

Machine Learning Approaches for Automated Land Use and Land Cover Classification

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Sundaravelan S 20BCT0248

Athish M 20BDS0139

Rithik RR 20BDS0201

Under the guidance of

Dr. Jayakumar

School of Computer Science and Engineering,

VIT, Vellore.



May, 2024

DECLARATION

I hereby declare that the thesis entitled "**Machine Learning Approaches for automated Land use and Land Cover Classification**" submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering with specialization in Data Science* to VIT is a record of bonafide work carried out by me under the supervision of Dr.Jayakumar K.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 09.05.24

R.R. R^o P^o K^o
Sundarandam
M. Kiz

Signature of the Candidate

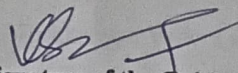
CERTIFICATE

This is to certify that the thesis entitled "**Machine Learning Approaches for Automated Land Use and Land Cover Classification**" submitted by **Rithik RR (20BDS0201)**, **Athish M (20BDS0139)**, **Sundaravelan S (20BCT0248)**, School of Computer Science and Engineering, VIT, for the award of the degree of *Bachelor of Technology in Programme*, is a record of bonafide work carried out by him/her under my supervision during the period, 01. 12. 2023 to 30.04.2024, as per the VIT code of academic and research ethics.

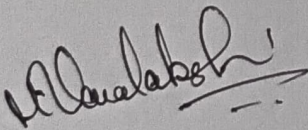
The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion, meets the necessary standards for submission.

Place: Vellore

Date: 09.05.2024

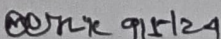

Signature of the Guide

Name: Dr. Jayakumar K
Date:



Internal Examiner

Name: V. PRALAKSHMI. M
Date: 9/5/24.


External Examiner

Name:
Date:

Dr. Murali S

Head of Department

CSE with spl. in Data Science

ACKNOWLEDGEMENTS

It is my pleasure to express with deep sense of gratitude to Dr. Jayakumar K, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr.

G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor i/c & Pro-Vice Chancellor, VIT Vellore and for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ramesh Babu K, Dean, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore for spending his valuable time and efforts in sharing his knowledge and for helping us in every aspect.

In a jubilant state, I express ingeniously my whole-hearted thanks to Dr. Murali S, Head of the Department, B.Tech. CSE with specialization in Data Science and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staff at Vellore Institute of Technology, Vellore who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Vellore

Date: 09.05.24

Rithik RR (20BDS0201)

Athish M (20BDS0139)

Sundaravelan S (20BCT0248)

Executive Summary

This capstone project explores land use and land cover classification utilizing machine learning algorithms, including CART, Random Forest, KNN, Support Vector Machine, Gradient Boosting, and Minimum Distance. We conducted a comprehensive comparison of these algorithms, focusing on their performance metrics in accurately classifying land use categories. Furthermore, we analyzed changes in urban land cover over time, particularly emphasizing the algorithms that yielded higher accuracy, such as Random Forest and CART. Through rigorous experimentation and evaluation, we identified the strengths and weaknesses of each algorithm in handling land use classification tasks. Our analysis revealed that Random Forest and CART exhibited superior accuracy in capturing urban land cover changes. By comparing these algorithms' performance metrics, we gained valuable insights into their effectiveness in monitoring dynamic environmental transformations. This project contributes to advancing land use and land cover classification methodologies, offering practical implications for urban development monitoring and environmental management. The findings underscore the significance of leveraging machine learning techniques, particularly Random Forest and CART, for accurate and efficient land cover classification. The novelty proves to be our Ensemble approach that compares and votes the best algorithm out of our top 3 according to accuracy.

CONTENTS		Page No.
	Acknowledgement	i.
	Executive Summary	ii.
	Table of Contents	iii.
	List of Figures	iv.
	List of Tables	vi.
	Abbreviations	vii.
1	INTRODUCTION	1
	1.1 Objectives	1
	1.2 Motivation	1
	1.3 Background	1
2	PROJECT DESCRIPTION AND GOALS	2
	2.1 Survey on Existing System	2
	2.2 Research Gap	6
	2.3 Problem Statement	6
3	TECHNICAL SPECIFICATION	7
	3.1 Requirements	7
	3.1.1 Functional	
	3.1.2 Non-Functional	
	3.2 Feasibility Study	8
	3.2.1 Technical Feasibility	
	3.2.2 Economic Feasibility	
	3.2.3 Social Feasibility	
	3.3 System Specification	9
	3.3.1 Hardware Specification	
	3.3.2 Software Specification	
	3.3.3 Standards and Policies	
4	DESIGN APPROACH AND DETAILS	11
	4.1 System Architecture	11
	4.2 Design	13
	4.2.1 Data Flow Diagram	

	4.2.2 Use Case Diagram	
	4.2.3 Class Diagram	
	4.2.4 Sequence Diagram	
	4.3 Constraints, Alternatives and Tradeoffs	19
5	SCHEDULE, TASKS AND MILESTONES	21
	5.1 Gantt Chart	21
	5.2 Module Description	21
	5.2.1 Data Import and Preprocessing	
	5.2.2 Feature Extraction	
	5.2.3 Model Training	
	5.2.4 Classification and Visualization	
	5.2.5 Evaluation and Comparison	
	5.2.6 Conclusion and Recommendations	
	5.3 Testing	22
	5.3.1 Unit Testing	
	5.3.2 Integration Testing	
6	PROJECT DEMONSTRATION	24
7	COST ANALYSIS / RESULT & DISCUSSION (as applicable)	28
8	SUMMARY	35
9	REFERENCES	36
	APPENDIX A – SAMPLE CODE	38

List of Figures

Figure No	Figure	Page No
1	Architecture of Proposed Work	11
2	Data Flow Diagram	13
3	Use Case Diagram	15
4	Class Diagram	17
5	Sequence Diagram	18
6	Gantt Chart	21
7	Shape File	24
8	Satellite image clipped to the image file size	24
9	Feature collection of vegetation	25
10	Feature collection of water body	25
11	Feature collection of barren land	26
12	Feature collection of urban land	26
13	Classification done by random forest model	28
14	Classification done by Aart algorithm	29
15	Classification done by SVM	30
16	Classification done by Gradient Boosting	31
17	Classification done by KNN	32
18	Classification done by Minimum Distance	33
19	Classification done by Ensemble Learning	34

List of Tables

Table No	Table Name	Page No
1	Percentage of each class classified by Random Forest	28
2	Accuracy and F1-Score of Random Forest	28
3	Percentage of each class classified by Cart	29
4	Accuracy and F1-Score of Cart	29
5	Percentage of each class classified by SVM	30
6	Accuracy and F1-Score of SVM	30
7	Percentage of each class classified by Gradient boosting	31
8	Accuracy and F1-Score of Gradient Boosting	31
9	Percentage of each class classified by KNN	32
10	Accuracy and F1-Score of KNN	32
11	Percentage of each class classified by Minimum Distance	33
12	Accuracy and F1-Score of Minimum Distance	33
13	Percentage of each class classified by Ensemble Learning	34

List of Abbreviations

RF	Random Forest
KNN	K- Nearest Neighbor
SVM	Support Vector Machine
MD	Minimum Distance
NDVI	Normalized Difference Vegetation Index
DVI	Difference Vegetation Index
RVI	Ratio Vegetation Index

1. INTRODUCTION

1.1. OBJECTIVE

Implement and compare the performance metrics of various machine learning algorithms in accurately classifying land use categories.

Analyze changes in urban land cover over time and identify the algorithm(s) that yield higher accuracy in capturing these changes, with a focus on Random Forest and CART algorithms. Evaluate the strengths and weaknesses of each algorithm in handling land use classification tasks, providing insights into their effectiveness for environmental monitoring and urban development analysis. Contribute to advancing methodologies for land use and land cover classification, offering practical implications for environmental management and decision-making processes. Provide recommendations for selecting the most suitable machine learning algorithm(s) for land use classification based on performance metrics and accuracy in capturing dynamic land cover changes.

1.2. MOTIVATION

In recent years, there has been a growing interest in using machine learning algorithms for land use and land cover classification. These algorithms have the ability to analyze geospatial data with great precision and speed, making them a valuable tool for this task. However, it is important to carefully evaluate and compare the performance of different machine learning techniques specifically designed for this purpose. In this project, we aim to provide a comprehensive analysis of the strengths, weaknesses, and overall suitability of various machine learning algorithms for land use and land cover classification. Our goal is to offer valuable insights that can help researchers and practitioners choose the most appropriate algorithm for their specific needs.

1.3. BACKGROUND

Traditional methods of land use and land cover classification often rely on manual interpretation and simplistic classification techniques, which can be time-consuming, subjective, and prone to errors. In contrast, machine learning algorithms offer a data-driven approach that leverages advanced computational techniques to automate the classification process based on statistical patterns and training data. This project builds upon the advancements in machine learning and remote sensing technologies to enhance the accuracy and scalability of land use classification methodologies.

2. PROJECT DESCRIPTION AND GOALS

2.1. SURVEY ON EXISTING SYSTEM

SI.NO	PAPER	METHODOLOGY	ADVANTAGES AND LIMITATIONS
1.	Mahmoud, R.; Hassanin, M.; Al Feel, H.; Badry, R.M. Machine Learning-Based Land Use and Land Cover Mapping Using Multi-Spectral Satellite Imagery: A Case Study in Egypt. Sustainability 2023, 15, 9467. https://doi.org/10.3390/su15129467	<ul style="list-style-type: none"> • This project involves collecting high-resolution satellite imagery and creating a mosaic image. Various objects within the image are identified based on shape, color, and land cover type. • The data is categorized into eight distinct categories with detailed sub-categories, and a training dataset is created through digitization. • Machine learning classifiers such as SVM, RF, DT, NB, and ANN are utilized which is trained and evaluated using precision, recall, f-score, kappa index, and confusion matrix. The kappa index is used to compare the efficiency of different models. 	<ul style="list-style-type: none"> • The advantage of the results is the demonstration of reliable classification performance and the potential for developing applications in urban planning and environmental monitoring. • Disadvantage is the complexity and resource requirements of the deep learning techniques used, which may pose challenges for implementation and maintenance.
2.	Aryal, J., Sitaula, C. & Frery, A.C. Land use and land cover (LULC) performance modelling using machine learning algorithms: a case study of the city of Melbourne, Australia. Sci Rep 13, 13510 (2023). https://doi.org/10.1038/s41598-023-40564-0	<ul style="list-style-type: none"> • The study focused on land use and land cover classification in Melbourne, Australia, using machine learning algorithms. • Six machine learning algorithms were employed, including Random Forest, SVM, ANN, Naïve Bayes. • The study conducted category-wise analysis, robustness studies, and statistical analysis of algorithm performance. • The workflow involved statistical and spectral feature extraction, indices-based feature extraction, classification, and implementation using the R programming language. 	<ul style="list-style-type: none"> • The advantage of the study is the stable and reliable performance of machine learning algorithms in land use and land cover classification. • Disadvantage is the potential variation in effectiveness in different geographic locations.

3.	<p>Atef, I., Ahmed, W. & Abdel Maguid, R.H. Modelling of land use land cover changes using machine learning and GIS techniques: a case study in El Fayoum Governorate, Egypt. Environ Monit Assess 195, 637 (2023). https://doi.org/10.1007/s10661-023-11224-7</p>	<p>The study utilized Landsat-8 OLI, Landsat-5, and Landsat 7 ETM+ satellite images from 2000, 2012, 2016, and 2020 sourced from Google Earth Engine and the USGS website. Three supervised classification methods (SVM, RF, MLH) were evaluated using a validation dataset.</p> <ul style="list-style-type: none"> • Accuracy assessment was done using overall accuracy (OA) and kappa coefficient (k). • SVM achieved the highest accuracy, and changes in agricultural, urban, desert, and water areas were detected by comparing LULC maps from different periods. 	<ul style="list-style-type: none"> • The study employs remote sensing, machine learning, and GIS technologies to accurately monitor spatiotemporal changes in LULC. • It identifies the most accurate classifier for LULC change detection, highlighting the challenges of selecting suitable algorithms. • Further research is needed to enhance classification accuracy, especially in regions undergoing rapid urban growth and encroachments on agricultural lands.
4.	<p>M, Arpitha and Ahmed, S and N, Harishnaika. Land use and land cover classification using machine learning algorithms in google earth engine Earth Science Informatics 16,08 (2023). https://doi.org/10.1007/s12145-023-01073-w</p>	<ul style="list-style-type: none"> • The paper focuses on classifying land use and land cover (LULC) in Karnataka using machine learning algorithms in the Google Earth Engine (GEE). • It employs RF, SVM, and CART methods for classification, utilizing multi-temporal images such as Sentinel-2, Landsat-8, and MODIS data. Supervised classification, NDVI assessment, and evaluation of LULC change percentages are conducted. • The study assesses classifier performance using accuracy assessment, correlation coefficients, and RMSE, highlighting the effectiveness of RF, CART, and SVM techniques for LULC classification. • It underscores the significant role of GEE in facilitating large-scale data processing and decision-making. 	<ul style="list-style-type: none"> • The use of Google Earth Engine for land use and land cover classification offers advantages such as convenient and adaptable large-scale data processing. • But may also present limitations in terms of the complexity of machine learning algorithms and the need for expertise in selecting and extracting training samples for supervised classification.
5.	<p>Hamad, R. (2020). An assessment of artificial neural networks support vector machines</p>	<ul style="list-style-type: none"> • Sentinel-2A satellite data covering the study area is initially acquired and pre-processed to correct for atmospheric effects and geometric distortions. The imagery is segmented into homogeneous objects or pixels, and relevant spectral, spatial, and 	<ul style="list-style-type: none"> • The study utilizes Sentinel-2A imagery for high-resolution and multispectral analysis, improving the accuracy of land cover classification.

