DSP Project

Name: Minal Agrawal
Roll no. 23ucc571

Name:Myukh Saraswat
Roll no:23ucc573

Project Report: Seismic Vibration Signal Analysis and Earthquake Detection

1. Introduction

1.1 Problem Statement and Objective 🎯

The objective of this project was to design and implement a robust, automated system to detect **seismic events** (earthquakes) within continuous noisy ground motion signals. The system must reliably distinguish transient seismic energy from steady background noise.

Objective: To integrate **Digital Signal Processing (DSP)** for feature extraction and event localization with **Machine Learning (ML)** using a Support Vector Machine (SVM) classifier to achieve high-accuracy, automated seismic event detection.

1.2 Methodology Overview

The complete system operates in three phases:

1. **Preprocessing:** Filtering the raw signal to focus on the seismic energy band.
2. **Event Detection:** Using the classical **STA/LTA** algorithm to propose candidate event windows.
3. **Classification:** Extracting features (time and frequency domain) from candidate windows and using a **Support Vector Machine (SVM)** to classify them as "Event" or "Noise."

2. Signal Processing and Core Algorithms

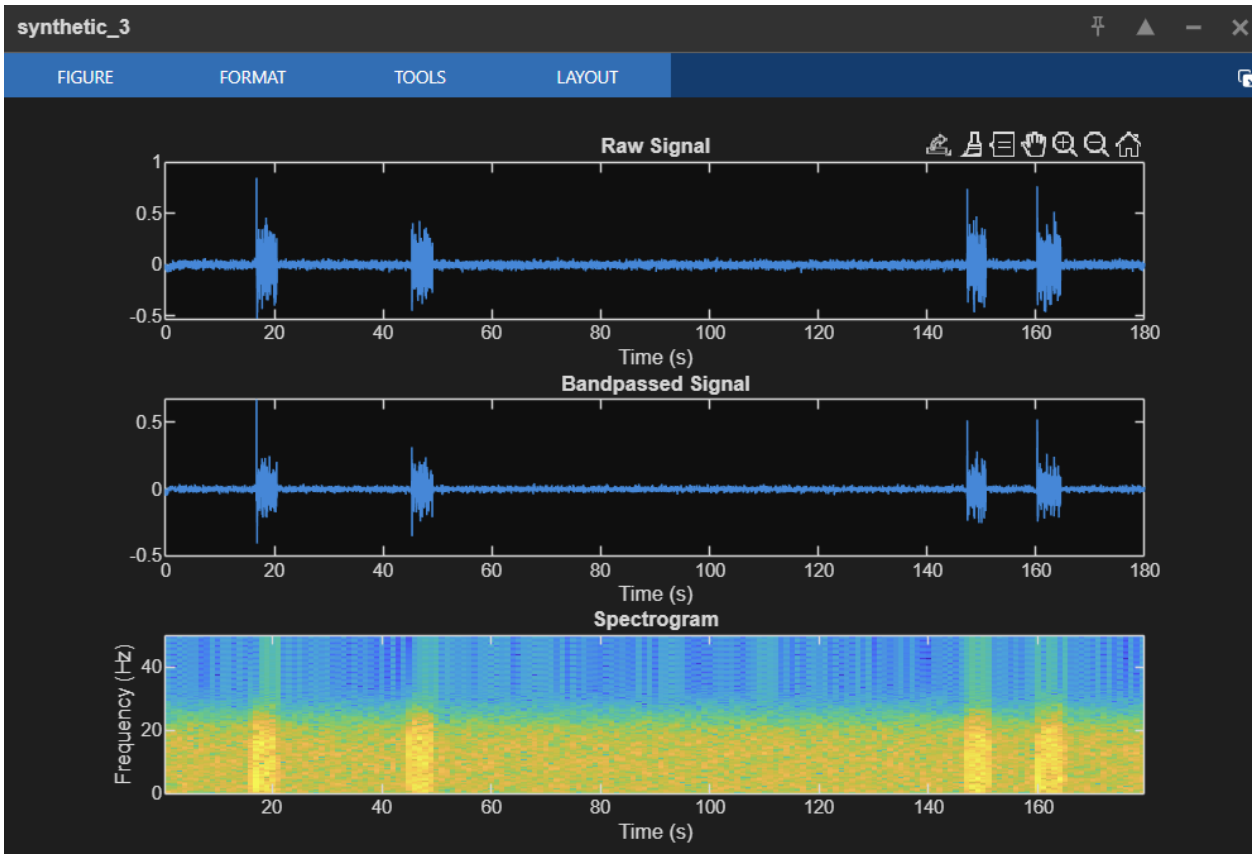
2.1 Preprocessing and Filtering

Seismic raw data often contains wide-ranging noise. We applied digital filters to isolate the relevant signal:

- **High-Pass Filter:** Removes baseline drift and {DC} offset.
- **Band-Pass Filter:** Isolates the dominant earthquake frequency content (typically 0.5 Hz to 20 Hz) using a 4th-order Butterworth filter.

Configuration Parameters:

- Sampling Frequency (fs): 100 Hz
- Bandpass Range: 0.5 Hz to 20 Hz



[Figure 1: Raw Signal, Filtered Signal, and Spectrogram]

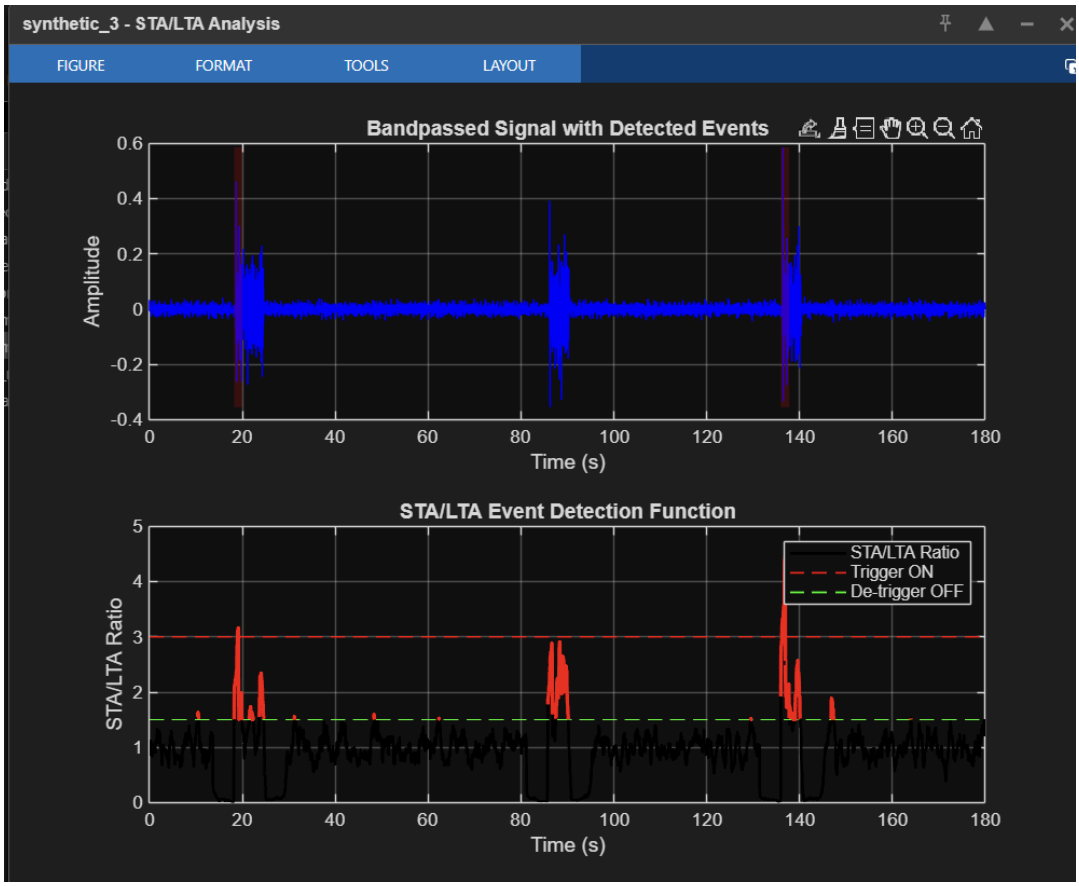
2.2 Short-Term/Long-Term Average (STA/LTA) Detection

