AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH
Faculty of Science and Technology

Report Cover Sheet

Report Title:	**Landslide Prediction Using Machine Learning Models: A Comparative Study of Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine**		
Report No:	01	Date of Submission:	23 September 2024
Course Title:	MACHINE LEARNING		
Course Code:	CSC4232	Section:	C
Semester:	Summer 2023-24	Course Teacher:	Prof. Dr. Md. Asraf Ali

Group Name/No.:

No	Name	ID	Program	Signature
1	NOSHIN FARZANA	21-44647-1	BSc [CSE]	
2	AVIJIT SAHA ANTO	21-44630-1	BSc [CSE]	
3	SHAKIL KHAN	21-45118-2	BSc [CSE]	

Faculty use only

FACULTY COMMENTS	**Marks Obtained**	
	Total Marks	

Landslide Prediction Using Machine Learning Models: A Comparative Study of Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine

Noshin Farzana, Avijit Saha Anto, Shakil Khan

1. Introduction

One of nature's most destructive hazards, landslides are frequently caused by environmental factors like heavy rains, seismic activity, unique features of the terrain, and human activity. They can result in serious fatalities, destruction of infrastructure, and financial losses, especially in hilly and mountainous areas. Because landslides are unpredictable, governments and disaster management organizations around the world are extremely concerned about them.

Recent advances in machine learning (ML) provide up new possibilities for constructing automated and efficient prediction models. Machine learning approaches can evaluate enormous volumes of environmental data, discover trends, and predict landslides with great precision.

1.1. Problem Statement

Conventional techniques for predicting landslides depend on geotechnical analyses, physical models, and geological surveys, all of which can take a significant amount of time, money, and experience. Although helpful, these methods might not be as accurate or efficient, particularly when working with large amounts of real-time data. A promising substitute has been made possible by the development of machine learning (ML), which enables the integration of multiple data kinds and offers more precise and accurate prediction models. Several machine learning models, including Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine (SVM), have shown promise in this area.

Managing Data Imbalance is a major challenge in using machine learning (ML) models for landslide prediction. Datasets are skewed because landslide events are uncommon in comparison to non-landslide occurrences. The overall effectiveness of the model in high-risk scenarios can be decreased by this imbalance, which can lead to models performing well in predicting non-landslide events but struggling to identify actual landslides.

The Selection of Appropriate Features, such as rainfall, slope, soil type, and seismic activity, that affect landslides presents another difficulty. There is no one-size-fits-all model because these factors differ greatly between regions. Choosing the appropriate feature set and performing efficient data preprocessing are essential steps in developing a model that performs well in a variety of geographical and environmental contexts.

Addressing these challenges is essential to improving the accuracy, efficiency, and real-world applicability of machine learning-based landslide prediction models.

1.2. Objective

This paper compares the performance of four machine learning models for predicting landslides: Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine. We evaluate their accuracy, precision, recall, F1 score, and computational efficiency. Furthermore, we hope to determine which model produces the most reliable results for practical deployment in landslide-prone areas.

2. Literature Review

Pham et al. applied Random Forest (RF), Alternating Decision Tree (ADT), and Logistic Model Tree (LMT) to model landslide susceptibility in Vietnam. The study compared these models and found RF as the most accurate with an AUC of 0.839, outperforming ADT and LMT. **[1]**

Dou et al. used Support Vector Machines (SVM) with ensemble methods (bagging, boosting, stacking) to predict landslides in Japan. Boosting-based SVM had the highest predictive accuracy with an AUC of 0.91. This study emphasized the importance of rainfall as a critical factor in susceptibility. **[2]**

This study, by Pham et al., compared the effectiveness of SVM and Bayesian-based algorithms for landslide susceptibility. SVM generally provided higher accuracy due to its ability to handle nonlinear patterns better, though Bayesian models are advantageous in dealing with uncertainty and small datasets. **[3]**

Ma et al., 2023 investigated a Case Study of Yinghu Lake Basin, shows how landslide data integrity, machine learning (ML) models, and the selection of non-landslide samples impact landslide susceptibility prediction (LSP). By comparing random forest (RF), Artificial Neural Networks (ANN), and K-means clustering, the research concludes that supervised machine learning models, especially RF, outperform unsupervised ones like K-means in predicting landslide zones. **[4]**

Ballabio et al., applied Support Vector Machines (SVM) to landslide susceptibility mapping, with a case study in the Staffora River Basin, Italy. The performance of SVM is compared to Logistic Regression and Naive Bayes classifiers. The authors found that SVM outperformed the other models by identifying the most susceptible landslide areas with a higher prediction rate. **[5]**

Song et al., 2023 addressed the issue of imbalanced datasets in landslide susceptibility prediction. Using Logistic Regression, Random Forest (RF), and LightGBM, the authors compared different sampling strategies, such as oversampling (SMOTE) and undersampling, to balance the dataset. This paper makes a substantial contribution to dealing with imbalanced data problems, offering practical solutions like oversampling and under-sampling techniques. **[6]**

Wang et al., explored machine learning techniques for identifying landslides, utilizing models like Decision Trees and Neural Networks. The authors highlighted the advantages of machine learning methods in processing large datasets and extracting nonlinear patterns that may be missed by traditional statistical approaches. By integrating geographical information system (GIS) data with machine learning algorithms, this research provides significant insights into spatial landslide prediction. **[7]**

Catani et al., introduces a Random Forests (RF) model to estimate landslide susceptibility in the Arno River basin in central Italy. The focus is on exploring the sensitivity and scalability of the RF model, particularly on varying input data resolution and model parameters. The study examined how the resolution of the mapping unit and the number of predictor variables affect model performance, emphasizing the need for a careful balance between the model's complexity and the scale of analysis. The paper found that Random Forests produce robust results, particularly when a variety of input data types (e.g., lithology, land use, geomorphology) are used. **[8]**

Lee stated the use of Logistic Regression (LR) models for landslide susceptibility mapping in Penang, Malaysia. It combined GIS and remote sensing data to build a spatial database, utilizing factors such as topography, geology, and land use to predict landslide-prone areas. The paper emphasized the importance of using a comprehensive spatial database for model accuracy, including factors like slope, curvature, drainage, lithology, and vegetation index. The validation of the LR model shows a high level of accuracy when applied to real-world landslide occurrences. **[9]**

3. Methodology

The methodology of this study is to predict landslides by comparing four machine learning models: Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine. These models were chosen for their varying abilities to handle complex, nonlinear relationships in data, making them appropriate for analyzing geospatial features related to landslide susceptibility. We apply these algorithms to see how well they perform in terms of prediction accuracy and model efficiency, given the unique characteristics of landslide data. To improve the study's robustness, we used a comprehensive dataset from Kaggle that included a variety of topographic and environmental features that contribute to landslides. Each model was trained and evaluated using standardized techniques such as data preprocessing, normalization, and feature extraction.

3.1. Data Collection Procedure

The dataset used in this research was sourced from Kaggle, titled "Landslide Data Large Numeric"[Kaggle Dataset Link: https://www.kaggle.com/datasets/vigneshwarchandran/landslide-data-large-numeric]. It contains a variety of environmental and topographic features such as Landslide, Aspect, Curvature, Earthquake, Elevation, Flow, Lithology, NDVI, NDWI, Plan, Precipitation, Profile, Slope, Temperature, Humidity, Rain, Moisture, Pressure that are influential in landslide prediction, including slope angle, soil type, and rainfall intensity. The dataset includes both landslide-prone and non-landslide regions, making it suitable for training supervised machine learning models.

To ensure the integrity and quality of the dataset, the data was inspected for missing values. The collection procedure was designed to maintain high-quality inputs for machine learning models, ensuring that all essential variables are represented adequately. The dataset was divided into training (80%) and testing sets (20%) to evaluate model performance and generalization.

3.2. Data Validation Procedure

Data validation is an important step in ensuring that the data used for landslide prediction is correct, complete, and suitable for training machine learning models. For this study, the validation process was carried out systematically in multiple stages, with the goal of preparing the dataset for optimal model performance. To ensure data quality and reliability, the procedure included cleaning, preprocessing, sampling, cross-validation, evaluation metrics, and model comparison.

1. Data cleaning: Data cleaning was the first step in the validation process, ensuring that any inconsistencies or errors were removed from the dataset. The following tasks were performed during data cleaning:

- **Managing Missing Data:** Missing values were identified and treated with appropriate imputation methods.

2. Data Visualization: Precipitation, Slope, and Elevation are distributed in the dataset, with different colors representing whether a landslide occurred or not. Correlation heatmap was generated to visualize the pairwise correlations between the features in the dataset.

3. Preprocessing: Effective preprocessing was used to convert the raw data into a structured format suitable for machine learning.

4. Sampling: To ensure that the dataset was representative and free of bias, the sampling procedure was designed to keep a balance of landslide-prone and non-landslide-prone data points.

- **Data Splitting:** The data was divided into two parts: 80% for training and 20% for testing, ensuring that the models were trained on a significant portion of the data but tested on unseen data to assess generalization.

5. Cross-validation: Cross-validation was used to ensure that the machine learning models were robust and did not overfit.

6. Evaluation Metrics: Throughout the validation process, several evaluation metrics were used to assess the performance of the models:

- **Accuracy:** The proportion of correctly predicted instances among all instances in the dataset.

