SRINIVAS UNIVERSITY

INSTITUTE OF ENGINEERING & TECHNOLOGY

(Subject: Fundamentals of AI & ML)

(SUBJECT CODE:24SBT110)

A MINI PROJECT REPORT ON

"Earthquake Severity Classification using MongoDB and Machine Learning"

Submitted in the partial fulfillment of the requirements for the Fourth semester

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE & ENGINEERING

Akshay Kumar	**01SU24CS015**
Aryan	**01SU24CS023**
Basavaraja Shekhappa Sajjana	**01SU24CS026**
Kishore M R	**01SU24CS070**
Lakshith	**01SU24CS075**

UNDER THE GUIDANCE OF

Prof Mahesh Kumar V B

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SRINIVAS UNIVERSITY INSTITUTE OF ENGINEERING & TECHNOLOGY

MUKKA, MANGALURU-574146

2025-26

SRINIVAS UNIVERSITY
INSTITUTE OF ENGINEERING & TECHNOLOGY

Department of Computer Science Engineering

CERTIFICATE

This is to certify that, **Mr.Kishore M R(01SU24CS070) , Mr.Aryan(01SU24CS023) , Mr.Lakshith(01SU24CS075) , Mr.Basavaraja Shekhappa Sajjana(01SU24CS026) , Mr.Akshay Kumar(01SU24CS015)** have satisfactorily completed the assessment (Report) in **Fundamentals of AI & ML (24SBT110)** prescribed by the Srinivas University for the 4th semester B. Tech course during the year **2025-26**.

MARKS AWARDED

Staff In charge

Name: Prof Mahesh Kumar V B

Date: 27-02-2026

TABLE OF CONTENTS

Abstract

Earthquakes are among the most destructive natural disasters, causing significant damage to life and property. Accurate classification of seismic events is essential for disaster preparedness, infrastructure planning, and risk mitigation. This project presents a Machine Learning-based Earthquake Classification System integrated with MongoDB for efficient data storage and processing.

The earthquake dataset, initially stored in CSV format, is inserted into a MongoDB database to simulate real-world data management systems. MongoDB enables scalable and flexible storage of structured seismic records. The data is then retrieved from the database and converted into a Pandas DataFrame for preprocessing and analysis.

Exploratory Data Analysis (EDA) is performed using visualization techniques such as correlation heatmaps, distribution plots, and scatter plots to understand relationships between seismic attributes like magnitude, depth, and geographical coordinates.

For classification, the HistGradientBoostingClassifier algorithm is implemented. This advanced gradient boosting technique improves predictive performance by combining multiple decision trees. Instead of relying on the default classification threshold, probability predictions are generated and an optimal threshold is selected using ROC curve analysis. This threshold tuning enhances model sensitivity and specificity.

The model's performance is evaluated using metrics such as Accuracy, Precision, Recall, F1 Score, and ROC analysis. The results demonstrate that the proposed system effectively classifies earthquake events and provides reliable predictive performance.

This project highlights the practical integration of database systems with advanced machine learning techniques, creating a scalable and efficient earthquake classification framework suitable for real-world seismic monitoring applications.

Introduction

Natural disasters pose significant threats to human civilization, and earthquakes are among the most unpredictable and devastating events. Monitoring and classifying earthquakes is crucial for disaster preparedness, structural engineering, and risk mitigation planning.

Traditionally, earthquake analysis relied on manual observation of seismic wave recordings and expert interpretation. However, with the advancement of Artificial Intelligence (AI) and Machine Learning (ML), automated systems can now analyze seismic datasets efficiently and classify earthquake events based on magnitude and other parameters.

Earthquake classification is a supervised machine learning problem where the objective is to categorize seismic events into predefined classes based on magnitude or intensity levels. It involves analyzing features such as depth, latitude, longitude, magnitude, and other geological indicators.

In real-world systems, seismic data is continuously generated and stored in databases. Therefore, integrating machine learning with database systems is essential. In this project, MongoDB is used as a NoSQL database to store and manage earthquake data before applying machine learning algorithms. This approach reflects practical industrial applications where raw data is stored in databases rather than static CSV files.