	and decision trees for land cover classification using sentinel-2A. Data Sci. 8, 459–464. doi: 10.12691/aees-8-6-18	temporal features are extracted as input variables for classification models. • ANNs, SVMs, and decision trees are trained separately on labelled training data to classify land cover types within the satellite imagery. Model performance is evaluated using metrics such as accuracy, precision, recall, and F1-score, through cross validation or validation against ground truth data.	<ul style="list-style-type: none"> Assessing multiple classification algorithms enhances understanding of their applicability and performance in remote sensing. However, focusing solely on Sentinel-2A imagery may limit applicability to other datasets or environmental conditions, potentially overlooking nuances in classification.
6.	Loukika, K. N., Keesara, V. R., and Sridhar, V. (2021). Analysis of land use and land cover using machine learning algorithms on google earth engine for Munneru River Basin, India. Sustainability 13, 13758. doi: 10.3390/su132413758	<ul style="list-style-type: none"> The study utilized machine learning algorithms on the Google Earth Engine platform to analyze land use and land cover (LULC) in the Munneru River Basin, India. Satellite imagery data covering the area and relevant time periods were pre processed. Training data representing different LULC classes were annotated, and algorithms like Random Forest or Support Vector Machine were applied for large-scale LULC classification. Validation techniques assessed classification accuracy, revealing spatial patterns and changes within the basin. 	<ul style="list-style-type: none"> Leveraging Google Earth Engine enables the analysis of large-scale satellite imagery datasets for comprehensive LULC analysis, while machine learning algorithms automate classification, reducing manual intervention. However, variability in satellite imagery data accuracy and availability may affect reliability. Automated methods may struggle with complex land cover types or subtle changes, requiring expert interpretation and validation.
7.	Ming, D., Zhou, T., Wang, M., and Tan, T. (2016). Land cover classification using random forest with genetic algorithm-based parameter optimization. J. Appl. Remote	<ul style="list-style-type: none"> The study proposes a hybrid model for forecasting changes in land use and land cover (LULC) using satellite techniques. It involves integrating satellite imagery, GIS, and machine learning algorithms. Initially, historical LULC data and environmental variables are pre processed. Feature selection techniques identify relevant predictors. Machine learning algorithms like artificial neural networks or random forests are trained on historical data to predict future LULC changes. Validation 	<ul style="list-style-type: none"> Satellite imagery enables broad spatial coverage and consistent data acquisition for analyzing large-scale LULC dynamics. The hybrid model's adaptability enhances its applicability to different scenarios. Limitations include reliance on satellite image quality, variable predictor availability,

	Sens. 10, 35021. doi: 10.1117/1.JRS.10.035021	procedures assess model accuracy and reliability through cross validation or comparison with ground truth data.	and the complexity of machine learning algorithms for interpretation and validation.
8.	Sonobe, R., Yamaya, Y., Tani, H., Wang, X., Kobayashi, N., and Mochizuki, K. I. (2017). Mapping crop cover using multi-temporal Landsat 8 OLI imagery. Int. J. Remote Sens. 38, 4348–4361. doi: 10.1080/01431161.2017.1323286	<ul style="list-style-type: none"> • Use of Landsat 8 OLI data for crop classification in Hokkaido, Japan. • Evaluation of vegetation indices (VIs) calculated from Landsat 8 OLI data • Evaluation of reflectance and combinations of reflectance wavebands • Evaluation of the six components of the Kauth-Thomas transform 	<ul style="list-style-type: none"> • Landsat 8 OLI data resulted in an overall accuracy of 94.5% for crop classification. • VIs derived from bands 6 or 7 were important for crop classification. • Combining all data (reflectance, KT transform, and VIs) achieved the best performance. • Collecting training data is a challenge and requires saving manual labor. • Cross-year training and classification should be analysed for future work.
9.	Wang, X., Gao, X., Zhang, Y., Fei, X., Chen, Z., Wang, J., et al. (2019). Landcover classification of coastal wetlands using the RF algorithm for Worldview-2 and Landsat 8 images. Remote Sens. 11, 1927. doi: 10.3390/rs11161927	<ul style="list-style-type: none"> • The study used Worldview-2 and Landsat-8 satellite images to classify land use and land cover. • They used they following models to find the best suited method: <ul style="list-style-type: none"> • Random Forest (RF) classification method • Support Vector Machine (SVM) method • k-nearest neighbour (k-NN) method 	RF algorithm achieved better classification results for some land cover types RF algorithm provided better classification accuracy compared to SVM and k-NN algorithms Low classification accuracy for buildings covered by vegetation Need for improved classification method or increased number of features
10.	Sang, X., Guo, Q., Wu, X., Xie, T., He, C., Zang, J., et al. (2021). The effect of DEM on the land	<ul style="list-style-type: none"> • Terrain steepness index (TSI) was used to study the effect of DEM on land use/cover classification accuracy. • Data preprocessing included image atmospheric correction and clipping. • Two kinds of image classification were 	<ul style="list-style-type: none"> • Models may not accurately describe the correlation between TSI and classification accuracies. • Study only suited for 30m resolution images. • Further

	use/cover classification accuracy of landsat OLI images. J. Ind. Soc. Remote Sens. 5, 1–12. doi: 10.1007/s12524 021-01318-5	executed separately by controlling the DEM data. • Mathematical regression relationship between classification accuracy and TSI was evaluated	research suggested on TSI range with other spatial resolutions.
--	--	---	---

2.2. RESEARCH GAP

While existing literature extensively discusses the classification of land cover using various machine learning algorithms and their associated accuracy metrics, there is a noticeable absence of efforts aimed at further enhancing this accuracy. Hence, our research seeks to bridge this gap by employing the Ensemble Voting technique, which combines multiple machine learning models, to achieve a higher level of accuracy in land cover classification. This approach represents a novel and innovative step towards improving the precision and reliability of land cover mapping methodologies.

Ensemble voting techniques are powerful strategies in machine learning where predictions from multiple individual models are combined to make a final decision. In hard voting, the majority class prediction is chosen, while soft voting averages probabilities across models.

These methods enhance accuracy, reduce overfitting, and improve robustness by leveraging the diversity of models. Ensemble voting is particularly effective in complex classification tasks and contributes significantly to achieving superior performance in predictive modeling.

2.3. PROBLEM STATEMENT

This project endeavors to create an accurate methodology for mapping land cover types within a specific district using Google Earth Engine. The goal is to refine preprocessing techniques to minimize cloud cover and enhance data quality. Through the implementation of feature extraction methods such as Remote Vegetation Index (RVI) and Difference Vegetation Index (DVI), the objective is to effectively capture spectral information crucial for distinguishing between various land cover categories, including vegetation, water bodies, urban areas, and barren land. By evaluating different machine learning algorithms based on established performance metrics like accuracy, precision, and recall, the project aims to identify the most suitable approach for land cover classification. Ultimately, this effort aims to provide actionable insights for improving environmental management and urban planning strategies in similar geographic contexts.

3. TECHNICAL SPECIFICATIONS

3.1. REQUIREMENTS

3.1.1. FUNCTIONAL REQUIREMENTS

- **Data Collection and Preprocessing:** Collect high-resolution satellite imagery along with the shape file for Chennai. Preprocess the data to correct for atmospheric effects and geometric distortions, and enhance image quality like NDVI RVI.
- **Land Use and Land Cover Classification:** Implement machine learning algorithms (e.g., CART, Random Forest, SVM) for land use and land cover classification.
- Train the algorithms using labeled training datasets derived from the satellite imagery. Evaluate the performance of each algorithm based on metrics such as accuracy, precision, recall, F1-score, and kappa coefficient.
- **Change Detection and Comparison:** Detect changes in urban areas and water bodies over time using classified land cover maps. Compare the extent and nature of changes between different time periods to analyze trends and patterns. Identify areas of significant urban expansion, water body decline, and potential environmental impacts.
- **Visualization and Reporting:** Visualize the classified land cover maps, and performance metrics results. Generate reports summarizing the findings, including maps, charts, and statistical analyses. Provide interactive visualization tools for stakeholders to explore the data and results.

3.1.2. NON-FUNCTIONAL REQUIREMENTS

- **Performance:** Ensure the classification algorithms achieve high accuracy (>80%) in land use and land cover classification. Optimize the processing speed and memory usage of the machine learning models to handle large-scale satellite imagery datasets efficiently.
- **Scalability:** Design the system to scale effectively with increasing data volume and complexity, accommodating future expansion and analysis requirements.
- **Reliability:** Ensure the reliability and robustness of the classification algorithms, minimizing errors and misclassifications in land cover mapping.
- **Usability:** Develop a user-friendly interface for data input, algorithm selection, visualization, and result interpretation. Provide documentation and tutorials to guide users in accessing and utilizing the system effectively.
- **Security:** Implement data encryption, access control mechanisms, and data integrity checks to protect sensitive information and ensure data confidentiality.
- **Compatibility:** Ensure compatibility with different satellite imagery formats, GIS software, and data storage systems for seamless integration and interoperability.
- **Compliance:** Adhere to relevant data privacy regulations, ethical guidelines, and industry standards in data collection, processing, and reporting. Ensure the project complies with environmental and urban planning regulations in Chennai and follows best practices for sustainable development and environmental monitoring.

3.2. FEASIBILITY STUDY

3.2.1. TECHNICAL FEASIBILITY

- **Data Availability:** Check the availability and accessibility of high-resolution satellite imagery and geographic data for Chennai, ensuring the data covers multiple time periods to analyze land cover changes.
- **Algorithm Suitability:** Evaluate the suitability of machine learning algorithms such as CART, Random Forest, and SVM for land use and land cover classification in the Chennai region, considering factors like algorithm performance, scalability, and computational resources.
- **Software and Tools:** Assess the availability and compatibility of the cloud computing platform Google Earth Engine for data processing, analysis, and visualization.
- **Expertise and Resources:** Determine the availability of skilled personnel with expertise in remote sensing, machine learning, and geospatial analysis to develop and implement the project. Consider the availability of hardware resources required for data processing and algorithm training.

3.2.2. ECONOMIC FEASIBILITY

- **Cost of Data Acquisition:** Estimate the costs associated with acquiring high-resolution satellite imagery, geographic data, and software licenses for data processing and analysis.
- **Development and Implementation Costs:** Assess the costs of developing the classification algorithms, designing the user interface, and integrating data visualization tools.
- **Operational Costs:** Consider ongoing operational costs such as data storage, cloud computing resources, and maintenance of software and hardware infrastructure.
- **Return on Investment (ROI):** Evaluate the potential benefits and value generated by the project, such as improved environmental monitoring, informed decision-making for urban planning, and potential cost savings in disaster management and resource allocation.

3.2.3. SOCIAL FEASIBILITY

- **Stakeholder Engagement:** Identify key stakeholders, including government agencies, urban planners, environmental organizations, and local communities, and assess their interest and support for the project.
- **Impact Assessment:** Consider the potential social impact of the project, such as improved flood resilience, sustainable urban development, and enhanced environmental awareness among citizens.

- **Ethical Considerations:** Address ethical considerations related to data privacy, consent, and responsible use of technology in environmental monitoring and urban planning.
- **Community Benefits:** Evaluate how the project's outcomes, such as accurate land use maps and environmental insights, can benefit the local community by informing policy decisions, supporting disaster preparedness, and promoting sustainable development practices.

3.3. SYSTEM SPECIFICATION

3.3.1. HARDWARE SPECIFICATION

- **Computer System:** Processor like Intel Core i5 or AMD Ryzen 5 (or higher) for smooth performance when running JavaScript or Python scripts in the Google Earth Engine Code Editor.
- **RAM:** Minimum 8GB RAM (16GB recommended) for handling large geospatial datasets and complex data processing tasks.
- **Storage:** Sufficient local storage space for storing project files, scripts, and downloaded datasets. SSD storage is preferable for faster read/write speeds.
- **Graphics Card:** A dedicated graphics card is not necessary for Google Earth Engine as it operates in a web-based environment. Integrated graphics should suffice.
- **Internet Connectivity:** A stable and high-speed internet connection is essential for accessing Google Earth Engine's online platform, streaming satellite imagery, and running analysis tasks.
- **External Resources:** Google Earth Engine leverages cloud computing resources provided by Google's infrastructure. You do not need to set up or manage hardware servers. However, ensure your internet connection can handle data transfer to and from Google's servers efficiently.

3.3.2. SOFTWARE SPECIFICATION

- **Operating System:** Windows 10, macOS, or Linux-based operating systems (Ubuntu, CentOS) are compatible with Google Earth Engine.
- **Web Browser:** Google Chrome is the recommended web browser for accessing and working with Google Earth Engine. Other modern browsers like Mozilla Firefox and Microsoft Edge can also be used, but Chrome tends to offer the best performance and compatibility.
- **Integrated Development Environment (IDE):** Google Earth Engine Code Editor: This web-based IDE provided by Google Earth Engine allows you to write, edit, and run JavaScript or Python scripts directly in your browser. It provides access to GEE's API and visualization tools for analyzing geospatial data.
- Google Earth Engine is a cloud-based platform developed by Google for planetary-scale environmental data analysis. It combines a vast archive of satellite imagery and

geospatial datasets with powerful computational tools to enable researchers, scientists, and policymakers to monitor and analyze changes in the Earth's environment.

- One of the key features of Google Earth Engine is its ability to process and analyze massive amounts of geospatial data quickly. This makes it particularly useful for tasks like land classification, where large datasets and computational power are required to classify and map land cover types accurately.