- **Precision:** The model's ability to accurately predict positive instances (e.g., landslide-prone areas).

- **Recall (Sensitivity):** The model's ability to identify true positive instances.

- **F1-Score:** The harmonic mean of precision and recall, which strikes a balance between the two measures.

7. Model Comparison: The final step of the validation process was to compare the performance of four machine learning models: Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine. Following training and testing, the models were evaluated using the previously mentioned metrics. The models were compared to determine which achieved the best balance of accuracy and computational efficiency. Graphs and charts were used to visualize the comparison, allowing for better interpretation and understanding of the findings.

3.3. Data Preprocessing Technique

To prepare the raw data for machine learning, a series of preprocessing steps were performed. We used Random Under Sampling to address class imbalance in the target variable—landslide occurrence.

3.4. Feature Selection Technique

Landslide prediction relies heavily on feature extraction, which identifies the most influential variables in the dataset. Several environmental factors were identified as important features in this study, including:

Slope Angle: Slope angle is an important predictor of landslides because steep slopes are more prone to them.

Precipitation: Raises the moisture content of the soil, which may cause landslides.

Elevation: Elevation influences the relationship between slope and precipitation as well as the way that climate variables like snow accumulation affect the likelihood of landslides.

3.5. Classification Algorithm

Decision Trees: The DT belongs to the supervised learning algorithm family. Unlike other supervised learning algorithms, the decision tree approach may also be used for regression and classification. Because of its resilience to noise, tolerance for missing information, management of irrelevant redundant predictive attribute values, low processing cost, interpretability, and robust predictors, the DT is one of the most popular and widely used machine learning algorithms.

Gradient Boosting: Gradient Boosting is an ensemble technique that creates models sequentially, optimizing residuals at each stage.

Random Forest: RF is an ensemble learning technique that builds several decision trees during training and produces the mean prediction (regression) or the mode of the classes (classification) for each individual tree. The reason RF is used is because of its capacity to manage high-dimensional data, keep accuracy when features are noisy, and lessen overfitting. Because it combines the predictions from several decision trees, it is resistant to overfitting.
Support Vector Machine (SVM): SVM is a technique for supervised machine learning that is used to solve classification and regression problems. In order to categorize the data into different classes, it generates a hyperplane or collection of hyperplanes in a high dimensional space.

3.6. Data Analysis Technique

Various performance evaluation metrics were used to assess the machine learning models' effectiveness in predicting landslides. A confusion matrix, which provides a detailed breakdown of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), served as the foundation for this binary classification task. The confusion matrix evaluates how well the models predict landslide-prone areas versus non-landslide-prone areas.

In addition to the confusion matrix, the following key metrics were used to evaluate the model performance:

Accuracy: Accuracy is the proportion of correct predictions (both landslides and non-landslides) to the total number of predictions. It is a simple but important metric for determining the overall correctness of the model. It's calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Precision measures the proportion of correctly predicted landslide-prone areas (TP) versus all instances predicted as landslide-prone (TP+FP). A high precision value indicates a low number of false positives, which is critical in landslide prediction to avoid overestimating risk areas. Precision is calculated as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

F1 Score: The F1 score balances precision and recall, providing a single measure of model accuracy by taking the harmonic mean of these two metrics. It is especially useful when there is an unequal class distribution or when both false positives and false negatives have serious consequences. The F1 score is computed as follows:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

AUC-ROC (Area Under Curve - Receiver Operating Characteristic): The AUC-ROC metric measures the model's ability to distinguish between two classes (landslide-prone and non-landslide-prone areas). The ROC curve compares the true positive rate (recall) to the false positive rate (FPR), with the AUC indicating the likelihood that the model ranks a randomly selected positive instance higher than a negative one. A higher AUC value (closer to one) indicates improved model performance.

$$\text{AUC-ROC} = \int_0^1 \text{ROC Curve}$$

Confusion Matrix: The confusion matrix offers a detailed view of the model's classification performance by showing the counts of:

- **True Positives (TP)**: Correctly predicted landslide-prone areas.
- **True Negatives (TN)**: Correctly predicted non-landslide-prone areas.
- **False Positives (FP)**: Areas predicted as landslide-prone that were not.
- **False Negatives (FN)**: Areas that were landslide-prone but were not predicted as such.

3.7. Block Diagram of Proposed Model

A block diagram is a drawing illustration of a system whose major parts or components are represented by blocks. These blocks are joined by lines to display the relationship between subsequent blocks. While flow diagram is used to show the flow of information within a system visually.

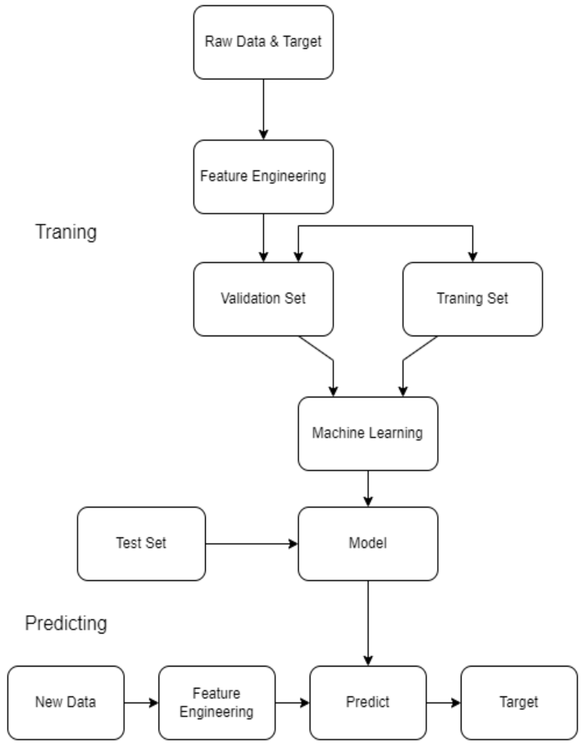
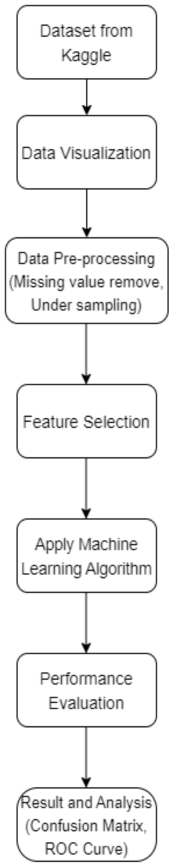


Fig: Block Diagram of Proposed Methodology



Dataset from Kaggle
Data Visualization
Data Pre-processing (Missing value remove, Under sampling)
Feature Selection
Apply Machine Learning Algorithm
Performance Evaluation
Result and Analysis (Confusion Matrix, ROC Curve)

Fig: Workflow Diagram of Proposed Model

3.8. Experimental Setup

The experiments were carried out using Python's Scikit-learn library. All models were trained and evaluated with an 80:20 train-test split and to ensure generalizability. We also used Google Colab for computational efficiency and resource management.

4. Result and Discussion

In comparison to existing solutions, such as empirical and geotechnical methods, machine learning models significantly improved prediction accuracy. The results of the machine learning classification were analyzed by comparing the performance of the Algorithms as discussed below:

4.1. Results Analysis by comparing existing solution

	Accuracy	Precision	Recall	F1-score
Decision Tree	0.9846449136276392	0.9809670781893004	0.981977342945417	0.9814719505918682
Gradient Boosting	0.9971711456859972	0.984970477724101	0.95822454308094	0.9714134462678665
Random Forest	0.9899765408402644	0.9846547314578005	0.9912461380020597	0.9879394405953296
Support Vector Machine	0.9908296011942845	0.9861751152073732	0.9917610710607621	0.9889602053915275

From the provided table, we can analyze the performance of four machine learning models: Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine. The metrics compared include Accuracy, Precision, Recall, and F1-score.

Summary of Results:

- **Accuracy**:
  - **Gradient Boosting** achieved the highest accuracy (0.9971), followed by **Support Vector Machine (SVM)** with 0.9908, **Random Forest** with 0.9899, and **Decision Tree** with 0.9846.
- **Precision**:
  - The highest precision was observed in **SVM** (0.9861), closely followed by **Random Forest** (0.9846) and **Gradient Boosting** (0.9849). **Decision Tree** had a slightly lower precision (0.9809).

➢ **Recall**:

- **SVM** also led in recall (0.9917), with **Random Forest** (0.9912) not far behind. **Decision Tree** (0.9819) and **Gradient Boosting** (0.9582) performed slightly lower in terms of recall.

➢ **F1-score**:

- The **SVM** again achieved the highest F1-score (0.9889), indicating a strong balance between precision and recall. **Random Forest** had a similarly high F1-score (0.9879), followed by **Decision Tree** (0.9814) and **Gradient Boosting** (0.9714).

Key Insights:

- **Gradient Boosting** exhibits the highest accuracy, making it the best model in terms of correctly classifying instances overall.

- **SVM** and **Random Forest** demonstrated strong performance across all metrics, with the highest precision, recall, and F1-scores, which indicates better generalization and handling of class imbalances.

- **Decision Tree**, while not the top performer, still shows competitive results, particularly in recall and F1-score, but is outperformed by the ensemble models (Gradient Boosting, Random Forest) and SVM.

In conclusion, **SVM** and **Random Forest** stand out as the most balanced models in terms of precision, recall, and F1-score, while **Gradient Boosting** excels in accuracy.

