Human Activity Recognition (HAR)

Technical Implementation Report

Technical Research Team

December 19, 2024

Contents

1 Technology Stack Overview

Figure 1: Technology Stack Mind Map

2 Development Environment

2.1 Required Software

- **Python Environment:**

```
    Python 3.8+
    Anaconda/Miniconda
    Jupyter Notebook/Lab
```

- **Version Control:**

```
    Git 2.x
    GitHub Desktop (optional)
```

- **Containerization:**

```
    Docker Desktop
    NVIDIA Docker (for GPU support)
```

2.2 Python Dependencies

```
numpy==1.21.0
pandas==1.3.0
scikit-learn==0.24.2
tensorflow==2.8.0
keras==2.8.0
matplotlib==3.4.2
seaborn==0.11.1
plotly==5.1.0
```

3 Data Processing Pipeline

Figure 2: Data Processing Pipeline

4 Model Architectures

4.1 Classical ML Pipeline

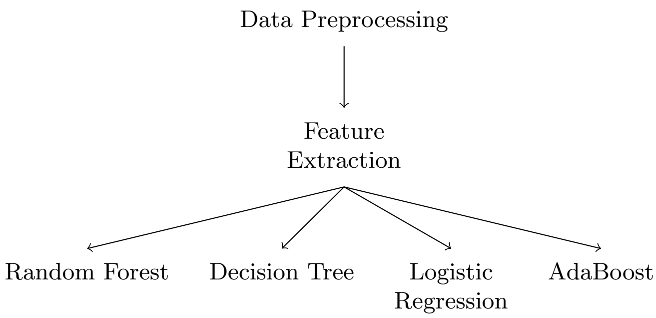

Figure 3: Classical ML Model Pipeline

4.2 Deep Learning Architecture

Figure 4: 1D-CNN Architecture

5 Performance Analysis

5.1 Model Accuracy Comparison

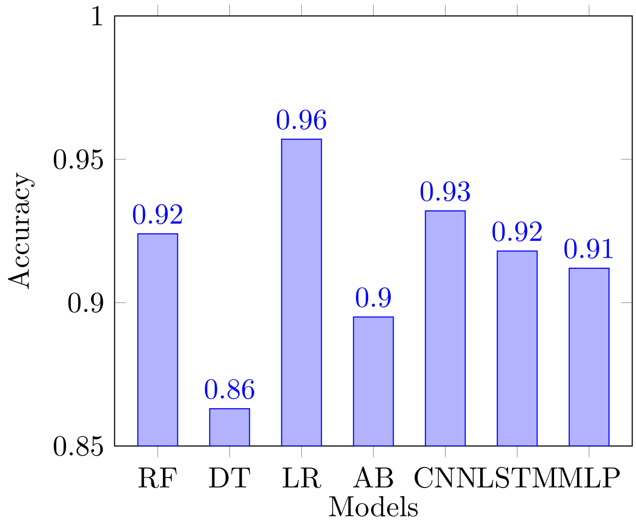

Figure 5: Model Accuracy Comparison

5.2 Training Time Comparison

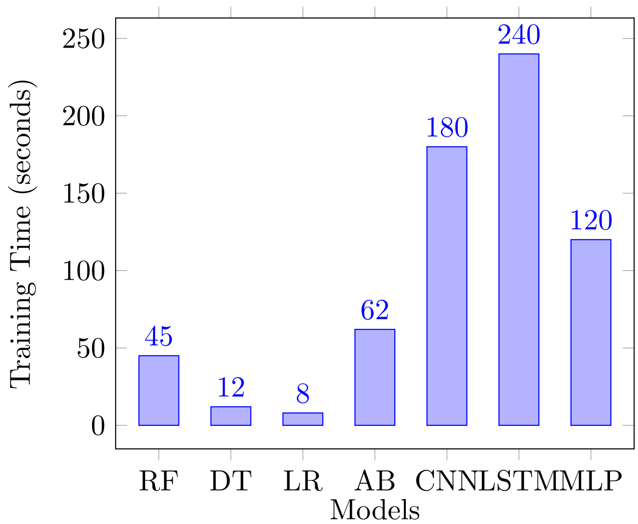

Figure 6: Training Time Comparison

6 Implementation Code

6.1 Data Preprocessing

```
def preprocess_data(raw_data):
    # Normalize data
    scaler = StandardScaler()
    normalized_data = scaler.fit_transform(raw_data)

    # Create windows
```

```
    window_size = 128
    stride = 64
    windows = create_windows(normalized_data,
                             window_size,
                             stride)
    return windows

def create_windows(data, window_size, stride):
    windows = []
    for i in range(0, len(data) - window_size, stride):
        window = data[i:i + window_size]
        windows.append(window)
    return np.array(windows)
```

6.2 Model Training

```
# Classical ML Training
def train_classical_models():
    models = {
        'random_forest': RandomForestClassifier(
            n_estimators=100,
            random_state=42
        ),
        'decision_tree': DecisionTreeClassifier(
            random_state=42
        ),
        'logistic_regression': LogisticRegression(
            max_iter=1000,
            random_state=42
        ),
        'adaboost': AdaBoostClassifier(
            random_state=42
        )
    }
    return models

# Deep Learning Training
def train_deep_learning_models():
    models = {
        'cnn': build_cnn_model(),
        'lstm': build_lstm_model(),
        'mlp': build_mlp_model()
    }

    for name, model in models.items():
        model.compile(
            optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )
    return models
```

7 Docker Deployment

7.1 Dockerfile

```
FROM python:3.8-slim

WORKDIR /app
```

```

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 8080

CMD ["python", "api.py"]
```

7.2 Docker Compose

```
version: '3'
services:
  har_api:
    build: .
    ports:
      - "8080:8080"
    volumes:
      - ./models:/app/models
    environment:
      - MODEL_PATH=/app/models
      - CUDA_VISIBLE_DEVICES=0
```

8 API Documentation

8.1 REST Endpoints

```
@app.route('/predict', methods=['POST'])
def predict():
    """
    Endpoint for real-time predictions

    Request Body:
    {
        "sensor_data": [...],  # Array of sensor readings
        "window_size": 128     # Optional window size
    }

    Response:
    {
        "activity": "WALKING",
        "confidence": 0.95
    }
    """
    pass
```

9 Performance Optimization

9.1 Model Quantization

```
# Convert model to TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_model = converter.convert()
```

10 Monitoring and Logging

10.1 Logging Configuration

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('har.log'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger('HAR')
```

11 Testing Framework

11.1 Unit Tests

```
import unittest

class TestHARModel(unittest.TestCase):
    def setUp(self):
        self.model = load_model()
        self.test_data = load_test_data()

    def test_prediction(self):
        prediction = self.model.predict(self.test_data)
        self.assertIsNotNone(prediction)
        self.assertTrue(0 <= prediction <= 1)
```

12 Conclusion

The implementation combines multiple technologies and frameworks to create a robust HAR system. Key technical highlights include:

- Comprehensive data processing pipeline
- Multiple model architectures
- Docker containerization
- REST API implementation
- Performance optimization
- Monitoring and testing frameworks