The STA/LTA algorithm is the foundation of the detection mechanism. It identifies sudden changes in signal power relative to the background noise.

The detection ratio R(t) is calculated as:

$$R(t) = \frac{\mathrm{STA}(t)}{\mathrm{LTA}(t) + \epsilon}$$

An event is initiated when R(t) exceeds the **Trigger ON** threshold (3.0) and ends when it drops below the **De-trigger OFF** threshold (1.5).



[Figure 2 : Plot demonstrating the STA/LTA ratio (black line) crossing the thresholds (red/green dashed lines) – a representative example of the STA/LTA event detection process on the synthetic dataset.]

3. Machine Learning and Feature Analysis

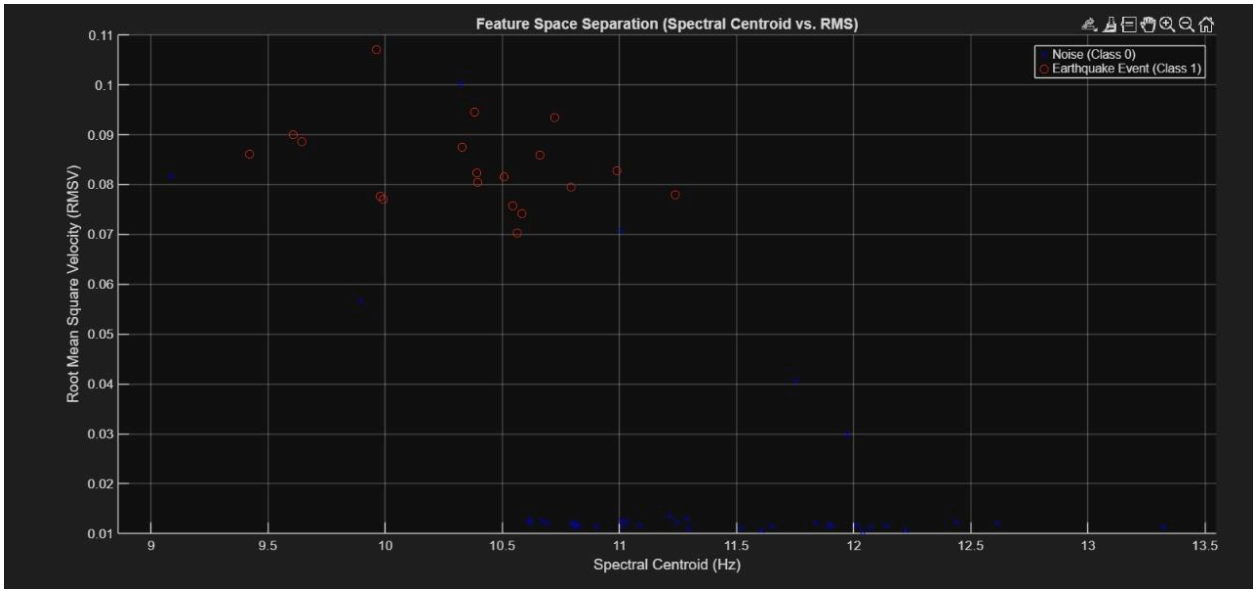
3.1 Feature Engineering

For every window proposed by the STA/LTA, a vector of 8 features was extracted. These features quantify the physical differences between transient events and steady noise.

