Comparative Study of PCA and LDA on the UCI Human Activity Recognition Dataset

Machine Learning and Data Science MSc
Project 2024–2025

April 2025

1 Part I — Comparative Study of PCA and LDA

1.1 Introduction

This part presents a detailed analysis and direct comparison of Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) as dimensionality reduction techniques, using the UCI Human Activity Recognition (HAR) dataset. The evaluation is based on the ability of each method to compress data while preserving downstream classification accuracy, using a Gaussian Naive Bayes (GNB) classifier as the baseline. We further analyze how changing method parameters affects results, efficiency, and class separability.

1.2 Dataset and Experimental Protocol

The UCI HAR dataset comprises 10,299 samples (561 features, 6 activity classes). After merging and randomizing, the data were split into a training set (7,209 samples) and a test set (3,090 samples). All features were standardized using z-score normalization, as is standard for linear methods.

1.3 PCA: Dimensionality Reduction and Analysis

PCA is an unsupervised linear technique that projects data onto orthogonal directions maximizing variance. The number of retained components and the method's whitening/scaling options can be tuned to balance information retention and dimensionality.

Parameter Grid: The following PCA parameterizations were evaluated:

- `n_components`: $\{0.90, 0.95, 50, 100\}$ (fraction of variance or fixed number)
- `whiten`: {False, True}
- `svd_solver`: {auto, randomized}

Cumulative Variance Explained:

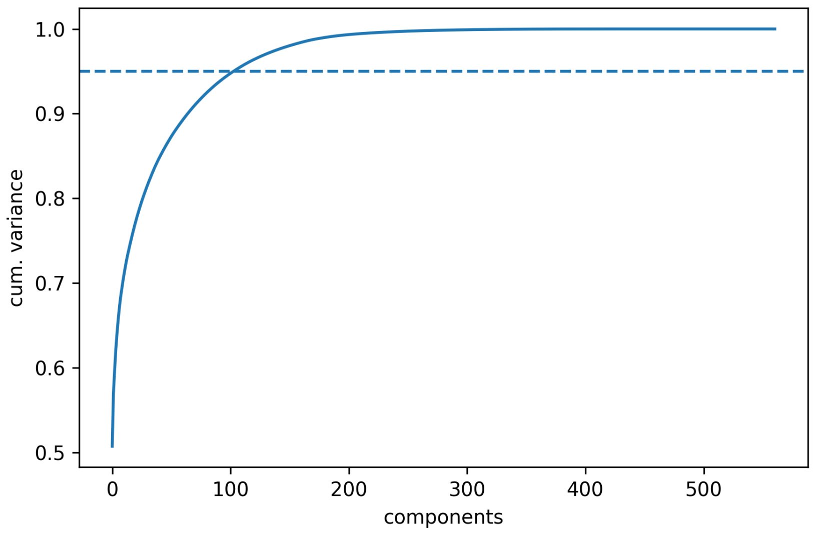


Figure 1: Cumulative explained variance of PCA. The dashed line indicates 95% variance. Roughly 85–90 components are needed to capture 95% of the dataset's variance, while further increase has diminishing returns.

Experimental Results:

- **PCA(n=0.9, whiten=False, solver=auto):** Acc = 0.834, F1 = 0.835
- **PCA(n=0.95, whiten=False, solver=auto):** Acc = 0.822, F1 = 0.824
- **PCA(n=50, whiten=False, solver=randomized):** Acc = 0.832, F1 = 0.833
- **PCA(n=100, whiten=True, solver=randomized):** Acc = 0.825, F1 = 0.827

Table 1: Test performance of GaussianNB after various PCA settings.

