Assignment 3 Report

Advanced Artificial Intelligence
Assignment 3 (Hands on ML Assignment)

Submitted By: 25k-7627 (Sanjinee)

1. Problem Statement

The problem is framed as a multi-class classification and an anomaly detection challenge based on time-series telemetry data. The goal is to accurately distinguish between the three defined classes using a diverse set of sensor and state features, specifically:

1. **Multi-Class Classification:** Employing supervised learning techniques, namely **Long Short-Term Memory (LSTM)** and **Support Vector Classification (SVC)**, to classify a given set of operational data as Normal, DoS Attack, or Malfunction.
2. **Anomaly Detection:** Utilizing an unsupervised learning approach, specifically a **Variational Autoencoder (VAE)**, to model the "Normal" operational profile and identify DoS Attacks and Malfunctions as significant deviations (anomalies) from this learned baseline.

The successful outcome is defined by high performance across key metrics (Precision, Recall, F1-Score, and Accuracy) and a clear, interpretable understanding of which drone features are most indicative of each threat state.

2. Data Preprocessing Documentation with Visualizations

2.1 Data Loading and Initial Inspection

The raw dataset, sourced from the 'Dos-Drone' directory, consists of multiple CSV files, each representing a segment of drone operation labeled implicitly by the file prefix (e.g., 'normal', 'dos', 'malfunction'). A total of **87,417** raw data points were loaded across all files, with approximately **80** features collected from various drone sensors and system logs, including GPS, battery, IMU, and RC controls.

Class Distribution

The raw class distribution revealed a significant imbalance, which is common in real-world security datasets:

- **Normal:** 49,800 instances
- **DoS_Attack:** 19,587 instances
- **Malfunction**: 18,030 instances
  This imbalance necessitates careful selection of evaluation metrics (favoring F1-Score over raw Accuracy) and may warrant class-weighting or resampling techniques during model training. The count of each class was visualized using a bar plot to confirm the distribution.

2.2 Data Cleaning and Time Synchronization

Data cleaning was crucial due to the nature of telemetry logs, which often contain missing values, infinite readings, and unsynchronized timestamps.

1. **Timestamp Handling:** The primary timestamp column, identified as setpoint_raw-global_Time, was converted to a uniform Timestamp index. Rows with missing timestamps were dropped, although this step resulted in zero row loss in the provided output, suggesting a clean time-series.
2. **Duplicate Timestamps:** Instances sharing identical timestamps were aggregated using the **median** for numeric features and the **mode** for categorical/non-numeric features, resulting in a dataset of **84,895** unique time-indexed rows.

3. **Handling Non-Finite Values:** All numeric columns were checked for NaN (Not a Number) and infinite (Inf, -Inf) values.
    - Infinite values were replaced with NaN.
    - All remaining NaN values were imputed using the **median** of their respective columns.

2.3 Feature Engineering

Two key feature engineering steps were performed to capture meaningful physical and navigational relationships:

1. **Haversine Distance:** A helper function was used to calculate the Haversine distance between coordinate pairs, which is essential for determining the distance between the drone's current location and its intended target, or the distance traveled between two time steps.
2. **Quaternion to Euler Conversion:** Quaternion data from the Inertial Measurement Unit (IMU) sensors (imu-data_orientation.x/y/z/w) was converted into the more physically intuitive **Roll, Pitch, and Yaw** (Euler angles). This added three new features (imu_roll, imu_pitch, imu_yaw), increasing the total feature count to **84** for the final processed dataset.

2.4 Feature Scaling and Transformation

The choice of scaling method was tailored to the specific model type:

- **SVC and Traditional ML Models:** A **Signed Log Transformation** was applied prior to scaling. This transformation compresses the scale of features with highly skewed, heavy-tailed distributions (like those expected in attack traffic) while preserving the sign. Subsequently, the data was scaled using **StandardScaler** to achieve a mean of 0 and a standard deviation of 1.
- **Deep Learning Models (LSTM, VAE):** The data was scaled using **StandardScaler**. This is crucial for deep learning to ensure stable gradient flow and faster convergence, as these models are highly sensitive to feature magnitudes.

