_score = mean(outer_f1_scores);
```

```
fprintf('Average Outer Accuracy: %.2f%%\n', mean_outer_accuracy * 100);
```

```
Average Outer Accuracy: 37.26%
```

```
fprintf('Average Outer Sensitivity: %.2f%%\n', mean_outer_sensitivity * 100);
```

```
Average Outer Sensitivity: 100.00%
```

```
fprintf('Average Outer Specificity: %.2f%%\n', mean_outer_specificity * 100);
```

```
Average Outer Specificity: NaN%
```

```
fprintf('Average Outer F1-score: %.2f%%\n', mean_outer_f1_score * 100);
```

```
Average Outer F1-score: 100.00%
```

Fine tuning the model on five top correlated subset

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
y = data.diagnosis;

% Convert y to -1 and 1
y(y == 0) = -1;
y(y == 1) = 1;

% Normalize the features (Z-score normalization)
X = zscore(X);

% Parameters for RBF kernel
sigma = 0.30;  % Adjust sigma for the RBF kernel
C = 5.0; % Regularization parameter (adjust as needed)
tolerance = 1e-5;
n_folds = 5;  % Number of folds for cross-validation

% Initialize metrics to store results for each fold
accuracy_scores = zeros(n_folds, 1);
sensitivity_scores = zeros(n_folds, 1);
specificity_scores = zeros(n_folds, 1);
precision_scores = zeros(n_folds, 1);
f1_scores = zeros(n_folds, 1);
auc_scores = zeros(n_folds, 1);
```

```
% Set random seed for reproducibility
rng(1);

% Perform k-fold cross-validation
cv = cvpartition(y, 'KFold', n_folds);

for fold = 1:n_folds
    % Split the data into training and validation sets
    train_idx = cv.training(fold);
    test_idx = cv.test(fold);

    X_train = X(train_idx, :);
    y_train = y(train_idx);
    X_test = X(test_idx, :);
    y_test = y(test_idx);

    % Get the number of training samples
    [n_samples_train, ~] = size(X_train);

    % RBF Kernel Calculation for Training Data
    H = (y_train * y_train') .* exp(-pdist2(X_train, X_train).^2 / (2 *
sigma^2));

    % Define the linear term in the dual objective
    f = -ones(n_samples_train, 1);

    % Solve the dual problem using CVX
    cvx_begin quiet
        variable alfa(n_samples_train)
        minimize(0.5 * alfa' * H * alfa + f' * alfa)
        subject to
            y_train' * alfa == 0;
            0 <= alfa <= C;
    cvx_end

    % Extract support vectors (0 < alfa < C)
    support_vector_indices = find(alfa > tolerance & alfa < C - tolerance);

    % Bias term (calculated from support vectors)
    if isempty(support_vector_indices)
        disp('No support vectors found in fold.');
        b = 0;
    else
        alfa_sv = alfa(support_vector_indices);
        y_sv = y_train(support_vector_indices);
        X_sv = X_train(support_vector_indices, :);

        b = mean(y_sv - sum(alfa_sv .* y_sv .* exp(-pdist2(X_sv, X_sv).^2 /
(2 * sigma^2)), 2));
```

```
    end

    %% Performance Metrics Calculation for Validation Data

    % Compute RBF kernel for predictions on test data
    K_test = exp(-pdist2(X_test, X_train).^2 / (2 * sigma^2));

    % Calculate decision function for test data
    decision_function_test = K_test * (alfa .* y_train) + b;

    % Predictions on the test set
    y_pred_test = sign(decision_function_test);
    y_pred_test(y_pred_test == 0) = 1; % Handle zero predictions

    % Compute performance metrics
    true_positive = sum((y_pred_test == 1) & (y_test == 1));
    true_negative = sum((y_pred_test == -1) & (y_test == -1));
    false_positive = sum((y_pred_test == 1) & (y_test == -1));
    false_negative = sum((y_pred_test == -1) & (y_test == 1));

    % Accuracy
    accuracy_scores(fold) = (true_positive + true_negative) / length(y_test);

    % Sensitivity (Recall)
    sensitivity_scores(fold) = true_positive / (true_positive +
false_negative);

    % Specificity
    specificity_scores(fold) = true_negative / (true_negative +
false_positive);

    % Precision
    precision_scores(fold) = true_positive / (true_positive +
false_positive);

    % F1 Score
    f1_scores(fold) = 2 * (precision_scores(fold)
* sensitivity_scores(fold)) / (precision_scores(fold) +
sensitivity_scores(fold));

    % ROC and AUC
    [roc_x, roc_y, ~, auc_scores(fold)] = perfcurve(y_test,
decision_function_test, 1);
end

% Calculate mean and standard deviation for each metric
mean_accuracy = mean(accuracy_scores);
std_accuracy = std(accuracy_scores);
mean_sensitivity = mean(sensitivity_scores);
std_sensitivity = std(sensitivity_scores);
```

```
mean_specificity = mean(specificity_scores);
std_specificity = std(specificity_scores);
mean_precision = mean(precision_scores);
std_precision = std(precision_scores);
mean_f1 = mean(f1_scores);
std_f1 = std(f1_scores);
mean_auc = mean(auc_scores);
std_auc = std(auc_scores);

% Display cross-validated results
fprintf('Cross-Validated Metrics over %d folds:\n', n_folds);
```

```
Cross-Validated Metrics over 5 folds:
```

```
fprintf('Accuracy: %.2f%% ± %.2f%%\n', mean_accuracy * 100, std_accuracy *
100);
```

```
Accuracy: 92.62% ± 0.80%
```

```
fprintf('Sensitivity: %.2f%% ± %.2f%%\n', mean_sensitivity * 100,
std_sensitivity * 100);
```

```
Sensitivity: 95.28% ± 2.35%
```

```
fprintf('Specificity: %.2f%% ± %.2f%%\n', mean_specificity * 100,
std_specificity * 100);
```

```
Specificity: 91.03% ± 1.65%
```

```
fprintf('Precision: %.2f%% ± %.2f%%\n', mean_precision * 100, std_precision
* 100);
```

```
Precision: 86.39% ± 1.83%
```

```
fprintf('F1 Score: %.2f%% ± %.2f%%\n', mean_f1 * 100, std_f1 * 100);
```

```
F1 Score: 90.59% ± 0.93%
```

```
fprintf('AUC: %.2f ± %.2f\n', mean_auc, std_auc);
```

```
AUC: 0.97 ± 0.02
```

Using Benchmark SVM from Matlab

```
% Load and preprocess data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, 3:10};
%X = data{:, {'concavePointsWorst', 'areaWorst'}};
y = data.diagnosis;
y(y == 0) = -1;
y(y == 1) = 1;
X = zscore(X);
C=1;
% Train SVM using built-in function
SVMModel = fitcsvm(X, y, 'KernelFunction', 'linear', 'BoxConstraint', C,
'Standardize', false);

% Predict using built-in SVM
[y_pred_builtin, score_builtin] = predict(SVMModel, X);

% Calculate performance metrics
confMat = confusionmat(y, y_pred_builtin);
TP = confMat(2,2);
TN = confMat(1,1);
FP = confMat(1,2);
FN = confMat(2,1);

accuracy = (TP + TN) / sum(confMat(:));
sensitivity = TP / (TP + FN);
specificity = TN / (TN + FP);
precision = TP / (TP + FP);
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
[~, ~, ~, auc_builtin] = perfcurve(y, score_builtin(:,2), 1);

% Display results
fprintf('Built-in SVM Results:\n');
```

```
Built-in SVM Results:
```

```
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
```

```
Accuracy: 94.38%
```

```
fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity * 100);
```

```
Sensitivity (Recall): 90.09%
```

```
fprintf('Specificity: %.2f%%\n', specificity * 100);
```

```
Specificity: 96.92%
```

```
fprintf('Precision: %.2f%%\n', precision * 100);
```

```
Precision: 94.55%
```

```
fprintf('F1 Score: %.2f%%\n', f1_score * 100);
```

```
F1 Score: 92.27%
```

```
fprintf('AUC: %.2f\n', auc_builtin);
```

```
AUC: 0.99
```

Built-in SVM with non-linear kernel

```
% Load and preprocess data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, 3:10};
%X = data{:, {'concavePointsWorst', 'areaWorst'}};
y = data.diagnosis;
y(y == 0) = -1;
y(y == 1) = 1;
X = zscore(X);

% Train SVM using built-in function
%SVMModel = fitcsvm(X, y, 'KernelFunction', 'RBF', 'BoxConstraint', C,
'Standardize', false);
SVMModel = fitcsvm(X, y, 'KernelFunction', 'RBF', 'BoxConstraint', c,
'Standardize', false);
% Predict using built-in SVM
[y_pred_builtin, score_builtin] = predict(SVMModel, X);

% Calculate performance metrics
confMat = confusionmat(y, y_pred_builtin);
TP = confMat(2,2);
TN = confMat(1,1);
FP = confMat(1,2);
FN = confMat(2,1);

accuracy = (TP + TN) / sum(confMat(:));
sensitivity = TP / (TP + FN);
specificity = TN / (TN + FP);
precision = TP / (TP + FP);
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);
[~, ~, ~, auc_builtin] = perfcurve(y, score_builtin(:,2), 1);

% Display results
fprintf('Built-in SVM Results:\n');
```

```
Built-in SVM Results:
```

```
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
```

```
Accuracy: 100.00%
```

```
fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity * 100);
```

```
Sensitivity (Recall): 100.00%
```

```
fprintf('Specificity: %.2f%%\n', specificity * 100);
```

```
Specificity: 100.00%
```

```
fprintf('Precision: %.2f%%\n', precision * 100);
```

```
Precision: 100.00%
```

```
fprintf('F1 Score: %.2f%%\n', f1_score * 100);
```

```
F1 Score: 100.00%
```

```
fprintf('AUC: %.2f\n', auc_builtin);
```

```
AUC: 1.00
```

```
clc;
clear all;
close all;

% Load the breast cancer dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
%X = data{:, 3:end}; % Assuming features start from the 3rd column
y = data.diagnosis; % Diagnosis labels

% Convert target labels to numerical values if necessary
y = double(categorical(y)); % Convert categorical to numeric (if needed)
y(y == 1) = -1;
y(y == 2) = 1;
% Data dimensions
m = size(X, 1); % Number of data points
n = size(X, 2); % Number of features

% Data mapping to high dimensions
dim = 900; % Target dimensionality
rng(2545); % Random number seed
R = 2 * randn(n, dim); % Random projection matrix (size n x dim)
```

```
% Map data to high-dimensional space
M_data = X * R; % Mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data

% Prepare for SVM classification using CVX
n_mapped = size(data_RKS, 2); % Number of features after mapping
e = ones(m, 1); % m-tuple one-vector
c = 2; % Penalty for error

% CVX optimization for SVM
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predictions on training data
r = sum(y == z); % Count correctly classified points
Acc = (r / m) * 100; % Calculate accuracy

fprintf('Accuracy: %.2f%%\n', Acc);
```

```
Accuracy: 100.00%
```

