

**AIDL_B_AS01 - Signal Processing, Pattern Recognition and
Machine Learning**

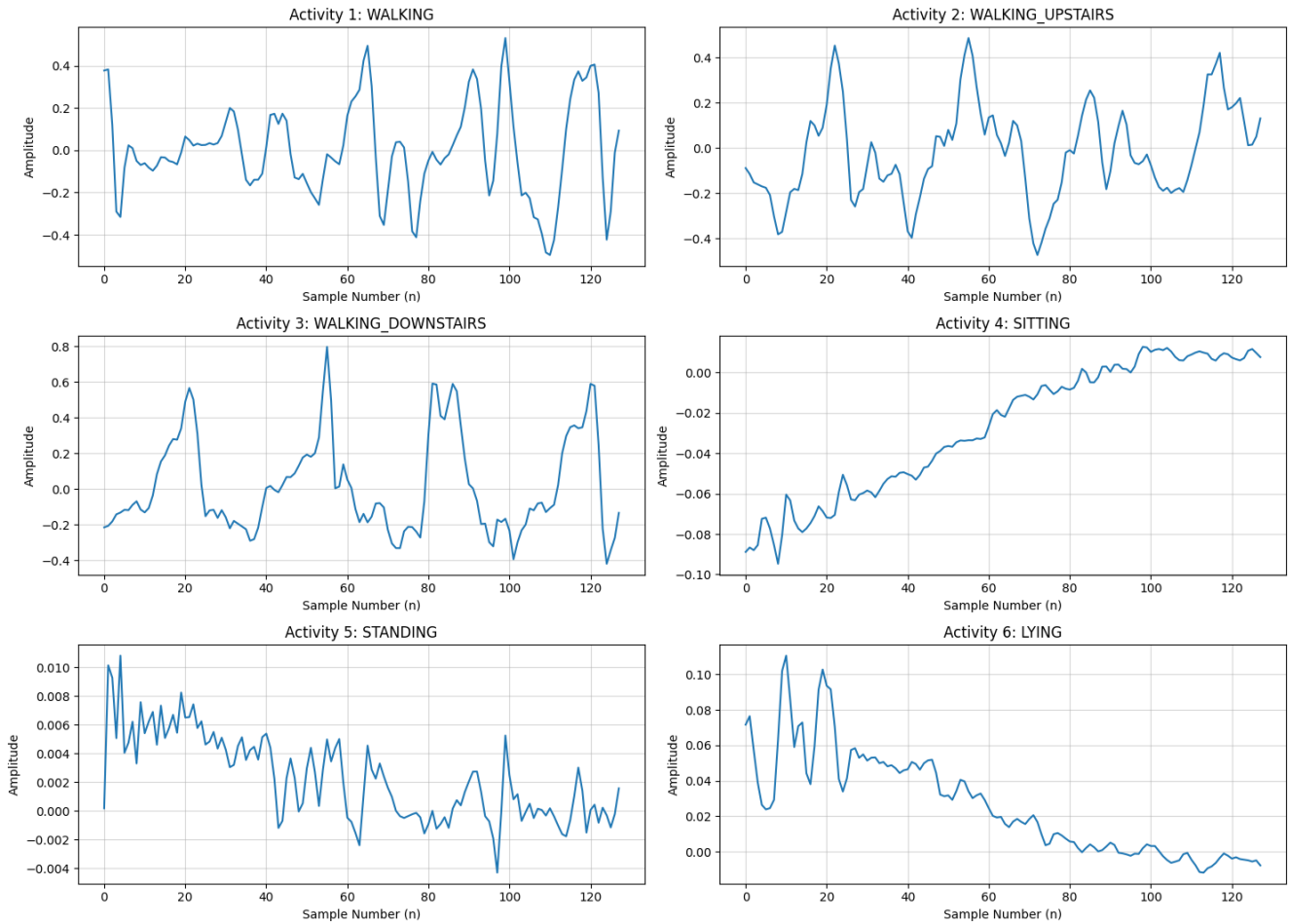
Assignment #1 - Features Extraction for Classification