3.3.3. STANDARDS AND POLICIES

- **Data Privacy and Security Policies:**
Compliance with data protection regulations is ensured, such as GDPR and CCPA. Encryption methods are implemented for sensitive data during transmission and storage. Access controls and authentication mechanisms are established to prevent unauthorized access.
- **Ethical Guidelines:**
Ethical guidelines are followed for handling data, particularly concerning personal information. Biases in data collection, preprocessing, and model training are avoided to maintain fairness. Privacy rights are respected, and consent is obtained when necessary for data usage.
- **Quality Assurance Standards:**
Industry best practices are followed for data quality assessment and validation. Regular quality checks are conducted on input data, preprocessing steps, and model training outcomes. Comprehensive documentation of quality control procedures is maintained to ensure data integrity.
- **Code of Conduct for Development:**
Coding standards and guidelines are adhered to for software development. Version control systems and collaborative development platforms are used for code management. Code reviews and testing are conducted to ensure code quality and reliability.
- **Documentation and Reporting Policies:**
Comprehensive documentation is maintained for all project components, including data sources, algorithms, and results. Clear documentation on the use of machine learning models, ensemble voting techniques, and accuracy assessment methods is provided. Regular reports on project progress, milestones achieved, and any issues encountered are generated.
- **Intellectual Property Rights (IPR):**
Intellectual property rights and licensing agreements for software tools, libraries, and datasets used in the project are respected. Necessary permissions or licenses are obtained for the commercial use of third-party resources.
- **Compliance with Environmental Regulations:**
Compliance with environmental regulations and guidelines, especially concerning land use and monitoring activities, is ensured. Environmental impact assessments and sustainability practices are considered in project implementation.
- **Accessibility and Inclusivity:**
User interfaces and visualizations are designed to be accessible to all users, including those with disabilities. Inclusive design principles are followed to ensure usability and accessibility throughout the project.

4. DESIGN, APPROACH AND DETAILS

4.1. SYSTEM ARCHITECTURE:

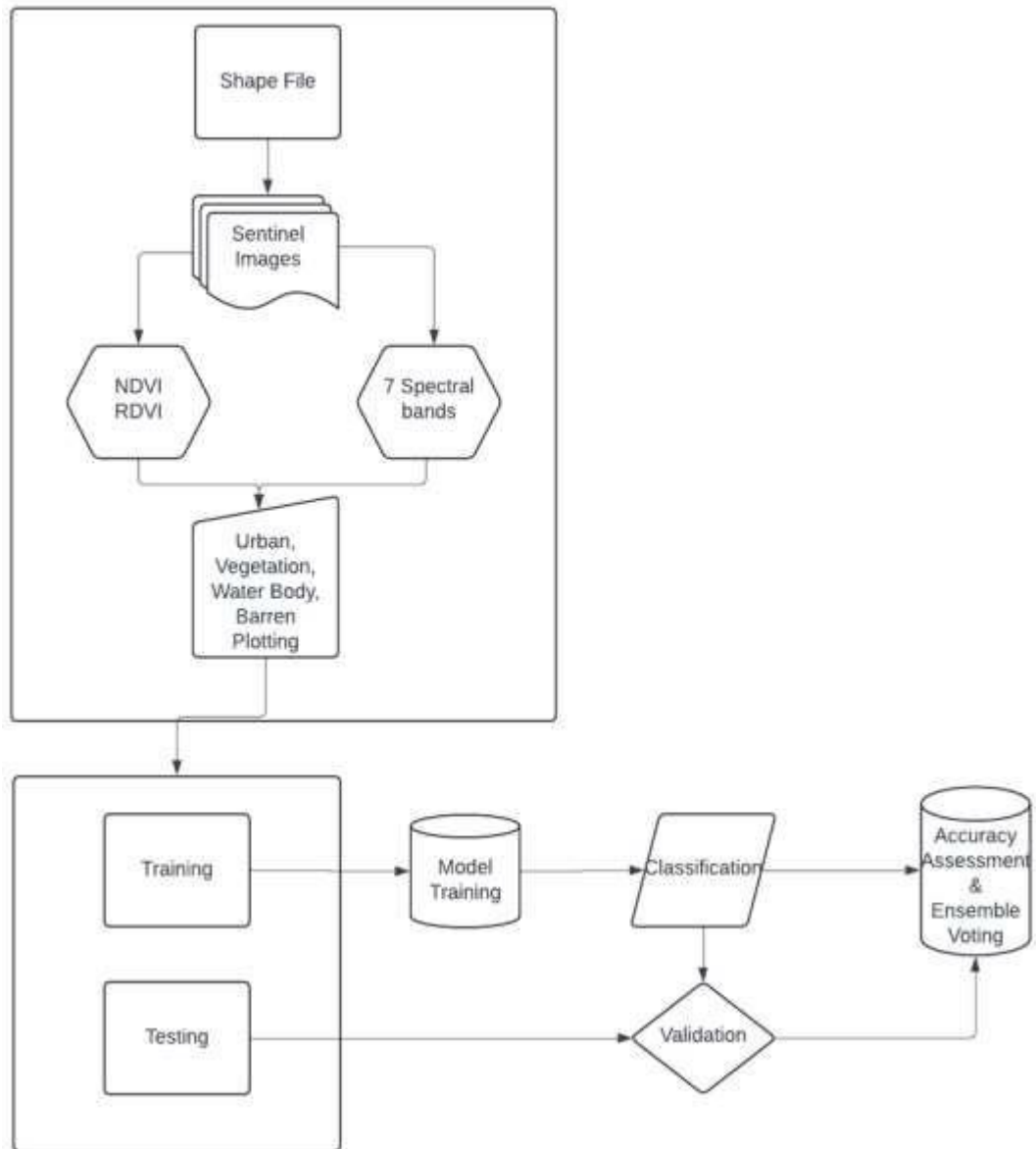


Figure 1: Architecture of proposed work

Data Acquisition and Preprocessing:

- Obtain the shapefile for Chennai, which delineates its geographical boundaries.
- Retrieve Sentinel satellite imagery data covering Chennai's region.

- Preprocess the satellite imagery to correct for atmospheric effects, geometric distortions, and resample to a common spatial resolution.

Feature Extraction and Index Calculation:

- Extract spectral bands (i.e., NIR, Red, Green, Blue) from the preprocessed Sentinel imagery.
- Calculate vegetation indices such as Normalized Difference Vegetation Index (NDVI), Ratio Vegetation Index (RVI), and Difference Vegetation Index (DVI) using the spectral bands.

Land Cover Classification:

- Use the shapefile to mask the spectral bands, focusing only on the area within Chennai.
- Apply machine learning algorithms (SVM, RF, KNN, Cart, Minimum Distance, Gradient Boosting) to classify land cover types (urban, vegetation, barren, water bodies) based on the calculated indices and spectral bands.

Training and Testing Data Split:

- Divide the dataset into training and testing subsets to train the machine learning models.
- Ensure a balanced representation of each land cover class in both training and testing sets.

Model Training and Validation:

- Train the machine learning models (SVM, RF, KNN, Cart, Minimum Distance, Gradient Boosting) using the training data.
- Validate the models using the testing data to assess their accuracy and performance metrics (precision, recall, F1-score).

Ensemble Voting Methodology:

- Implement ensemble voting to combine predictions from all six algorithms (SVM, RF, KNN, Cart, Minimum Distance, Gradient Boosting) for robust classification results.
- Utilize hard or soft voting mechanisms to determine the final classification output.

Accuracy Assessment and Validation:

- Evaluate the accuracy of the ensemble voting approach using metrics like overall accuracy, kappa coefficient, confusion matrix, and ROC curve analysis.
- Validate the classified land cover map against ground truth data or reference datasets to ensure reliability.

Result Visualization and Reporting:

- Visualize the classified land cover map overlaid on the Chennai shapefile to depict urban, vegetation, barren, and water bodies distribution.
- Generate reports or dashboards summarizing the project's methodology, results, accuracy metrics, and recommendations for land cover management and analysis.

4.2 DESIGN

4.2.1 DATA FLOW DIAGRAM

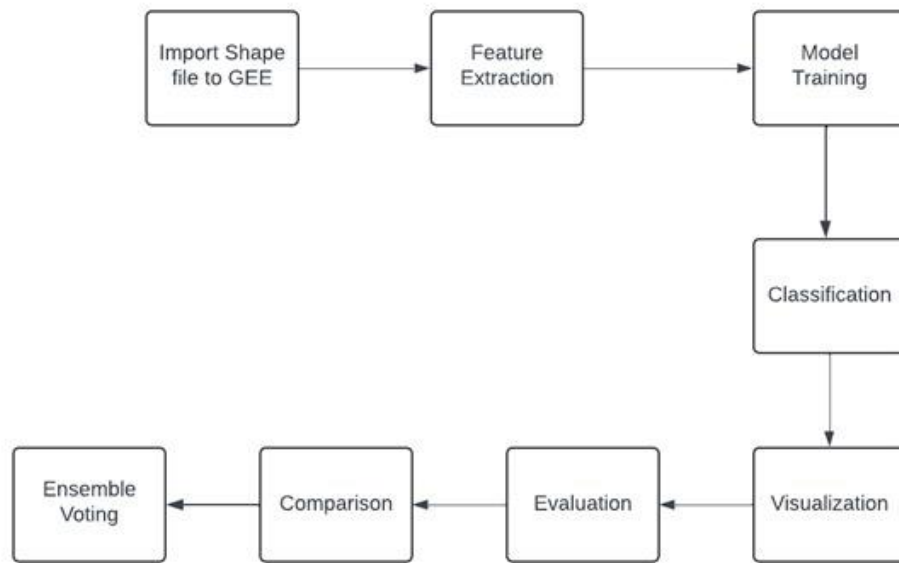


Figure 2: Data Flow Diagram

Data Acquisition:

Data is initially sourced from external or internal repositories, sensors, or input interfaces. In our project, satellite imagery data is acquired from sources like Sentinel using Google Earth Engine.

Preprocessing:

Upon acquisition, the raw data undergoes preprocessing steps such as cleaning, filtering, normalization, and transformation. In your case, preprocessing involves correcting atmospheric effects, geometric distortions, and extracting spectral indices.

Feature Extraction:

Relevant features or attributes are extracted from the preprocessed data. This step involves identifying key patterns, characteristics, or variables that are crucial for subsequent analysis or modeling. In our project, features like NDVI, RVI, and DVI are extracted from the spectral bands.

Model Training:

The extracted features are used to train machine learning models, such as SVM, RF, KNN, Cart, Minimum Distance and Gradient Boosting. This step involves feeding the training data into the models and optimizing their parameters to learn the underlying patterns and relationships in the data.

Classification and Prediction:

Trained models are then used to classify or predict outcomes based on new or unseen data instances. For instance, in land cover classification, the models predict whether a given pixel corresponds to urban, vegetation, barren land, or water bodies.

Validation and Evaluation:

The predicted outcomes are compared against known ground truth data or validation sets to assess the accuracy, precision, recall, F1-score, and other performance metrics of the models.

Ensemble Voting:

In ensemble voting, predictions from multiple models are aggregated using a voting mechanism (hard or soft) to generate a final prediction or decision. This step ensures robustness and improves accuracy by leveraging the collective intelligence of diverse models.

Result Presentation:

Finally, the classified land cover map, accuracy metrics, and evaluation results are presented to users through visualization tools, reports, or dashboards for analysis, interpretation, and decision-making.

4.2.2 USE CASE DIAGRAM**Actors:**

- **User:** Interacts with the system to perform tasks such as data input, model

training, and result analysis.

- **Google Earth Engine:** Represents the land cover classification system itself.
- **External sources:** Provide additional support such as providing shapefile for Chennai.

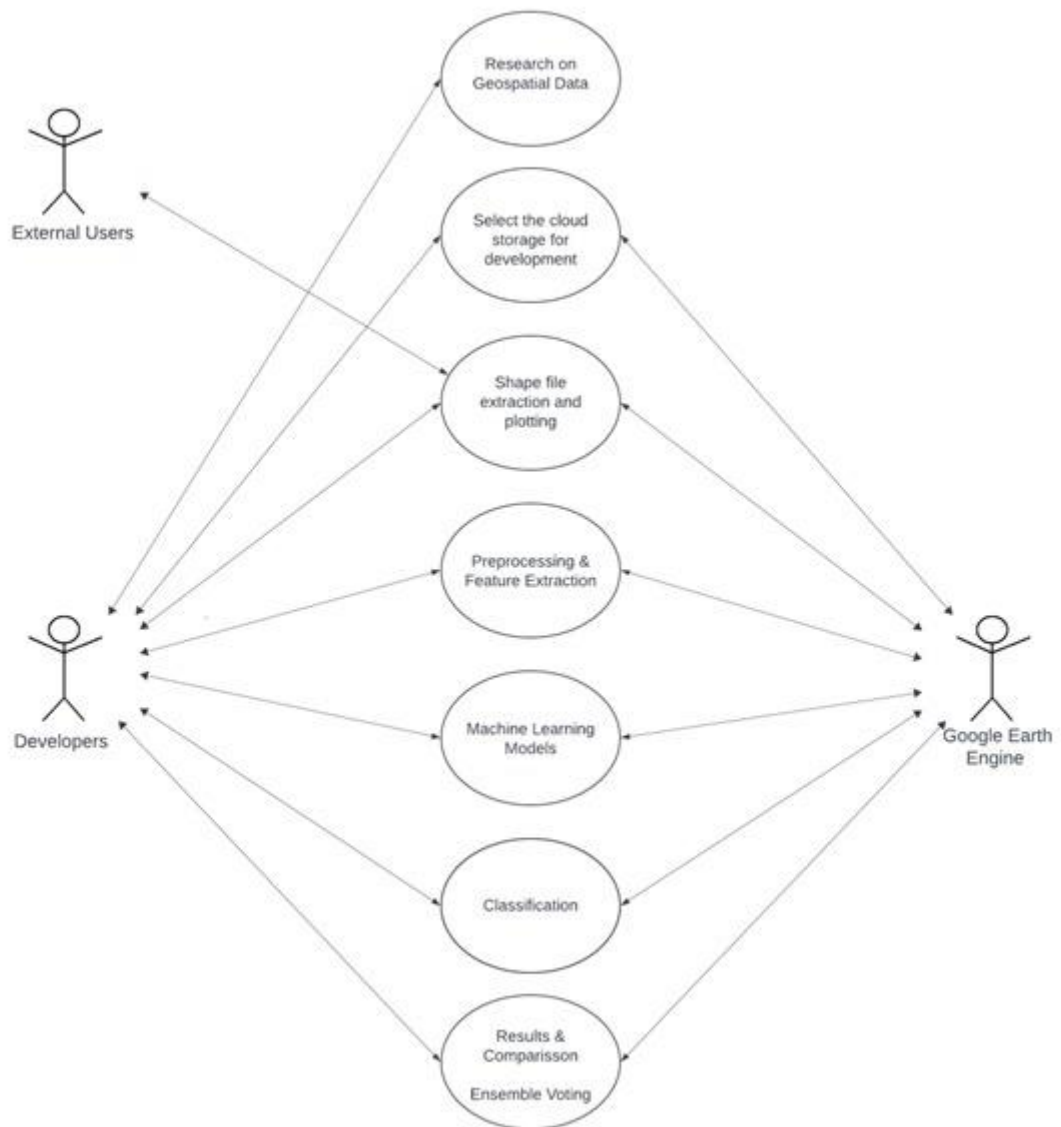


Figure 3: Use Case Diagram

Use Cases:

- **Input Satellite Data:** Allows the user to upload or input satellite imagery data

(i.e Sentinel) for land cover analysis.