The project workflow includes:

- Loading earthquake data from CSV
- Storing the dataset in MongoDB
- Retrieving and converting data into Pandas DataFrame
- Performing data preprocessing and visualization
- Training classification models
- Evaluating model performance

By combining database integration with machine learning, the project demonstrates a complete data pipeline suitable for real-time seismic monitoring systems.

Problem Statement

Earthquake monitoring systems generate large volumes of seismic data containing multiple attributes such as magnitude, depth, latitude, longitude, and time. Manual analysis of such data is time-consuming and inefficient.

There is a need for an automated system that:

- Stores earthquake data efficiently using MongoDB
- Retrieves and preprocesses seismic data
- Classifies earthquakes based on magnitude
- Evaluates classification accuracy using performance metrics

The goal is to develop a scalable earthquake classification system that can assist in disaster management and seismic risk assessment.

Objectives

- To store earthquake dataset in MongoDB
- To retrieve and preprocess data from MongoDB
- To perform Exploratory Data Analysis (EDA)
- To build machine learning classification models
- To evaluate model performance using standard metrics
- To visualize classification results

Technologies Used

1. Python

Python is the primary programming language used in this project. It provides simplicity, readability, and extensive libraries for machine learning and data analysis.

Role:
- Implementing program logic
- Database connectivity
- Data preprocessing
- Model training and evaluation
- Visualization

2. MongoDB

MongoDB is a NoSQL database used to store earthquake data.

Role:
• Database Name: mlproject
• Collection Name: earthquake
• Prevents duplicate insertion
• Demonstrates real-world database integration
• Stores structured seismic records

Connection URL used:
mongodb://localhost:27017/

MongoDB allows flexible document-based storage, making it ideal for large seismic datasets.

3. Pandas

Used for:
• Reading CSV file
• Converting MongoDB data into DataFrame
• Data cleaning and preprocessing
• Feature selection

4. NumPy

Used for:
• Numerical computations
• Array operations
• Mathematical calculations

5. Matplotlib

Used for:
• Scatter plots
• Classification visualization
• Performance graphs

6. Seaborn

Used for:
• Correlation heatmaps
• Distribution plots
• Enhanced visualizations

7. Scikit-learn

Scikit-learn is the primary Machine Learning library used in this project.

Role in the Project:

• Splitting dataset using train_test_split
• Implementing **HistGradientBoostingClassifier**
• Generating probability predictions using predict_proba()
• Computing ROC curve using roc_curve()
• Selecting optimal classification threshold
• Evaluating model performance

Dataset: Description, Features and Splitting

Dataset Description

The earthquake dataset contains seismic activity records with multiple attributes.

Structure:
• Each row represents one earthquake event
• Each column represents a feature

Total records : 23119 (from 1969-2018)

The dataset is structured and suitable for supervised learning classification.

Features in Dataset

1. Latitude – Geographic coordinate
2. Longitude – Geographic coordinate
3. Depth – Depth of earthquake (km)
4. Magnitude – Strength of earthquake
5. Time – Timestamp of event
6. Location – Region of occurrence

Target Variable:
Earthquake Category (derived from magnitude ranges)

The dataset represents a supervised classification problem.

Dataset Splitting

To evaluate performance:

Training Set (80%)
• Used to train the classification model

Testing Set (20%)
• Used to evaluate performance on unseen data

Methodology

Step 1: Data Collection

Earthquake dataset collected from open-source seismic records.

Step 2: Data Storage in MongoDB

• CSV loaded using Pandas
• Inserted into MongoDB collection
• Duplicate prevention logic applied

Step 3: Data Retrieval

• Data fetched using PyMongo
• Converted into Pandas DataFrame

Step 4: Data Preprocessing

• Handling missing values
• Encoding categorical variables
• Feature selection
• Splitting dataset

Step 5: Exploratory Data Analysis

• Correlation heatmap
• Magnitude distribution
• Depth vs Magnitude analysis

Step 6: Model Building

The following algorithm was implemented:

HistGradientBoostingClassifier

The model was trained using:

• max_depth = 10 to control tree complexity
• learning_rate = 0.1 to regulate boosting contribution

• max_iter = 300 to define number of boosting iterations
• random_state = 42 for reproducibility

The model was trained using:

model.fit(X_train, y_train)

This algorithm improves classification performance by combining multiple weak learners (decision trees) into a strong predictive model.