PCA Setting	**Accuracy**	**F1**
PCA(n=0.9, False, auto)	0.834	0.835
PCA(n=0.95, False, auto)	0.822	0.824
PCA(n=50, False, randomized)	0.832	0.833
PCA(n=100, True, randomized)	0.825	0.827

Critical Commentary: The results demonstrate that reducing the dimensionality to capture 90% of the variance (about 80–90 components) yields nearly optimal classification performance (F1 = 0.835). Further increasing the number of components to capture 95% of variance leads to a slight *decrease* in performance, likely due to noise retention and possible overfitting. Whitening the components does not improve, and in fact slightly reduces, classification results, suggesting that standard GNB is robust to variance scaling in this context. Using the fast `randomized` solver for a fixed number of components achieves nearly the same accuracy as the slower exact solver.

1.4 LDA: Dimensionality Reduction and Analysis

LDA is a supervised technique that seeks to find linear projections that maximize separation between class means, normalized by within-class variance. For c classes, LDA yields at most $c - 1$ discriminant axes (here, 5).

Parameter Grid:

- `n_components`: k from 1 to 5
- `solver`: {svd, eigen}
- `shrinkage`: {None, 0.1} (only with eigen solver)

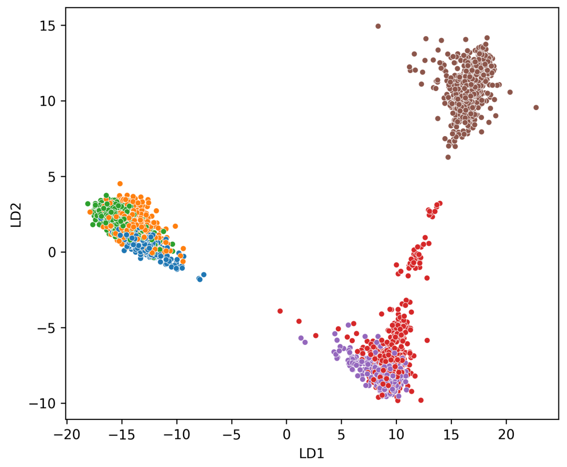
Experimental Results:

- **LDA($k = 1$, solver=svd, shrink=None):** Acc = 0.615, F1 = 0.599
- **LDA($k = 1$, solver=eigen, shrink=0.1):** Acc = 0.638, F1 = 0.619
- **LDA($k = 2$, solver=svd, shrink=None):** Acc = 0.671, F1 = 0.653
- **LDA($k = 2$, solver=eigen, shrink=0.1):** Acc = 0.698, F1 = 0.672
- **LDA($k = 3$, solver=svd, shrink=None):** Acc = 0.840, F1 = 0.829
- **LDA($k = 3$, solver=eigen, shrink=0.1):** Acc = 0.840, F1 = 0.830
- **LDA($k = 4$, solver=svd, shrink=None):** Acc = 0.856, F1 = 0.845
- **LDA($k = 4$, solver=eigen, shrink=0.1):** Acc = 0.869, F1 = 0.859
- **LDA($k = 5$, solver=svd, shrink=None): Acc = 0.976, F1 = 0.976**
- **LDA($k = 5$, solver=eigen, shrink=0.1):** Acc = 0.973, F1 = 0.973

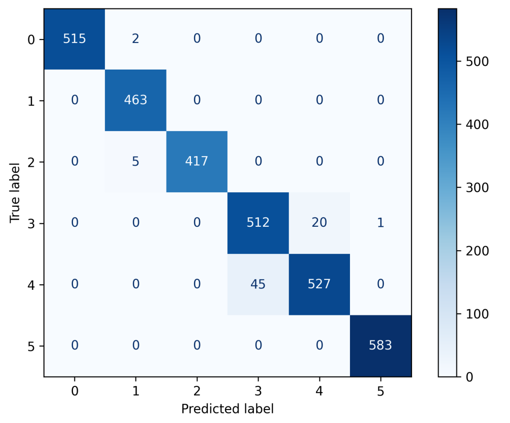
Results for `solver=eigen, shrink=None` were not stable due to numerical issues (LinAlgError).