- **Preprocess Data:** Preprocesses the input data, including atmospheric correction, geometric distortions, and spectral index calculation (NDVI, RVI, DVI).
- **Train Models:** Trains machine learning models (SVM, RF, KNN, Cart, Gradient Boosting and Minimum Distance.) using the preprocessed data for land cover classification.
- **Classify Land Cover:** Classifies land cover types (urban, vegetation, barren, water bodies) using trained models and spectral indices.
- **Evaluate Accuracy:** Evaluates the accuracy of classification results using validation datasets and performance metrics.
- **Apply Ensemble Voting:** Applies ensemble voting technique to combine predictions from multiple models for improved accuracy.
- **Present Results:** Presents the classified land cover map, accuracy metrics, and visualizations for user analysis.

Relationships:

- User interacts with all use cases mentioned above, representing the user-system interactions.
- The system interacts with internal components (e.g., data preprocessing module, model training module, ensemble voting module) to execute use cases.

System Boundary:

- The system boundary encloses all use cases and actors, defining the scope of the land cover classification system.

4.2.3 CLASS DIAGRAM

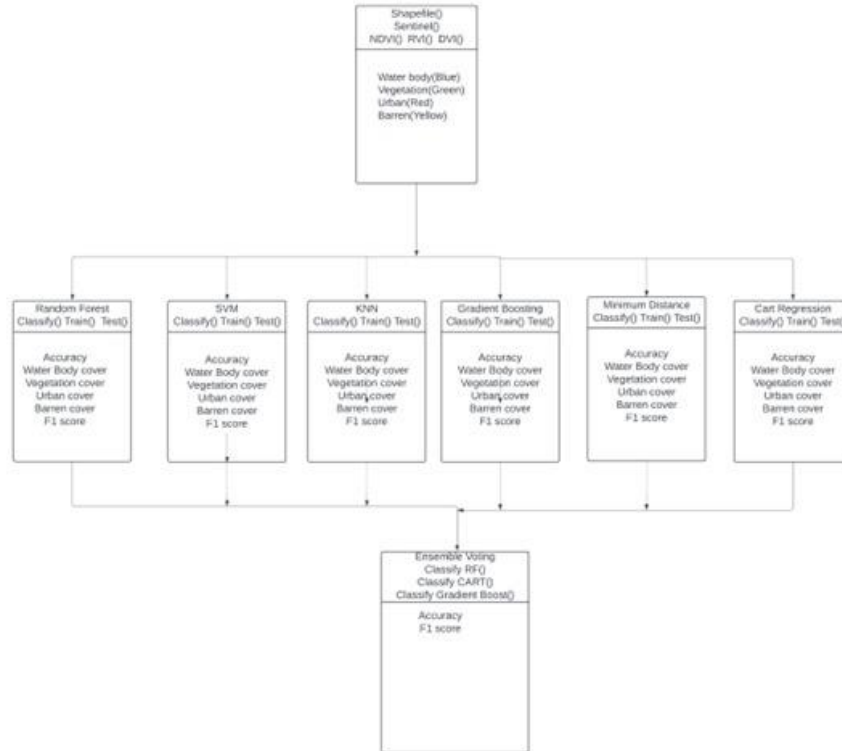


Figure 4: Class Diagram

DataProcessor Class:

- Handles preprocessing of satellite imagery data.
- **Methods:** Preprocess(Sentinel images) and CalculateIndices(NDVI, RVI, DVI).

ModelTrainer Class:

- Trains machine learning models (SVM, RF, KNN, Cart, Gradient Boosting, Minimum Distance) for classification.
- **Method:** TrainModel(For all 6 algorithms).

EnsembleVoter Class:

- Implements ensemble voting to combine predictions from models.
- **Method:** ApplyEnsembleVoting(modelPredictions).

ResultPresenter Class:

- Displays classified land cover map and accuracy metrics.

4.2.4 SEQUENCE DIAGRAM

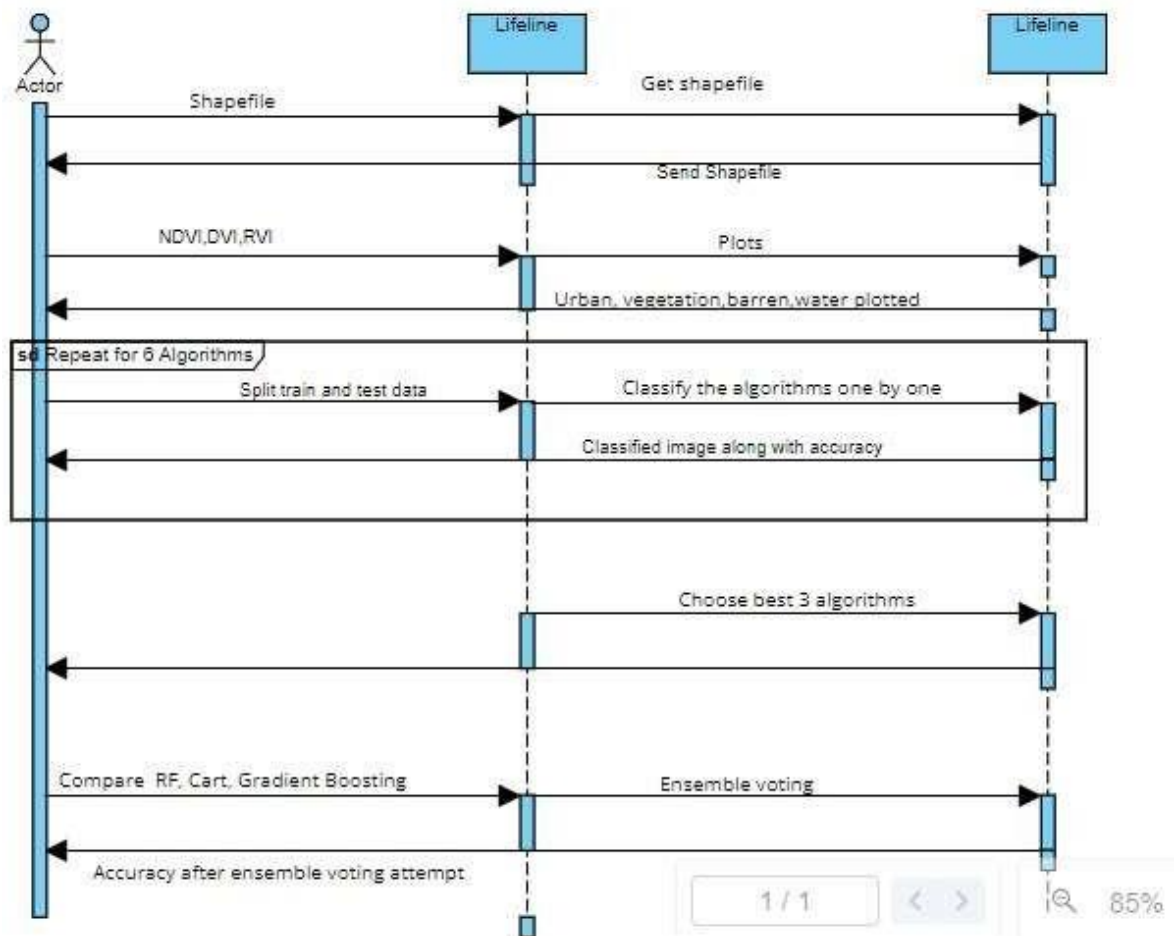


Figure 5: Sequence Diagram

User Uploads Data and Initiates Processing:

- User interacts with the system by uploading satellite data.
- System triggers DataProcessor to preprocess the uploaded data (Preprocess method).

Data Preprocessing:

- DataProcessor processes the input data, calculates spectral indices, and generates preprocessed data.

Model Training Phase:

- User selects the ML model type (SVM, RF, KNN, Cart, Gradient Boosting, Minimum Distance) and triggers ModelTrainer to train the selected model.
- ModelTrainer utilizes preprocessed data to train the model and stores the trained model.

Classification and Ensemble Voting:

- User requests land cover classification using trained models.
- System activates ModelTrainer to classify land cover types based on trained models and spectral indices.
- EnsembleVoter applies ensemble voting to combine predictions from multiple models for enhanced accuracy.

Result Presentation:

- ResultPresenter retrieves the classified land cover map and accuracy metrics.
- ResultPresenter presents the results to the user through visualizations and metrics display.

4.3. CONSTRAINTS, ALTERNATIVES AND TRADEOFFS

Constraints:

- Data Availability: Limited access to high-resolution satellite imagery and ground truth data poses a constraint on achieving optimal classification accuracy.
- Computational Resources: The project may face constraints related to computational resources, such as processing power and memory, which could limit the complexity of algorithms and size of datasets that can be processed.
- Budget: Budgetary constraints may impact the acquisition of premium datasets, hiring of domain experts, or procurement of specialized software tools.

Alternatives:

- Algorithm Exploration: Exploring alternative machine learning algorithms or ensemble techniques offers potential alternatives to enhance classification accuracy.
- Data Augmentation: Implementing data augmentation techniques can augment the training dataset's diversity and size, especially in areas with limited ground truth data.
- Transfer Learning: Leveraging transfer learning approaches or pre-trained models presents an alternative strategy to adapt existing models for land cover classification tasks.

Trade-offs:

- Accuracy vs. Computational Cost: Balancing the desire for higher accuracy with the computational cost associated with complex algorithms, which may impact processing time and resource utilization.
- Generalization vs. Overfitting: Managing the trade-off between model generalization (avoiding overfitting) and achieving high accuracy specifically on the training dataset.
- Data Quality vs. Quantity: Prioritizing data quality improvements (e.g., noise reduction, error correction) over increasing data quantity to ensure better model performance while considering dataset diversity.
- Resource Allocation Trade-offs:
- Hardware vs. Software Investments: Evaluating the trade-off between investing in high-performance hardware (e.g., GPUs, cloud computing) or optimizing software

algorithms for efficient processing.

- Expertise vs. Outsourcing: Considering the trade-off between hiring domain experts for tasks like data annotation and model tuning versus outsourcing certain project aspects to external vendors.
- Data Acquisition vs. Data Enhancement: Weighing the benefits of acquiring diverse, high-quality datasets against implementing data enhancement techniques (e.g., image fusion, resolution enhancement) on existing datasets.

5. SCHEDULE, TASKS AND MILESTONES

5.1. GANTT CHART:

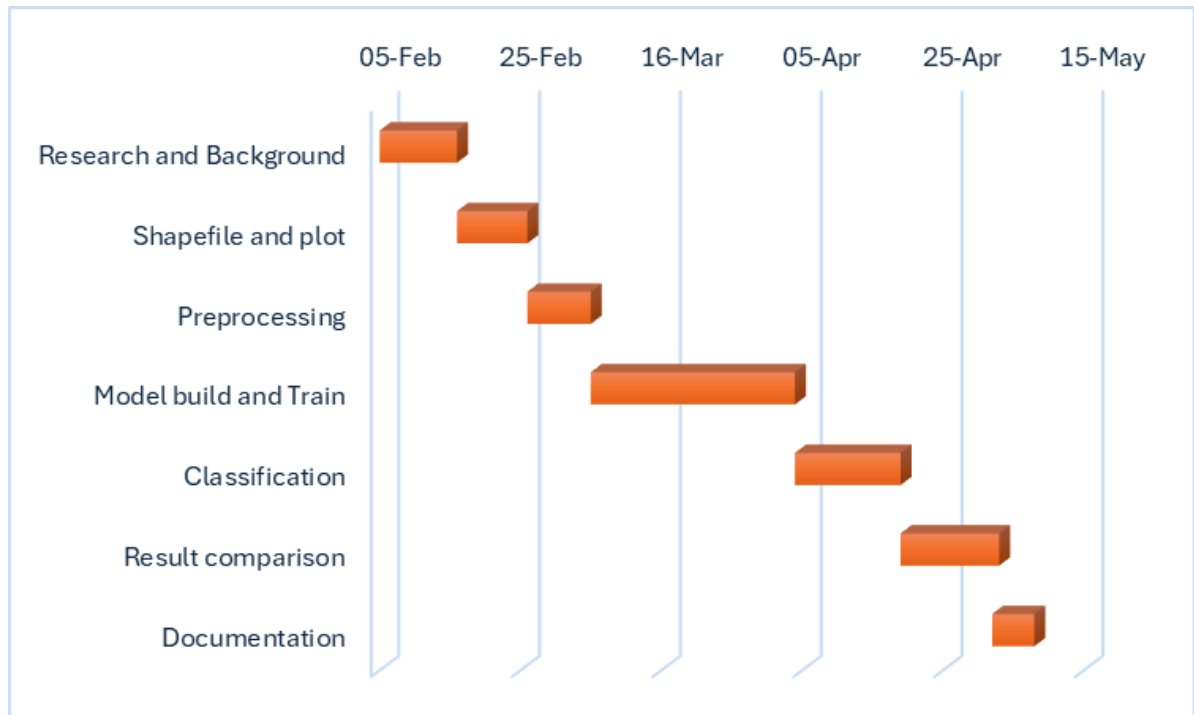


Figure 6: Gantt Chart

5.2. MODULE DESCRIPTION:

Data Import and Preprocessing:

- Importing Shapefile: The module begins by importing a shapefile of a specific district, defining the study area for land cover classification.
- Preprocessing: Removal of cloud cover from satellite imagery using Google Earth Engine's capabilities to ensure clear and reliable data for analysis.