2.5 Visualizations for Preprocessing

Visualizations were generated to inspect the data quality and feature characteristics:

1. **Distribution Plots (Histograms & KDE):** Histograms and Kernel Density Estimates (KDE) were used to examine the distribution of individual numeric features. This step confirmed the presence of highly skewed and multi-modal distributions, validating the necessity of the Signed Log Transformation.
2. **Boxplots:** Boxplots were generated across all numerical features to visualize the presence of outliers, which were numerous due to the inclusion of attack and malfunction data. This visualization confirmed the decision to use a robust scaler/transformation approach.
3. **Time-Series Plots:** Plots of key features over time were created, grouped by Source_File and Class, providing visual evidence of the attack and malfunction signatures (e.g., constant or zeroed readings during a DoS attack).
4. **Correlation Analysis:** A heatmap was generated to show the correlation matrix between the features. This analysis revealed features highly correlated with the target variable, such as setpoint_raw-global_longitude and global_position-global_longitude

2.6 Data Splitting

The cleaned and scaled dataset was split into training, validation, and test sets. The split ratio was targeted as 60% Train, 20% Validation, and 20% Test. The final dataset shapes were:

- **Training Set (X_train, y_train):** (2,547, 84)
- **Validation Set (X_val, y_val):** (849, 84)
- **Test Set (X_test, y_test):** (849, 84)

3. Model Architectures and Hyperparameter Tuning Results

Three distinct models were selected to address the security problem, providing a comparison between traditional kernel-based methods, advanced sequence modeling, and unsupervised anomaly detection.

3.1 Long Short-Term Memory (LSTM) - Sequence Classification

1. LSTM Model Architectures

Three main LSTM model architectures were explored:

1.1. Initial LSTMClassifier (Standard LSTM for sequence classification.)

Structure:

- nn.LSTM layer: Takes input sequences and outputs the hidden states.
- nn.Linear layer (self.fc): Maps the final hidden state (hn[-1]) from the LSTM to the number of output classes.

Key Characteristic: Relies solely on the final hidden state for classification, potentially losing information from earlier time steps.

1.2. Modified LSTMClassifier for Random Search (Type: Enhanced standard LSTM.)

Structure:

- nn.LSTM layer: Similar to the initial version, but bidirectional option was added.
- Activation Function: An activation function (nn.ReLU, nn.Sigmoid, or nn.Tanh) is applied to the output of the LSTM before the final linear layer.
- nn.Linear layer (self.fc): Maps the (activated) last hidden state (or concatenation of forward/backward last hidden states for bidirectional) to the number of output classes.

Key Characteristic: Introduced bidirectional capability and a configurable activation function post-LSTM to explore more complex temporal dependencies and non-linearities. It uses the output of the last timestep (out[:, -1, :]) after the LSTM layer.

1.3. LSTMWithAttention Model (Type: LSTM with an additive attention mechanism.)

Structure:

- nn.LSTM layer: Similar to the modified version, supporting bidirectional processing.
- Attention Mechanism (Attention class):
- Learns to assign weights to each hidden state across the entire sequence.
- Uses nn.Linear layers (self.W, self.v) and F.softmax to compute attention scores.
- Calculates a "context vector" by taking a weighted sum of all hidden states, allowing the model to focus on the most relevant parts of the input sequence.
- Activation Function: Applied to the context vector.
- nn.Linear layer (self.fc): Maps the attention-weighted context vector to the output classes.

Key Characteristic: Addresses the limitation of standard LSTMs by not relying solely on the final hidden state. It uses attention to dynamically select and weight relevant information from all hidden states across the sequence, improving the model's ability to handle long-term dependencies and identify crucial temporal features.

2. Hyperparameter Tuning Results

2.1. Initial LSTMClassifier Training

Configuration:

- INPUT_SIZE: 74
- HIDDEN_SIZE: 128
- NUM_LAYERS: 2
- DROPOUT: 0.3
- EPOCHS: 30
- LR: 1e-3
- Criterion: nn.CrossEntropyLoss with balanced class weights.
- Optimizer: Adam

Performance:

- train_loss decreased significantly, reaching 0.1562 by Epoch 30.
- train_acc increased dramatically, reaching 0.9270 by Epoch 30.
- val_loss steadily increased, reaching 4.2303 by Epoch 30.
- val_acc hovered around 0.33-0.39.