Built-in SVM with train-test split

```
% Load and preprocess data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, {'concavePointsWorst',
'areaWorst','perimeterWorst','concavePointsMean','radiusWorst' }};
%X = data{:, 3:30}; % Adjust column indices as necessary
y = data.diagnosis; % Assuming 'diagnosis' is the target variable

% Convert labels: 0 to -1 and 1 to 1
y(y == 0) = -1;
y(y == 1) = 1;

% Standardize features
X = zscore(X);

% 80-20 Train-Test Split
```

```
cv = cvpartition(y, 'HoldOut', 0.2);
X_train = X(cv.training, :);
y_train = y(cv.training);
X_test = X(cv.test, :);
y_test = y(cv.test);

% Set the BoxConstraint parameter
C = 5;

% 5-Fold Cross-Validation on the Training Data
cv_kfold = cvpartition(y_train, 'KFold', 5);
accuracy = zeros(cv_kfold.NumTestSets, 1);
sensitivity = zeros(cv_kfold.NumTestSets, 1);
specificity = zeros(cv_kfold.NumTestSets, 1);
precision = zeros(cv_kfold.NumTestSets, 1);
f1 = zeros(cv_kfold.NumTestSets, 1);
auc = zeros(cv_kfold.NumTestSets, 1);

for i = 1:cv_kfold.NumTestSets
    % Get training and validation sets for the current fold
    X_train_fold = X_train(cv_kfold.training(i), :);
    y_train_fold = y_train(cv_kfold.training(i));
    X_val_fold = X_train(cv_kfold.test(i), :);
    y_val_fold = y_train(cv_kfold.test(i));

    % Train the SVM model
    SVMModel = fitcsvm(X_train_fold, y_train_fold, 'KernelFunction',
'linear', 'BoxConstraint', C, 'Standardize', false);

    % Predict on the validation fold
    [y_pred_val, score_val] = predict(SVMModel, X_val_fold);

    % Calculate performance metrics for the validation fold
    confMat = confusionmat(y_val_fold, y_pred_val);
    TP = confMat(2,2);
    TN = confMat(1,1);
    FP = confMat(1,2);
    FN = confMat(2,1);

    accuracy(i) = (TP + TN) / sum(confMat(:));
    sensitivity(i) = TP / (TP + FN);
    specificity(i) = TN / (TN + FP);
    precision(i) = TP / (TP + FP);
    f1(i) = 2 * (precision(i) * sensitivity(i)) / (precision(i) +
sensitivity(i));
    [~, ~, ~, auc(i)] = perfcurve(y_val_fold, score_val(:,2), 1);
end

% Display Cross-Validation Results
fprintf('Cross-Validation Results (5-Fold):\n');
```

```
Cross-Validation Results (5-Fold):
```

```
fprintf('Mean Accuracy: %.2f%%\n', mean(accuracy) * 100);
```

```
Mean Accuracy: 94.96%
```

```
fprintf('Mean Sensitivity (Recall): %.2f%%\n', mean(sensitivity) * 100);
```

```
Mean Sensitivity (Recall): 91.18%
```

```
fprintf('Mean Specificity: %.2f%%\n', mean(specificity) * 100);
```

```
Mean Specificity: 97.20%
```

```
fprintf('Mean Precision: %.2f%%\n', mean(precision) * 100);
```

```
Mean Precision: 95.11%
```

```
fprintf('Mean F1 Score: %.2f%%\n', mean(f1) * 100);
```

```
Mean F1 Score: 93.00%
```

```
fprintf('Mean AUC: %.2f\n\n', mean(auc));
```

```
Mean AUC: 0.99
```

```

% Final Model Training on the entire training set
SVMModel_final = fitcsvm(X_train, y_train, 'KernelFunction', 'linear',
'BoxConstraint', C, 'Standardize', false);

% Predict on the test dataset
[y_pred_test, score_test] = predict(SVMModel_final, X_test);

% Calculate performance metrics on the test dataset
confMat_test = confusionmat(y_test, y_pred_test);
TP_test = confMat_test(2,2);
TN_test = confMat_test(1,1);
FP_test = confMat_test(1,2);
FN_test = confMat_test(2,1);

accuracy_test = (TP_test + TN_test) / sum(confMat_test(:));
sensitivity_test = TP_test / (TP_test + FN_test);
specificity_test = TN_test / (TN_test + FP_test);
precision_test = TP_test / (TP_test + FP_test);
f1_score_test = 2 * (precision_test * sensitivity_test) / (precision_test +
sensitivity_test);
[~, ~, ~, auc_test] = perfcurve(y_test, score_test(:,2), 1);

% Display results on test dataset
fprintf('Test Dataset Results:\n');
```

```
Test Dataset Results:
```

```
fprintf('Accuracy: %.2f%%\n', accuracy_test * 100);
```

```
Accuracy: 94.69%
```

```
fprintf('Sensitivity (Recall): %.2f%%\n', sensitivity_test * 100);
```

```
Sensitivity (Recall): 92.86%
```

```
fprintf('Specificity: %.2f%%\n', specificity_test * 100);
```

```
Specificity: 95.77%
```

```
fprintf('Precision: %.2f%%\n', precision_test * 100);
```

```
Precision: 92.86%
```

```
fprintf('F1 Score: %.2f%%\n', f1_score_test * 100);
```

```
F1 Score: 92.86%
```

```
fprintf('AUC: %.2f\n', auc_test);
```

```
AUC: 0.99
```

Data normalization and RKSA

```
% Load the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, 3:end}; % Assuming the features start from the 3rd column
%y = data.diagnosis;

% Preprocess: Normalize features
X = (X - mean(X)) ./ std(X);

% Define parameters
nd = 100; % number of data points in each class (adjust as necessary)
dim = 2700; % Increase dimension for Random Kitchen Sink

% Generate random projection matrix
rng(2545); % random seed
R = 2 * randn(size(X, 2), dim); % matrix for high dimensional mapping

% Perform Random Kitchen Sink transformation
M_data = X * R; % First level mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data
```

```
% Prepare for SVM classification using CVX
n_mapped = size(data_RKS, 2); % Number of features after mapping
e = ones(m, 1); % m-tuple one-vector
c = 10000;%for SVM
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predictions on training data
r = sum(y == z); % Count correctly classified points
Acc = (r / m) * 100; % Calculate accuracy

fprintf('Accuracy: %.2f%%\n', Acc);
```

```
Accuracy: 37.26%
```

Using PCA and Cross validation

```
% Load and preprocess the data
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, 3:end}; % Assuming the features start from the 3rd column
%y = data.diagnosis;

% Normalize features
X = (X - mean(X)) ./ std(X);

% Apply PCA
[coeff, score, latent] = pca(X);
X_pca = score(:, 1:30); % Retain the first 30 principal components

% Random Kitchen Sink Parameters
dim = 2700; % Dimension for Random Kitchen Sink
rng(2545);
R = 2 * randn(size(X_pca, 2), dim); % Random projection matrix

% Map data to high-dimensional space
M_data = X_pca * R;
data_RKS = [cos(M_data) sin(M_data)];
```

```
% Train SVM with a different kernel
SVMModel = fitcsvm(data_RKS, y, 'KernelFunction', 'gaussian', 'Standardize',
true);

% Cross-Validation
CVSVMModel = crossval(SVMModel);
classLoss = kfoldLoss(CVSVMModel);
fprintf('Cross-Validated Loss: %f\n', classLoss);
```

```
Cross-Validated Loss: 0.372583
```

```
% Load your dataset
% Load the dataset without specifying variable names
data = readtable('breast-cancer-wisconsin-data.csv');
% Define the correct variable names based on your previous message
correctVarNames = {'id', 'diagnosis', 'radius mean', 'texture mean',
'perimeter mean', ...
                   'area mean', 'smoothness mean', 'compactness mean',
'concavity mean', ...
                   'concave points mean', 'symmetry mean', 'fractal
dimension mean', ...
                   'radius se', 'texture se', 'perimeter se', 'area se', ...
                   'smoothness se', 'compactness se', 'concavity se', ...
                   'concave points se', 'symmetry se', 'fractal dimension
se', ...
                   'radius worst', 'texture worst', 'perimeter worst', ...
                   'area worst', 'smoothness worst', 'compactness worst', ...
                   'concavity worst', 'concave points worst', 'symmetry
worst', ...
                   'fractal dimension worst'};

% Assign the correct variable names if needed
data.Properties.VariableNames = correctVarNames;
data.Properties.VariableNames = cellstr(data.Properties.VariableNames);
data.diagnosis = double(strcmp(data.diagnosis, 'M'));

%data = readtable('transformed_data.csv');
%X = data{:, {'concave points worst', 'area worst','perimeter
worst','concave points mean','radius worst' }};
X = data{:, 3:end}; % Extract features (assuming features start from the 3rd
column)
y = data.diagnosis; % Target variable

% Convert target labels to numerical values if necessary
y = double(categorical(y)); % Convert categorical to numeric (if needed)
% Convert y to -1 and 1
y(y == 1) = -1;
y(y == 2) = 1;
```

```
% Data dimensions
m = size(X, 1); % Number of data points
n = size(X, 2); % Number of features

% Data mapping to high dimensions
dim = 150; % Target dimensionality
rng(25); % Random number seed
R = 2 * randn(n, dim); % Random projection matrix (size n x dim)

% Map data to high-dimensional space
M_data = X * R; % Mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data

% Prepare for SVM classification using CVX
n_mapped = size(data_RKS, 2); % Number of features after mapping
e = ones(m, 1); % m-tuple one-vector
c = 5; % Penalty for error

% CVX optimization for SVM
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predictions on training data
r = sum(y == z); % Count correctly classified points
Acc = (r / m) * 100; % Calculate accuracy

fprintf('Accuracy: %.2f%%\n', Acc);
```

```
Accuracy: 96.84%
```