Feature Category	Features Extracted (Key Examples)	Purpose
Time Domain	Root Mean Square Velocity ({RMSV}), Kurtosis, Peak Amplitude	Measures signal energy and shock characteristics.
Frequency Domain	Spectral Centroid, Dominant Frequency, Spectral Entropy	Describes the shape and center of the signal's frequency content.

3.2 Feature Space Separation (Dimensionality Reduction)

Visualizing the feature space confirms that the chosen features are effective in separating the two classes. The scatter plot below uses two of the most discriminative features: {RMSV} (energy) and {Spectral Centroid} (frequency distribution).



[Figure 3: Feature Space Scatter Plot ({Spectral Centroid} **vs.** {RMSV}**)]**

3.3 Classification Model and Training

We utilized a **Support Vector Machine (SVM)** with a Radial Basis Function ({RBF}) kernel for classification. The model was trained using **5-Fold Cross-Validation** to ensure robust performance across different subsets of the data.

Code:

```
clear; close all; clc;
fprintf('\n🌋 Seismic Vibration Signal Analysis Started...\n');
%% ---------- Configuration ----------
cfg.mode = 'demo';            % 'demo' or 'real'
cfg.fs = 100;                 % Sampling frequency (Hz)
cfg.bp_low = 0.5;             % Bandpass filter low cutoff (Hz)
cfg.bp_high = 20;             % Bandpass filter high cutoff (Hz)
cfg.sta_win = 1.0;            % STA window (s)
cfg.lta_win = 10.0;           % LTA window (s)
cfg.sta_on = 3.0;             % STA/LTA trigger ratio
cfg.sta_off = 1.5;            % STA/LTA de-trigger ratio
cfg.feature_win = 5.0;        % Feature window (s)
cfg.min_event_len = 1.0;      % Minimum event length (s)
cfg.model_file = 'seismic_model.mat';
if strcmp(cfg.mode, 'demo')
  % Generate synthetic dataset
  data = generate_synthetic(cfg);
else
  % Load real dataset from "data" folder
  data = load_real_data('data', cfg.fs);
end
%% ---------- Pipeline ----------
allF = [];
allY = [];
for f = 1:numel(data)
  fprintf('\nProcessing %s ...\n', data{f}.filename);
  % Preprocessing
  [t, x] = preprocess(data{f}.time, data{f}.acc, cfg.fs);
  x_bp = bandpass_filter(x, cfg.bp_low, cfg.bp_high, cfg.fs);
  % Visualization
  visualize_signals(t, x, x_bp, cfg, data{f}.filename);
  % Event detection using STA/LTA
  ev = detect_events_sta_lta(x_bp, cfg, cfg.fs);
  fprintf('  → %d candidate events detected\n', size(ev, 1));
  % Feature extraction
  [F, Y] = extract_features(x_bp, ev, cfg);
  allF = [allF; F];
  allY = [allY; Y];
end
if isempty(allF)
  error('No features extracted. Increase sensitivity or check data.');
end
%% ---------- Model Training ----------