Finding: Severe overfitting was observed. High training accuracy coupled with low validation accuracy indicated the model was memorizing the training data rather than learning generalizable patterns. The model struggled to generalize to unseen data.

2.2. Second LSTMClassifier Training (Reduced Capacity, Lower LR)

Configuration:

- INPUT_SIZE: 74
- HIDDEN_SIZE: 64 (reduced from 128)

- NUM_LAYERS: 2
- DROPOUT: 0.4 (increased from 0.3)
- EPOCHS: 30
- LR: 0.0001 (reduced from 0.001)
- Criterion: nn.CrossEntropyLoss with balanced class weights.
- Optimizer: Adam

Performance:

- train_loss reached 1.0270 by Epoch 30.
- train_acc reached 0.4184 by Epoch 30.
- val_loss reached 1.1804 by Epoch 30.
- val_acc reached 0.3103 by Epoch 30.

Finding: While overfitting was somewhat mitigated (train/val metrics were closer), the overall performance remained poor, with validation accuracy still low. The model appeared to be underfitting or struggling to learn effectively with the reduced capacity and lower learning rate.

2.3. Random Search for LSTMClassifier Hyperparameters (40 Trials)

- Search Space: Varied num_layers (1-3), hidden_size (32-256), dropout (0.0-0.5), lr (1e-3, 1e-4, 1e-5), batch_size (16-128), epochs (50-200), optimizer (Adam, RMSprop, SGD), activation (tanh, relu, sigmoid), and bidirectional (True/False).
- Performance: The random search trials showed mixed results. Many configurations resulted in low validation accuracy (often around 0.20-0.56). Some trials showed training accuracy increasing but validation accuracy stagnating or fluctuating, indicating continued challenges with generalization.

Example of "better" trials (relatively, still poor overall):

- Trial 2: hidden_size=256, dropout=0.0, lr=1e-5, batch_size=16, epochs=200, optimizer=SGD, activation=tanh, bidirectional=False -> train_acc=0.5728, val_acc=0.5624 (This configuration seemed to predict the majority class).
- Trial 9: hidden_size=256, dropout=0.2, lr=1e-5, batch_size=32, epochs=100, optimizer=SGD, activation=tanh, bidirectional=False -> train_acc=0.4547, val_acc=0.4722.
- Trial 25: hidden_size=32, dropout=0.0, lr=0.0001, batch_size=16, epochs=100, optimizer=Adam, activation=sigmoid, bidirectional=True -> train_acc=0.5094, val_acc=0.5550.

Finding: Even with extensive random search, consistently high and stable validation accuracy remained elusive for the LSTMClassifier. Many models converged to predicting the majority class (as suggested by validation accuracies around 0.56, which is the proportion of the largest class). This suggested that the basic LSTM architecture, even with various hyperparameter combinations, was not robust enough for this imbalanced multiclass problem, or the reliance on the last hidden state was problematic.

3. Models Evaluation and Comparison

- The evaluation was performed on the LSTMWithAttention model, specifically the one from the third run, which had the best (lowest) validation loss among the attention models.
- 3.1. Final Test Accuracy:
- Final Test Accuracy: 0.5513
- Observation: This test accuracy matches the validation accuracy and is very close to the proportion of the majority class in the test set (446 / 809 ≈ 0.5513), confirming the model's bias towards the majority class.

3.2 Support Vector Classification (SVC) - Multi-Class Classification

I explored two main SVM tuning phases, each with distinct preprocessing and hyperparameter search strategies

Phase 1: Initial Tuning with Signed Log Transformation and StandardScaler

- Preprocessing: Applied a signed logarithmic transformation (np.sign(X) * np.log1p(np.abs(X))) followed by StandardScaler to X_train, X_val, and X_test. SMOTE was then applied only to the training data to address class imbalance, balancing Counter({2: 1464, 0: 596, 1: 487}) to Counter({1: 1464, 2: 1464, 0: 1464}).
- Model: SVC initialized with class_weight='balanced'.
- Hyperparameter Grid: {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto']}.
- Tuning Results: GridSearchCV identified best parameters as {'C': 0.1, 'gamma': 'auto'} with a best cross-validation F1-weighted score of 1.0.
- Issues: The notebook explicitly noted that this perfect score was a strong indicator of data leakage or an overly simplistic problem, making the results unreliable for generalization. It also mentioned numerical instability during fitting.