Soft Margin SVM with RKS kernel

```
% Load your dataset
% Load the dataset without specifying variable names
data = readtable('breast-cancer-wisconsin-data.csv');
% Define the correct variable names based on your previous message
correctVarNames = {'id', 'diagnosis', 'radius mean', 'texture mean',
'perimeter mean', ...
                   'area mean', 'smoothness mean', 'compactness mean',
'concavity mean', ...
                   'concave points mean', 'symmetry mean', 'fractal
dimension mean', ...
```

```
                   'radius se', 'texture se', 'perimeter se', 'area se', ...
                   'smoothness se', 'compactness se', 'concavity se', ...
                   'concave points se', 'symmetry se', 'fractal dimension
se', ...
                   'radius worst', 'texture worst', 'perimeter worst', ...
                   'area worst', 'smoothness worst', 'compactness worst', ...
                   'concavity worst', 'concave points worst', 'symmetry
worst', ...
                   'fractal dimension worst'};

% Assign the correct variable names if needed
data.Properties.VariableNames = correctVarNames;
data.Properties.VariableNames = cellstr(data.Properties.VariableNames);
data.diagnosis = double(strcmp(data.diagnosis, '1'));

data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
X = data{:, 3:end}; % Extract features (assuming features start from the 3rd
column)
y = data.diagnosis; % Target variable

% Convert target labels to numerical values if necessary
y = double(categorical(y)); % Convert categorical to numeric (if needed)
% Convert y to -1 and 1
y(y == 1) = -1;
y(y == 2) = 1;

% Data dimensions
m = size(X, 1); % Number of data points
n = size(X, 2); % Number of features

% Data mapping to high dimensions
dim = 150; % Target dimensionality
rng(2545); % Random number seed
R = 2 * randn(n, dim); % Random projection matrix (size n x dim)

% Map data to high-dimensional space
M_data = X * R; % Mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data

% Prepare for SVM classification using CVX
n_mapped = size(data_RKS, 2); % Number of features after mapping
e = ones(m, 1); % m-tuple one-vector
c = 10; % Penalty for error

% CVX optimization for SVM
```

```
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z_test = sign(data_RKS * w - g); % Predictions on training data
% Compute performance metrics
    TP = sum((z_test == 1) & (y == 1)); % True Positives
    TN = sum((z_test == -1) & (y == -1)); % True Negatives
    FP = sum((z_test == 1) & (y == -1)); % False Positives
    FN = sum((z_test == -1) & (y == 1)); % False Negatives

    % Accuracy
    acc = (TP + TN) / length(y);

    % Precision, Recall, F1-Score
    sensitivity = TP / (TP + FN);
    specificity = TN / (TN + FP);
    precision = TP / (TP + FP);
    %precision = TP / (TP + FP);
    recall = TP / (TP + FN);
    f1_score = 2 * (precision * recall) / (precision + recall);
    fprintf('Accuracy: %.2f%%\n', acc);
```

```
Accuracy: 1.00%
```

```
    fprintf('Speciificity: %.2f%%\n', specificity);
```

```
Speciificity: 1.00%
```

```
    fprintf('Sensitivity: %.2f%%\n', sensitivity);
```

```
Sensitivity: 0.99%
```

```
    fprintf('F1-score: %.2f%%\n', f1_score);
```

```
F1-score: 1.00%
```

SVM-RKS with test train split

```
    % Load your dataset
data = readtable('breast-cancer-wisconsin-data.csv');

% Define the correct variable names
```

```
correctVarNames = {'id', 'diagnosis', 'radius mean', 'texture mean',
'perimeter mean', ...
                   'area mean', 'smoothness mean', 'compactness mean',
'concavity mean', ...
                   'concave points mean', 'symmetry mean', 'fractal
dimension mean', ...
                   'radius se', 'texture se', 'perimeter se', 'area se', ...
                   'smoothness se', 'compactness se', 'concavity se', ...
                   'concave points se', 'symmetry se', 'fractal dimension
se', ...
                   'radius worst', 'texture worst', 'perimeter worst', ...
                   'area worst', 'smoothness worst', 'compactness worst', ...
                   'concavity worst', 'concave points worst', 'symmetry
worst', ...
                   'fractal dimension worst'};

% Assign the correct variable names if needed
data.Properties.VariableNames = correctVarNames;

% Convert diagnosis to numeric (1 for malignant, -1 for benign)
data.diagnosis = double(strcmp(data.diagnosis, '1'));
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
y = data.diagnosis; % Target variable
X = data{:, 3:end}; % Extract features (assuming features start from the 3rd
column)
%X = (X - mean(X)) ./ std(X);
% Split data into 80% training and 20% testing
cv = cvpartition(size(X, 1), 'HoldOut', 0.2);
idx = cv.test;

% Separate to training and testing data
X_train = X(~idx, :);
y_train = y(~idx, :);
X_test = X(idx, :);
y_test = y(idx, :);

% Initialize cross-validation parameters
k = 2; % Number of folds
indices = crossvalind('Kfold', y_train, k);
rng(2545); % Random number seed
% Variables to store performance metrics
accuracy = zeros(k, 1);
sensitivity = zeros(k, 1);
specificity = zeros(k, 1);
f1_scores = zeros(k, 1);
```

```
for i = 1:k
    test_idx = (indices == i); % Index for test fold
    train_idx = ~test_idx; % Index for train fold

    % Extract train and test sets for this fold
    X_train_cv = X_train(train_idx, :);
    y_train_cv = y_train(train_idx, :);
    X_test_cv = X_train(test_idx, :);
    y_test_cv = y_train(test_idx, :);

    % Data mapping to high dimensions
    dim = 150; % Target dimensionality
    R = 2 * randn(size(X_train_cv, 2), dim); % Random projection matrix
(size n x dim)

    % Map data to high-dimensional space
    M_data_train = X_train_cv * R; % Mapped training data
    M_data_test = X_test_cv * R; % Mapped testing data
    data_RKS_train = [cos(M_data_train), sin(M_data_train)]; % RKS mapped
training data
    data_RKS_test = [cos(M_data_test), sin(M_data_test)]; % RKS mapped
testing data

    % Prepare for SVM classification using CVX
    n_mapped = size(data_RKS_train, 2); % Number of features after mapping
    e = ones(size(data_RKS_train, 1), 1); % m-tuple one-vector
    c = 5; % Penalty for error

    % CVX optimization for SVM
    cvx_begin quiet
        variables w(n_mapped) g Psi(size(data_RKS_train, 1))
        minimize ((0.5 * w' * w) + (c * sum(Psi)))
        subject to
            y_train_cv .* (data_RKS_train * w - g * e) + Psi - e >= 0;
            Psi >= 0;
    cvx_end

    % Classification on the test fold
    z_test_cv = sign(data_RKS_train * w - g); % Predictions on training data

    % Compute performance metrics
    TP = sum((z_test_cv == 1) & (y_train_cv == 1)); % True Positives
    TN = sum((z_test_cv == -1) & (y_train_cv == -1)); % True Negatives
    FP = sum((z_test_cv == 1) & (y_train_cv == -1)); % False Positives
    FN = sum((z_test_cv == -1) & (y_train_cv == 1)); % False Negatives

    % Store metrics
    accuracy(i) = (TP + TN) / length(y_train_cv);
    sensitivity(i) = TP / (TP + FN);
    specificity(i) = TN / (TN + FP);
    precision = TP / (TP + FP);
    recall = TP / (TP + FN);
    f1_scores(i) = 2 * (precision * recall) / (precision + recall);
end

% Average metrics across folds
mean_accuracy = mean(accuracy);
mean_sensitivity = mean(sensitivity);
mean_specificity = mean(specificity);
mean_f1_score = mean(f1_scores);

fprintf('Average Accuracy: %.2f%%\n', mean_accuracy * 100);
```

```
Average Accuracy: 37.72%
```

```
fprintf('Average Sensitivity: %.2f%%\n', mean_sensitivity * 100);
```

```
Average Sensitivity: 100.00%
```

```
fprintf('Average Specificity: %.2f%%\n', mean_specificity * 100);
```

```
Average Specificity: NaN%
```

```
fprintf('Average F1-score: %.2f%%\n', mean_f1_score * 100);
```

```
Average F1-score: 100.00%
```

```

% Test on the separate test dataset
M_data_test_final = X_test * R; % Mapped test data
data_RKS_test_final = [cos(M_data_test_final), sin(M_data_test_final)]; %
RKS mapped test data

z_test_final = sign(data_RKS_test_final * w - g); % Predictions on test data

% Compute test performance metrics
TP_test = sum((z_test_final == 1) & (y_test == 1)); % True Positives
TN_test = sum((z_test_final == -1) & (y_test == -1)); % True Negatives
FP_test = sum((z_test_final == 1) & (y_test == -1)); % False Positives
FN_test = sum((z_test_final == -1) & (y_test == 1)); % False Negatives

% Test accuracy and other metrics
test_accuracy = (TP_test + TN_test) / length(y_test);
test_sensitivity = TP_test / (TP_test + FN_test);
test_specificity = TN_test / (TN_test + FP_test);
test_precision = TP_test / (TP_test + FP_test);
test_recall = TP_test / (TP_test + FN_test);
test_f1_score = 2 * (test_precision * test_recall) / (test_precision +
test_recall);

fprintf('Test Accuracy: %.2f%%\n', test_accuracy * 100);
```

```
Test Accuracy: 35.40%
```

```
fprintf('Test Sensitivity: %.2f%%\n', test_sensitivity * 100);
```

```
Test Sensitivity: 100.00%
```

```
fprintf('Test Specificity: %.2f%%\n', test_specificity * 100);
```

```
Test Specificity: NaN%
```

```
fprintf('Test F1-score: %.2f%%\n', test_f1_score * 100);
```