Feature Extraction:

- Remote Sensing Indices: Calculation of Remote Vegetation Index (RVI) and Difference Vegetation Index (DVI) from satellite imagery to capture vegetation health and density within the study area.

Model Training:

- Selection of Machine Learning Algorithms: Various algorithms such as Random Forest, CART-Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Gradient Boosting, and Minimum Distance are implemented for model training.
- Training Data Preparation: Splitting the preprocessed data into training and testing datasets for model evaluation.

- **Model Training:** Training each selected algorithm using the training dataset to learn the relationships between spectral features and land cover classes.

Classification and Visualization:

- **Land Cover Classification:** Application of trained models to classify land cover types including vegetation, water bodies, urban areas, and barren land across the study area.
- **Visualization:** Plotting the classified points on the shapefile to visually represent the distribution of land cover classes within the district.

Evaluation and Comparison:

- **Performance Metrics Calculation:** Calculation of accuracy, precision, recall, F1 score, and confusion matrix for each machine learning algorithm to evaluate classification performance.
- **Comparative Analysis:** Comparison of performance metrics to identify the most effective model for land cover classification in Google Earth Engine based on the specific study area.

Conclusion and Recommendations:

- **Summary of Findings:** Summarizing the results obtained from the evaluation and comparison of machine learning algorithms.
- **Recommendations:** Providing recommendations for the selection of the most suitable algorithm for future land cover classification projects in similar geographic regions.
- **Future Directions:** Discussing potential enhancements or modifications to improve the accuracy and efficiency of land cover classification using Google Earth Engine.

5.3. TESTING

5.3.1 UNIT TESTING:

Data Import and Preprocessing:

- Verify correct shapefile import and study area definition.
- Validate cloud cover removal for clear and reliable imagery.

Feature Extraction:

- Confirm accurate calculation of RVI and DVI for vegetation health assessment.

Model Training:

- Ensure proper data splitting and accurate model training for each algorithm.

Classification and Visualization:

- Validate accurate land cover classification and plotting on the shapefile.

Evaluation and Comparison:

- Verify correct calculation of performance metrics and identification of the most effective model.

5.3.2 INTEGRATION TESTING:

End-to-End Testing:

- Test the entire workflow for seamless integration between modules.

Data Flow and Model Integration:

- Ensure consistent data flow and compatibility of different machine learning algorithms.

Visualization Integration:

- Verify accurate representation of classified land cover types on the shapefile.

Performance and Error Handling:

- Evaluate system performance and test error handling mechanisms.

6. PROJECT DEMONSTRATION:

Loading Shape File:

```
i 1 var Geo = ee.FeatureCollection('projects/ee-rithikragupathi/assets/CHENNAI')  
i 2 Map.addLayer(Geo, {}, 'Geo')
```



Figure 7: Shape File of Chennai

Getting Satellite Image .

```
4 //Loading satellite  
5 var image = ee.ImageCollection("COPERNICUS/S2_SR_HARMONIZED")  
6   .filterDate('2020-01-01','2020-01-31')  
7   .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE',1))  
8   .filterBounds(Geo)  
9   .mean();  
10  
11 // Clip Sentinel-2 image to the extent of the shapefile  
12 var clippedImage = image.clip(Geo);
```



Figure 8: Satellite image clipped to the shape file size

Preprocessing: (Finding NDVI, RVI, DVI)

```
18 //NDVI
19 var ndvi = clippedImage.normalizedDifference(['B8','B4']).rename('NDVI');
20 print(ndvi)
21
22 //DVI
23 var dvi = clippedImage.select('B8').subtract(clippedImage.select('B4')).rename('DVI');
24 print(dvi)
25
26 //RVI
27 var rvi = clippedImage.select('B4').divide(clippedImage.select('B8')).rename('RVI');
28 print(rvi)
29
30 //Adding the new bands to exisiting ones
31 var combinedFeatures = clippedImage.addBands([ndvi, dvi, rvi]);
32 print(combinedFeatures)
```

Feature Collection:

```
34 // Labelling each class with their corresponding IDs for each point (LC for Land Cover, set any lat
35 var labelled_vegetation = Vegetation.map(function(feature){return feature.set({'LC':0})});
36 var labelled_water = WaterBody.map(function(feature){return feature.set({'LC':1})});
37 var labelled_barren = Barren.map(function(feature){return feature.set({'LC':2})});
38 var labelled_urban = Urban.map(function(feature){return feature.set({'LC':3})});
39
40 // Merging these data points and calling them GCPs
41 var GCPs = labelled_vegetation.merge(labelled_water).merge(labelled_barren).merge(labelled_urban);
42 // Lets print the GCPs
43 print(GCPs, 'GCPs Merged');
```



Figure 9: Feature Collection of Vegetation

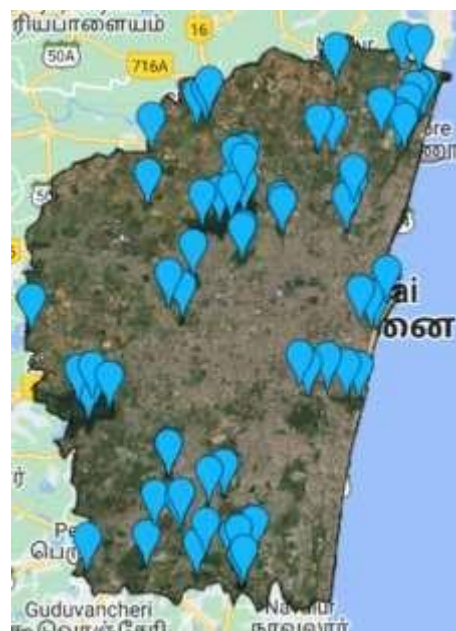


Figure 10: Feature Collection of Water Body



Figure 11: Feature Collection of Barren Land

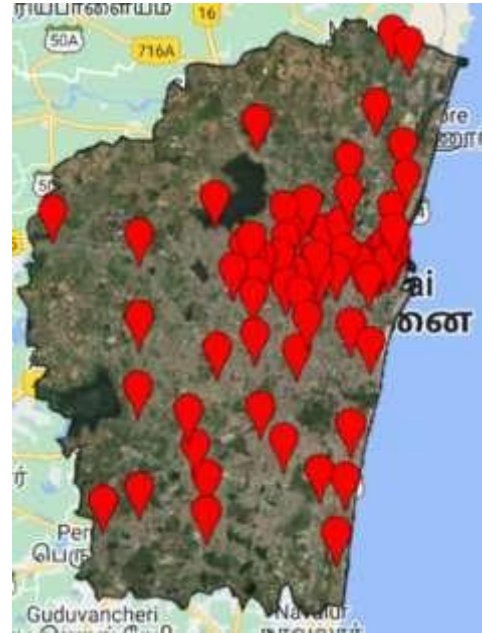


Figure 12: Feature Collection of Urban Land

TrainSet and TestSet Split:

```
// Now we will sample the spectral values of each band for these points (GCPs);
var sampled_GCPs = input.sampleRegions({
  collection:GCPs,
  properties:[label],
  scale:10
});
print(sampled_GCPs);

// Split the data into training and testing sets (e.g., 80% training, 20% testing)
var split = 0.8; // Ratio of training to testing data
var trainSet = sampled_GCPs.randomColumn('random').filter(ee.Filter.lt('random', split));
print(trainSet)

var testSet = sampled_GCPs.randomColumn('random').filter(ee.Filter.gte('random', split));
print(testSet)
```

Model Building and classification:

```
69 var classifierRF = ee.Classifier.smileRandomForest(10).train(trainSet, label, bands);
70
71 // Classify the clipped image using the trained classifier
72 var classifiedRF = input.classify(classifierRF);
```

```
70 var classifierCART = ee.Classifier.smileCart().train(trainSet, label, bands);
71
72 // Classify the clipped image using the trained classifier
73 var classifiedCART = input.classify(classifierCART);
```

```
67 // Train a SVM classifier using the training data
68 var classifierSVM = ee.Classifier.libsvm().train(trainSet, label, bands);
69
70 // Classify the clipped image using the trained classifier
71 var classifiedSVM = input.classify(classifierSVM);
```



```

68 // Train a Gradient Boost classifier using the training data
69 var classifierGB = ee.Classifier.smileGradientTreeBoost(10).train(trainSet, label, bands);
70
71 // Classify the clipped image using the trained classifier
72 var classifiedGB = input.classify(classifierGB);

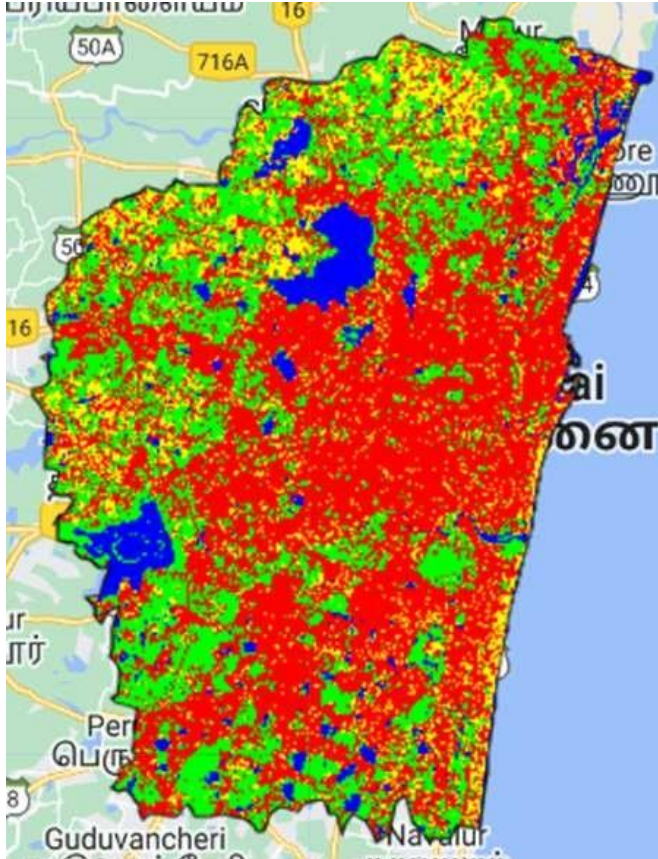
68 // Train a k-Nearest Neighbors (kNN) classifier using the training data
69 var classifierKNN = ee.Classifier.smileKNN().train(trainSet, label, bands);
70
71 // Classify the clipped image using the trained classifier
72 var classifiedKNN = input.classify(classifierKNN);

68 // Train a Minimum Distance Classifier using the training data
69 var classifierMinDist = ee.Classifier.minimumDistance().train(trainSet, label, bands);
70
71 // Classify the clipped image using the trained classifier
72 var classifiedMinDist = input.classify(classifierMinDist);

```

7. RESULTS AND DISCUSSION:

Classification By Random Forest:



Green: Vegetation

Blue: Water

Yellow: Barren

Red: Urban

Table 1: Percentage of each class classified by RF

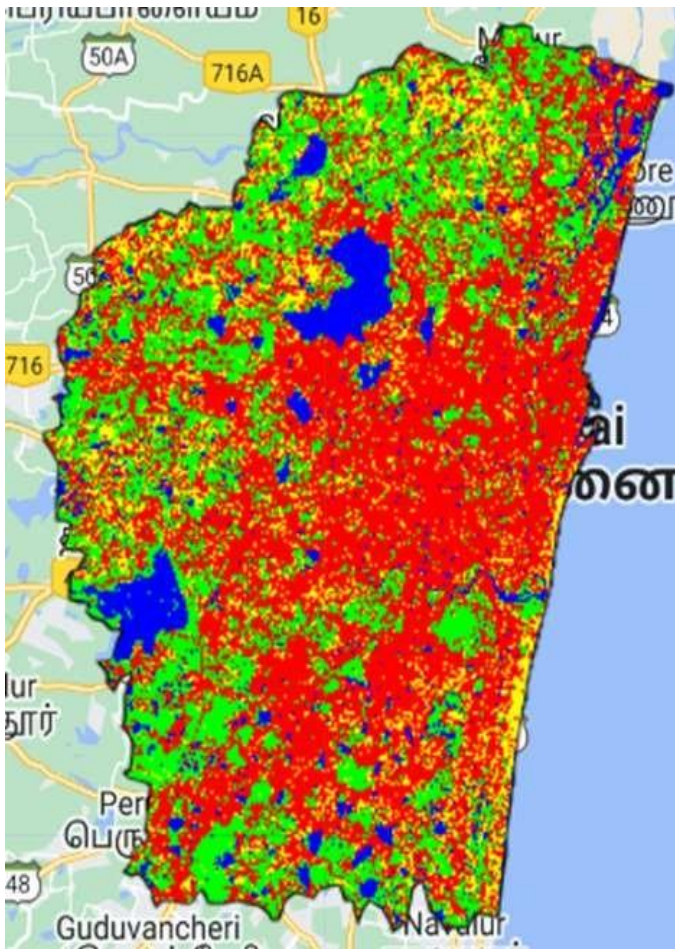
% of Vegetation Area	35.52
% of Water Area	7.62
% of Barren Area	16.42
% of Urban Area	40.42