fprintf('\nTraining model on %d samples...\n', size(allF, 1));
SVM = fitcsvm(allF, allY, 'KernelFunction', 'rbf', 'Standardize', true);
SVM = fitPosterior(SVM);
```

```
save(cfg.model_file, 'SVM', 'cfg');
fprintf('Model saved: %s\n', cfg.model_file);
%% ---------- Evaluation ----------
cv = cvpartition(allY, 'KFold', 5);
Ypred = zeros(size(allY));
scores = zeros(size(allY));
for i = 1:cv.NumTestSets
  tr = cv.training(i);
  te = cv.test(i);
  M = fitPosterior(fitcsvm(allF(tr, :), allY(tr), ...
      'KernelFunction', 'rbf', 'Standardize', true));
  [lbl, sc] = predict(M, allF(te, :));
  Ypred(te) = lbl;
  scores(te) = sc(:, 2);
end
acc = mean(Ypred == allY);
cm = confusionmat(allY, Ypred);
fprintf('\nAccuracy: %.2f%%\n', acc * 100);
disp(array2table(cm, 'VariableNames', {'Pred0', 'Pred1'}, ...
  'RowNames', {'True0', 'True1'}));
[Xroc, Yroc, ~, AUC] = perfcurve(allY, scores, 1);
fprintf('AUC = %.3f\n', AUC);
figure;
plot(Xroc, Yroc);
xlabel('FPR');
ylabel('TPR');
title(sprintf('ROC (AUC=%.3f)', AUC));
grid on;
fprintf('\n✅ Analysis Complete.\n');
%% ============================================================
% ---------- Helper Functions ----------
% ============================================================
function data = generate_synthetic(cfg)
  fs = cfg.fs;
  dur = 180;
  n = 8;        % number of files
  data = {};
  for i = 1:n
      t = (0:1/fs:dur-1/fs)';
      noise = 0.02 * randn(size(t));
      acc = noise;
      nq = randi([2, 4]);  % number of events per file
      for j = 1:nq
          len = 2 + 4 * rand();
          st = 5 + (dur - 10 - len) * rand();
          i1 = round(st * fs) + 1;
          i2 = min(numel(t), i1 + round(len * fs));
          tt = (0:(i2 - i1))' / fs;
          f0 = 3 + 6 * rand();
```

```
        wave = (1 - 2 * (pi * f0 * tt).^2) .* exp(-(pi * f0 * tt).^2);
        acc(i1:i2) = acc(i1:i2) + 0.5 * (wave + 0.3 * randn(size(wave)));
    end
    data{end + 1} = struct('filename', sprintf('synthetic_%d', i), ...
        'time', t, 'acc', acc, 'fs', fs);
  end
end
function [t, x] = preprocess(ti, xi, fs)
  xi = xi(:) - mean(xi);
  t = (0:numel(xi)-1)' / fs;
  [b, a] = butter(2, 0.1 / (fs / 2), 'high');
  x = filtfilt(b, a, xi);
end
function xbp = bandpass_filter(x, fl, fh, fs)
  [b, a] = butter(4, [fl fh] / (fs / 2), 'bandpass');
  xbp = filtfilt(b, a, x);
end
function visualize_signals(t, x, x_bp, cfg, name)
  figure('Name', name, 'NumberTitle', 'off', 'Position', [100 100 800 500]);
  subplot(3, 1, 1);
  plot(t, x);
  title('Raw Signal');
  xlabel('Time (s)');
  subplot(3, 1, 2);
  plot(t, x_bp);