4.2. Results validation by Graphical Representation

We validated our results using several plots, including:

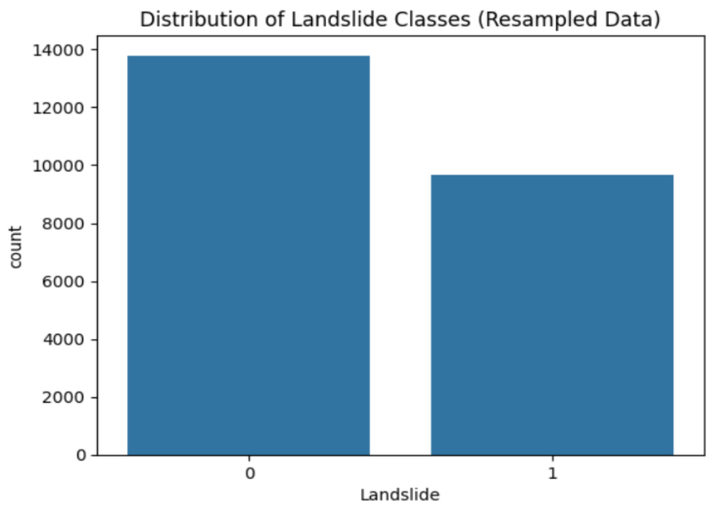


Figure: Data Visualization

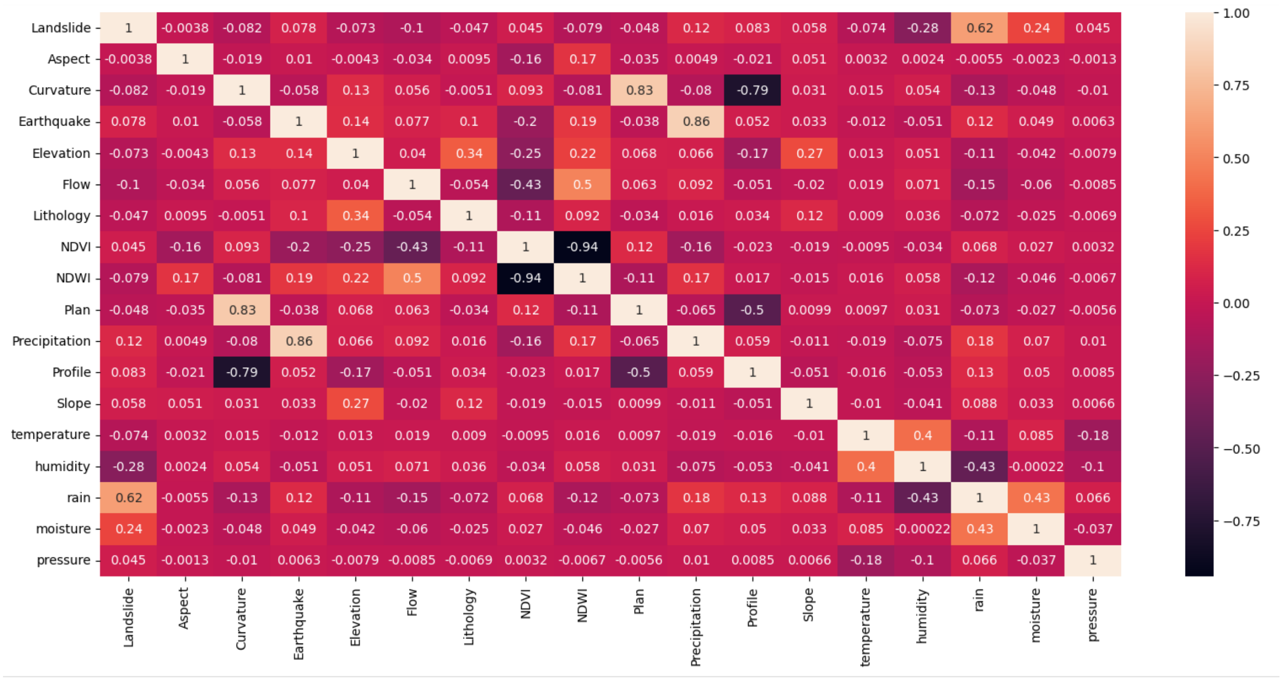


Figure: Correlation Heat Map (Decision Tree)

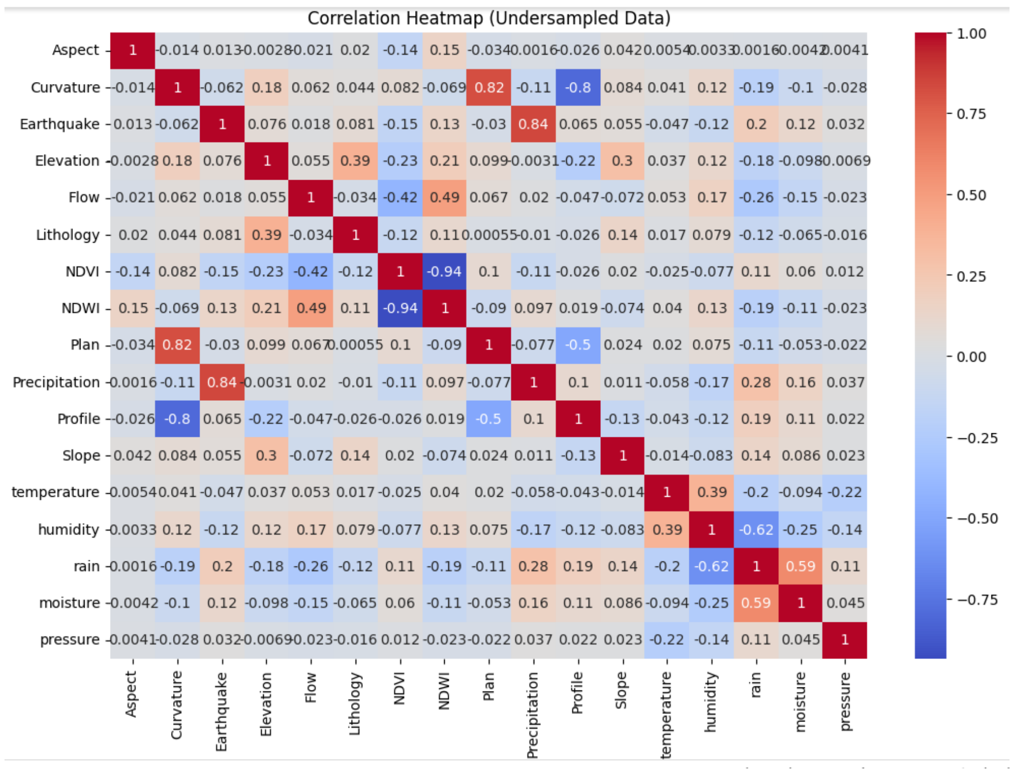


Figure: Correlation Heat Map (Gradient Boosting)

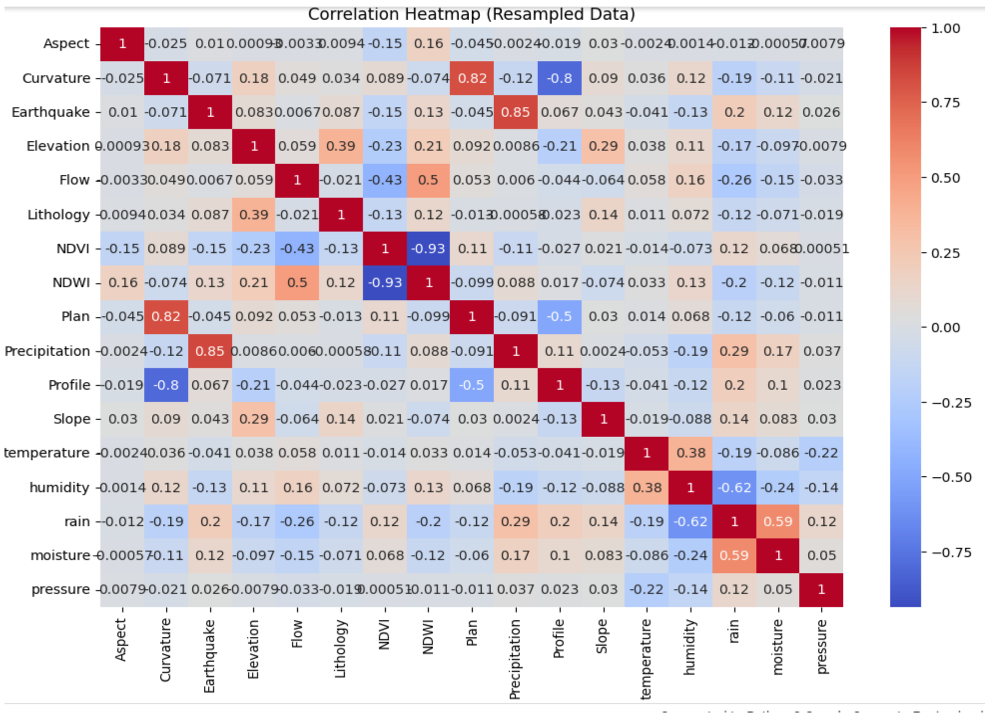


Figure: Correlation Heat Map (Random Forest)