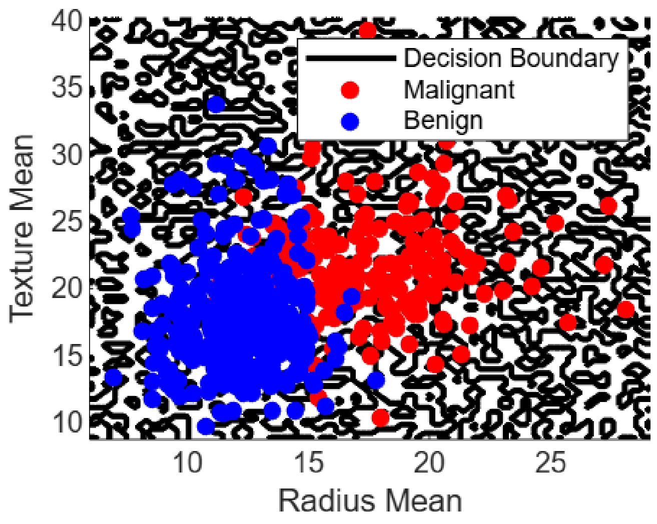
```
Test F1-score: 100.00%
```

```
% Use only the first two features (e.g., 'radius_mean' and 'texture_mean')
for visualization
X = data{:, {'radius mean', 'texture mean'}};

% Data dimensions
m = size(X, 1); % Number of data points
n = size(X, 2); % Number of features (2 in this case)

% Data mapping to high dimensions using Random Kitchen Sink (RKS)
dim = 900; % Target high-dimensional space dimensionality
rng(2545); % Random number seed
R = 2 * randn(n, dim); % Random projection matrix (size n x dim)

% Map data to high-dimensional space
M_data = X * R; % First step mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS feature mapping

% SVM optimization using CVX
n_mapped = size(data_RKS, 2); % Number of features after RKS mapping
e = ones(m, 1); % Vector of ones for bias term
c = 1; % Penalty parameter for SVM

% CVX for SVM optimization
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + c * sum(Psi))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predicted labels
r = sum(y == z); % Correctly classified samples
Acc = (r / m) * 100; % Accuracy
```

```
fprintf('Accuracy: %.2f%%\n', Acc);
```

```
Accuracy: 99.82%
```

```

% Visualize the decision boundary
% Create a grid over the first two features for plotting the boundary
[x1Grid, x2Grid] = meshgrid(linspace(min(X(:,1))-1, max(X(:,1))+1, 100), ...
                            linspace(min(X(:,2))-1, max(X(:,2))+1, 100));
gridPoints = [x1Grid(:), x2Grid(:)]; % Grid points

% Apply the same RKS transformation to grid points
M_grid = gridPoints * R; % Project grid points to the higher-dimensional
space
grid_RKS = [cos(M_grid) sin(M_grid)]; % RKS transformation

% Classify each grid point using the learned SVM model
zGrid = sign(grid_RKS * w - g); % Classify grid points
zGrid = reshape(zGrid, size(x1Grid)); % Reshape into the shape of the grid

% Plot the decision boundary
figure;
hold on;
contour(x1Grid, x2Grid, zGrid, [0 0], 'k', 'LineWidth', 2); % Plot decision
boundary
scatter(X(y==1,1), X(y==1,2), 'r', 'filled'); % Malignant (class +1)
scatter(X(y==-1,1), X(y==-1,2), 'b', 'filled'); % Benign (class -1)
%title('SVM Decision Boundary with RKS for Breast Cancer Dataset');
xlabel('Radius Mean');
ylabel('Texture Mean');
legend('Decision Boundary', 'Malignant', 'Benign');
hold off;
```



Train over entire feature set

```
X = data{:, 3:end}; % Use all features for training

% Data dimensions
m = size(X, 1); % Number of data points
n = size(X, 2); % Number of features (full dataset)

% Data mapping to high dimensions using Random Kitchen Sink (RKS)
dim = 10; % Target high-dimensional space dimensionality
rng(2545); % Random number seed
R = 2 * randn(n, dim); % Random projection matrix (size n x dim)

% Map full data to high-dimensional space
M_data = X * R; % First step mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS feature mapping

% SVM optimization using CVX
n_mapped = size(data_RKS, 2); % Number of features after RKS mapping
e = ones(m, 1); % Vector of ones for bias term
c = 1; % Penalty parameter for SVM

% CVX for SVM optimization
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + c * sum(Psi))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predicted labels
r = sum(y == z); % Correctly classified samples
Acc = (r / m) * 100; % Accuracy
fprintf('Accuracy: %.2f%%\n', Acc);
```

```
Accuracy: 37.26%
```

```

% --- Visualization of the decision boundary using only two selected
features ---
% Select two features for visualization (e.g., 'radius_mean' and
'texture_mean')
X_visualize = data{:, {'radiusMean', 'textureMean'}};

% Create a grid over the first two features for plotting the boundary
[x1Grid, x2Grid] = meshgrid(linspace(min(X_visualize(:,1))-1,
max(X_visualize(:,1))+1, 100), ...
                            linspace(min(X_visualize(:,2))-1,
max(X_visualize(:,2))+1, 100));
```

```
gridPoints = [x1Grid(:), x2Grid(:)]; % Grid points

% Map the grid points to high-dimensional space (only the selected two
features)
R_visualize = R(1:2, :); % Extract the random projection matrix for the
first two features
M_grid = gridPoints * R_visualize; % Project grid points to high-dimensional
space
grid_RKS = [cos(M_grid) sin(M_grid)]; % Apply RKS transformation

% Classify each grid point using the learned SVM model
zGrid = sign(grid_RKS * w - g); % Classify grid points
zGrid = reshape(zGrid, size(x1Grid)); % Reshape into the shape of the grid

% Plot the decision boundary
figure;
hold on;
contour(x1Grid, x2Grid, zGrid, [0 0], 'k', 'LineWidth', 2); % Plot decision
boundary
```

```
Warning: Contour not rendered for constant ZData
```

```
scatter(X_visualize(y==1,1), X_visualize(y==1,2), 'r', 'filled'); %
Malignant (class +1)
scatter(X_visualize(y==-1,1), X_visualize(y==-1,2), 'b', 'filled'); % Benign
(class -1)
xlabel('Radius Mean');
ylabel('Texture Mean');
legend('Decision Boundary', 'Malignant', 'Benign');
hold off;
```

SVM with RKS kernel with 5-fold cross validation and other performance metrics

```
% Use the entire feature set for training (all features starting from column
3)
X = data{:, 3:end}; % Use all features for training

% Data dimensions
m = size(X, 1); % Number of data points
n = size(X, 2); % Number of features (full dataset)

% --- Cross-validation parameters ---
k = 5; % Number of folds
cv = cvpartition(m, 'KFold', k); % Create a cross-validation partition

% Initialize arrays to store performance metrics for each fold
```

```
acc = zeros(k, 1); % Accuracy
precision = zeros(k, 1);
recall = zeros(k, 1);
f1_score = zeros(k, 1);

for fold = 1:k
    % Training and test indices for this fold
    trainIdx = training(cv, fold); % Indices for training data
    testIdx = test(cv, fold); % Indices for test data

    X_train = X(trainIdx, :); % Training data
    y_train = y(trainIdx); % Training labels

    X_test = X(testIdx, :); % Test data
    y_test = y(testIdx); % Test labels

    % Data mapping to high dimensions using Random Kitchen Sink (RKS)
    dim = 1500; % Target high-dimensional space dimensionality
    rng(2545); % Random number seed
    R = 2 * randn(n, dim); % Random projection matrix (size n x dim)

    % Map full data to high-dimensional space
    M_data_train = X_train * R; % First step mapped data for training
    M_data_test = X_test * R;   % Mapped data for test
    data_RKS_train = [cos(M_data_train) sin(M_data_train)]; % RKS feature
mapping for training
    data_RKS_test = [cos(M_data_test) sin(M_data_test)];    % RKS feature
mapping for test

    % SVM optimization using CVX
    n_mapped = size(data_RKS_train, 2); % Number of features after RKS
mapping
    e = ones(sum(trainIdx), 1); % Vector of ones for bias term
    c = 5; % Penalty parameter for SVM

    % CVX for SVM optimization
    cvx_begin quiet
        variables w(n_mapped) g Psi(sum(trainIdx))
        minimize ((0.5 * w' * w) + c * sum(Psi))
        subject to
            y_train .* (data_RKS_train * w - g * e) + Psi - e >= 0;
            Psi >= 0;
    cvx_end

    % Classification on test data
    z_test = sign(data_RKS_test * w - g); % Predicted labels for test set

    % Compute performance metrics
    TP = sum((z_test == 1) & (y_test == 1)); % True Positives
    TN = sum((z_test == -1) & (y_test == -1)); % True Negatives
    FP = sum((z_test == 1) & (y_test == -1)); % False Positives
    FN = sum((z_test == -1) & (y_test == 1)); % False Negatives

    % Accuracy
    acc(fold) = (TP + TN) / length(y_test);

    % Precision, Recall, F1-Score
    precision(fold) = TP / (TP + FP);
    recall(fold) = TP / (TP + FN);
    f1_score(fold) = 2 * (precision(fold) * recall(fold)) / (precision(fold)
+ recall(fold));
end

% Calculate and print the mean performance metrics across all folds
meanAcc = mean(acc) * 100;
meanPrecision = mean(precision) * 100;
meanRecall = mean(recall) * 100;
meanF1Score = mean(f1_score) * 100;

fprintf('Mean Accuracy across %d-fold Cross-Validation: %.2f%%\n', k,
meanAcc);
```

```
Mean Accuracy across 5-fold Cross-Validation: 54.83%
```

```
fprintf('Mean Precision across %d-fold Cross-Validation: %.2f%%\n', k,
meanPrecision);
```

```
Mean Precision across 5-fold Cross-Validation: 37.56%
```

```
fprintf('Mean Recall across %d-fold Cross-Validation: %.2f%%\n', k,
meanRecall);
```

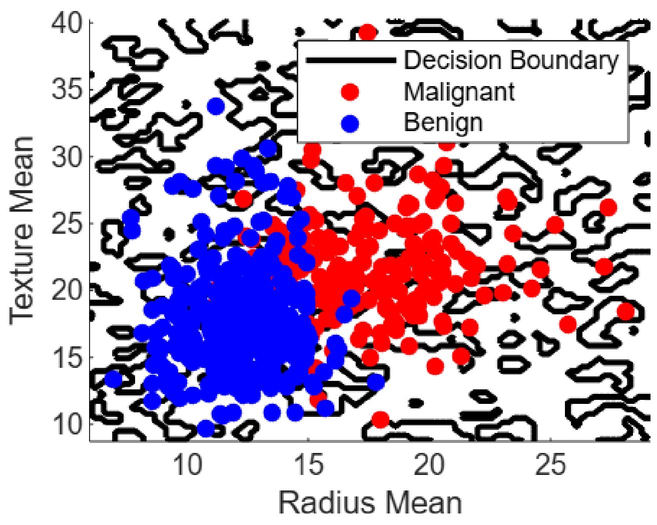
```
Mean Recall across 5-fold Cross-Validation: 29.70%
```

```
fprintf('Mean F1-Score across %d-fold Cross-Validation: %.2f%%\n', k,
meanF1Score);
```

```
Mean F1-Score across 5-fold Cross-Validation: 32.27%
```

```

% --- Visualization of the decision boundary using only two selected
features ---
% Select two features for visualization (e.g., 'radius_mean' and
'texture_mean')
X_visualize = data{:, {'radius mean', 'texture mean'}};

% Create a grid over the first two features for plotting the boundary
[x1Grid, x2Grid] = meshgrid(linspace(min(X_visualize(:,1))-1,
max(X_visualize(:,1))+1, 100), ...
                            linspace(min(X_visualize(:,2))-1,
max(X_visualize(:,2))+1, 100));
gridPoints = [x1Grid(:), x2Grid(:)]; % Grid points