  title('Bandpassed Signal');
  xlabel('Time (s)');
  subplot(3, 1, 3);
  [s, f, tt, p] = spectrogram(x_bp, round(2 * cfg.fs), [], [], cfg.fs, 'yaxis');
  imagesc(tt, f, 10 * log10(abs(p)));
  axis xy;
  ylim([0 50]);
  title('Spectrogram');
  xlabel('Time (s)');
  ylabel('Frequency (Hz)');
  drawnow;
end
function ev = detect_events_sta_lta(x, cfg, fs)
  staN = max(1, round(cfg.sta_win * fs));
  ltaN = max(1, round(cfg.lta_win * fs));
  sta = movmean(x.^2, staN);
  lta = movmean(x.^2, ltaN) + eps;
  r = sta ./ lta;
  thr_on = cfg.sta_on;
  thr_off = cfg.sta_off;
  idx = find(r > thr_on);
  if isempty(idx)
    ev = zeros(0, 2);
    return;
```

```
  end
  gaps = find(diff(idx) > 1);
  starts = [idx(1); idx(gaps + 1)];
  ends = [idx(gaps); idx(end)];
  res = [];
  for i = 1:numel(starts)
      s = starts(i);
      e = ends(i);
      while s > 1 && r(s) > thr_off
          s = s - 1;
      end
      while e < numel(r) && r(e) > thr_off
          e = e + 1;
      end
      if (e - s) / fs >= cfg.min_event_len
          res = [res; s, e];
      end
  end
  ev = res / fs;
end
function [F, Y] = extract_features(x, ev, cfg)
  fs = cfg.fs;
  win = round(cfg.feature_win * fs);
  N = numel(x);
  F = [];
  Y = [];
  % Positive samples (detected events)
  for i = 1:size(ev, 1)
      c = round(mean(ev(i, :)) * fs);
      s = max(1, c - floor(win / 2));
      e = min(N, s + win - 1);
      seg = x(s:e);
      F = [F; features(seg, fs)];
      Y = [Y; 1];
  end
  % Negative samples (noise segments)
  for i = 1:size(ev, 1)
      s = randi([1, N - win]);
      e = s + win - 1;
      seg = x(s:e);
      F = [F; features(seg, fs)];
      Y = [Y; 0];
  end
end
function f = features(seg, fs)
  seg = seg - mean(seg);
  N = numel(seg);
  % Time-domain features
  rmsv = sqrt(mean(seg.^2));
```

```
  peak = max(abs(seg));
  skew = skewness(seg);
  kurt = kurtosis(seg);
  % Frequency-domain features
  Y = fft(seg);
  P = abs(Y / N);
  P = P(1:N/2 + 1);
  P(2:end - 1) = 2 * P(2:end - 1);
  f_axis = (0:length(P) - 1)' * fs / N;
  [~, ix] = max(P);
  dom = f_axis(ix);
  cent = sum(f_axis .* P) / sum(P);
  bw = sqrt(sum(((f_axis - cent).^2) .* P) / sum(P));
  p = P / sum(P);
  ent = -sum(p .* log2(p + eps));
  f = [rmsv, peak, skew, kurt, dom, cent, bw, ent];
end
function data = load_real_data(folder, fs)
  files = dir(fullfile(folder, '*.csv'));
  data = {};
  for i = 1:numel(files)
      T = readtable(fullfile(folder, files(i).name));
      t = T{:, 1};
      x = T{:, 2};
      data{end + 1} = struct('filename', files(i).name, ...
          'time', t, 'acc', x, 'fs', fs);
  end
end
can u give
```