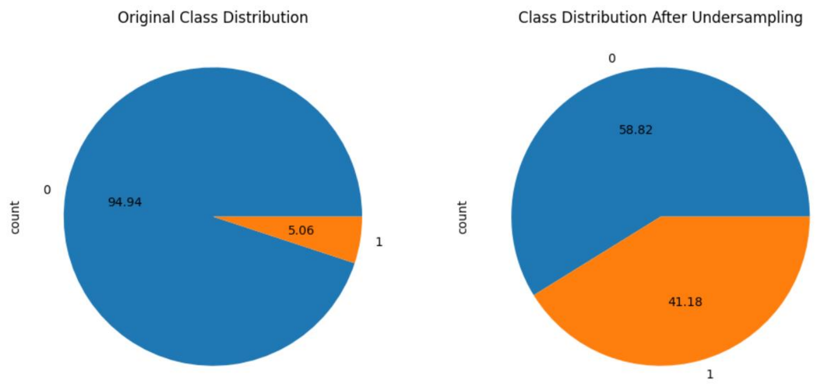
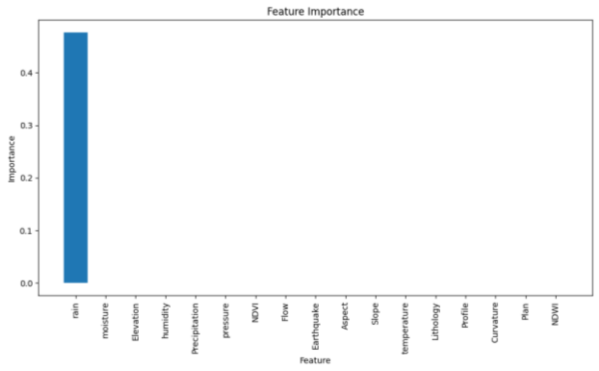


Figure: Undersampling



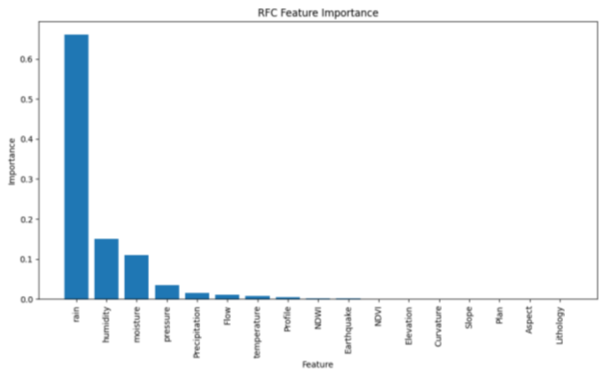


Figure: Feature Importance (Decision Tree)

Figure: Feature Importance (Random Forest)

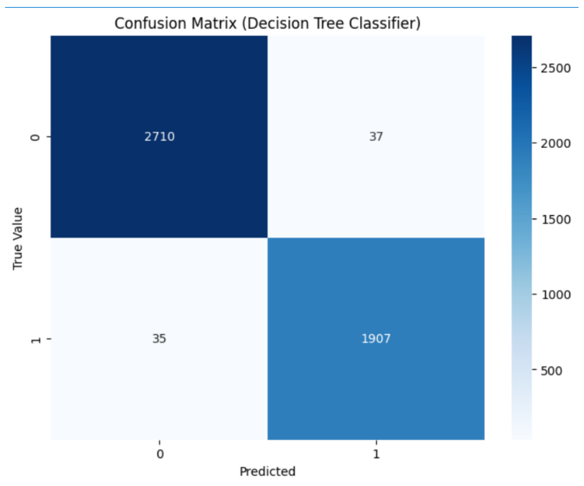


Figure: Confusion Matrix (Decision Tree)

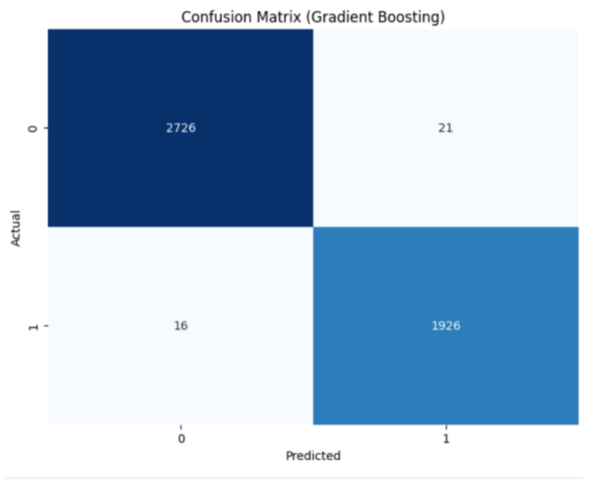


Figure: Confusion Matrix (Gradient Boosting)

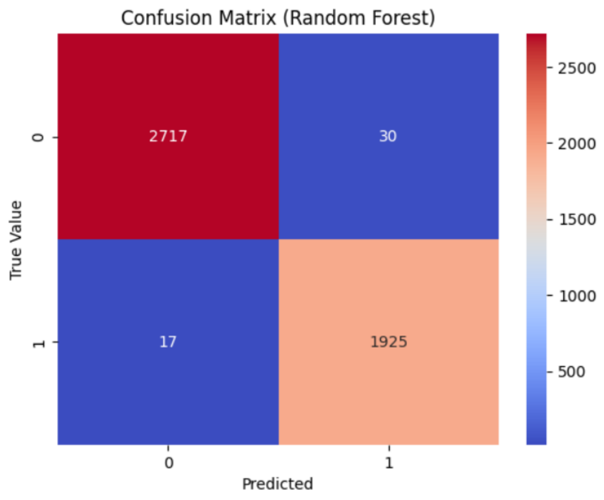


Figure: Confusion Matrix (Random Forest)

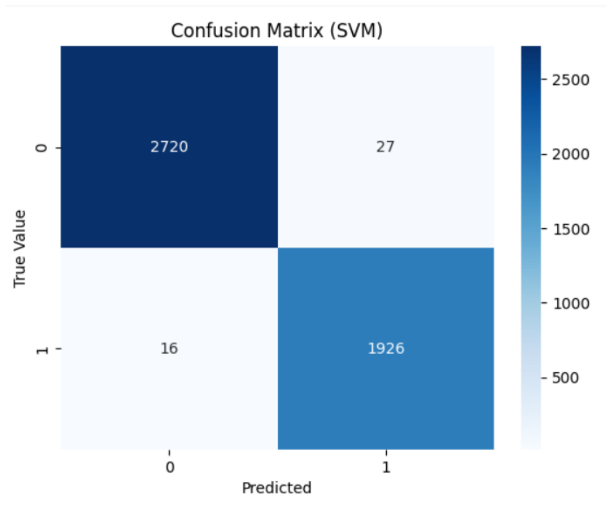


Figure: Confusion Matrix (Support Vector Machine)

The performance of the Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine models could be evaluated and compared by analyzing the confusion matrix and associated metrics. These metrics were useful in determining not only the accuracy of the predictions, but also the balance between identifying true landslide-prone areas and minimizing false predictions. The models were further evaluated using visual tools such as ROC curves to gain a thorough understanding of their predictive abilities.

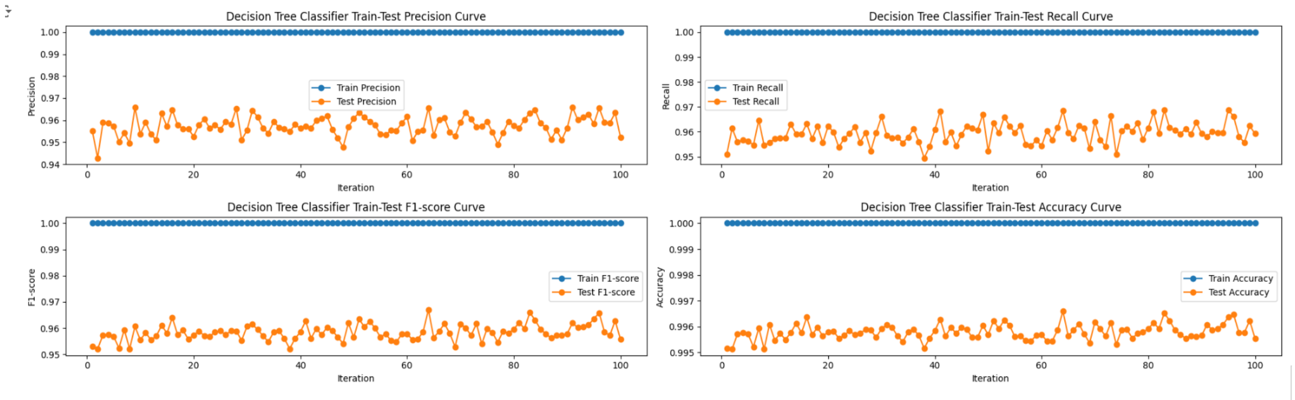


Figure: Precision curve, Recall curve, F1-score curve, Accuracy curve of Decision Tree