Thodoris Charos mscaidl-0077

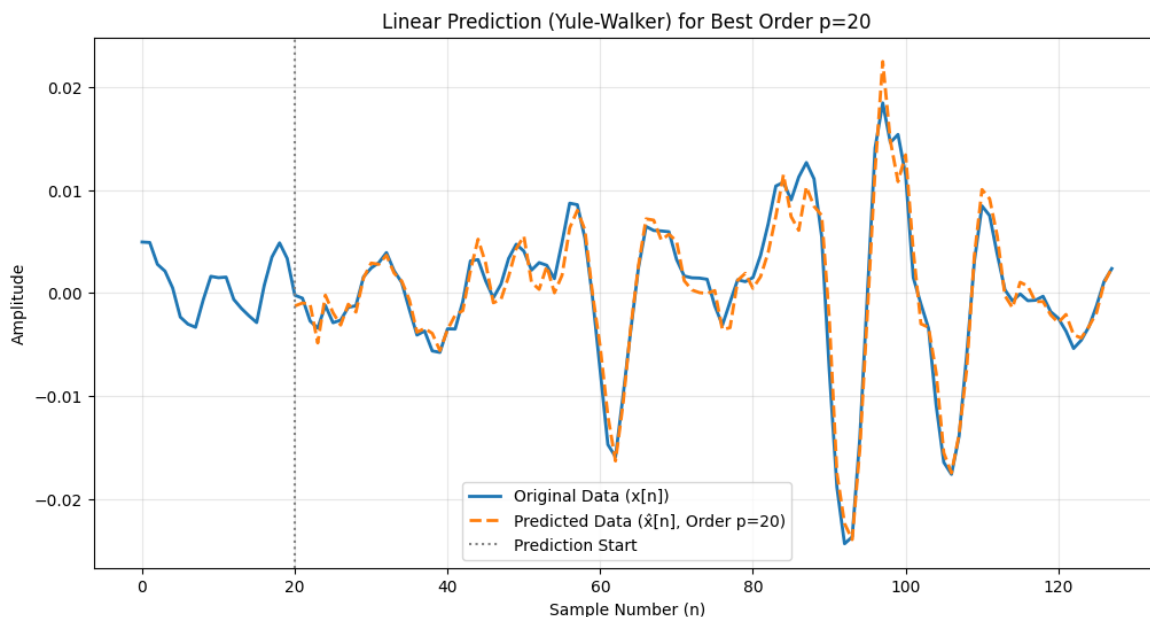
1. Data inspection and acquisition

```
--- Loading TRAIN Signals from: ./data set/UCI HAR Dataset/train/Inertial Signals
Loaded body_acc_x_train.txt: Shape (7352, 128)
Loaded total_acc_x_train.txt: Shape (7352, 128)
Loaded body_gyro_x_train.txt: Shape (7352, 128)
Loaded y_train.txt: Shape (7352,)
--- Loading TEST Signals from: ./data set/UCI HAR Dataset/test/Inertial Signals
Loaded body_acc_x_test.txt: Shape (2947, 128)
Loaded total_acc_x_test.txt: Shape (2947, 128)
Loaded body_gyro_x_test.txt: Shape (2947, 128)
Loaded y_test.txt: Shape (2947,)
```

Sample Time Series for Sensor: BODY_ACC_X



2. Yule–Walker linear prediction



What happens when p is too high?

Overfitting and Noise Modeling

A high-order model is overly flexible, allowing it to fit not only the underlying, predictable structure of the signal (the actual dynamics of the human activity) but also the random, non-repeating fluctuations and noise.

Poor Generalization

A high p might lead to a very low prediction error on the 10 training segments used to find the best p (especially if p approaches $N=128$), the resulting LPC coefficients (a) will be highly sensitive to those specific noise patterns. This means the model loses its ability to generalize to new, unseen segments of the same activity, leading to higher prediction errors when tested on the entire dataset. The model overfits.

Computational and Stability Issues

Increasing p increases the size of the Toeplitz autocorrelation matrix R_p

How is linear prediction related to AR modeling? When is the output of a linear predictor an AR process?

Linear Prediction is the method we use to find the numerical rules (the AR coefficients) that best describe how a signal's past values predict its future values. If we assume a signal is a song for which we try to predict the next note based on the previous: the AR model is the rulebook of the music ("next note is always a mix of previous five notes with little randomness") --> true weights and the Linear Prediction is the tool we use to figure out that rulebook by studying a piece of music --> finding the correct weights that minimize the error.

The Linear Prediction coefficients define the mathematical Auto Regressive model.

3. KNN algorithm

4. KNN benchmark

Benchmark Training Set Shape: (7352, 384)
Benchmark Testing Set Shape: (2947, 384)

```
--- Task 4: KNN Benchmark Results ---  
Total correctly classified: 2259  
Total instances examined: 2947  
Total Accuracy (K=3, Raw Data): 76.65%
```

5. KNN data pre-processing

```
--- Task 5: KNN Normalized Results using Z-score ---  
Total correctly classified: 2349  
Normalized Accuracy (K=3): 79.71%
```

```
--- Task 5: KNN Min-Max Results using Min-Max Scaling ---  
Total correctly classified: 2342  
Min-Max Accuracy (K=3): 79.47%
```

Sensitivity to Outliers: Min-Max is sensitive to outliers. If a sensor has one extreme "spike" (noise), it will squash all other "normal" data into a very tiny range, which can hurt KNN's ability to distinguish classes.

Standard Deviation: Z-score is generally more robust for sensor data because it handles the natural variance of human movement better. It centers the data at zero, which aligns well with sensors that oscillate (like the gyroscope or body acceleration).

Z-score normalization slightly outperformed Min-Max scaling but both preprocessing techniques improved classification performance compared to the benchmark.

6. Selecting the most important training set

Total Acceleration (total_acc): This is the raw signal directly from the smartphone's accelerometer. It includes two components: Gravity (the constant pull of the earth) and Body Motion.

Body Acceleration (body_acc): This signal is derived by applying a high-pass filter to the total acceleration to remove the gravity component, leaving only the user's movement.

Gyroscope (body_gyro): This sensor measures angular velocity (rotation). It captures the twisting and turning of the body, which is very different from linear acceleration.

