MLEng Assignment 3: Advanced Classification for Time Series Data

Highlight Reel

1 Objectives

By the end of this assignment, you will be able to:

- Understand the structure and approach of time-series classification
- Explore different classification algorithms beyond Random Forest
- Compare classifier performance using appropriate metrics
- Apply feature engineering techniques to improve classification results
- Implement and evaluate a superior classifier for the given dataset

2 Background

In previous assignments, you worked with tracking data to extract and visualize various features. Now, we'll make the transition to using this data for classification. Classification is a supervised learning task where we predict categorical labels based on input features. In our context, we'll be classifying time segments of motion data into different categories.

Time series classification presents unique challenges compared to standard classification problems because the temporal ordering of data points contains important information. A model that understands how features evolve over time will often outperform one that treats each data point independently.

3 Understanding the Provided Classifier

The file `time_classification.py` contains a baseline classifier implementation using Random Forest. Let's understand its overall structure and approach:

3.1 Data Preparation

The classifier begins by:

- Loading the tracking data (positions, dimensions, and effort measurements)
- Processing missing values in the effort column through interpolation
- Loading target labels that specify the classification for each frame
- Merging the tracking data with the target labels
- Standardizing the features to ensure all have similar scales

3.2 Time Series Feature Engineering

Rather than treating each frame independently, the classifier creates "lag features" by:

- Taking a window of consecutive frames (using a window size of 2)
- Creating new features that include values from previous time steps
- This allows the model to capture temporal patterns and changes over time

For example, if we have features $x_1, x_2, \ldots, x_n$ for each frame, the lag features for a window size of 2 would include $x_1, x_2, \ldots, x_n$ for the current frame as well as $x_1, x_2, \ldots, x_n$ for the previous frame.

3.3 Training and Evaluation

The classifier then:

- Splits the data chronologically (to respect the time series nature)
- Trains a Random Forest classifier on the first 70% of the data
- Predicts labels for the remaining 30%
- Evaluates performance using precision, recall, and F1-score metrics
- Outputs the predictions to a CSV file

3.4 Limitations of the Current Approach

While Random Forest is a robust classifier, it has some limitations for time series data:

- It doesn't inherently model the sequential nature of the data
- It relies on the manually created lag features to capture temporal relationships
- It treats each feature as independent, potentially missing complex interactions over time
- It may not capture long-range dependencies in the data

4 Your Task: Implement a Superior Classifier

Your task is to implement a classifier that outperforms the provided Random Forest baseline. You are free to explore any classification algorithm or approach, but your implementation should demonstrate a clear understanding of the unique challenges of time series classification.

4.1 Required Steps

1. Run the provided `time_classification.py` to understand the baseline performance
2. Analyze the features and target labels to gain insights about the classification task
3. Research and select at least one alternative classification algorithm
4. Implement your chosen algorithm(s)
5. Compare your results with the baseline using appropriate metrics
6. Document your approach, findings, and recommendations

4.2 Alternative Classifier Options

You may consider algorithms such as:

- **Support Vector Machines (SVM)**: Might better handle the high-dimensional feature space created by lag features
- **Gradient Boosting Machines**: Algorithms like XGBoost, LightGBM, or CatBoost often outperform Random Forest
- **Recurrent Neural Networks (RNN/LSTM/GRU)**: Specifically designed for sequential data, these models inherently capture temporal dependencies
- **Temporal Convolutional Networks (TCN)**: Combines convolutional operations with dilated kernels to capture long-range dependencies
- **Transformer models**: State-of-the-art for many sequence modeling tasks
- **Dynamic Time Warping (DTW)**: A specialized approach for time series that measures similarity between temporal sequences
- **Ensemble methods**: Combining multiple diverse classifiers may yield better results

4.3 Beyond Algorithm Selection

Remember that achieving superior performance often requires more than just switching algorithms. Consider:

- **Feature engineering**: Can you create better features that capture the motion characteristics?
- **Parameter tuning**: Most algorithms have hyperparameters that significantly affect performance
- **Data preprocessing**: Different normalization or transformation techniques might help
- **Handling class imbalance**: If some classes are underrepresented, special techniques may be needed
- **Cross-validation strategies**: Appropriate for time series (e.g., time series cross-validation)

5 Tools and Libraries

You're free to use any of the following libraries:

- **scikit-learn**: For traditional machine learning algorithms and evaluation metrics
- **PyTorch or TensorFlow**: For implementing neural network-based approaches
- **XGBoost, LightGBM, CatBoost**: For gradient boosting algorithms
- **tslearn**: Specialized tools for time series analysis
- **sktime**: A unified framework for machine learning with time series

Submit the improved classifier Python script to your repository, and post visualizations and metrics to the assignment thread in our Discord.

6 Tips for Success

- Start by thoroughly understanding the baseline classifier's approach
- Visualize the data and class distributions to gain insights
- Research algorithms specifically designed for time series classification
- Implement simple improvements first before moving to more complex approaches
- Consider the trade-off between model complexity and performance gain

7 Deadline

Submit all required materials by our next workshop.

Good luck!