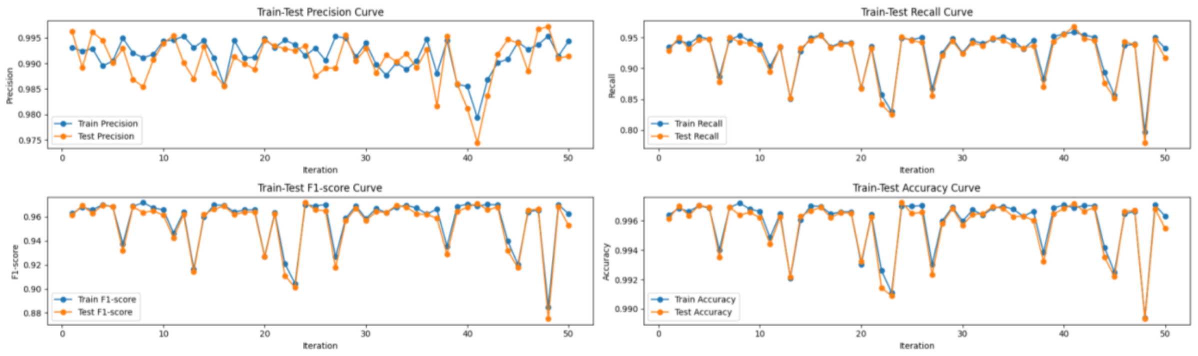


Figure: Precision curve, Recall curve, F1-score curve, Accuracy curve of Random Forest

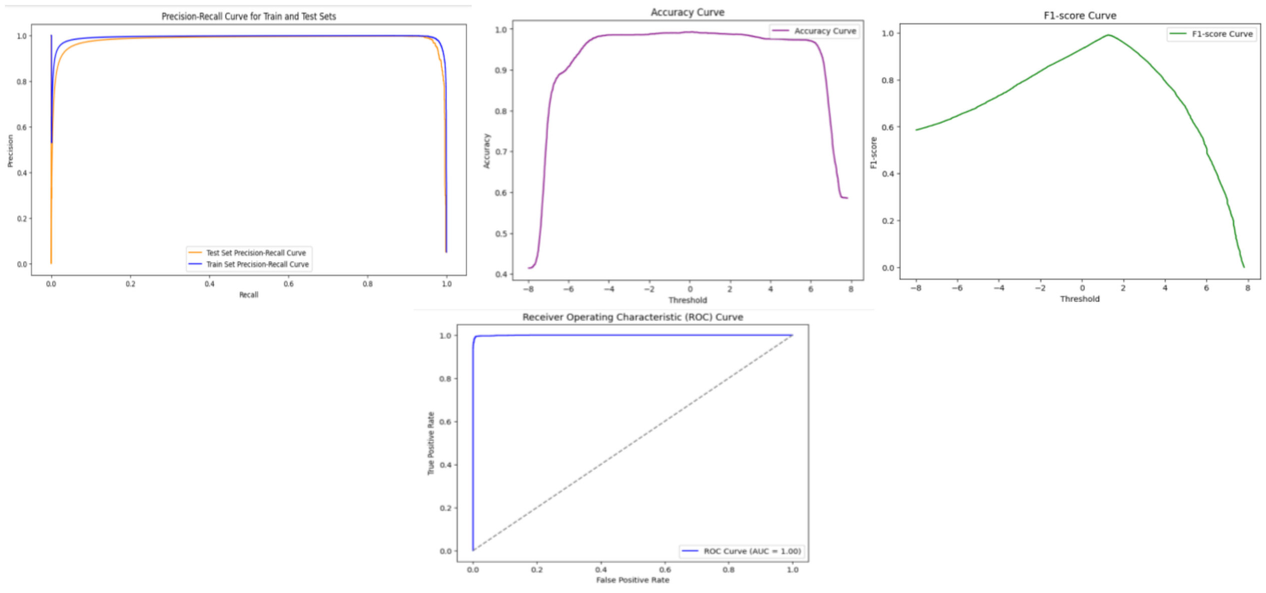


Figure: Precision curve, Recall curve, F1-score curve, Accuracy curve, ROC Curve of Gradient Boosting

5. Conclusion and Future Recommendations

5.1. The significance of outcomes

In this comparative study of machine learning models for landslide prediction, four popular algorithms were tested: Decision Tree, Gradient Boosting, Random Forest, and Support Vector Machine. Gradient Boosting outperformed other methods in correctly classifying landslide data, with an accuracy of 99 %.

The findings of this comparative study have significant implications for landslide prediction in regions prone to such natural disasters.

- The Gradient Boosting model is highly accurate and effective for predicting landslides with minimal error.

- SVM and Random Forest provide a good balance of precision and recall, making them suitable for managing false positives and false negatives.

The models can help develop early warning systems, potentially lowering the loss of life and property by making timely and accurate predictions.

5.2. Recommendations for Future Work

Feature Engineering: Future research could focus on feature engineering to improve model performance. Additional features, such as weather conditions, soil moisture levels, and topographic factors, may improve prediction accuracy.

Model Optimization: While Gradient Boosting performed well, there is still room for hyperparameter optimization and fine-tuning to improve predictive accuracy and lower computational costs.

Ensemble Learning: Given the power of ensemble models such as Gradient Boosting and Random Forest, future research could look into stacking multiple algorithms or developing hybrid models to capture the strengths of each approach and make even more accurate predictions.

Real-Time Prediction: Using these models in real-time systems for landslide-prone areas could provide practical benefits, such as continuous monitoring and immediate action in the event of an impending landslide.

Dataset Expansion: Future research could include more diverse landslide events from different geographical regions. This would improve the model's ability to generalize and perform on previously unseen data.

By implementing these recommendations, the models' accuracy and applicability in real-world landslide prediction scenarios can be enhanced.

6. References

1. B. T. Pham, D. T. Bui, and I. Prakash, "Landslide susceptibility modelling using different advanced decision trees methods," *Civil Engineering and Environmental Systems*, published online, vol. 36, no. 1, pp. 1-17, Jan. 2019. [Online]. Available: https://doi.org/10.1080/10286608.2019.1568418.

2. J. Dou, A. P. Yunus, D. T. Bui, A. Merghadi, M. Sahana, Z. Zhu, C.-W. Chen, Z. Han, and B. T. Pham, "Improved landslide assessment using support vector machine with bagging, boosting, and stacking ensemble machine learning framework in a mountainous watershed, Japan," *Landslides*, vol. 16, no. 12, pp. 1-14, 2019. [Online]. Available: https://doi.org/10.1007/s10346-019-01286-5.

3. B. T. Pham, I. Prakash, K. Khosravi, K. Chapi, P. T. Trinh, T. Q. Ngo, S. V. Hosseini, and D. T. Bui, "A comparison of support vector machines and Bayesian algorithms for landslide susceptibility modeling," *Geocarto International*, vol. 34, no. 9, pp. 1-15, 2018. [Online]. Available: https://doi.org/10.1080/10106049.2018.1489422.

4. S. Ma, J. Chen, S. Wu, and Y. Li, "Landslide susceptibility prediction using machine learning methods: A case study of landslides in the Yinghu Lake Basin in Shaanxi," *Sustainability*, vol. 13, no. 1, pp. 1-17, 2021. [Online]. Available: https://doi.org/10.3390/su13010123.

5. C. Ballabio and S. Sterlacchini, "Support vector machines for landslide susceptibility mapping: The Staffora River Basin case study, Italy," *Math. Geosci.*, vol. 44, no. 1, pp. 47-70, 2012. [Online]. Available: https://doi.org/10.1007/s11004-011-9379-9.

6. Y. Song, D. Yang, W. Wu, X. Zhang, J. Zhou, Z. Tian, C. Wang, and Y. Song, "Evaluating landslide susceptibility using sampling methodology and multiple machine learning models," *Int. J. Geo-Inf.*, vol. 9, no. 2, pp. 1-25, 2020. [Online]. Available: https://doi.org/10.3390/ijgi9020074.

7. H. Wang, L. Zhang, K. Yin, H. Luo, and J. Li, "Landslide identification using machine learning," *Geoscience Frontiers*, vol. 12, pp. 351–364, 2021. https://doi.org/10.1016/j.gsf.2020.09.019.

8. F. Catani, D. Lagomarsino, S. Segoni, and V. Tofani, "Landslide susceptibility estimation by random forests technique: sensitivity and scaling issues," *Nat. Hazards Earth Syst. Sci.*, vol. 13, no. 11, pp. 2815-2831, Nov. 2013. https://doi.org/10.5194/nhess-13-2815-2013.

9. S. Lee, "Application of logistic regression model and its validation for landslide susceptibility mapping using GIS and remote sensing data," *Int. J. Remote Sens.*, vol. 26, no. 7, pp. 1477–1491, Apr. 2005. https://doi.org/10.1080/01431160512331331012.

Appendixes

Decision Tree Model

Import Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
accuracy_score, mean_squared_error, roc_curve, auc
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.inspection import permutation_importance
from sklearn.tree import DecisionTreeClassifier
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import auc
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
```

Data Input

```
# Load the dataset
data = pd.read_csv("/content/drive/MyDrive/ML/Machine-Learning-Landslide-Prediction-
main/Landslide_dataset.csv")
data.head()
```

Data Analysis/Cleaning

```
# Check for missing values
features_na = [features for features in data.columns if data[features].isnull().sum() > 0]
for feature in features_na:
    print(feature, np.round(data[feature].isnull().mean(), 4),  ' % missing values')
else:
    print("No missing value found")
```

Precipitation, Slope, and Elevation are distributed in the dataset, with different colors representing whether a landslide occurred or not.