--- Statistical Analysis per Activity ---

Signal Activity	Mean			StdDev	
	body_acc_x	body_gyro_x	total_acc_x	body_acc_x	body_gyro_x
DOWNSTAIRS	0.002183	-0.050523	0.991595	0.377717	0.711868
LYING	-0.001728	0.011256	0.071710	0.036015	0.092041
SITTING	-0.000850	-0.010474	0.950771	0.015156	0.059806
STANDING	0.000354	0.001394	1.001449	0.010478	0.075598
UPSTAIRS	-0.003224	0.050122	0.949443	0.258872	0.589723
WALKING	-0.000271	-0.003691	0.995493	0.228925	0.506082

Signal Activity	total_acc_x
DOWNSTAIRS	0.377792
LYING	0.141267
SITTING	0.104541
STANDING	0.025349
UPSTAIRS	0.262396
WALKING	0.230328

Based on statistical analysis of the training data, I selected **total_acc_x** (Total Acceleration) and **body_gyro_x** (Gyroscope) as the most informative signals. The total_acc_x mean values show a clear distinction between static activities (e.g., LYING mean ≈ 0.07 vs. STANDING mean ≈ 1.00) due to the inclusion of the gravity vector, a distinction that is lost in the filtered body_acc_x signal. Furthermore, the body_gyro_x standard deviation provides a contrast between dynamic and static states (e.g., 0.711 for DOWNSTAIRS vs. 0.075 for STANDING), offering unique rotational data that complements the linear acceleration data. This combination ensures the classifier can distinguish both the orientation and the intensity of the movement.

--- Task 6.4: KNN Reduced Set (256 features) Results ---

Total correctly classified: 2205

Reduced Set Accuracy (K=3): 74.82%

Upon reducing the dataset to only total_acc_x and body_gyro_x, the accuracy dropped from 76,65% to 74,82%. This indicates that while total_acc_x provides essential orientation data, the body_acc_x signal contained non-redundant information regarding high-frequency body movements that assisted the classifier in distinguishing between similar dynamic activities.

7. KNN features design

Why did Config 3 (Time/Freq) produced better results?

The results show that the basic physical characteristics, how much the person is moving (Time domain: Mean/StdDev) and how fast they are moving (Frequency domain: FFT), are the most discriminative markers for these 6 activities. The LPC coefficients (Config 1 & 2) were too sensitive to the specific "shape" of the signal, which can vary too much between different people walking or sitting.

Theoretical Question

Is frequency domain analysis (FFT) related to spectrograms? How are they similar/different?

Both **FFT** and **spectrograms** analyze a signal in the **frequency domain**.

- The **FFT (Fast Fourier Transform)** tells you *which frequencies are present* in a signal. The FFT takes a signal (or a part) and converts it from time to frequency. The output is a single frequency spectrum that shows how strong each frequency is. We don't have time info.
- A **spectrogram** shows *which frequencies are present **and when***. It splits the signal to small time windows and applies FFT to each of them. We have a display of the frequency vs time as a 2d plot. We see how frequencies change over time.

8. KNN fine-tuning

--- Task 8: Hyperparameter Tuning (K-Value) ---

K = 1	Accuracy: 78.18%
K = 3	Accuracy: 81.20%
K = 5	Accuracy: 81.68%
K = 7	Accuracy: 81.81%
K = 9	Accuracy: 81.81%
K = 11	Accuracy: 81.88%
K = 13	Accuracy: 82.19%
K = 15	Accuracy: 82.15%

Optimal K-value: 13 with 82.19% accuracy

During the hyperparameter tuning phase the classifier's performance improved from K=1 to K=13. At K=1, the lower accuracy (78.18%) suggests that the model was sensitive to noise or variations. By increasing the number of neighbors to 13, the classifier had a larger local consensus, which improved generalization. The peak accuracy of 82.19% at K=13 shows the optimal point where the model is complex enough to capture activity patterns but robust enough to ignore signal outliers.

What happens when K increases/decrease and why?

As **K increases** the decision boundary of the classifier becomes smoother and less sensitive to individual data points. With a higher K the classification is determined by a larger "consensus" of neighbors. This reduces the impact of outliers or noisy sensor readings.

As **K decreases** the model becomes highly sensitive to the local structure of the data. If a specific "Walking" window happens to look slightly like "Sitting" due to a sensor glitch or a user hitting their phone, a small K will incorrectly follow that single outlier. This is why the accuracy was lower at K=1 compared to K=13.

How is this related to underfitting/overfitting?

Small K leads to Overfitting: The model memorizes the training data, including its noise and random changes. It performs perfectly on training data but fails to generalize to the test data.

Large K leads to Underfitting: If K is too large (e.g., K=100), the model becomes too "simple". It might start predicting the most common activity for every window because the neighborhood is so large it includes almost the entire dataset, ignoring the subtle differences between activities.

Why select odd values of K?

We select odd values for K to prevent ties during the majority voting process. In a 2-class (or multi-class) problem, if K is an even number (like K=4), it is possible for two different activities to receive exactly 2 votes. An odd K ensures that there is always a clear winner in the majority vote.