Step 7: Probability Prediction and Threshold Optimization

Instead of directly using default predictions, probability scores were generated:

y_proba = model.predict_proba(X_test)[:, 1]

To improve classification performance, the ROC curve was computed:

fpr, tpr, thresholds = roc_curve(y_test, y_proba, pos_label=3)

The optimal threshold was selected using:

best_threshold = thresholds[np.argmax(tpr - fpr)]

This method selects the threshold that maximizes the difference between True Positive Rate (TPR) and False Positive Rate (FPR), improving model sensitivity and specificity.

This threshold tuning improves classification accuracy compared to the default 0.5 threshold.

The model was evaluated using:

• ROC Curve
• Optimal Threshold Selection
• Accuracy Score
• Confusion Matrix
• Precision
• Recall
• F1 Score

Using ROC-based threshold optimization ensures better class separation, especially when dealing with imbalanced earthquake categories.

System Architecture

1. Load Earthquake CSV Dataset
2. Store Data in MongoDB
3. Retrieve Data from MongoDB
4. Perform Data Analysis
5. Train Classification Model
6. Evaluate Model
7. Visualize Results

This architecture uses database-driven machine learning workflow.

Implementation and Code

The implementation includes:

• Importing required libraries
• Connecting to MongoDB using MongoClient
• Creating database and collection
• Inserting CSV data into MongoDB
• Retrieving data using collection.find()
• Converting data to DataFrame
• Performing correlation heatmap
• Splitting dataset using train_test_split
• Training classification model
• Predicting test data
• Evaluating using accuracy and classification report
• Plotting confusion matrix

Code:

```
# Imports
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pymongo import MongoClient
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score,
roc_curve
from sklearn.inspection import permutation_importance
```

```
# Import data from MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["mlproject"]
collection = db["earthquake"]
df = pd.DataFrame(list(collection.find({}, {"_id": 0})))
```

```
#Format data and categorize into groups based on coordinates
df["occurred_on"] = pd.to_datetime(df["occurred_on"], errors="coerce")
df["year"] = df["occurred_on"].dt.year
df.drop(columns=["occurred_on"], inplace=True)

df["depth_category"] = pd.cut(df["depth"], bins=[0, 70, 300, 700],
labels=["shallow", "intermediate", "deep"])

def region_bucket(lat, lon):
    if lat > 50 and lon < -150: return "Alaska_Aleutian"
    elif -15 < lat < 10 and 120 < lon < 160: return "Indonesia"
    elif 30 < lat < 45 and 130 < lon < 150: return "Japan"
    elif -25 < lat < -10 and 160 < lon < 180: return "Fiji_Tonga"
    else: return "Other"
```

```
df["tectonic_region"] = df.apply(lambda row: region_bucket(row["latitude"],
row["longitude"]), axis=1)
df["severity"] = df["magnitude"].apply(lambda x: 3 if x >= 6.0 else 2)
```

```
# We remove 'magnitude' because severity is a direct function of it.
X = df.drop(columns=["earthquake_id", "place", "network_id", "magnitude",
"severity"], errors='ignore')
y = df["severity"]
X = pd.get_dummies(X, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
# Train the model
```

```
model = HistGradientBoostingClassifier(max_depth=10, learning_rate=0.1,
max_iter=300, random_state=42)
model.fit(X_train, y_train)
y_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba, pos_label=3)
best_threshold = thresholds[np.argmax(tpr - fpr)]
```

```
#Determine metrics
y_pred_optimal = np.where(y_proba > best_threshold, 3, 2)

print(f"Dataset shape: {df.shape}")
print(f"Optimal Decision Threshold: {best_threshold:.4f}")
print(f"\nAccuracy: {accuracy_score(y_test, y_pred_optimal):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba):.4f}")
print(classification_report(y_test, y_pred_optimal, target_names=["Moderate",
"Strong"]))
```

```
# Sample prediction
result = permutation_importance(model, X_test, y_test, n_repeats=5,
random_state=42)
print("\nTop Important Features:\n", pd.Series(result.importances_mean,
index=X.columns).sort_values(ascending=False).head(5))
sample_proba = model.predict_proba(X_test.iloc[0:1])[:, 1][0]
sample_pred = 3 if sample_proba > best_threshold else 2
print(f"\nSample Prediction - Prob: {sample_proba:.4f} | Result: {'Strong' if
sample_pred == 3 else 'Moderate'}")
```

```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True),
            annot=True,
            cmap="coolwarm",
            fmt=".2f")

plt.title("Feature Correlation Heatmap")
plt.tight_layout()
plt.show()
```

```
roc_auc = roc_auc_score(y_test, y_proba)

# Best threshold index
best_idx = np.argmax(tpr - fpr)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.4f}")
plt.scatter(fpr[best_idx], tpr[best_idx], color='red',
            label=f"Best Threshold = {thresholds[best_idx]:.4f}")

plt.plot([0, 1], [0, 1], linestyle='--')

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Positive Class = Strong Earthquake)")
plt.legend(loc="lower right")
plt.grid()
plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(8, 5))
sns.histplot(df["magnitude"], bins=30, kde=True)
plt.title("Earthquake Magnitude Distribution")
plt.xlabel("Magnitude")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

Output:

The system generates:

• Dataset preview retrieved from MongoDB
• Dataset shape: (23119, 13)
• Correlation heatmap
• Magnitude distribution plot
• ROC curve visualization
• Optimal decision threshold: 0.3275
• Accuracy: 68.21%
• ROC-AUC Score: 0.7235
• Classification report showing precision, recall, and F1-scoreThe optimized threshold improves classification reliability and reduces false positives and false negatives.