```

```
% Map the grid points to high-dimensional space (only the selected two
features)
R_visualize = R(1:2, :); % Extract the random projection matrix for the
first two features
M_grid = gridPoints * R_visualize; % Project grid points to high-dimensional
space
grid_RKS = [cos(M_grid) sin(M_grid)]; % Apply RKS transformation

% Classify each grid point using the learned SVM model
zGrid = sign(grid_RKS * w - g); % Classify grid points
zGrid = reshape(zGrid, size(x1Grid)); % Reshape into the shape of the grid

% Plot the decision boundary
figure;
hold on;
contour(x1Grid, x2Grid, zGrid, [0 0], 'k', 'LineWidth', 2); % Plot decision
boundary
scatter(X_visualize(y==1,1), X_visualize(y==1,2), 'r', 'filled'); %
Malignant (class +1)
scatter(X_visualize(y==-1,1), X_visualize(y==-1,2), 'b', 'filled'); % Benign
(class -1)
%title(sprintf('SVM Decision Boundary with RKS (Cross-Validated)'));
xlabel('Radius Mean');
ylabel('Texture Mean');
legend('Decision Boundary', 'Malignant', 'Benign');
hold off;
```



Implementation of Decision Tree Algorithm for Classification

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Extract features and target variable
X = data{:, 3:end}; % Assuming the features start from the 3rd column
%y = double(data.diagnosis == '1'); % Convert 'M' (malignant) to 1 and 'B'
(benign) to 0
y=data.diagnosis;
% Define parameters
[n_samples, n_features] = size(X);
max_depth = 3;  % Example constraint for maximum depth
min_samples_leaf = 3;  % Minimum samples in leaf

% Define Lagrangian multipliers
lambda = zeros(n_samples, 1); % Placeholder for Lagrange multipliers

% Initialize CVX
cvx_begin quiet
    variable beeta(n_features); % Coefficients for the decision tree
    variable predictions(n_samples); % Predictions for the samples

    % Define the objective function (Negative Log-Likelihood)
    loss = -sum(y .* log(predictions) + (1 - y) .* log(1 - predictions));

    % Set the objective to minimize the Lagrangian
    minimize(loss)

    % Define constraints
    subject to
        predictions >= 0;
        predictions <= 1;
        sum(beeta) <= max_depth; % Example of constraint for max depth
        sum(predictions) >= min_samples_leaf; % Minimum samples in leaf
cvx_end

% Display the results
disp('beeta Coefficients:');
```

```
beeta Coefficients:
```

```
disp(beeta);
```

```
     3
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
```

```
%disp('Predictions:');
%disp(predictions);
% Convert predictions to binary output (0 or 1) based on a threshold
threshold = 0.5;
predicted_classes = predictions >= threshold;
predicted_classes = double(predicted_classes);
% Calculate confusion matrix
confusionMat = confusionmat(y, predicted_classes);

% Extract True Positives, True Negatives, False Positives, False Negatives
TP = confusionMat(2, 2); % True Positive
TN = confusionMat(1, 1); % True Negative
FP = confusionMat(1, 2); % False Positive
FN = confusionMat(2, 1); % False Negative

% Calculate metrics
accuracy = (TP + TN) / sum(confusionMat(:));
sensitivity = TP / (TP + FN); % Recall
specificity = TN / (TN + FP);
precision = TP / (TP + FP);
F1 = 2 * (precision * sensitivity) / (precision + sensitivity);

% Display metrics
fprintf('Accuracy: %.2f\n', accuracy);
```

```
Accuracy: 1.00
```

```
fprintf('Sensitivity (Recall): %.2f\n', sensitivity);
```

```
Sensitivity (Recall): 1.00
```

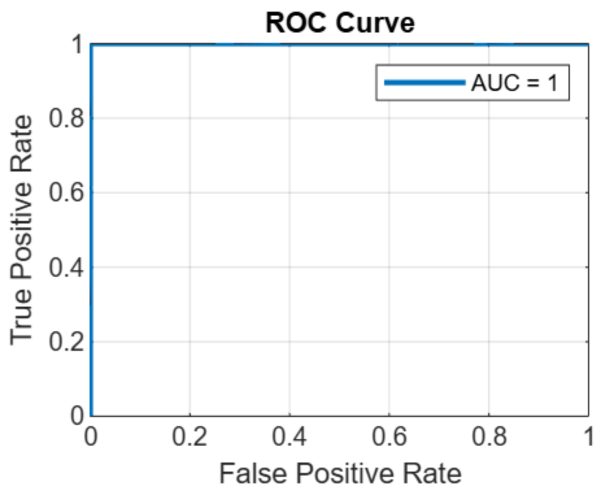
```
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 1.00
```

```
fprintf('F1 Score: %.2f\n', F1);
```

```
F1 Score: 1.00
```

```
% Compute the ROC curve
[x, y, ~, AUC] = perfcurve(y, predictions, 1);

% Plot ROC curve
figure;
plot(x, y, 'LineWidth', 2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
legend(['AUC = ', num2str(AUC)]);
grid on;
```



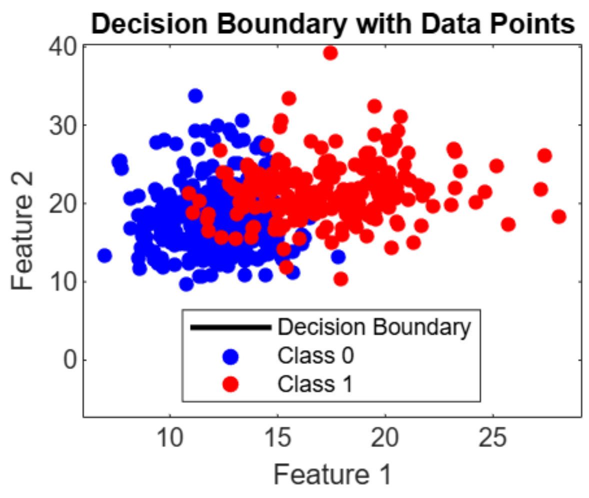
```
% Load your dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```


% Extract features and target variable
X = data{:, 3:end}; % Assuming features start from the 3rd column
y = data.diagnosis; % Ensure y has the correct number of rows
```

```
% Ensure y is a column vector
if iscolumn(y) == false
    y = y(:);  % Convert to column vector if it's not
end

% Ensure the dimensions match
if length(y) ~= size(X, 1)
    error('y must have the same number of rows as X');
end

% Make predictions
%predicted_probs = 1 ./ (1 + exp(-X * beeta)); % Logistic regression
predictions
%predicted_classes = predicted_probs > 0.5; % Classify based on threshold
%predicted_classes=double(predicted_classes);
% Create confusion matrix
confusion_mat = confusionmat(y, predicted_classes);

% Calculate performance metrics
TP = confusion_mat(2,2); % True Positives
TN = confusion_mat(1,1); % True Negatives
FP = confusion_mat(1,2); % False Positives
FN = confusion_mat(2,1); % False Negatives

accuracy = (TP + TN) / sum(confusion_mat(:));
sensitivity = TP / (TP + FN); % Recall
specificity = TN / (TN + FP);
precision = TP / (TP + FP);
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

% Prepare to plot decision boundary using the first two features
figure;
x1_range = linspace(min(X(:,1)), max(X(:,1)), 100);
x2_range = linspace(min(X(:,2)), max(X(:,2)), 100);
[x1_grid, x2_grid] = meshgrid(x1_range, x2_range);

% Create grid data with only the first two features
grid_data = [x1_grid(:), x2_grid(:)];

% Ensure beeta is a column vector for the first two features
beeta = beeta(1:2);  % Use only the coefficients for the first two features

% Calculate predicted probabilities for the grid
predicted_grid_probs = 1 ./ (1 + exp(-grid_data * beeta));
predicted_grid_probs = reshape(predicted_grid_probs, size(x1_grid));

% Plot decision boundary
contour(x1_grid, x2_grid, predicted_grid_probs, [0.5 0.5], 'LineColor', 'k',
'LineWidth', 2);
```

```
hold on;

% Scatter plot of data points, color-coded by class (0 and 1)
scatter(X(y==0,1), X(y==0,2), 30, 'blue', 'filled'); % Class 0 points
scatter(X(y==1,1), X(y==1,2), 30, 'red', 'filled');  % Class 1 points
% Set axes limits
xlim([min(X(:,1)) - 1, max(X(:,1)) + 1]); % Set limits for x-axis
ylim([min(X(:,2)) - 17, max(X(:,2)) + 1]); % Set limits for y-axis
% Add labels and title
title('Decision Boundary with Data Points');
xlabel('Feature 1');
ylabel('Feature 2');
legend({'Decision Boundary', 'Class 0', 'Class 1'}, 'Location', 'best');
hold off;
```



```
% Display performance metrics
fprintf('Accuracy: %.2f\n', accuracy);
```

```
Accuracy: 1.00
```

```
fprintf('Sensitivity (Recall): %.2f\n', sensitivity);
```

```
Sensitivity (Recall): 1.00
```

```
fprintf('Specificity: %.2f\n', specificity);
```

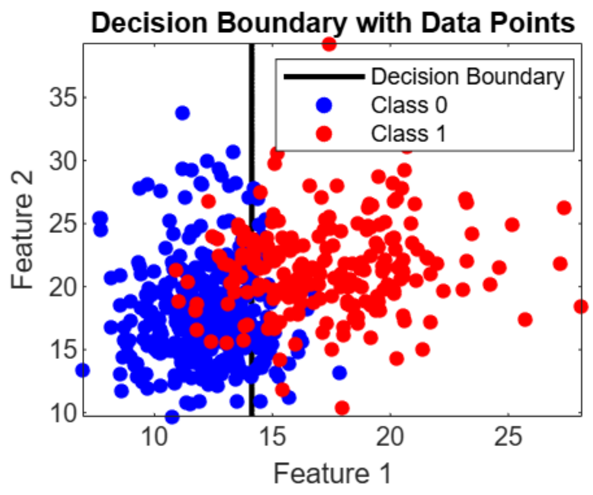
```
Specificity: 1.00
```