Phase 2: Comprehensive Tuning with RobustScaler

- Preprocessing: RobustScaler was applied directly to the original X_train, X_val, and X_test data (after confirming no NaNs). SMOTE was also applied to the training data, as in Phase 1.
- Model: SVC initialized with class_weight='balanced'.
- Hyperparameter Grid: A comprehensive grid was used, exploring different kernels (linear, rbf, poly, sigmoid) with varying C, gamma, and degree parameters. For example:
- {'kernel': ['linear'], 'C': [0.1, 1, 10, 100]}
- {'kernel': ['rbf'], 'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1]}
- {'kernel': ['poly'], 'C': [0.1, 1, 10, 100], 'degree': [2, 3, 4, 5], 'gamma': ['scale', 'auto', 0.001, 0.01, 0.1]}
- {'kernel': ['sigmoid'], 'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto']}
- Tuning Results: GridSearchCV found best parameters as {'C': 0.1, 'gamma': 'auto', 'kernel': 'sigmoid'} with a best cross-validation F1-weighted score of 0.855.
- Issues: Despite finding a realistic best score, this phase also encountered FitFailedWarning (432 out of 580 fits failed) due to non-finite dual coefficients, indicating numerical instability for certain parameter combinations.

Model Evaluation and Comparison

Evaluation of Initial Tuned Model (with Signed Log + StandardScaler)

- Validation Performance: Achieved perfect accuracy and F1-weighted scores of 1.0. The classification report showed 100% precision, recall, and f1-score for all classes (DoS_Attack, Malfunction, Normal). The confusion matrix also showed perfect classification.
- Test Performance: Also achieved perfect accuracy and F1-weighted scores of 1.0, with identical perfect metrics across all classes and a perfect confusion matrix.
- Comparison/Conclusion: The notebook concluded that this perfect performance was highly suspicious, indicating potential data leakage or an overly simplistic problem, rendering these results unreliable for real-world generalization.

Evaluation of Best Comprehensive Model (with RobustScaler)

- Validation Performance: Accuracy: 0.837, F1-weighted: 0.849.
- DoS_Attack: F1-score 0.71 (Precision: 0.70, Recall: 0.73)
- Malfunction: F1-score 0.81 (Precision: 0.73, Recall: 0.90)
- Normal: F1-score 0.90 (Precision: 0.95, Recall: 0.86)
- Confusion Matrix showed some misclassifications, particularly for 'DoS_Attack' and 'Normal' classes.
- Test Performance: Accuracy: 0.846, F1-weighted: 0.85.
- DoS_Attack: F1-score 0.75 (Precision: 0.72, Recall: 0.79)
- Malfunction: F1-score 0.81 (Precision: 0.76, Recall: 0.87)
- Normal: F1-score 0.90 (Precision: 0.95, Recall: 0.86)
- Test confusion matrix showed similar patterns of misclassification as the validation set.

Comparison/Conclusion: This model, obtained with C=0.1, gamma='auto', and kernel='sigmoid', provided robust and realistic performance. It demonstrated good generalization across validation and test sets and achieved balanced F1-scores across all three classes, confirming its ability to classify different types of events effectively. This was ultimately selected as the best model due to its valid and realistic performance.

Overall Best Model Selection The SVC model derived from the comprehensive GridSearchCV using RobustScaler and SMOTE ({'C': 0.1, 'gamma': 'auto', 'kernel': 'sigmoid'}) was chosen as the best performing model. Its consistent and realistic performance on unseen data distinguished it from the initially perfect but unreliable results.