Figure 13: Classification done by Random Forest Model

Table 2: Accuracy and F1-score of RF

Measurement Parameters(RF)	Values
Accuracy	0.888
F1-Score(Vegetation)	0.888
F1-Score(Water)	0.967
F1-Score(Barren)	0.869
F1-Score(Urban)	0.814

Classification by CART:



Green: Vegetation

Blue: Water

Yellow: Barren

Red: Urban

Table 3: Percentage of each class classified by CART

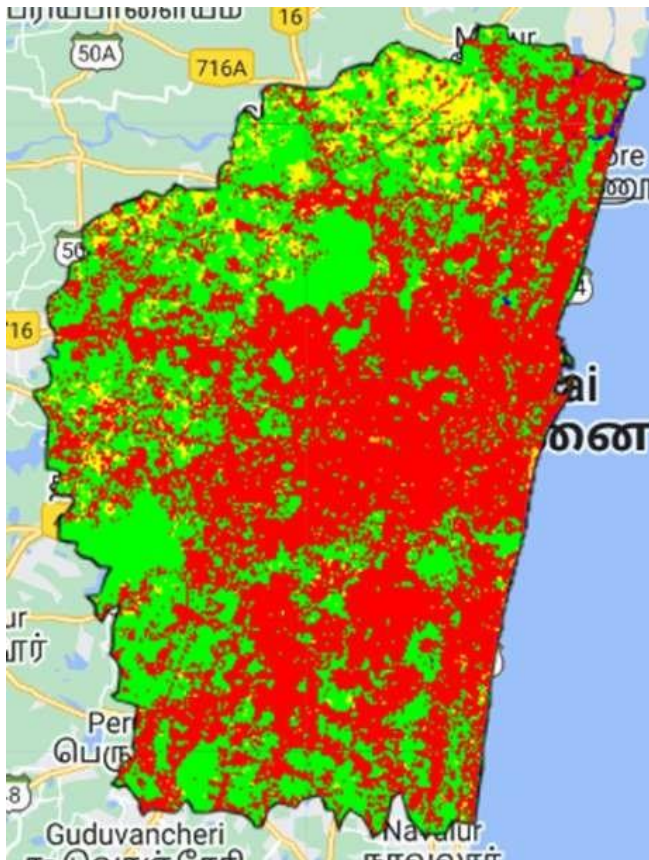
% of Vegetation Area	33.89
% of Water Area	10.11
% of Barren Area	13.36
% of Urban Area	42.62

Figure 14: Classified image Done by CART Algorithm

Table 4: Accuracy and F-1 Score of CART

Measurement Parameters (CART)	Values
Accuracy	0.87
F1-Score(Vegetation)	0.833
F1-Score(Water)	1
F1-Score(Barren)	0.818
F1-Score(Urban)	0.800

Classification by SVM:



Green: Vegetation

Blue: Water

Yellow: Barren

Red: Urban

Table 5: Percentage of each class classified by SVM

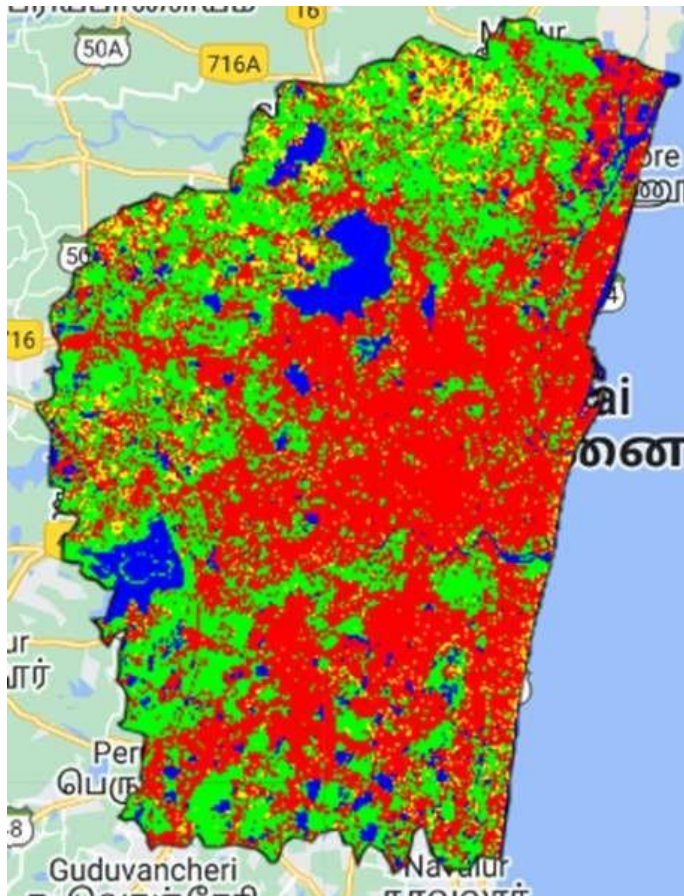
% of Vegetation Area	46.94
% of Water Area	1.73
% of Barren Area	10.04
% of Urban Area	41.28

Figure 15: Classification done by SVM

Table 6: Accuracy and F1-Score of SVM

Measurement Parameters (SVM)	Values
Accuracy	0.592
F1-Score(Vegetation)	0.533
F1-Score(Water)	NAN
F1-Score(Barren)	0.960
F1-Score(Urban)	0.727

Classification by Gradient Boost:



Green: Vegetation

Blue: Water

Yellow: Barren

Red: Urban

Table 7: Percentage of each class classified by GB

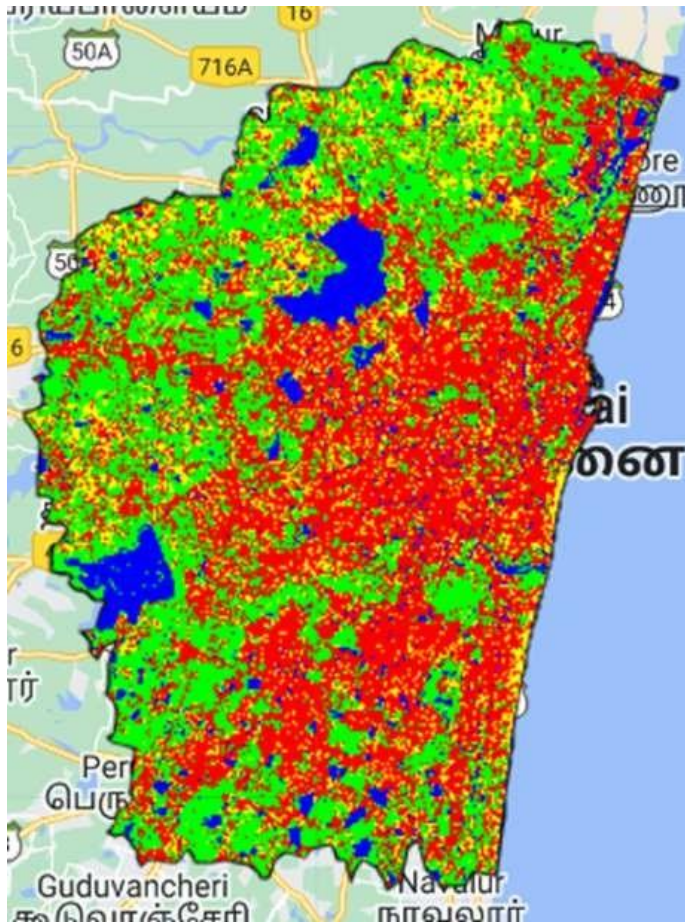
% of Vegetation Area	39.97
% of Water Area	9.13
% of Barren Area	9.08
% of Urban Area	41.80

Figure 16: Classification done by GB

Table 8: Accuracy and F1-Score of GB

Measurement Parameters (GB)	Values
Accuracy	0.833
F1-Score(Vegetation)	0.827
F1-Score(Water)	0.937
F1-Score(Barren)	0.818
F1-Score(Urban)	0.719

Classification by KNN:



Green: Vegetation

Blue: Water

Yellow: Barren

Red: Urban

Table 9: Percentage of each class classified by KNN

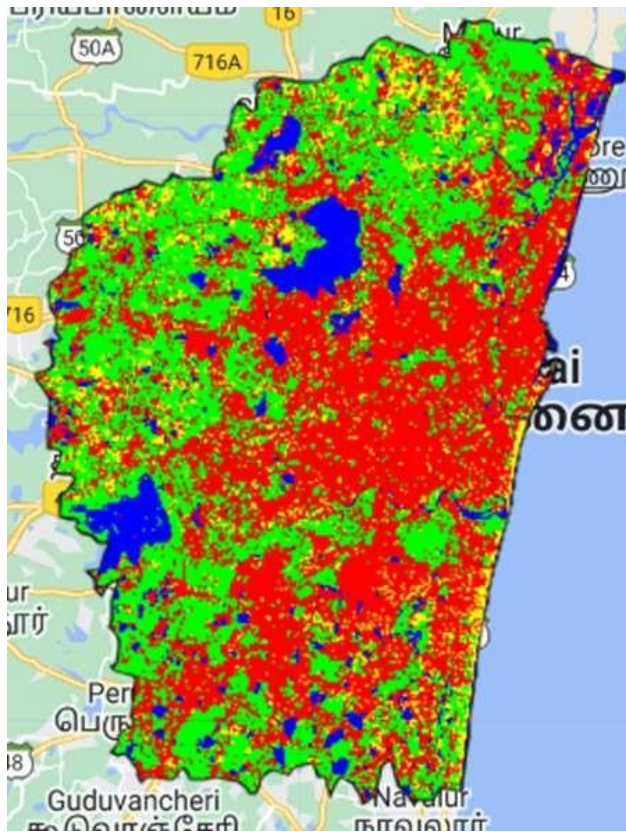
% of Vegetation Area	37.51
% of Water Area	9.80
% of Barren Area	13.59
% of Urban Area	39.09

Figure 17: Classification done by KNN

Table 10: Accuracy and F1-Score of SVM

Measurement Parameters(KNN)	Values
Accuracy	0.851
F1-Score(Vegetation)	0.857
F1-Score(Water)	0.967
F1-Score(Barren)	0.818
F1-Score(Urban)	0.740

Classification by Minimum Distance



Green: Vegetation

Blue: Water

Yellow: Barren

Red: Urban

Table 11: Percentage of each class classified by MD

% of Vegetation Area	40.84
% of Water Area	9.04
% of Barren Area	11.50
% of Urban Area	38.96

Figure 18: Classification done by MD

Table 12: Accuracy and F1-Score of MD

Measurement Parameters (MD)	Values
Accuracy	0.851
F1-Score(Vegetation)	0.888
F1-Score(Water)	0.967
F1-Score(Barren)	0.818
F1-Score(Urban)	0.714

Classification by Ensemble Approach:

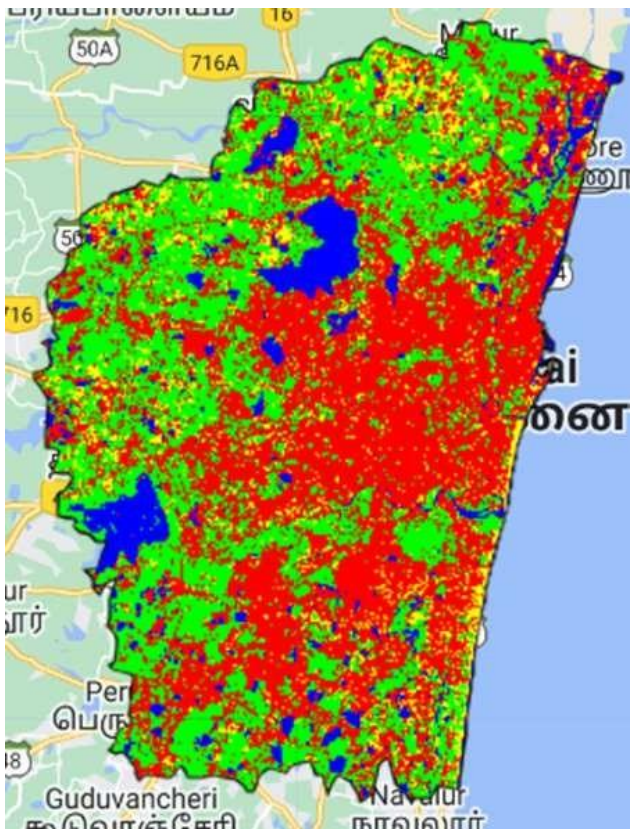


Figure 19: Classification done by Ensemble Learning

Green: Vegetation	
Blue: Water	
Yellow: Barren	
Red: Urban	
Table 13: Percentage of each class classified by Ensemble Learning	
% of Vegetation Area	29.906
% of Water Area	16.249
% of Barren Area	18.127
% of Urban Area	35.716

8. SUMMARY

In this study, we conducted land cover classification of Chennai using Sentinel-2A satellite imagery and various machine learning algorithms including Random Forest (RF), Classification and Regression Trees (CART), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Gradient Boosting (GB), and Multidimensional Scaling (MD). Additionally, we constructed an ensemble model by combining RF, CART, and GB classifiers to enhance classification accuracy.