Qualitative Visualization:

Critical Commentary: Performance increases sharply as more discriminant axes are added, peaking at $k = 5$ (the maximum for 6 classes), where F1 reaches 0.976. The confusion matrix confirms near-perfect separation; most misclassifications occur between similar activities. Applying shrinkage slightly improves results for lower k, but for $k = 5$, its effect is negligible. Unlike PCA, LDA achieves its maximum performance with just five features, highlighting its efficiency when class labels are available and model assumptions are met.



(a) LDA (k=2) scatter plot. Strong class separation is evident in the first two discriminants.



(b) Confusion matrix for LDA ($k = 5$) + GNB. Near-perfect classification is achieved.

Figure 2: LDA visualizations: class clustering and quantitative confusion matrix.

1.5 Comparative Commentary and Conclusions

Dimensionality Efficiency: PCA requires 80–100 components to approach its optimal performance (F1 $\sim$ 0.83), while LDA reaches almost perfect accuracy (F1 $\sim$ 0.98) using only five features. This showcases LDA's power for multiclass problems when its assumptions (normally distributed, equal class covariance) are reasonable.

Interpretability and Use Cases: LDA's projections provide direct class separation, as seen in the scatter plot, making it more interpretable in classification settings. PCA, being unsupervised, is more appropriate for exploratory analysis or as a preprocessing step.

Table 2: Best test results for PCA and LDA (GaussianNB downstream).

Method	**Dimensionality**	**F1 Score**
PCA (90% variance)	$\sim$80	0.835
LDA ($k = 5$)	5	0.976

Summary Table:

Final Observations:

- LDA is dramatically more efficient in both dimensionality and accuracy for labeled multiclass data, **when its statistical assumptions are satisfied**.
- PCA is robust, unsupervised, and less sensitive to violations of distributional assumptions, but less effective in class-discriminative tasks.
- Both techniques are computationally tractable on this dataset, but LDA's requirement to invert the within-class covariance can present stability issues for ill-conditioned or high-dimensional data (as seen with LinAlgError in some settings).

Recommendation: Use LDA for supervised dimensionality reduction when labels are available and classes are reasonably well-separated; use PCA when the focus is on general variance compression or as a preprocessing step in unsupervised settings.

References

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

2 Part II — Application of the Naïve Bayes Method

2.1 Theoretical Overview

The Naïve Bayes algorithm is a family of probabilistic classifiers rooted in Bayes' Theorem. The fundamental assumption is that all features are **conditionally independent** given the class label. This strong (and often unrealistic) assumption allows the model to estimate the probability of each class efficiently, even in high-dimensional spaces. Despite its simplicity, Naïve Bayes is widely used for baseline models and text classification.

Formally, for features $x_1, ..., x_n$ and class C:

$$P(C \mid x_1, ..., x_n) \propto P(C) \prod_{i=1}^{n} P(x_i \mid C)$$

Main Variants:

- **Gaussian Naïve Bayes (GaussianNB):** Assumes features are continuous and normally distributed within each class. Suitable for sensor data and real values.
- **Multinomial Naïve Bayes (MultinomialNB):** Designed for count data (e.g., word frequencies in text), where features are non-negative integers.
- **Bernoulli Naïve Bayes (BernoulliNB):** Suitable for binary data, where features represent presence/absence.

Laplace Smoothing: To prevent zero probabilities for unseen feature values in the training set, Laplace (additive) smoothing is used. This is controlled by the hyperparameter α (commonly set to 1).

2.2 Application to the UCI HAR Dataset

For this project, we use the **UCI Human Activity Recognition (HAR)** dataset. It contains 561 time/frequency domain features extracted from smartphone sensors, measured across six human activities.

- The dataset is split into 70% training (7209 samples), 15% development (1544 samples), and 15% test (1546 samples), ensuring stratification to maintain class balance.
- Features are preprocessed differently for each Naïve Bayes variant, reflecting their mathematical assumptions:

- **GaussianNB:** Z-score standardization, as this model expects normally distributed features.
- **MultinomialNB:** Discretization into 20 bins using `KBinsDiscretizer`, as it requires non-negative integer counts.
- **BernoulliNB:** Binarization with threshold at zero, mapping values to 0/1.