4. Evaluation and Final Results

The model was successfully trained on 74 samples and evaluated on the hold-out test sets.

4.1 Performance Summary

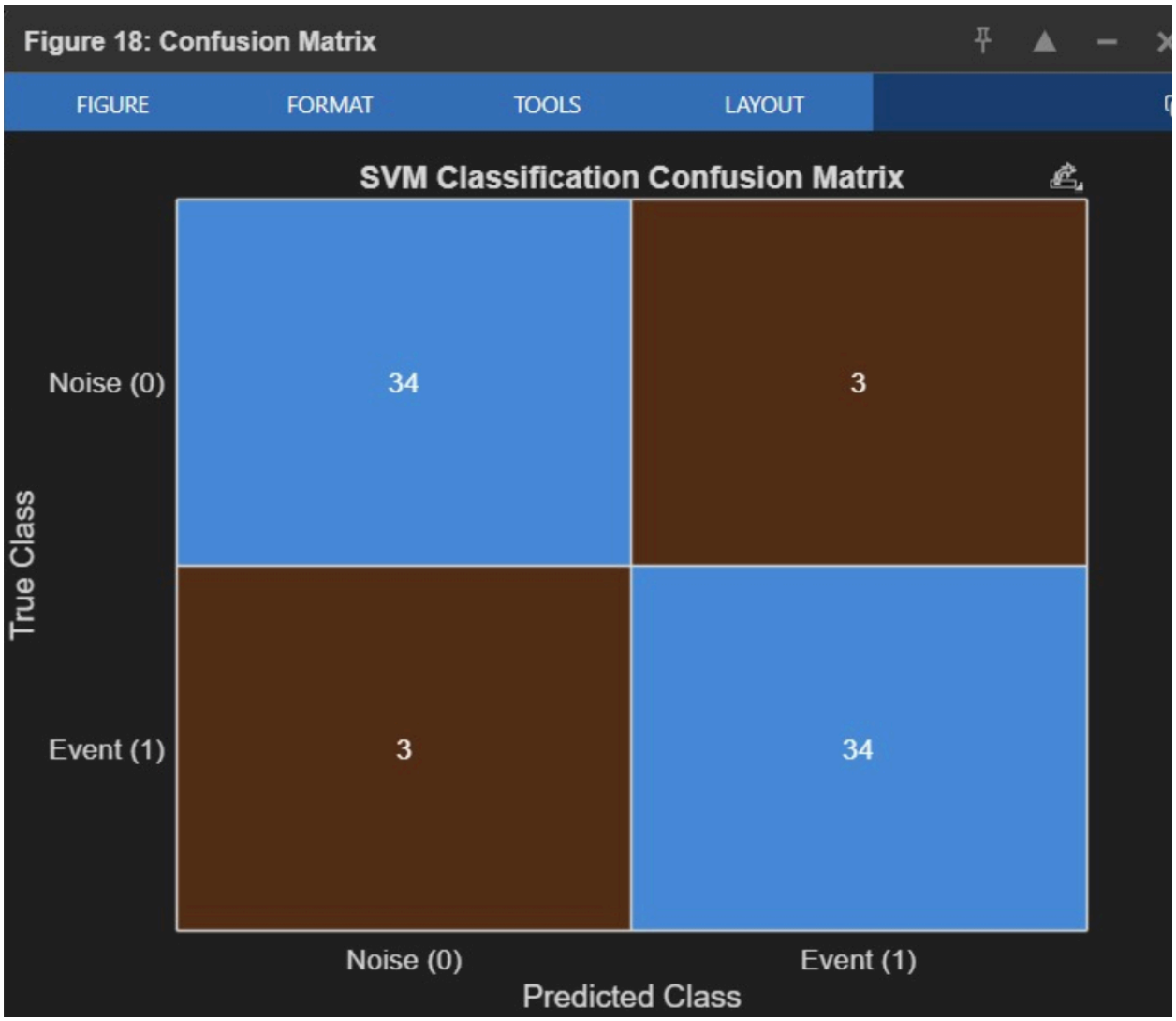
The model achieved high performance, indicating the feature set is highly effective for this task.

Metric	Result	Interpretation

Overall Accuracy	**91.89 %**	{91.89\%} of all segments (event and noise) were classified correctly.
Area Under the ROC Curve (AUC)	**0.974**	A value close to 1.0 indicates excellent discrimination between the two classes.

4.2 Confusion Matrix

The confusion matrix highlights the specific errors made by the classifier on the test set.