Initially, we preprocessed the Sentinel-2A imagery by filtering based on date and cloud cover, clipping to the study area, and deriving spectral indices such as NDVI, DVI, and RVI. We labeled training data points for vegetation, water, barren land, and urban areas, and split the dataset into training and testing subsets.

Subsequently, we trained each classifier individually using the training data and classified the image. Moreover, we implemented an ensemble voting technique to merge the predictions of RF, CART, and GB classifiers. The performance of each classifier and the ensemble model was evaluated using confusion matrices and metrics like accuracy, precision, recall, and F1-score.

Our findings indicate variations in the performance of different classifiers for classifying various land cover types. RF, CART, and GB classifiers exhibited promising results, with the ensemble model demonstrating improved accuracy compared to individual classifiers. The inclusion of spectral indices enhanced the discrimination between land cover classes.

In conclusion, our study underscores the significance of selecting appropriate machine learning algorithms and preprocessing methods for accurate land cover classification. The outcomes offer valuable insights for land management, urban planning, and environmental monitoring in Chennai and similar regions. Future research could explore additional features, optimize model parameters, and investigate advanced machine learning techniques to further enhance classification accuracy and robustness.

References

- [1] Mahmoud, R.; Hassanin, M.; Al Feel, H.; Badry, R.M. Machine Learning-Based Land Use and Land Cover Mapping Using Multi-Spectral Satellite Imagery: A Case Study in Egypt. *Sustainability* 2023, 15, 9467. <https://doi.org/10.3390/su15129467>
- [2] Aryal, J., Sitaula, C. & Frery, A.C. Land use and land cover (LULC) performance modeling using machine learning algorithms: a case study of the city of Melbourne, Australia. *Sci Rep* 13, 13510 (2023). <https://doi.org/10.1038/s41598-023-40564-0>
- [3] Atef, I., Ahmed, W. & Abdel-Maguid, R.H. Modelling of land use land cover changes using machine learning and GIS techniques: a case study in El-Fayoum Governorate, Egypt. *Environ Monit Assess* 195, 637 (2023). <https://doi.org/10.1007/s10661-023-11224-7>
- [4] M, Arpitha and Ahmed, S and N, Harishnaika.
Land use and land cover classification using machine learning algorithms in google earth engine
- [5] Hamad, R. (2020). An assessment of artificial neural networks support vector machines and decision trees for land cover classification using sentinel-2A. *Data Sci.* 8, 459–464. doi: 10.12691/aees-8-6-18
- [6] Alshari, E. A., and Gawali, B. W. (2022b). Analysis of machine learning techniques for sentinel-2A satellite images. *J. Electr. Comput. Eng.* 2022:9092299. doi: 10.1155/2022/9092299
- [7] Loukika, K. N., Keesara, V. R., and Sridhar, V. (2021). Analysis of land use and land cover using machine learning algorithms on google earth engine for Munneru River Basin, India. *Sustainability* 13, 13758. doi: 10.3390/su132413758
- [8] Ming, D., Zhou, T., Wang, M., and Tan, T. (2016). Land cover classification using random forest with genetic algorithm-based parameter optimization. *J. Appl. Remote Sens.* 10, 35021. doi: 10.1117/1.JRS.10.035021
- [9] Matosak, B. M., Fonseca, L. M. G., Taquary, E. C., Maretto, R. V., Bendini, H. D. N., and Adami, M. (2022). Mapping deforestation in cerrado based on hybrid deep learning architecture and medium spatial resolution satellite time series. *Remote Sens.* 14, 209. doi: 10.3390/rs14010209
- [10] Sonobe, R., Yamaya, Y., Tani, H., Wang, X., Kobayashi, N., and Mochizuki, K. I. (2017). Mapping crop cover using multi-temporal Landsat 8 OLI imagery. *Int. J. Remote Sens.* 38, 4348–4361. doi: 10.1080/01431161.2017.1323286

- [11] Wang, X., Gao, X., Zhang, Y., Fei, X., Chen, Z., Wang, J., et al. (2019). Landcover classification of coastal wetlands using the RF algorithm for Worldview-2 and Landsat 8 images. *Remote Sens.* 11, 1927. doi: 10.3390/rs11161927
- [12] Sang, X., Guo, Q., Wu, X., Xie, T., He, C., Zang, J., et al. (2021). The effect of DEM on the land use/cover classification accuracy of landsat OLI images. *J. Ind. Soc. Remote Sens.* 5, 1–12. doi: 10.1007/s12524-021-01318-5

APPENDIX A – SAMPLE CODE

```
var Geo = ee.FeatureCollection('projects/ee-rithikragupathi/assets/CHENNAI')
Map.addLayer(Geo,{ },'Geo')

//Loading satellite
var image = ee.ImageCollection("COPERNICUS/S2_SR_HARMONIZED")
  .filterDate('2020-01-01','2020-01-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE',1))
  .filterBounds(Geo)
  .mean();

// Clip Sentinel-2 image to the extent of the shapefile
var clippedImage = image.clip(Geo);

//Visualizing Selected Area
var visParamsTrue = {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000, gamma: 1.4};
Map.addLayer(clippedImage, visParamsTrue, "Sentinel");

//NDVI
var ndvi =clippedImage.normalizedDifference(['B8','B4']).rename('NDVI');
print(ndvi)

//DVI
var dvi = clippedImage.select('B8').subtract(clippedImage.select('B4')).rename('DVI');
print(dvi)

//RVI
var rvi = clippedImage.select('B4').divide(clippedImage.select('B8')).rename('RVI');
print(rvi)
```

```

//Adding the new bands to exisiting ones
var combinedFeatures = clippedImage.addBands([ndvi, dvi, rvi]);
print(combinedFeatures)

// Labelling each class with their corresponding IDs for each point (LC for Land Cover, set
any label key you want but they should start from 0)
var labelled_vegetation = Vegetation.map(function(feature){return feature.set({'LC':0})});
var labelled_water = WaterBody.map(function(feature){return feature.set({'LC':1})});
var labelled_barren = Barren.map(function(feature){return feature.set({'LC':2})});
var labelled_urban = Urban.map(function(feature){return feature.set({'LC':3})});

// Merging these data points and calling them GCPs
var GCPs =
labelled_vegetation.merge(labelled_water).merge(labelled_barren).merge(labelled_urban);
// Lets print the GCPs
print(GCPs,'GCPs Merged');

var label = 'LC';
var bands = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'NDVI', 'RVI', 'DVI'];
var input = combinedFeatures.select(bands);

// Now we will sample the spectral values of each band for these points (GCPs);
var sampled_GCPs = input.sampleRegions({
  collection:GCPs,
  properties:[label],
  scale:10
});
print(sampled_GCPs);

// Split the data into training and testing sets (e.g., 80% training, 20% testing)
var split = 0.8; // Ratio of training to testing data

```

```
var trainSet = sampled_GCPs.randomColumn('random').filter(ee.Filter.lt('random', split));  
print(trainSet)
```

```
var testSet = sampled_GCPs.randomColumn('random').filter(ee.Filter.gte('random', split));  
print(testSet)
```

```
//START OF RANDOM FOREST
```

```
print("RANDOM FOREST START")
```

```
// Train a Random Forest classifier using the training data
```

```
var classifierRF = ee.Classifier.smileRandomForest(10).train(trainSet, label, bands);
```

```
// Classify the clipped image using the trained classifier
```

```
var classifiedRF = input.classify(classifierRF);
```

```
var landcoverPaletteRF = [
```

```
  '#00FF00', // Green for Vegetation
```

```
  '#0000FF', // Blue for Water
```

```
  '#FFFF00', // Yellow for Barren
```

```
  '#FF0000', // Red for Urban
```

```
];
```

```
Map.addLayer(classifiedRF, {palette: landcoverPaletteRF, min: 0, max:3 }, 'classification  
RF');
```

```
var legendRF = ui.Panel({
```

```
  style: {
```

```
    position: 'bottom-left',
```

```
    padding: '8px 15px'
```

```
  },
```

```
  widgets: [
```

```

    ui.Label('Random Forest Classification', {fontWeight: 'bold', fontSize: '16px'}),
    ui.Label('Green: Vegetation', {color: 'green'}),
    ui.Label('Blue: Water', {color: 'blue'}),
    ui.Label('Yellow: Barren', {color: 'yellow'}),
    ui.Label('Red: Urban', {color: 'red'})
  ]
});

// Add the legend to the map
Map.add(legendRF);

var confusionMatrixRF = ee.ConfusionMatrix(testSet.classify(classifierRF)
  .errorMatrix({
    actual: 'LC',
    predicted: 'classification'
  }));

print('Confusion matrix of Random Forest:', confusionMatrixRF);
print('Overall Accuracy of Random Forest:', confusionMatrixRF.accuracy());
print('Overall Accuracy of Random Forest:', confusionMatrixRF.fscore());

var totalCountImage = classifiedRF.reduceRegion({
  reducer: ee.Reducer.count(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

// Get the total count of pixels
var totalCountPixels = ee.Number(totalCountImage.get('classification')).toInt();

```

```

// Calculate the count of pixels classified as urban
var VegetationCountImage = classifiedRF.eq(0).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var WaterCountImage = classifiedRF.eq(1).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var BarrenCountImage = classifiedRF.eq(2).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var urbanCountImage = classifiedRF.eq(3).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

// Get the count of pixels classified as Vegetation, Water, Barren, Urban
var VegetationCountPixels =

```

```

ee.Number(VegetationCountImage.get('classification')).toInt();
var VegetationPercentage =
VegetationCountPixels.divide(totalCountPixels).multiply(100);

var WaterCountPixels = ee.Number(WaterCountImage.get('classification')).toInt();
var WaterPercentage = WaterCountPixels.divide(totalCountPixels).multiply(100);

var BarrenCountPixels = ee.Number(BarrenCountImage.get('classification')).toInt();
var BarrenPercentage = BarrenCountPixels.divide(totalCountPixels).multiply(100);

var urbanCountPixels = ee.Number(urbanCountImage.get('classification')).toInt();
var urbanPercentage = urbanCountPixels.divide(totalCountPixels).multiply(100);

print('Percentage of Vegetation Area:', VegetationPercentage);
print('Percentage of Water Area:', WaterPercentage);
print('Percentage of Barren Area:', BarrenPercentage);
print('Percentage of Urban Area:', urbanPercentage);
print("RANDOM FOREST ALGORITHM END");

//START OF CART ALGORITHM
print("CART ALGORITHM START")
// Train a Random Forest classifier using the training data
var classifierCART = ee.Classifier.smileCart().train(trainSet, label, bands);

// Classify the clipped image using the trained classifier
var classifiedCART = input.classify(classifierCART);

var landcoverPaletteCART = [
  '#00FF00', // Green for Vegetation
  '#0000FF', // Blue for Water

```

```

'#FFFF00', // Yellow for Barren
'#FF0000', // Red for Urban
];

Map.addLayer(classifiedCART, {palette: landcoverPaletteCART, min: 0, max:3 },
'classification CART');

var legendCART = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  },
  widgets: [
    ui.Label('CART Classification', {fontWeight: 'bold', fontSize: '16px'}),
    ui.Label('Green: Vegetation', {color: 'green'}),
    ui.Label('Blue: Water', {color: 'blue'}),
    ui.Label('Yellow: Barren', {color: 'yellow'}),
    ui.Label('Red: Urban', {color: 'red'})
  ]
});

// Add the legend to the map
Map.add(legendCART);

var confusionMatrixCART = ee.ConfusionMatrix(testSet.classify(classifierCART)
  .errorMatrix({
    actual: 'LC',
    predicted: 'classification'
  }));

print('Confusion matrix of CART:', confusionMatrixCART);

```

```

print('Overall Accuracy of CART:', confusionMatrixCART.accuracy());
print('F1-Score of CART:', confusionMatrixCART.fscore());

var totalCountImage = classifiedCART.reduceRegion({
  reducer: ee.Reducer.count(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

// Get the total count of pixels
var totalCountPixels = ee.Number(totalCountImage.get('classification')).toInt();

// Calculate the count of pixels classified as urban
var VegetationCountImage = classifiedCART.eq(0).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var WaterCountImage = classifiedCART.eq(1).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var BarrenCountImage = classifiedCART.eq(2).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),

```



```

    scale: 10,
    maxPixels: 1e9
  });

var urbanCountImage = classifiedCART.eq(3).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

// Get the count of pixels classified as urban
var VegetationCountPixels =
ee.Number(VegetationCountImage.get('classification')).toInt();
var VegetationPercentage =
VegetationCountPixels.divide(totalCountPixels).multiply(100);

var WaterCountPixels = ee.Number(WaterCountImage.get('classification')).toInt();
var WaterPercentage = WaterCountPixels.divide(totalCountPixels).multiply(100);

var BarrenCountPixels = ee.Number(BarrenCountImage.get('classification')).toInt();
var BarrenPercentage = BarrenCountPixels.divide(totalCountPixels).multiply(100);

var urbanCountPixels = ee.Number(urbanCountImage.get('classification')).toInt();
var urbanPercentage = urbanCountPixels.divide(totalCountPixels).multiply(100);

print('Percentage of Vegetation Area:', VegetationPercentage);
print('Percentage of Water Area:', WaterPercentage);
print('Percentage of Barren Area:', BarrenPercentage);
print('Percentage of Urban Area:', urbanPercentage);
print("CART ALGORITHM END");