```
fprintf('F1 Score: %.2f\n', f1_score);
```

```
F1 Score: 1.00
```

```
% Ensure features are normalized (if normalization was used during training)
mean_X = mean(X);
std_X = std(X);

% Scale the grid data if the original data was scaled
```

```
grid_data_scaled = (grid_data - mean_X(1:2)) ./ std_X(1:2);

% Use only the first two coefficients from beeta for plotting
beta_plot = beeta(1:2);

% Calculate predicted probabilities for the grid (using scaled data)
predicted_grid_probs = 1 ./ (1 + exp(-grid_data_scaled * beta_plot));

% Reshape the predicted probabilities to fit the grid
predicted_grid_probs = reshape(predicted_grid_probs, size(x1_grid));

% Plot decision boundary
figure;
contour(x1_grid, x2_grid, predicted_grid_probs, [0.5 0.5], 'LineColor', 'k',
'LineWidth', 2);
hold on;

% Scatter plot of data points, color-coded by class (0 and 1)
scatter(X(y==0,1), X(y==0,2), 30, 'blue', 'filled'); % Class 0 points
scatter(X(y==1,1), X(y==1,2), 30, 'red', 'filled');  % Class 1 points

% Add labels and title
title('Decision Boundary with Data Points');
xlabel('Feature 1');
ylabel('Feature 2');
legend({'Decision Boundary', 'Class 0', 'Class 1'}, 'Location', 'best');
hold off;
```



Updated with L2 regularization

```
% Load your dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Extract features and target variable
X = data{:, 3:end}; % Assuming the features start from the 3rd column
y = data.diagnosis;

% Standardize features (mean normalization and variance scaling)
mean_X = mean(X);
std_X = std(X);
X_scaled = (X - mean_X) ./ std_X;

% Define L2 regularization parameter (lambda)
lambda = 0.1; % Regularization strength (tune this)

% Setup CVX problem with Lagrangian formulation and L2 regularization
[m, n] = size(X_scaled);
cvx_begin quiet
    variables beeta(n) b(1)
    % Objective: Minimize negative log-likelihood (logistic loss) with L2
regularization
    loss = sum(log(1 + exp(-y .* (X_scaled * beeta + b)))); % Logistic loss
    regularization = lambda * sum_square(beeta); % L2 regularization term
    minimize(loss + regularization) % Minimize the combined objective
cvx_end

% Predictions on the same data for testing (you may use test data)
pred_probs = 1 ./ (1 + exp(-(X_scaled * beeta + b))); % Predicted
probabilities

% Convert predicted probabilities to binary class (using 0.5 threshold)
pred_classes = pred_probs >= 0.5;
pred_classes=double(pred_classes);
% Performance Metrics
conf_mat = confusionmat(y, pred_classes);
accuracy = sum(diag(conf_mat)) / sum(conf_mat(:));
specificity = conf_mat(2, 2) / (conf_mat(2, 1) + conf_mat(2, 2));
sensitivity = conf_mat(1, 1) / (conf_mat(1, 1) + conf_mat(1, 2));
f1_score = 2 * (specificity * sensitivity) / (specificity + sensitivity);

% Display performance metrics
fprintf('Accuracy: %.4f\n', accuracy);
```

```
Accuracy: 0.3726
```

```
fprintf('Specificity: %.4f\n', specificity);
```

```
Specificity: 1.0000
```

```
fprintf('Sensitivity: %.4f\n', sensitivity);
```

```
Sensitivity: 0.0000
```

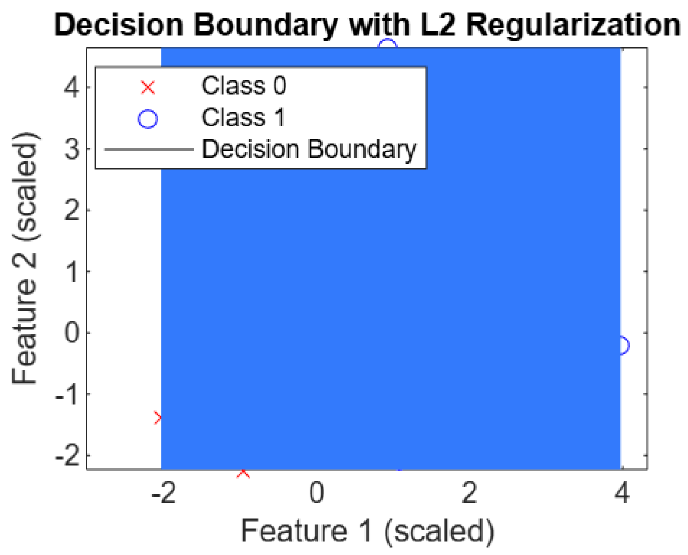
```
fprintf('F1-Score: %.4f\n', f1_score);
```

```
F1-Score: 0.0000
```

```

% Visualize decision boundary (using first two features for 2D plot)
x1_range = linspace(min(X_scaled(:, 1)), max(X_scaled(:, 1)), 100);
x2_range = linspace(min(X_scaled(:, 2)), max(X_scaled(:, 2)), 100);
[x1_grid, x2_grid] = meshgrid(x1_range, x2_range);
grid_data = [x1_grid(:), x2_grid(:)];

% Predict on the grid
grid_data_scaled = (grid_data - mean_X(1:2)) ./ std_X(1:2);
predicted_grid_probs = 1 ./ (1 + exp(-(grid_data_scaled * beeta(1:2) + b)));

% Plot decision boundary
figure;
gscatter(X_scaled(:, 1), X_scaled(:, 2), y, 'rb', 'xo');
hold on;
%contour(x1_grid, x2_grid, predicted_grid_probs, [0.5 0.5], 'LineColor',
'k', 'LineWidth', 2);
contourf(x1_grid, x2_grid, reshape(predicted_grid_probs, size(x1_grid)),
[0.5 0.5], 'LineColor', 'k');
```

```
Warning: Contour not rendered for constant ZData
```

```
title('Decision Boundary with L2 Regularization');
xlabel('Feature 1 (scaled)');
ylabel('Feature 2 (scaled)');
legend('Class 0', 'Class 1', 'Decision Boundary');
hold off;
```



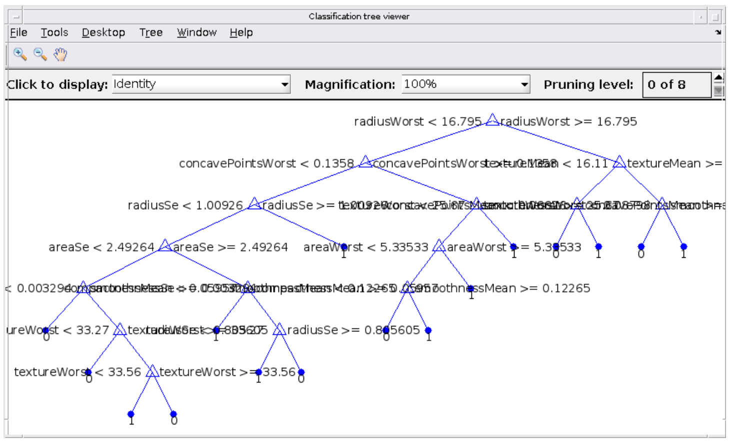
```

```

All the efforts are failed so go for built-in functions and methods.

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```

% Extract features and target variable
X = data{:, 3:end}; % Assuming the features start from the 3rd column
y = data.diagnosis; % Assuming 'diagnosis' is the target variable (binary
classification)

% Fit the Decision Tree model
treeModel = fitctree(X, y, 'PredictorNames',
data.Properties.VariableNames(3:end), 'ResponseName', 'Diagnosis');

% View the Decision Tree
view(treeModel, 'Mode', 'graph');
```



Click to display: Identity
Magnification: 100%
Pruning level: 0 of 8

```

% Make predictions using the trained model
predictions = predict(treeModel, X);

% Calculate performance metrics
confusionMat = confusionmat(y, predictions);
% Calculate performance metrics
TP = confusionMat(2,2); % True Positives
TN = confusionMat(1,1); % True Negatives
FP = confusionMat(1,2); % False Positives
FN = confusionMat(2,1); % False Negatives

accuracy = (TP + TN) / sum(confusionMat(:));
```

```
sensitivity = TP / (TP + FN); % Recall
specificity = TN / (TN + FP);
precision = TP / (TP + FP);
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity);

accuracy = sum(diag(confusionMat)) / sum(confusionMat(:));
disp(['Accuracy: ', num2str(accuracy)]);
```

```
Accuracy: 0.98946
```

```
disp(['Sensitivity: ', num2str(sensitivity)]);
```

```
Sensitivity: 0.99528
```

```
disp(['Specificity: ', num2str(specificity)]);
```

```
Specificity: 0.98599
```

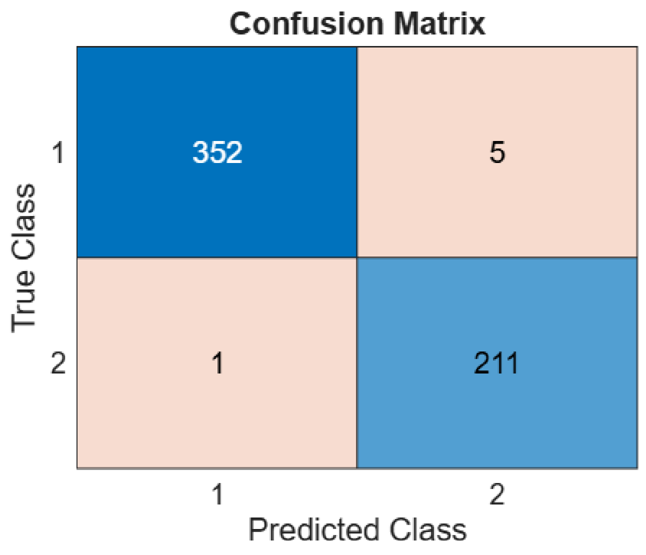
```
disp(['Precision: ', num2str(precision)]);
```

```
Precision: 0.97685
```

```
disp(['F1-score: ', num2str(f1_score)]);
```