```
Dataset shape: (23119, 13)

Optimal Decision Threshold: 0.3275
Accuracy: 0.6821
ROC-AUC: 0.7235
```

	precision	recall	f1-score	support
Moderate	0.81	0.71	0.76	3212
Strong	0.48	0.62	0.54	1412
accuracy			0.68	4624
macro avg	0.65	0.67	0.65	4624
weighted avg	0.71	0.68	0.69	4624

Results from the Model and Graphs

Results from the Model

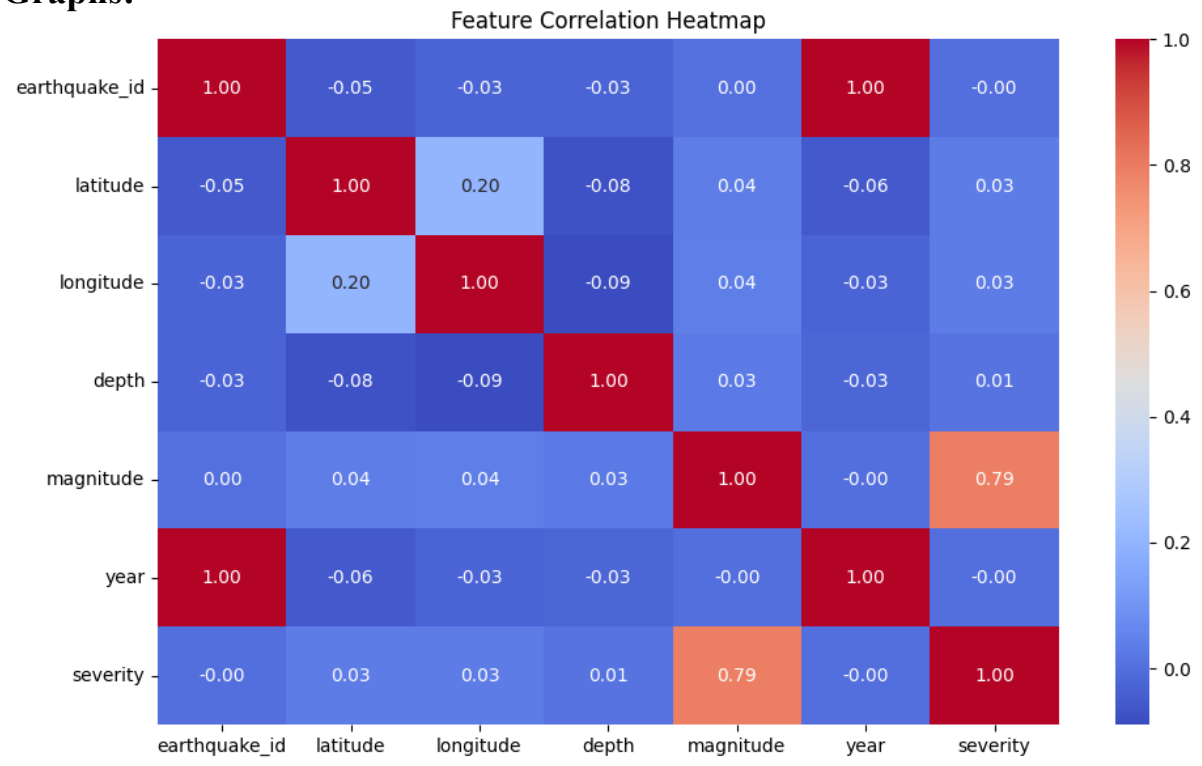
The model performs better in identifying **Moderate earthquakes**, achieving:

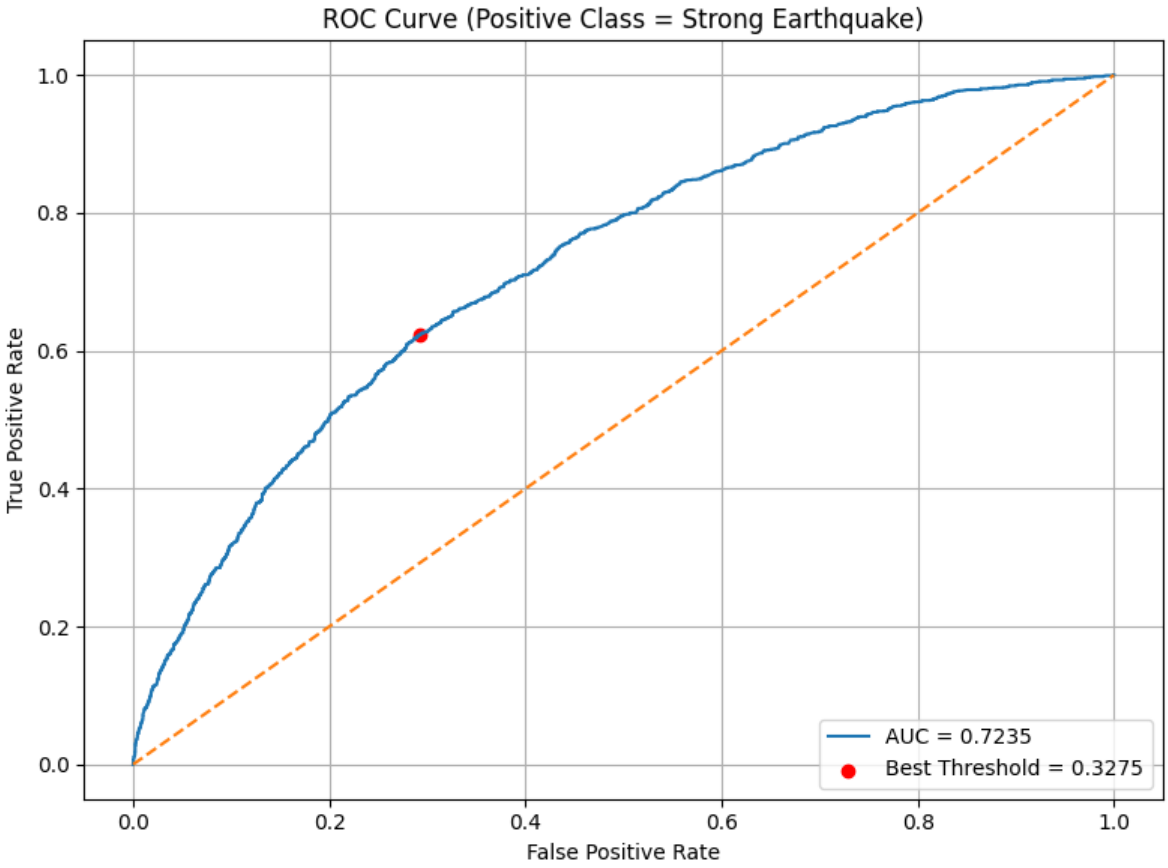
• High precision (0.81)
• Strong F1-score (0.76)

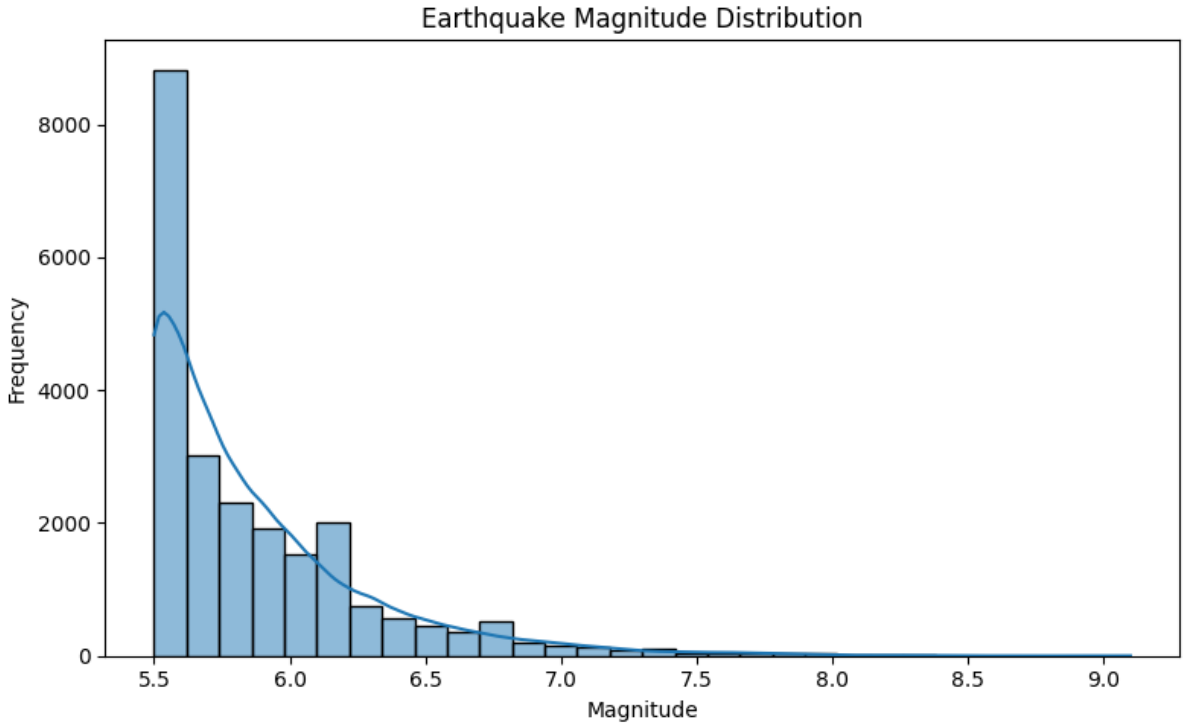
For **Strong earthquakes**, recall (0.62) is higher than precision (0.48), indicating that the model successfully detects many strong events but also produces some false positives.The ROC-AUC score of 0.7235 indicates good class separability and confirms that the HistGradientBoostingClassifier is learning meaningful patterns from the dataset.

The optimized threshold (0.3275) improves detection of strong earthquakes compared to the default 0.5 threshold, which is important in disaster monitoring systems where missing a strong earthquake could have serious consequences.

Graphs:




ROC Curve (Positive Class = Strong Earthquake)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.7235
Best Threshold = 0.3275


Earthquake Magnitude Distribution
Frequency
Magnitude
0
2000
4000
6000
8000
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0

Conclusion

The Earthquake Classification project successfully implemented HistGradientBoostingClassifier integrated with MongoDB for scalable data management. The final dataset contained 23,119 seismic records with 13 features.

Using ROC-based threshold optimization, the model achieved:

- Accuracy of 68.21%
- ROC-AUC score of 0.7235

The model shows strong performance in identifying Moderate earthquakes and reasonable detection capability for Strong earthquakes. The optimized decision threshold improved recall for stronger seismic events, making the system more suitable for real-world disaster monitoring applications.

The integration of MongoDB ensures efficient storage and retrieval of seismic data, while gradient boosting enhances predictive accuracy. The project demonstrates a practical, scalable, and technically sound earthquake classification framework.