3.3 Variational Autoencoder (VAE) - Anomaly Detection

The Variational Autoencoder (CustomVAE) was constructed with a flexible architecture, defined by the following components:

- Encoder: Comprises Dense layers with relu or leaky_relu activation, followed by two output Dense layers for z_mean and z_log_var (representing the mean and log-variance of the latent space distribution).

- Reparameterization Trick: A Lambda layer implementing the reparameterization trick was used to sample from the latent space, enabling backpropagation through the stochastic sampling process.
- Decoder: Mirrors the encoder structure, using Dense layers with relu or leaky_relu activation, culminating in a final Dense output layer with sigmoid activation to reconstruct the input data. The output dimension matches the input_dim.
- Custom Loss Function: The VAE utilizes a custom loss function that is the sum of two components:
- Reconstruction Loss: Mean Squared Error (MSE) between the original input and the reconstructed output.
- KL Divergence Loss: Measures the divergence between the learned latent distribution and a standard normal distribution, weighted by a beta hyperparameter.

2. Hyperparameter Tuning Results

A random search was performed on a predefined hyperparameter grid to find the optimal VAE configuration. The search space included:

- latent_dim: [2, 4, 8, 16]
- encoder_layers: [(64, 32), (128, 64), (64,)]
- decoder_layers: [(32, 64), (64, 128), (64,)]
- learning_rate: [0.001, 0.0005, 0.0001]
- batch_size: [32, 64, 128]
- beta: [0.1, 0.5, 1.0, 2.0]
- activation: ['relu', 'leaky_relu']
- epochs: [50, 100]

After evaluating 20 random combinations, the best-performing hyperparameters that yielded the lowest validation loss were identified as:

- latent_dim: 8
- encoder_layers: (128, 64)
- decoder_layers: (64, 128)
- learning_rate: 0.001
- batch_size: 64
- beta: 1.0
- activation: 'relu'
- epochs: 100
- The best validation loss achieved was approximately 0.0776.

3. Model Evaluation and Comparison

- The VAE model configured with the best hyperparameters was evaluated on the X_test_scaled dataset.
- The Test Loss for the best VAE model was approximately 1.1380.

4. **Explainable AI analysis with interpretations**

- **Global Feature Importance (SHAP):** Across both SVC and VAE models, system resource usage (e.g., **Used_RAM_MB, CPU_Percent**) and control/motion features (e.g., **rc-out_channels_2, vfr_hud_heading**) consistently emerged as the strongest drivers of predictions, confirming strong alignment with real-world robotic system behavior.
- **Non-Linear Model Behavior: SHAP dependence plots and Partial Dependence Plots (PDPs)** revealed clear **non-linear relationships**, where class probabilities and anomaly scores often changed sharply after specific thresholds (e.g., **battery_header.seq**), validating the models' ability to capture complex dynamics.
- **Class-Specific Feature Contributions:** The same feature was shown to have **opposite effects across classes**, where a single value could increase **DoS_Attack** probability while decreasing **Normal**, demonstrating strong **class-conditional behavior**.
- **Local Explainability & Stability (SHAP & LIME): Force plots, waterfall plots, and LIME explanations** provided reliable instance-level interpretations, showing that predictions are driven by **combinations of abnormal feature values rather than isolated features**, ensuring high diagnostic value.
- **Time-Series Awareness: Time-averaged SHAP analysis** captured sustained feature influence across sequences, proving essential for understanding **temporal patterns in LSTM-based predictions** rather than only instantaneous effects.
- **Feature Interactions:** SHAP dependence plots confirmed that **feature interactions exist**, meaning the effect of one feature changes depending on another, strengthening the justification for using non-linear models.
- **Anomaly Detection Interpretation (VAE):** The VAE successfully used **reconstruction error as a meaningful anomaly score**, where key indicators such as **RSSI_Quality, rc-out_channels_2, and vfr_hud_airspeed** consistently explained abnormal behavior.
- **Thresholds & Contextual Anomalies:** PDPs and correlation plots revealed **critical thresholds** where reconstruction error sharply increases, confirming that anomalies are **contextual and multi-feature driven**.
- **Correlation & Redundancy:** Several **geographic position features showed moderate correlation (~0.46)** with the target, indicating **potential redundancy** and opportunities for **feature selection and model simplification**.