```
# Visualize precipitation, slope, elevation distributions
plt.figure(figsize=(18, 6))

# Precipitation Distribution
plt.subplot(1, 3, 1)
sns.histplot(data, x="Precipitation", hue='Landslide', kde=True)
plt.title("Precipitation Distribution", fontsize=14)

# Slope Distribution
```

```
plt.subplot(1, 3, 2)
sns.histplot(data, x="Slope", hue='Landslide', kde=True)
plt.title("Slope Distribution", fontsize=14)

# Elevation Distribution
plt.subplot(1, 3, 3)
sns.histplot(data, x="Elevation", hue='Landslide', kde=True)
plt.title("Elevation Distribution", fontsize=14)

plt.tight_layout()
plt.show()
```

orrelation heatmap to visualize the pairwise correlations between the features in the dataset.

```
# Plot the correlation heatmap
plt.figure(figsize=(18, 8))
sns.heatmap(data.corr(), annot=True)
plt.show()
```

Data Processing

```
# Specify your features and target
X = data.drop(columns=['Landslide'])
y = data['Landslide']

#Checking different class numbers in the target variable
plt.subplot(2, 2, 1)
y.value_counts().plot.pie(autopct='%.2f')
y.value_counts()

#undersampling
rus=RandomUnderSampler(sampling_strategy=0.7)
X_res, Y_res = rus.fit_resample(X, y)
plt.subplot(2, 2, 2)
ax=Y_res.value_counts().plot.pie(autopct='%.2f')
_=ax.set_title("Under sampling")
Y_res.value_counts()
```

X_train: The training set features (80% of X_res).
X_test: The test set features (20% of X_res).
y_train: The training set target variable (80% of Y_res). y_test: The test set target variable (20% of Y_res).

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_res, Y_res, test_size=0.2, random_state=42)
# Fit Decision Tree Classifier
dtc_model = DecisionTreeClassifier()
dtc_model.fit(X_train, y_train)
```

Feature Importance

```
# Create and fit a classifier model (Random Forest, for example)
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Calculate permutation feature importance
```

```
result = permutation_importance(model, X_test, y_test, n_repeats=10, random_state=42,
n_jobs=-1)

# Get feature importances and indices
importances = result.importances_mean
indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(12, 6))
plt.bar(range(X_test.shape[1]), importances[indices], align="center")
plt.xticks(range(X_test.shape[1]), X_test.columns[indices], rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature Importance")
plt.show()
```

Decision Tree Classifier

```
# Evaluate the model on the test set
dtc_model_score = dtc_model.score(X_test, y_test)
print("Decision Tree Classifier Accuracy:", dtc_model_score)

# Confusion matrix for Decision Tree Classifier
dtc_y_pred = dtc_model.predict(X_test)
dtc_cm = confusion_matrix(y_test, dtc_y_pred)
print("Confusion Matrix (Decision Tree):")
print(dtc_cm)

# Additional Evaluation Metrics
dtc_precision = precision_score(y_test, dtc_y_pred)
dtc_recall = recall_score(y_test, dtc_y_pred)
dtc_f1 = f1_score(y_test, dtc_y_pred)

print("Precision:", dtc_precision)
print("Recall:", dtc_recall)
print("F1-score:", dtc_f1)

# Plot ROC Curve
dtc_y_scores = dtc_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, dtc_y_scores)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree Classifier')
plt.legend(loc="lower right")
plt.show()
```

Performance of Decision Tree model

```
Decision Tree Classifier Accuracy: 0.9846449136276392
Confusion Matrix (Decision Tree):
[[2710   37]
 [  35 1907]]
Precision: 0.9809670781893004
Recall: 0.981977342945417
F1-score: 0.9814719505918682
```

```
# Confusion matrix for Decision Tree Classifier
dtc_y_pred = dtc_model.predict(X_test)
dtc_cm = confusion_matrix(y_test, dtc_y_pred)
print("Confusion Matrix (Decision Tree Classifier):")
print(dtc_cm)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(dtc_cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True Value')
plt.title('Confusion Matrix (Decision Tree Classifier)')
plt.show()
```

```
# Confusion matrix for Decision Tree Classifier
dtc_y_pred = dtc_model.predict(X_test)
dtc_cm = confusion_matrix(y_test, dtc_y_pred)
print("Confusion Matrix (Decision Tree Classifier):")
print(dtc_cm)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(dtc_cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True Value')
plt.title('Confusion Matrix (Decision Tree Classifier)')
plt.show()
```

```
Confusion Matrix (Decision Tree Classifier):
[[2710   37]
 [  35 1907]]
```

Precision curve, Recall curve, F1-score curve, Accuracy curve of DecisionTreeClassifier:

```
# Initialize lists to store train and test scores for each iteration
dtc_train_precision, dtc_test_precision = [], []
dtc_train_recall, dtc_test_recall = [], []
dtc_train_f1, dtc_test_f1 = [], []
dtc_train_accuracy, dtc_test_accuracy = [], []

# Number of iterations for train-test splits
num_iterations = 100

for i in range(num_iterations):
    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=i)

    # Step 3: Model training
    dtc_model = DecisionTreeClassifier()
    dtc_model.fit(X_train, y_train)

    # Step 4: Model evaluation on train and test sets
    y_train_pred = dtc_model.predict(X_train)
    y_test_pred = dtc_model.predict(X_test)

    # Calculate precision, recall, F1-score, and accuracy for train and test sets
    dtc_train_precision.append(precision_score(y_train, y_train_pred))
    dtc_test_precision.append(precision_score(y_test, y_test_pred))

    dtc_train_recall.append(recall_score(y_train, y_train_pred))
    dtc_test_recall.append(recall_score(y_test, y_test_pred))

    dtc_train_f1.append(f1_score(y_train, y_train_pred))
    dtc_test_f1.append(f1_score(y_test, y_test_pred))

    dtc_train_accuracy.append(accuracy_score(y_train, y_train_pred))
    dtc_test_accuracy.append(accuracy_score(y_test, y_test_pred))

# Plot train-test curves for Decision Tree Classifier
iterations = np.arange(1, num_iterations + 1)

plt.figure(figsize=(20, 6))

# Precision curve
plt.subplot(2, 2, 1)
plt.plot(iterations, dtc_train_precision, marker='o', label='Train Precision')
plt.plot(iterations, dtc_test_precision, marker='o', label='Test Precision')
plt.xlabel('Iteration')
plt.ylabel('Precision')
plt.title('Decision Tree Classifier Train-Test Precision Curve')
plt.legend()

# Recall curve
plt.subplot(2, 2, 2)
plt.plot(iterations, dtc_train_recall, marker='o', label='Train Recall')
plt.plot(iterations, dtc_test_recall, marker='o', label='Test Recall')
plt.xlabel('Iteration')
plt.ylabel('Recall')
plt.title('Decision Tree Classifier Train-Test Recall Curve')
plt.legend()

# F1-score curve
plt.subplot(2, 2, 3)
plt.plot(iterations, dtc_train_f1, marker='o', label='Train F1-score')
plt.plot(iterations, dtc_test_f1, marker='o', label='Test F1-score')
plt.xlabel('Iteration')
plt.ylabel('F1-score')
plt.title('Decision Tree Classifier Train-Test F1-score Curve')
plt.legend()

# Accuracy curve
plt.subplot(2, 2, 4)
plt.plot(iterations, dtc_train_accuracy, marker='o', label='Train Accuracy')
plt.plot(iterations, dtc_test_accuracy, marker='o', label='Test Accuracy')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.title('Decision Tree Classifier Train-Test Accuracy Curve')
plt.legend()

plt.tight_layout()
plt.show()
```

Gradient Boosting Model

Import Library

```
import pandas as pd
import numpy as np
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall_curve, auc,
roc_curve, r2_score, mean_squared_error
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
accuracy_score, classification_report, precision_recall_curve, roc_curve, auc
from sklearn.ensemble import GradientBoostingClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, auc
from google.colab import drive
drive.mount('/content/drive')
```

Data Input

```
# Load the dataset
data = pd.read_csv("/content/drive/MyDrive/ML/Machine-Learning-Landslide-Prediction-
main/Landslide_dataset.csv")
data.head()
```

Data Analysis/Cleaning

```
# Check for missing values
```

```
features_na = [features for features in data.columns if data[features].isnull().sum() > 0]
for feature in features_na:
    print(feature, np.round(data[feature].isnull().mean(), 4),  ' % missing values')
else:
    print("No missing value found")


X = data.drop(['Landslide'], axis=1)
y = data['Landslide']

print(X, y)
```