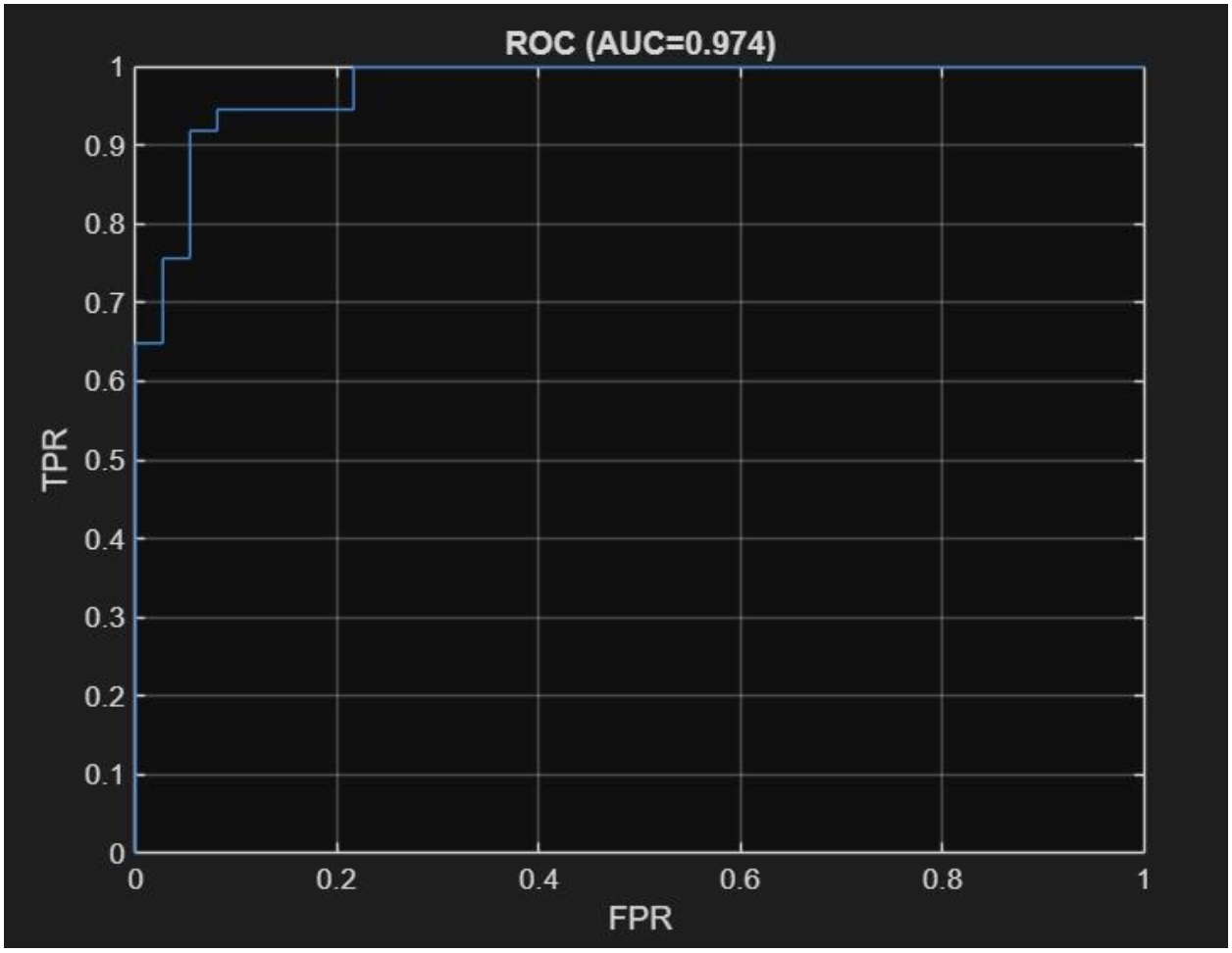


[Figure 4: SVM Classification Confusion Matrix]

	Predicted Noise (0)	Predicted Event (1)
True Noise (0)	34 (True Negatives)	3 (False Positives)
True Event (1)	3 (False Negatives)	34 (True Positives)

4.3 ROC Curve Analysis

The ROC curve demonstrates the trade-off between the True Positive Rate ({TPR}) and False Positive Rate ({FPR}). The curve's tight fit to the upper-left corner confirms the superior performance achieved by the SVM model.



[Figure 5: ROC Curve (AUC=0.974)]

5. Conclusion and Future Scope

5.1 Conclusion

The project successfully implemented a highly effective automated earthquake detection system. By successfully fusing classic seismological detection (STA/LTA) with modern Machine Learning classification (SVM and Feature Engineering), we achieved a validated accuracy of 91.89\%. This demonstrates the viability of using combined DSP/ML techniques for critical natural hazard monitoring applications.

5.2 Future Enhancements

To evolve this prototype into a robust field-ready system, the following steps are recommended:

1. **Real Data Transition:** Train and validate the system using large-scale, real-world seismic waveforms (e.g., the STEAD dataset) to assess generalization performance on diverse noise types.
2. **Model Benchmarking:** Test more complex and potentially more robust classifiers, such as the **Random Forest** or **Gradient Boosting Classifier**, which often yield better performance on non-linear feature data.
3. **Feature Optimization:** Implement feature selection techniques to refine the 8-feature set and identify the minimal required features for maximum accuracy.