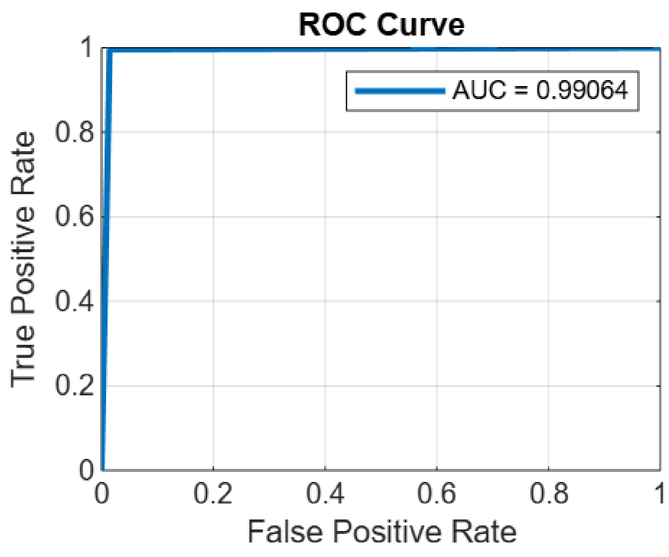
```
F1-score: 0.98598
```

```



% If needed, plot the confusion matrix
figure;
confusionchart(confusionMat);
title('Confusion Matrix');
```



```

% Visualize the decision boundary for the first two features
% Use only the first two features for visualization purposes
X_visual = X(:, 1:2); % First two features
```

```
x1_range = linspace(min(X_visual(:, 1)), max(X_visual(:, 1)), 100);
x2_range = linspace(min(X_visual(:, 2)), max(X_visual(:, 2)), 100);
[x1_grid, x2_grid] = meshgrid(x1_range, x2_range);
grid_data = [x1_grid(:), x2_grid(:)];

% Predict on the grid
grid_predictions = predict(treeModel, [grid_data, zeros(size(grid_data, 1),
size(X, 2) - 2)]); % Fill other features with zeros

% Reshape predictions to match grid
grid_predictions = reshape(grid_predictions, size(x1_grid));
% Compute the ROC curve
[x, y, ~, AUC] = perfcurve(y, predictions, 1);

% Plot ROC curve
figure;
plot(x, y, 'LineWidth', 2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve');
legend(['AUC = ', num2str(AUC)]);
grid on;
```



Decision tree with cross validation

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Extract features and target variable
X = data{:, 3:end}; % Assuming the features start from the 3rd column
y = data.diagnosis; % Assuming 'diagnosis' is the target variable (binary
classification)

% Convert target variable to categorical (important for classification)
y = categorical(y);
rng(1);
% Define the number of folds for cross-validation
kFolds = 10;

% Perform K-fold cross-validation
cv = cvpartition(height(data), 'KFold', kFolds);

% Initialize metrics for each fold
accuracy = zeros(kFolds, 1);
sensitivity = zeros(kFolds, 1);
specificity = zeros(kFolds, 1);
precision = zeros(kFolds, 1);
f1_score = zeros(kFolds, 1);
auc = zeros(kFolds, 1);

for i = 1:kFolds
    % Get the indices for the current fold
    trainIdx = find(cv.training(i)); % Logical index to actual indices
    testIdx = find(cv.test(i));

    % Split data into training and validation sets
    XTrain = X(trainIdx, :);
    yTrain = y(trainIdx);
    XTest = X(testIdx, :);
    yTest = y(testIdx);

    % Train the Decision Tree model on training data
    treeModel = fitctree(XTrain, yTrain, 'PredictorNames',
data.Properties.VariableNames(3:end), 'ResponseName', 'Diagnosis');

    % Make predictions on validation set with scores
    [predictions, scores] = predict(treeModel, XTest);

    % Calculate performance metrics for this fold
    confusionMat = confusionmat(yTest, predictions);
    TP = confusionMat(2,2); % True Positives
    TN = confusionMat(1,1); % True Negatives
    FP = confusionMat(1,2); % False Positives
    FN = confusionMat(2,1); % False Negatives

    accuracy(i) = (TP + TN) / sum(confusionMat(:));
    sensitivity(i) = TP / (TP + FN);
    specificity(i) = TN / (TN + FP);
    precision(i) = TP / (TP + FP);
    f1_score(i) = 2 * (precision(i) * sensitivity(i)) / (precision(i) +
sensitivity(i));

    % Compute the ROC curve and AUC using scores for the positive class
    positiveClassIndex = find(categories(y) == "1"); % Adjust class index
based on target encoding

end

% Calculate average metrics across all folds
avgAccuracy = mean(accuracy);
avgSensitivity = mean(sensitivity);
avgSpecificity = mean(specificity);
avgPrecision = mean(precision);
avgF1_score = mean(f1_score);
avgAUC = mean(auc);

% Display average results
disp(['Average Accuracy: ', num2str(avgAccuracy)]);
```

```
Average Accuracy: 0.92102
```

```
disp(['Average Sensitivity: ', num2str(avgSensitivity)]);
```

```
Average Sensitivity: 0.91103
```

```
disp(['Average Specificity: ', num2str(avgSpecificity)]);
```

```
Average Specificity: 0.92776
```

```
disp(['Average Precision: ', num2str(avgPrecision)]);
```

```
Average Precision: 0.88234
```

```
disp(['Average F1-score: ', num2str(avgF1_score)]);
```

```
Average F1-score: 0.89497
```

```
disp(['Average AUC: ', num2str(avgAUC)]);
```

```
Average AUC: 0
```

```
[x, y, ~, AUC] = perfcurve(yTest, scores(:, positiveClassIndex), '1');
disp(AUC)
```

```
    0.9291
```

Decision Tree with Test-train split

```
% Load the dataset
data = readtable('transformed_data.csv');
```

```
Warning: Column headers from the file were modified to make them valid MATLAB identifiers
before creating variable names for the table. The original column headers are saved in the
VariableDescriptions property.
Set 'VariableNamingRule' to 'preserve' to use the original column headers as table variable names.
```

```
% Extract features and target variable
X = data{:, 3:end}; % Assuming the features start from the 3rd column
%y = data.diagnosis; % Assuming 'diagnosis' is the target variable (binary
classification)

% Convert target variable to categorical (important for classification)
y = categorical(data.diagnosis);

% Step 1: Perform 80-20 train-test split
cv = cvpartition(height(data), 'HoldOut', 0.2); % 20% for testing
trainIdx = training(cv); % Logical index for training set
testIdx = test(cv); % Logical index for testing set

% Split data into training and test sets
XTrain = X(trainIdx, :);
yTrain = y(trainIdx);
XTest = X(testIdx, :);
yTest = y(testIdx);

% Step 2: Define the number of folds for cross-validation
kFolds = 5; % Using 5-fold cross-validation

% Create a cross-validation partition for training data
cvTrain = cvpartition(height(XTrain), 'KFold', kFolds);

% Initialize metrics for each fold
accuracy = zeros(kFolds, 1);
sensitivity = zeros(kFolds, 1);
specificity = zeros(kFolds, 1);
precision = zeros(kFolds, 1);
f1_score = zeros(kFolds, 1);
auc = zeros(kFolds, 1);

% Step 3: Perform cross-validation on training data
for i = 1:kFolds
    % Get the indices for the current fold
    trainIdx = find(cvTrain.training(i)); % Logical index to actual indices
    valIdx = find(cvTrain.test(i));

    % Split data into training and validation sets for current fold
    XFoldTrain = XTrain(trainIdx, :);
    yFoldTrain = yTrain(trainIdx);
    XFoldVal = XTrain(valIdx, :);
    yFoldVal = yTrain(valIdx);
    % Train the Decision Tree model on training data
    treeModel = fitctree(XFoldTrain, yFoldTrain, 'PredictorNames',
data.Properties.VariableNames(3:end), 'ResponseName', 'Diagnosis');

    % Make predictions on validation set
    [predictions, scores] = predict(treeModel, XFoldVal);

    % Calculate performance metrics for this fold
    confusionMat = confusionmat(yFoldVal, predictions);
    TP = confusionMat(2, 2); % True Positives
    TN = confusionMat(1, 1); % True Negatives
    FP = confusionMat(1, 2); % False Positives
    FN = confusionMat(2, 1); % False Negatives

    accuracy(i) = (TP + TN) / sum(confusionMat(:));
    sensitivity(i) = TP / (TP + FN);
    specificity(i) = TN / (TN + FP);
    precision(i) = TP / (TP + FP);
    f1_score(i) = 2 * (precision(i) * sensitivity(i)) / (precision(i) +
sensitivity(i));

    % Compute the ROC curve and AUC using scores for the positive class
    positiveClassIndex = find(categories(y) == "1"); % Adjust class index
based on target encoding
    %[x, y, t, AUC_fold] = perfcurve(yFoldVal, scores(:,
positiveClassIndex), '1');
    %auc(i) = AUC_fold;
end

% Step 4: Calculate average metrics across all folds
avgAccuracy = mean(accuracy);
avgSensitivity = mean(sensitivity);
avgSpecificity = mean(specificity);
avgPrecision = mean(precision);
avgF1_score = mean(f1_score);
%avgAUC = mean(auc);

% Display average results for training performance
disp('Training Performance (Cross-Validation):');
```

```
Training Performance (Cross-Validation):
```

```
disp(['Average Accuracy: ', num2str(avgAccuracy)]);
```

```
Average Accuracy: 0.91878
```

```
disp(['Average Sensitivity: ', num2str(avgSensitivity)]);
```

```
Average Sensitivity: 0.91416
```

```
disp(['Average Specificity: ', num2str(avgSpecificity)]);
```

```
Average Specificity: 0.92126
```

```
disp(['Average Precision: ', num2str(avgPrecision)]);
```

```
Average Precision: 0.87607
```

```
disp(['Average F1-score: ', num2str(avgF1_score)]);
```

```
Average F1-score: 0.89416
```

```
%disp(['Average AUC: ', num2str(avgAUC)]);

% Step 5: Evaluate the model on the test set
% Train the final model on the entire training set
finalModel = fitctree(XTrain, yTrain);

% Make predictions on the test set
[testPredictions, testScores] = predict(finalModel, XTest);

