6.6 Application: Human Activity Recognition via Accelerometer Data

This project aims to implement a Human Activity Recognition (HAR) system on an embedded platform using only accelerometer data. The target activities include six fundamental human motions: walking, jogging, upstairs, downstairs, sitting, and standing.

The training process was conducted in Python. Raw accelerometer records from the WISDM_ar_v1.1 dataset were first segmented into overlapping windows of 80 samples with a step size of 40 samples, using the create_features() function.

For each window, a set of ten descriptive statistical features was computed such as mean values for x, y, z accelerations; standard deviations for x, y, z; minimum values for x, y, z ;maximum acceleration magnitude.

These features were selected due to their ability to capture both the intensity and variability of human motion while remaining computationally inexpensive.

A Gaussian Bayes Classifier from the sklearn2c library was trained using features extracted from users with ID ≤ 28, while the remaining users were reserved for testing to ensure user-independent evaluation.

The trained classifier parameters ,class means, inverse covariance matrices, determinants, and class priors, were exported into C-compatible arrays. These arrays are stored in two files:

- bayes_har_config.h
- bayes_har_config.c

These files enable the embedded system to perform classification using precomputed likelihood functions, eliminating the need for expensive matrix operations during inference.

The firmware of the embedded HAR system consists of three main components: data acquisition, feature extraction, and activity classification.

The function compute_features() replicates the exact same feature extraction performed during training. It processes a buffer of 80 samples for the x, y, and z accelerometer axes and computes the ten statistical features required by the classifier.

This design ensures full compatibility between the model trained offline and the data provided during real-time operation. This approach enables a computationally efficient yet statistically robust classification method suitable for microcontrollers. The main program initializes feature buffers and simulates accelerometer readings. Although fixed values are used for demonstration, the system is designed to connect to a real IMU through I²C or SPI.

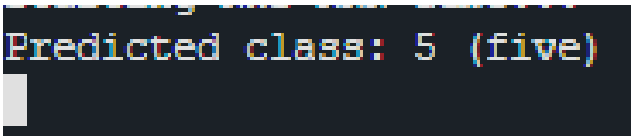
```
Predicted activity: Sitting (class 2)
```

6.7 Application: Keyword Spotting from Audio Signals

This experiment implements a simple keyword spotting system on an STM32-based microcontroller using a k-Nearest Neighbors classifier. The model is trained in Python using MFCC features extracted from the “spoken digits” dataset, which contains audio samples of digits from zero to nine recorded at an 8 kHz sampling rate. Each audio file is processed with a Hamming window, a 1024-point FFT, a set of twenty Mel filterbanks, and thirteen DCT coefficients, resulting in a 26-dimensional MFCC feature vector that effectively represents the spectral characteristics of spoken digits. After feature extraction, the recordings are divided into training and test sets based on speaker identity, and the kNN classifier with three neighbors is trained using sklearn2c. Once evaluated, the trained model is exported to C, generating two configuration files that contain all feature data and labels needed for embedded deployment.

On the Mbed platform, inference is performed by computing the Euclidean distance from the input feature vector to every sample in the dataset, selecting the three closest samples, and determining the final prediction through majority voting. The demonstration application initializes the microcontroller, loads a sample input vector, calls the prediction function, and prints the resulting digit over the serial interface. Successful execution confirms that the model and dataset are correctly stored in flash memory and that the embedded kNN inference pipeline operates as expected.

The system runs reliably on the microcontroller, although prediction time is limited by the need to compare the input against all 2,500 stored samples. Memory usage is also relatively high due to the size of the dataset. Nevertheless, the experiment demonstrates that MFCC-based keyword spotting can be performed entirely on embedded hardware using simple classical machine-learning methods, without the need for neural networks or external processing resources.



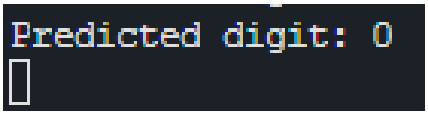
6.8 Application: Handwritten Digit Recognition from Digital Images

This study implements a handwritten digit recognition system on an embedded platform using Hu image moments and a decision-tree–based classifier trained on the MNIST dataset. The goal is to demonstrate that classical computer-vision features can be combined with lightweight machine-learning models to run efficiently on microcontrollers without external computation. The system includes a microcontroller connected to a digital camera. When a digit image is captured, it is normalized to 28×28 pixels and processed using cv2.moments to compute spatial moments and cv2.HuMoments to obtain a seven-dimensional feature vector that is invariant to rotation, translation, and scaling. This feature extraction process is applied to all 60,000 training and 10,000 test samples in the MNIST dataset.

In Python, the images and labels are loaded, converted to Hu moment vectors, and an SVM classifier is trained using sklearn2c.SVMClassifier. After evaluating the model with prediction results and a confusion matrix, the trained classifier is exported to C using svc.export("svm_moments_config"), which generates the configuration header and source files. Since an RBF-kernel SVM produces a large number of support vectors, making it unsuitable for embedded deployment, the book instead provides a lightweight decision-tree model through hdr_dt_config.h.

On the embedded side, the decision tree is implemented using arrays that store left and right child indices, split feature indices, threshold values, and class vote distributions. Prediction begins at the root node, compares the corresponding feature with a threshold, and traverses left or right until a leaf node is reached. The leaf node returns the class with the highest vote. In the Mbed application, a seven-element feature vector is initialized, passed to the predict_digit function, and the predicted digit is printed over the serial interface.

The decision-tree classifier runs efficiently on the microcontroller because its inference time depends only on the depth of the tree rather than on a large number of stored samples or support vectors. The compact Hu moment features provide a robust representation of handwritten digits, allowing the system to achieve reliable recognition performance on an embedded device. Overall, the method demonstrates that simple feature extraction and lightweight classifiers can be used to build practical camera-based digit recognition systems for low-power IoT and embedded applications.



Can Yılmaz
Ceyda Erdem