```

```

print("SVM ALGORITHM START")

// Train a SVM classifier using the training data
var classifierSVM = ee.Classifier.libsvm().train(trainSet, label, bands);

// Classify the clipped image using the trained classifier
var classifiedSVM = input.classify(classifierSVM);

var landcoverPaletteSVM = [
  '#00FF00', // Green for Vegetation
  '#0000FF', // Blue for Water
  '#FFFF00', // Yellow for Barren
  '#FF0000', // Red for Urban
];

Map.addLayer(classifiedSVM, {palette: landcoverPaletteSVM, min: 0, max:3 },
'classification SVM');

var legendSVM = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  },
  widgets: [
    ui.Label('SVM Classification', {fontWeight: 'bold', fontSize: '16px'}),
    ui.Label('Green: Vegetation', {color: 'green'}),
    ui.Label('Blue: Water', {color: 'blue'}),
    ui.Label('Yellow: Barren', {color: 'yellow'}),
    ui.Label('Red: Urban', {color: 'red'})
  ]
});

```

```

    ]
  });

  // Add the legend to the map
  Map.add(legendSVM);

  var confusionMatrixSVM = ee.ConfusionMatrix(testSet.classify(classifierSVM)
    .errorMatrix({
      actual: 'LC',
      predicted: 'classification'
    }));

  print('Confusion matrix of SVM:', confusionMatrixSVM);
  print('Overall Accuracy of SVM:', confusionMatrixSVM.accuracy());
  print('F1-Score of SVM:', confusionMatrixSVM.fscore());

  var totalCountImage = classifiedSVM.reduceRegion({
    reducer: ee.Reducer.count(),
    geometry: Geo.geometry(),
    scale: 10,
    maxPixels: 1e9
  });

  // Get the total count of pixels
  var totalCountPixels = ee.Number(totalCountImage.get('classification')).toInt();

  // Calculate the count of pixels classified as urban
  var VegetationCountImage = classifiedSVM.eq(0).reduceRegion({
    reducer: ee.Reducer.sum(),
    geometry: Geo.geometry(),
    scale: 10,

```

```

    maxPixels: 1e9
  });

var WaterCountImage = classifiedSVM.eq(1).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var BarrenCountImage = classifiedSVM.eq(2).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var urbanCountImage = classifiedSVM.eq(3).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

// Get the count of pixels classified as urban
var VegetationCountPixels =
ee.Number(VegetationCountImage.get('classification')).toInt();
var VegetationPercentage =
VegetationCountPixels.divide(totalCountPixels).multiply(100);

var WaterCountPixels = ee.Number(WaterCountImage.get('classification')).toInt();

```

```

var WaterPercentage = WaterCountPixels.divide(totalCountPixels).multiply(100);

var BarrenCountPixels = ee.Number(BarrenCountImage.get('classification')).toInt();
var BarrenPercentage = BarrenCountPixels.divide(totalCountPixels).multiply(100);

var urbanCountPixels = ee.Number(urbanCountImage.get('classification')).toInt();
var urbanPercentage = urbanCountPixels.divide(totalCountPixels).multiply(100);

print('Percentage of Vegetation Area:', VegetationPercentage);
print('Percentage of Water Area:', WaterPercentage);
print('Percentage of Barren Area:', BarrenPercentage);
print('Percentage of Urban Area:', urbanPercentage);
print("SVM ALGORITHM END");

print("GRADIENT BOOSTING CLASSIFIER START");
// Train a Gradient Boost classifier using the training data
var classifierGB = ee.Classifier.smileGradientTreeBoost(10).train(trainSet, label, bands);

// Classify the clipped image using the trained classifier
var classifiedGB = input.classify(classifierGB);

var landcoverPaletteGB = [
  '#00FF00', // Green for Vegetation
  '#0000FF', // Blue for Water
  '#FFFF00', // Yellow for Barren
  '#FF0000', // Red for Urban
];

Map.addLayer(classifiedGB, {palette: landcoverPaletteGB, min: 0, max:3 }, 'classification
GB');

```

```

var legendGB = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  },
  widgets: [
    ui.Label('Gradient Boosting Classification', {fontWeight: 'bold', fontSize: '16px'}),
    ui.Label('Green: Vegetation', {color: 'green'}),
    ui.Label('Blue: Water', {color: 'blue'}),
    ui.Label('Yellow: Barren', {color: 'yellow'}),
    ui.Label('Red: Urban', {color: 'red'})
  ]
});

// Add the legend to the map
Map.add(legendGB);

var confusionMatrixGB = ee.ConfusionMatrix(testSet.classify(classifierGB)
  .errorMatrix({
    actual: 'LC',
    predicted: 'classification'
  }));

print('Confusion matrix of Gradient Boosting:', confusionMatrixGB);
print('Overall Accuracy of Gradient Boosting:', confusionMatrixGB.accuracy());
print('F1-Score of Gradient Boosting:', confusionMatrixGB.fscore());

var totalCountImage = classifiedGB.reduceRegion({
  reducer: ee.Reducer.count(),
  geometry: Geo.geometry(),

```

```

    scale: 10,
    maxPixels: 1e9
  });

// Get the total count of pixels
var totalCountPixels = ee.Number(totalCountImage.get('classification')).toInt();

// Calculate the count of pixels classified as urban
var VegetationCountImage = classifiedGB.eq(0).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var WaterCountImage = classifiedGB.eq(1).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var BarrenCountImage = classifiedGB.eq(2).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var urbanCountImage = classifiedGB.eq(3).reduceRegion({
  reducer: ee.Reducer.sum(),

```

```

    geometry: Geo.geometry(),
    scale: 10,
    maxPixels: 1e9
  });

// Get the count of pixels classified as urban
var VegetationCountPixels =
ee.Number(VegetationCountImage.get('classification')).toInt();
var VegetationPercentage =
VegetationCountPixels.divide(totalCountPixels).multiply(100);

var WaterCountPixels = ee.Number(WaterCountImage.get('classification')).toInt();
var WaterPercentage = WaterCountPixels.divide(totalCountPixels).multiply(100);

var BarrenCountPixels = ee.Number(BarrenCountImage.get('classification')).toInt();
var BarrenPercentage = BarrenCountPixels.divide(totalCountPixels).multiply(100);

var urbanCountPixels = ee.Number(urbanCountImage.get('classification')).toInt();
var urbanPercentage = urbanCountPixels.divide(totalCountPixels).multiply(100);

print('Percentage of Vegetation Area:', VegetationPercentage);
print('Percentage of Water Area:', WaterPercentage);
print('Percentage of Barren Area:', BarrenPercentage);
print('Percentage of Urban Area:', urbanPercentage);
print("GB ALGORITHM END");

print("KNN ALGORITHM START");
// Train a k-Nearest Neighbors (kNN) classifier using the training data
var classifierKNN = ee.Classifier.smileKNN().train(trainSet, label, bands);

```



```

// Classify the clipped image using the trained classifier
var classifiedKNN = input.classify(classifierKNN);

var landcoverPaletteKNN = [
    '#00FF00', // Green for Vegetation
    '#0000FF', // Blue for Water
    '#FFFF00', // Yellow for Barren
    '#FF0000', // Red for Urban
];

Map.addLayer(classifiedKNN, {palette: landcoverPaletteKNN, min: 0, max:3 },
'classification kNN');

var legendKNN = ui.Panel({
    style: {
        position: 'bottom-left',
        padding: '8px 15px'
    },
    widgets: [
        ui.Label('k-Nearest Neighbors (kNN) Classification', {fontWeight: 'bold', fontSize:
'16px'}),
        ui.Label('Green: Vegetation', {color: 'green'}),
        ui.Label('Blue: Water', {color: 'blue'}),
        ui.Label('Yellow: Barren', {color: 'yellow'}),
        ui.Label('Red: Urban', {color: 'red'})
    ]
});

// Add the legend to the map
Map.add(legendKNN);

```

```

var confusionMatrixKNN = ee.ConfusionMatrix(testSet.classify(classifierKNN)
    .errorMatrix({
        actual: 'LC',
        predicted: 'classification'
    }));

```

```

print('Confusion matrix of k-Nearest Neighbors (kNN):', confusionMatrixKNN);
print('Overall Accuracy of k-Nearest Neighbors (kNN):',
confusionMatrixKNN.accuracy());
print('F1-Score of k-Nearest Neighbors (kNN):', confusionMatrixKNN.fscore());

```

```

var totalCountImage = classifiedKNN.reduceRegion({
    reducer: ee.Reducer.count(),
    geometry: Geo.geometry(),
    scale: 10,
    maxPixels: 1e9
});

```

```

// Get the total count of pixels

```

```

var totalCountPixels = ee.Number(totalCountImage.get('classification')).toInt();

```

```

// Calculate the count of pixels classified as urban

```

```

var VegetationCountImage = classifiedKNN.eq(0).reduceRegion({
    reducer: ee.Reducer.sum(),
    geometry: Geo.geometry(),
    scale: 10,
    maxPixels: 1e9
});

```

```

var WaterCountImage = classifiedKNN.eq(1).reduceRegion({
    reducer: ee.Reducer.sum(),

```

```
geometry: Geo.geometry(),
scale: 10,
maxPixels: 1e9
});
```

```
var BarrenCountImage = classifiedKNN.eq(2).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});
```

```
var urbanCountImage = classifiedKNN.eq(3).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});
```

```
// Get the count of pixels classified as urban
var VegetationCountPixels =
ee.Number(VegetationCountImage.get('classification')).toInt();
var VegetationPercentage =
VegetationCountPixels.divide(totalCountPixels).multiply(100);
```

```
var WaterCountPixels = ee.Number(WaterCountImage.get('classification')).toInt();
var WaterPercentage = WaterCountPixels.divide(totalCountPixels).multiply(100);
```

```
var BarrenCountPixels = ee.Number(BarrenCountImage.get('classification')).toInt();
var BarrenPercentage = BarrenCountPixels.divide(totalCountPixels).multiply(100);
```

```

var urbanCountPixels = ee.Number(urbanCountImage.get('classification')).toInt();
var urbanPercentage = urbanCountPixels.divide(totalCountPixels).multiply(100);

print('Percentage of Vegetation Area:', VegetationPercentage);
print('Percentage of Water Area:', WaterPercentage);
print('Percentage of Barren Area:', BarrenPercentage);
print('Percentage of Urban Area:', urbanPercentage);
print("KNN ALGORITHM END");

print("Minimum Distance Classifier START");
// Train a Minimum Distance Classifier using the training data
var classifierMinDist = ee.Classifier.minimumDistance().train(trainSet, label, bands);

// Classify the clipped image using the trained classifier
var classifiedMinDist = input.classify(classifierMinDist);

var landcoverPaletteMD = [
  '#00FF00', // Green for Vegetation
  '#0000FF', // Blue for Water
  '#FFFF00', // Yellow for Barren
  '#FF0000', // Red for Urban
];

Map.addLayer(classifiedMinDist, {palette: landcoverPaletteMD, min: 0, max:3 },
'classification MD');

var legendMinDist = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

```

```

    },
    widgets: [
        ui.Label('Minimum Distance Classification', {fontWeight: 'bold', fontSize: '16px'}),
        ui.Label('Green: Vegetation', {color: 'green'}),
        ui.Label('Blue: Water', {color: 'blue'}),
        ui.Label('Yellow: Barren', {color: 'yellow'}),
        ui.Label('Red: Urban', {color: 'red'})
    ]
});

// Add the legend to the map
Map.add(legendMinDist);

var confusionMatrixMinDist = ee.ConfusionMatrix(testSet.classify(classifierMinDist)
    .errorMatrix({
        actual: 'LC',
        predicted: 'classification'
    }));

print('Confusion matrix of Minimum Distance Classifier:', confusionMatrixMinDist);
print('Overall Accuracy of Minimum Distance Classifier:',
confusionMatrixMinDist.accuracy());
print('F1-Score of Minimum Distance Classifier:', confusionMatrixMinDist.fscore());

var totalCountImage = classifiedMinDist.reduceRegion({
    reducer: ee.Reducer.count(),
    geometry: Geo.geometry(),
    scale: 10,
    maxPixels: 1e9
});

```

```

// Get the total count of pixels
var totalCountPixels = ee.Number(totalCountImage.get('classification')).toInt();

// Calculate the count of pixels classified as urban
var VegetationCountImage = classifiedMinDist.eq(0).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var WaterCountImage = classifiedMinDist.eq(1).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var BarrenCountImage = classifiedMinDist.eq(2).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

var urbanCountImage = classifiedMinDist.eq(3).reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: Geo.geometry(),
  scale: 10,
  maxPixels: 1e9
});

```

```

// Get the count of pixels classified as urban
var VegetationCountPixels =
ee.Number(VegetationCountImage.get('classification')).toInt();
var VegetationPercentage =
VegetationCountPixels.divide(totalCountPixels).multiply(100);

var WaterCountPixels = ee.Number(WaterCountImage.get('classification')).toInt();
var WaterPercentage = WaterCountPixels.divide(totalCountPixels).multiply(100);

var BarrenCountPixels = ee.Number(BarrenCountImage.get('classification')).toInt();
var BarrenPercentage = BarrenCountPixels.divide(totalCountPixels).multiply(100);

var urbanCountPixels = ee.Number(urbanCountImage.get('classification')).toInt();
var urbanPercentage = urbanCountPixels.divide(totalCountPixels).multiply(100);

print('Percentage of Vegetation Area:', VegetationPercentage);
print('Percentage of Water Area:', WaterPercentage);
print('Percentage of Barren Area:', BarrenPercentage);
print('Percentage of Urban Area:', urbanPercentage);
print("MINIMUM DISTANCE ALGORITHM END");

```