% Calculate performance metrics on the test set
testConfusionMat = confusionmat(yTest, testPredictions);
TP_test = testConfusionMat(2, 2); % True Positives
TN_test = testConfusionMat(1, 1); % True Negatives
FP_test = testConfusionMat(1, 2); % False Positives
FN_test = testConfusionMat(2, 1); % False Negatives

testAccuracy = (TP_test + TN_test) / sum(testConfusionMat(:));
testSensitivity = TP_test / (TP_test + FN_test);
testSpecificity = TN_test / (TN_test + FP_test);
testPrecision = TP_test / (TP_test + FP_test);
testF1_score = 2 * (testPrecision * testSensitivity) / (testPrecision +
testSensitivity);

% Compute the ROC curve and AUC using scores for the positive class on test
set
[x_test, y_test, t_test, testAUC] = perfcurve(yTest, testScores(:,
positiveClassIndex), '1');

% Display performance metrics for the test set
disp('Test Performance:');
```

```
Test Performance:
```

```
disp(['Test Accuracy: ', num2str(testAccuracy)]);
```

```
Test Accuracy: 0.90265
```

```
disp(['Test Sensitivity: ', num2str(testSensitivity)]);
```

```
Test Sensitivity: 0.85714
```

```
disp(['Test Specificity: ', num2str(testSpecificity)]);
```

```
Test Specificity: 0.92308
```

```
disp(['Test Precision: ', num2str(testPrecision)]);
```

```
Test Precision: 0.83333
```

```
disp(['Test F1-score: ', num2str(testF1_score)]);
```

```
Test F1-score: 0.84507
```

```
disp(['Test AUC: ', num2str(testAUC)]);
```

```
Test AUC: 0.86355
```

```
[x, y, ~, AUC] = perfcurve(yTest, scores(:, positiveClassIndex), '1');
```

```
Error using perfcurve>preparedata (line 1375)
The size of scores does not match the size of labels.

Error in perfcurve (line 398)
[scores,cls,weights,ncv] = preparedata(scores,cls,weights);
```

```
disp(AUC)
```

Bibliography

[1] Yuyan Xu, Maoyuan Gong, Yue Wang, Yang Yang, Shu Liu, and Qibing Zeng. Global trends and forecasts of breast cancer incidence and deaths. *Scientific Data*, 10(1):334, 2023. pages 1

[2] I. F. Gareen and C. Gatsonis. Primer on multiple regression models for diagnostic imaging research. *Radiology*, 229:305–10, 2003. pages 2

[3] J. Concato, A. R. Feinstein, and T. R. Holford. The risk of determining risk with multivariable models. *Ann Intern Med*, 118:201–10, 1993. pages 2

[4] W.Nick Street, W. H. Wolberg, and O.L.Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In Raj S. Acharya and Dmitry B. Goldgof, editors, *Biomedical Image Processing and Biomedical Visualization*, volume 1905, pages 861 – 870. International Society for Optics and Photonics, SPIE, 1993. pages 5

[5] Howard John Hamilton, Nick Cercone, and Ning Shan. *RIAC: a rule induction algorithm based on approximate classification*. Citeseer, 1996. pages 5

[6] Kemal Polat and Salih Güneş. Breast cancer diagnosis using least square support vector machine. *Digital signal processing*, 17(4):694–701, 2007. pages 5

[7] Zakia Salod and Yashik Singh. A five-year (2015 to 2019) analysis of studies focused on breast cancer prediction using machine learning: A systematic review and bibliometric analysis. *Journal of Public Health Research*, 9(1):jphr.2020.1772, 2020. PMID: 32642458. pages 5

[8] Mohammed Amine Naji, Sanaa El Filali, Kawtar Aarika, EL Habib Benlahmar, Rachida Ait Abdelouhahid, and Olivier Debauche. Machine learning algorithms for breast cancer prediction and diagnosis. *Procedia Computer Science*, 191:487–492, 2021. pages 6

[9] Nikhilanand Arya and Sriparna Saha. Multi-modal advanced deep learning architectures for breast cancer survival prediction. *Knowledge-Based Systems*, 221:106965, 2021. pages 6

[10] Laila Khairunnahar, Mohammad Abdul Hasib, Razib Hasan Bin Rezanur, Mohammad Rakibul Islam, and Md Kamal Hosain. Classification of malignant

and benign tissue with logistic regression. *Informatics in Medicine Unlocked*, 16:100189, 2019. pages 6

[11] Shankar Thawkar, Satish Sharma, Munish Khanna, and Law kumar Singh. Breast cancer prediction using a hybrid method based on butterfly optimization algorithm and ant lion optimizer. *Computers in Biology and Medicine*, 139:104968, 2021. pages 6

[12] Weang-Kee Ho, Mei-Chee Tai, Joe Dennis, Xiang Shu, Jingmei Li, Peh Joo Ho, Iona Y Millwood, Kuang Lin, Yon-Ho Jee, Su-Hyun Lee, et al. Polygenic risk scores for prediction of breast cancer risk in asian populations. *Genetics in Medicine*, 24(3):586–600, 2022. pages 6

[13] A Saranya and R Subhashini. A systematic review of explainable artificial intelligence models and applications: Recent developments and future trends. *Decision analytics journal*, 7:100230, 2023. pages 6

[14] Malti Bansal, Apoorva Goyal, and Apoorva Choudhary. A comparative analysis of k-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning. *Decision Analytics Journal*, 3:100071, 2022. pages 7

[15] Mikael Eriksson, Stamatia Destounis, Kamila Czene, Andrew Zeiberg, Robert Day, Emily F Conant, Kathy Schilling, and Per Hall. A risk model for digital breast tomosynthesis to predict breast cancer and guide clinical care. *Science Translational Medicine*, 14(644):eabn3971, 2022. pages 7

[16] Y. F. Hernández-Julio, M. J. Prieto-Guevara, W. Nieto-Bernal, et al. Framework for the development of data-driven mamdani-type fuzzy clinical decision support systems. *Diagnostics (Basel)*, 9:52, 2019. pages 8

[17] B. K. Singh. Determining relevant biomarkers for prediction of breast cancer using anthropometric and clinical features: A comparative investigation in machine learning paradigm. *Biocybernet Biomed Engin*, 39:393–409, 2019. pages 8

[18] K. Polat and U. Senturk. A novel ml approach to prediction of breast cancer: Combining of mad normalization, kmc based feature weighting and adaboostm1 classifier. In *Proceedings 2nd Int Symp on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey, 2018. pages 8

[19] S. B. Akben. Determination of the blood, hormone and obesity value ranges that indicate the breast cancer, using data mining based expert system. *IRBM*, 40:355–60, 2019. pages 8

[20] Md Mohaimenul Islam and Tahmina Narin Poly. Machine learning models of breast cancer risk prediction. *BioRxiv*, page 723304, 2019. pages 8

[21] Vinícius Jonathan Silva Araújo, Augusto Junio Guimarães, Paulo Vitor de Campos Souza, Thiago Silva Rezende, and Vanessa Souza Araújo. Using resistin, glucose, age and bmi and pruning fuzzy neural network for the construction of expert systems in the prediction of breast cancer. *Machine Learning and Knowledge Extraction*, 1(1):466–482, 2019. pages 8

[22] Muhammet Fatih Aslan, Yunus Celik, Kadir Sabancı, and Akif Durdu. Breast cancer diagnosis by different machine learning methods using blood analysis data. 2018. pages 8

[23] Ioannis E Livieris. Improving the classification efficiency of an ANN utilizing a new training methodology. In *Informatics*, volume 6, page 1. MDPI, 2018. pages 8

[24] Miguel Patrício, José Pereira, Joana Crisóstomo, Paulo Matafome, Manuel Gomes, Raquel Seiça, and Francisco Caramelo. Using resistin, glucose, age and BMI to predict the presence of breast cancer. *BMC cancer*, 18:1–8, 2018. pages 8

[25] Y. Li and Z. Chen. Performance evaluation of machine learning methods for breast cancer prediction. *Appl Comput Math*, 7:212–6, 2018. pages 8

[26] P. D. Hung, T. D. Hanh, and V. T. Diep. Breast cancer prediction using spark mllib and ml packages. In *Proceedings 5th Int Conf on Bioinformatics Research and Applications (ICBRA 2018)*, Hong Kong. pages 8

[27] M. Abdar and V. Makarenkov. CWV-BANN-SVM ensemble learning classifier for an accurate diagnosis of breast cancer. *Measurement*, 146:557–70, 2019. pages 8

[28] Madeeh Nayer Elgedawy. Prediction of breast cancer using random forest, support vector machines and naïve bayes. *International Journal of Engineering and Computer Science*, 6(1):19884–19889, 2017. pages 8

[29] Dr Vikas Chaurasia and Saurabh Pal. A novel approach for breast cancer detection using data mining techniques. *International journal of innovative research in computer and communication engineering (An ISO 3297: 2007 Certified Organization) Vol*, 2, 2017. pages 8

[30] Hiba Asri, Hajar Mousannif, Hassan Al Moatassime, and Thomas Noel. Using machine learning algorithms for breast cancer risk prediction and diagnosis. *Procedia Computer Science*, 83:1064–1069, 2016. pages 8

[31] Abeer Alzubaidi, Georgina Cosma, David Brown, and A Graham Pockley. Breast cancer diagnosis using a hybrid genetic algorithm for feature selection based on mutual information. In *2016 International Conference on Interactive Technologies and Games (ITAG)*, pages 70–76. IEEE, 2016. pages 8

[32] Md Milon Islam, Hasib Iqbal, Md Rezwanul Haque, and Md Kamrul Hasan. Prediction of breast cancer using support vector machine and K-Nearest neighbors. In *2017 IEEE region 10 humanitarian technology conference (R10-HTC)*, pages 226–229. IEEE, 2017. pages 8

[33] Vikas Chaurasia, Saurabh Pal, and BB Tiwari. Prediction of benign and malignant breast cancer using data mining techniques. *Journal of Algorithms & Computational Technology*, 12(2):119–126, 2018. pages 8

[34] Dana Bazazeh and Raed Shubair. Comparative study of machine learning algorithms for breast cancer detection and diagnosis. In *2016 5th international conference on electronic devices, systems and applications (ICEDSA)*, pages 1–4. IEEE, 2016. pages 8

[35] L Li, J Yu, and JL Chen. A review of china's health care reform after the reform and open policy. *Chinese Health Economics*, 27(2):5–9, 2008. pages 8

[36] Meerja Akhil Jabbar. Breast cancer data classification using ensemble machine learning. *Engineering & Applied Science Research*, 48(1), 2021. pages 8

[37] K.P. Soman, R. LOGANATHAN, and V. AJAY. *Machine Learning with SVM and Other Kernel Methods*. PHI Learning, 2009. pages 29