Evaluating a Gradient Boosting Classifier on an undersampled dataset, including visualizations of performance metrics and error calculations.

```
# Checking different class numbers in the target variable
plt.figure(figsize=(12, 6))

# Subplot 1 - Original Class Distribution
plt.subplot(1, 2, 1)
y.value_counts().plot.pie(autopct='%.2f')
plt.title('Original Class Distribution')

# Subplot 2 - Class Distribution After Undersampling
plt.subplot(1, 2, 2)
rus = RandomUnderSampler(sampling_strategy=0.7)
X_resampled, y_resampled = rus.fit_resample(X, y)
y_resampled.value_counts().plot.pie(autopct='%.2f')
plt.title('Class Distribution After Undersampling')

plt.show()

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2,
random_state=42)

# Creating and training the Gradient Boosting Classifier on the undersampled data
gb_classifier = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_classifier.fit(X_train, y_train)

# Making predictions on the test set
y_pred = gb_classifier.predict(X_test)

# Evaluating the model performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Correlation heatmap with undersampled data
numeric_data_undersampled = pd.DataFrame(X_resampled,
columns=X.columns).select_dtypes(include=[np.number])
corr_undersampled = numeric_data_undersampled.corr()
```

```
plt.figure(figsize=(12, 8))
sns.heatmap(corr_undersampled, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap (Undersampled Data)')
plt.show()

# Confusion Matrix
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, gb_classifier.decision_function(X_test))

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='Precision-Recall Curve', color='darkorange')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()

# F1-score Curve
f1_scores = 2 * (precision * recall) / (precision + recall)

# Define thresholds here
thresholds = np.linspace(
    min(gb_classifier.decision_function(X_test)),
    max(gb_classifier.decision_function(X_test)),
    len(f1_scores)
)

plt.figure(figsize=(8, 6))
plt.plot(thresholds, f1_scores, label='F1-score Curve', color='green')
plt.xlabel('Threshold')
plt.ylabel('F1-score')
plt.title('F1-score Curve')
plt.legend()
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, gb_classifier.decision_function(X_test))
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC Curve (AUC = {:.2f})'.format(roc_auc), color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()

# Accuracy Curve
accuracies = [accuracy_score(y_test, gb_classifier.decision_function(X_test) > threshold) for
threshold in thresholds]

plt.figure(figsize=(8, 6))
plt.plot(thresholds, accuracies, label='Accuracy Curve', color='purple')
plt.xlabel('Threshold')
plt.ylabel('Accuracy')
plt.title('Accuracy Curve')
plt.legend()
plt.show()

# Error calculation
gb_y_pred = gb_classifier.predict(X_test)
print("R2 score of our model is:", r2_score(y_test, gb_y_pred))
print("Mean Squared Error of our model is:", mean_squared_error(y_test, gb_y_pred))

# Show Train-Test Curve
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Gradient Boosting Classifier on the training data
gb_classifier_train = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_classifier_train.fit(X_train, y_train)

# Making predictions on the test set
y_pred_train = gb_classifier_train.predict(X_test)

#Evaluating the model performance on the test set
accuracy_train = accuracy_score(y_test, y_pred_train)
print(f"Accuracy on Test Set: {accuracy_train}")

# Precision, Recall, and F1-score on the test set
precision_test = precision_score(y_test, y_pred_train)
recall_test = recall_score(y_test, y_pred_train)
f1_test = f1_score(y_test, y_pred_train)

print("Precision on Test Set:", precision_test)
print("Recall on Test Set:", recall_test)
print("F1-score on Test Set:", f1_test)

# Visualize Precision-Recall Curve for Train and Test Sets
precision_test, recall_test, _ = precision_recall_curve(y_test,
gb_classifier_train.decision_function(X_test))
precision_train, recall_train, _ = precision_recall_curve(y_train,
gb_classifier_train.decision_function(X_train))
```

```
plt.figure(figsize=(12, 6))

# Test set Precision-Recall curve
plt.plot(recall_test, precision_test, label='Test Set Precision-Recall Curve', color='darkorange')

# Train set Precision-Recall curve
plt.plot(recall_train, precision_train, label='Train Set Precision-Recall Curve', color='blue')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Train and Test Sets')
plt.legend()
plt.show()
```

```
Performance for Gradient Boosting model: 0.9923224568138196
R2 score of our model is: 0.9683572042077923
Mean Squared Error of our model is: 0.007677543186180422
Accuracy on Test Set: 0.9971711456859972
Precision on Test Set: 0.984970477724101
Recall on Test Set: 0.95822454308094
F1-score on Test Set: 0.9714134462678665
```

Plotting Learning Curve for a Machine Learning Model using Scikit-learn

```
from sklearn.model_selection import learning_curve

# Function to plot learning curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure(figsize=(8, 6))
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Plotting the learning curve
plot_learning_curve(gb_classifier, "Learning Curve (Gradient Boosting)", X_resampled,
y_resampled, cv=5, n_jobs=-1)
plt.show()
```

Random Forest Model

```
Import Library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
accuracy_score, precision_recall_curve, roc_curve, auc, mean_squared_error
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.under_sampling import RandomUnderSampler

from google.colab import drive
drive.mount('/content/drive')

Data Input
data = pd.read_csv('/content/drive/MyDrive/ML/Machine-Learning-Landslide-Prediction-
main/Landslide_dataset.csv')
data.head()
Data Analysis/Cleaning
features_list = [features for features in data.columns if data[features].isnull().sum() > 0]
for feature in features_list:
    #data[feature].isnull() indicates whether each value in the column is missing or not
    print(feature, np.round(data[feature].isnull().mean(), 4),  ' % missing values')
else:
    print("There are no missing values.")
There are no missing values.
#Separate features and target variable
X=data.drop('Landslide', axis=1)
Y=data['Landslide']
```

```
#Checking different class numbers in the target variable
Y.value_counts().plot.pie(autopct='%.2f')
Y.value_counts()

Undersampling
rus=RandomUnderSampler(sampling_strategy=0.7)
X_res, Y_res = rus.fit_resample(X, Y)
ax=Y_res.value_counts().plot.pie(autopct='%.2f')
_=ax.set_title("Under sampling")
Y_res.value_counts()
Data Processing
#Separate features and target variable
# X=data.drop('Landslide', axis=1)
# Y=data['Landslide']

#Splitting the dataset into the training set and test set
X_train, X_test, Y_train, Y_test = train_test_split(X_res, Y_res, test_size=0.2, random_state=42)
Exploratory Data Analysis (EDA) and Visualization
sns.countplot(x=Y_res, data=pd.DataFrame({'Landslide': Y_res}))
plt.title('Distribution of Landslide Classes (Resampled Data)')
plt.show()

# Correlation heatmap with resampled data
numeric_data_resampled = pd.DataFrame(X_res,
columns=X.columns).select_dtypes(include=[np.number])
corr_resampled = numeric_data_resampled.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_resampled, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap (Resampled Data)')
plt.show()
Fit model - Random Forest
# Fit Random Forest Classifier
model_rfc = RandomForestClassifier(criterion='entropy', max_depth=3, max_features='sqrt',
n_estimators=100)
model_rfc.fit(X_train, Y_train)
Feature Importance
#Predict on the test set
Y_pred = model_rfc.predict(X_test)

#Get feature importances from the trained Random Forest model
feature_importances = model_rfc.feature_importances_

#Get feature names from the X_train columns
feature_names = X_train.columns

#Sort in descending order
sorted = np.argsort(feature_importances)[::-1]

# Plot
plt.figure(figsize=(12, 6))
plt.bar(range(X_train.shape[1]), feature_importances[sorted], align="center")
plt.xticks(range(X_train.shape[1]), feature_names[sorted], rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("RFC Feature Importance")
plt.show()
```

Precision, Recall, F1 and Accuracy values

```
# Predict on the test set
Y_pred = model_rfc.predict(X_test)

# Evaluate additional metrics
precision = precision_score(Y_test, Y_pred)
recall = recall_score(Y_test, Y_pred)
f1 = f1_score(Y_test, Y_pred)
accuracy = accuracy_score(Y_test, Y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Accuracy:", accuracy)

# Plot Precision-Recall curve

#This line predicts the probability estimates of the positive class
Y_pred_probs = model_rfc.predict_proba(X_test)[:, 1]
#calculates precision and recall values at different probability thresholds
precision, recall, _ = precision_recall_curve(Y_test, Y_pred_probs)

plt.step(recall, precision, color='r', alpha=0.7, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.show()

# Plot F1-score curve
f1_values = 2 * (precision * recall) / (precision + recall)
plt.plot(recall, f1_values, color='k', label='F1-score')
plt.xlabel('Recall')
plt.ylabel('F1-score')
plt.title('F1-score Curve')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.legend()
plt.show()

# Calculate confusion matrix(used to evaluate the performance of a classification model)
cm = confusion_matrix(Y_test, Y_pred)

#Create a heatmap for the confusion matrix
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm')
plt.xlabel('Predicted')
plt.ylabel('True Value')
plt.title('Confusion Matrix')
plt.show()

# Calculate ROC curve and AUC (fpr, tpr, thresholds --  arrays)
Y_pred_probs = model_rfc.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve (graphical representation of the trade-off between true positive rate and false
positive rate at various thresholds)
plt.plot(fpr, tpr, color='b', label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='r', linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Performance Accuracy for Random Forest Model

```
Precision: 0.9845995893223819
Recall: 0.9876416065911432
F1-score: 0.9861182519280205
Accuracy: 0.9884836852207294
```

Train and Test results curve

```
#lists to store train and test scores for each iteration
train_precision, test_precision = [], []
train_recall, test_recall = [], []
train_f1, test_f1 = [], []
train_accuracy, test_accuracy = [], []

#Number of iterations for train-test splits
num_iterations = 50

for i in range(num_iterations):
    # Split the data into train and test sets
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=i)

    # Step 3: Model training
    model_rfc = RandomForestClassifier(criterion='entropy', max_depth=3, max_features='sqrt',
n_estimators=100)
    model_rfc.fit(X_train, Y_train)

    # Step 4: Model evaluation on train and test sets
    Y_train_pred = model_rfc.predict(X_train)
    Y_test_pred = model_rfc.predict(X_test)
    # Calculate precision, recall, F1-score, and accuracy for train and test sets
    train_precision.append(precision_score(Y_train, Y_train_pred))
    test_precision.append(precision_score(Y_test, Y_test_pred))

    train_recall.append(recall_score(Y_train, Y_train_pred))
    test_recall.append(recall_score(Y_test, Y_test_pred))

    train_f1.append(f1_score(Y_train, Y_train_pred))
    test_f1.append(f1_score(Y_test, Y_test_pred))

    train_accuracy.append(accuracy_score(Y_train, Y_train_pred))
    test_accuracy.append(accuracy_score(Y_test, Y_test_pred))

# Plot train-test curves
iterations = np.arange(1, num_iterations + 1)

plt.figure(figsize=(20, 6))

# Precision curve
plt.subplot(2, 2, 1)
plt.plot(iterations, train_precision, marker='o', label='Train Precision')
plt.plot(iterations, test_precision, marker='o', label='Test Precision')
plt.xlabel('Iteration')
plt.ylabel('Precision')
plt.title('Train-Test Precision Curve')
plt.legend()

# Recall curve
plt.subplot(2, 2, 2)
plt.plot(iterations, train_recall, marker='o', label='Train Recall')
plt.plot(iterations, test_recall, marker='o', label='Test Recall')
plt.xlabel('Iteration')
plt.ylabel('Recall')
plt.title('Train-Test Recall Curve')
plt.legend()

# F1-score curve
plt.subplot(2, 2, 3)
plt.plot(iterations, train_f1, marker='o', label='Train F1-score')
plt.plot(iterations, test_f1, marker='o', label='Test F1-score')
plt.xlabel('Iteration')
plt.ylabel('F1-score')
plt.title('Train-Test F1-score Curve')
plt.legend()

# Accuracy curve
plt.subplot(2, 2, 4)
plt.plot(iterations, train_accuracy, marker='o', label='Train Accuracy')
plt.plot(iterations, test_accuracy, marker='o', label='Test Accuracy')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.title('Train-Test Accuracy Curve')
plt.legend()

plt.tight_layout()
plt.show()
```

Support Vector Machine Model

Import Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
accuracy_score, classification_report, precision_recall_curve, roc_curve, auc
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
import seaborn as sns
from sklearn.inspection import permutation_importance
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, auc
from imblearn.under_sampling import RandomUnderSampler

from google.colab import drive
drive.mount('/content/drive')
```

Data Input

```
import pandas as pd
import os
print(os.getcwd())

# Load the dataset
#df = pd.read_csv(file_path)
df = pd.read_csv('/content/drive/MyDrive/ML/Machine-Learning-Landslide-Prediction-
main/Landslide_dataset.csv')
df.head()
```

Data Analysis/Cleaning

```
# Check for missing values
features_na = [features for features in df.columns if df[features].isnull().sum() > 0]
for feature in features_na:
    print(feature, np.round(df[feature].isnull().mean(), 4),  ' % missing values')
else:
    print("No missing value found")
No missing value found
```

Specify the target variable and features;X= Independent variable;y= Dependent variable

```
#Specify the target variable
target_column = 'Landslide'
label_encoder = LabelEncoder()
df[target_column] = label_encoder.fit_transform(df[target_column])

# Specify your features and target
X = df.drop(columns=[target_column])
y = df[target_column]
```

Target variable (y): The Landslide column is encoded as numeric values (0 or 1);Features (X): The columns Earthquake, Flow, and Slope are categorical features

```
#Specify the target variable
from sklearn.preprocessing import LabelEncoder
target_column = 'Landslide'
label_encoder = LabelEncoder()
df[target_column] = label_encoder.fit_transform(df[target_column])

# Specify your features and target
X = df.drop(columns=[target_column])
y = df[target_column]

# Encode categorical variables
label_encoder = LabelEncoder()
columns_to_encode = ['Earthquake', 'Flow', 'Slope']
for column in columns_to_encode:
    X[column] = label_encoder.fit_transform(X[column].astype(str))
```

The training set gets 80% of the data, and the test set gets 20%.[from random data]

```
from imblearn.under_sampling import RandomUnderSampler
#Separate features and target variable
# Separate features and target variable
X = df.drop('Landslide', axis=1)
y = df['Landslide']

# Create a subplot with one row and two columns
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plotting the original class distribution
axes[0].pie(y.value_counts(), autopct='%.2f', labels=y.value_counts().index)
axes[0].set_title('Original Class Distribution')

# Applying RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy=0.7)
X_res, y_res = rus.fit_resample(X, y)
```

```
# Plotting the class distribution after undersampling
ax = axes[1].pie(y_res.value_counts(), autopct='%.2f', labels=y_res.value_counts().index)
axes[1].set_title('Class Distribution After Undersampling')

plt.show()
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)
```

Initialize and Train the SVM model

```
svm_model = SVC(kernel="linear")

# Fit the SVM model
svm_model.fit(X_train, y_train)
```

Cross-validation using SVM; Plotting training details for SVM; Computes the confusion matrix, showing how many predictions were true positives, true negatives, false positives, and false negatives

```
from sklearn.model_selection import cross_val_score

# Add Support Vector Machine (SVM) model
# Cross-validate SVM model
svm_scores = cross_val_score(estimator=svm_model, X=X_train, y=y_train, cv=5)
svm_mean_score = np.mean(svm_scores)

print("Support Vector Machine (SVM) Scores:")
print(svm_scores)
print("Mean SVM Score:", svm_mean_score)

# Plot training details for SVM
plt.figure(figsize=(8, 5))
plt.plot(range(1, 6), svm_scores, marker='o', linestyle='--', label='Accuracy')
plt.plot(range(1, 6), [svm_mean_score] * 5, color='r', linestyle='-', label='Mean Accuracy')
plt.xticks(range(1, 6))
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Training Details during 5-fold Cross-Validation (SVM)')
plt.legend()
plt.show()

# Evaluate SVM model on the test set
svm_model_score = svm_model.score(X_test, y_test)
print("Support Vector Machine (SVM) Accuracy:", svm_model_score)

# Confusion matrix for SVM
svm_y_pred = svm_model.predict(X_test)
svm_cm = confusion_matrix(y_test, svm_y_pred)
print("Confusion Matrix (SVM):")
print(svm_cm)

# Evaluate additional metrics for SVM
svm_precision = precision_score(y_test, svm_y_pred)
svm_recall = recall_score(y_test, svm_y_pred)
svm_f1 = f1_score(y_test, svm_y_pred)
```

```
svm_accuracy = accuracy_score(y_test, svm_y_pred)

print("SVM Precision:", svm_precision)
print("SVM Recall:", svm_recall)
print("SVM F1-score:", svm_f1)
print("SVM Accuracy:", svm_accuracy)

# Classification report for SVM
svm_class_report = classification_report(y_test, svm_y_pred)
print("Classification Report (SVM):")
print(svm_class_report)

# Plot Precision-Recall curve for SVM
svm_y_pred_probs = svm_model.decision_function(X_test)
svm_precision, svm_recall, _ = precision_recall_curve(y_test, svm_y_pred_probs)

plt.step(svm_recall, svm_precision, color='b', alpha=0.7, where='post')
plt.fill_between(svm_recall, svm_precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (SVM)')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.show()

# Plot F1-score curve for SVM
svm_f1_values = 2 * (svm_precision * svm_recall) / (svm_precision + svm_recall)
plt.plot(svm_recall, svm_f1_values, color='b', label='F1-score')
plt.xlabel('Recall')
plt.ylabel('F1-score')
plt.title('F1-score Curve (SVM)')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.legend()
plt.show()

# Calculate confusion matrix for SVM
svm_cm = confusion_matrix(y_test, svm_y_pred)

# Create a heatmap for the confusion matrix (SVM)
sns.heatmap(svm_cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True Value')
plt.title('Confusion Matrix (SVM)')
plt.show()

# Calculate ROC curve and AUC for SVM
svm_y_pred_probs = svm_model.decision_function(X_test)
svm_fpr, svm_tpr, svm_thresholds = roc_curve(y_test, svm_y_pred_probs)
svm_roc_auc = auc(svm_fpr, svm_tpr)
```

```
# Plot ROC curve for SVM
plt.plot(svm_fpr, svm_tpr, color='b', label='ROC curve (AUC = %0.2f)' % svm_roc_auc)
plt.plot([0, 1], [0, 1], color='r', linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve (SVM)')
plt.legend(loc="lower right")
plt.show()
```

Performance Accuracy for Support Vector Machine

```
Support Vector Machine (SVM) Scores:
[0.99067164 0.98880299 0.99066915 0.98960277 0.99120235]
Mean SVM Score: 0.9901897802377079
Support Vector Machine (SVM) Accuracy: 0.9908296011942845
Confusion Matrix (SVM):
[[2720   27]
 [  16 1926]]
SVM Precision: 0.9861751152073732
SVM Recall: 0.9917610710607621
SVM F1-score: 0.9889602053915275
SVM Accuracy: 0.9908296011942845
Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2747
           1       0.99      0.99      0.99      1942

    accuracy                           0.99      4689
   macro avg       0.99      0.99      0.99      4689
weighted avg       0.99      0.99      0.99      4689
```