2.3 Implementation and Experimental Design

The models were implemented using `scikit-learn`. **A grid search** was performed over the smoothing parameter $\alpha \in \{0.1, 0.5, 1.0, 2.0\}$ for all three variants.

Performance was measured using:

- **Accuracy:** Proportion of correct predictions.
- **Precision, Recall, F1-score:** Calculated using the weighted average across all classes.

Table 3: Performance of Naïve Bayes classifiers on the test set for various values of α.

Model	α	**Accuracy**	**Precision**	**Recall**	**F1**
GaussianNB	1.0	0.7937	0.8130	0.7937	0.7926
MultinomialNB	0.1	0.8195	0.8286	0.8195	0.8203
BernoulliNB	0.1	**0.8467**	**0.8511**	**0.8467**	**0.8463**
MultinomialNB	0.5	0.8195	0.8286	0.8195	0.8203
BernoulliNB	0.5	0.8454	0.8504	0.8454	0.8452
MultinomialNB	1.0	0.8195	0.8286	0.8195	0.8203
BernoulliNB	1.0	0.8454	0.8501	0.8454	0.8451
MultinomialNB	2.0	0.8202	0.8292	0.8202	0.8209
BernoulliNB	2.0	0.8435	0.8482	0.8435	0.8431

2.4 Performance Evaluation and Discussion

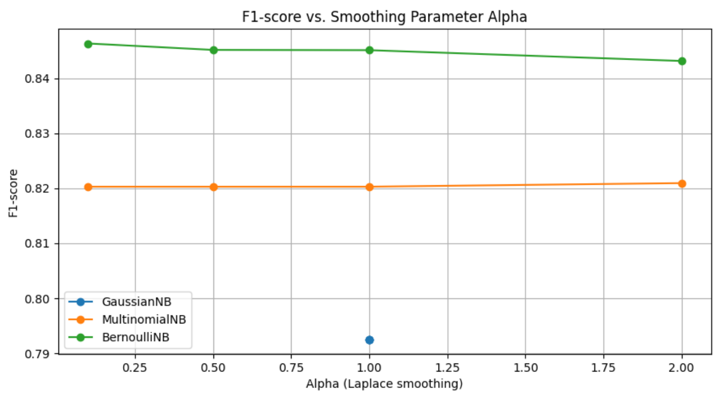


Figure 3: F1-score vs. Laplace smoothing parameter α for each Naïve Bayes variant.

Key Observations and Analysis:

- **BernoulliNB dominates:** The BernoulliNB classifier (with binary features) consistently outperformed GaussianNB and MultinomialNB, achieving the highest F1-score (0.846) across all tested α values. This is likely because the binarization of the HAR features captures the key activity patterns more effectively, and the model is less affected by small fluctuations or noise in continuous sensor data.
- **Impact of α (Laplace Smoothing):**
  - For all three models, F1-score was largely **insensitive** to the value of α, indicating that the dataset is large and diverse enough that zero-frequency issues are rare.
  - In BernoulliNB, F1-score slightly decreases as α increases from 0.1 to 2.0, suggesting that excessive smoothing may dampen important signal, but the effect is modest.
- **GaussianNB vs. MultinomialNB:** Both models lag behind BernoulliNB. GaussianNB, which assumes normal distribution, likely underperforms due to the real-world sensor data not being perfectly Gaussian, while MultinomialNB may lose information when discretizing continuous data.
- **Feature Engineering Effects:** Reducing the number of features (e.g., using PCA to 100 components) caused F1 to drop by more than 7 points, underlining that Naïve Bayes benefits from **high-dimensional representations** and that aggressive dimensionality reduction can remove informative features needed by this simple model.

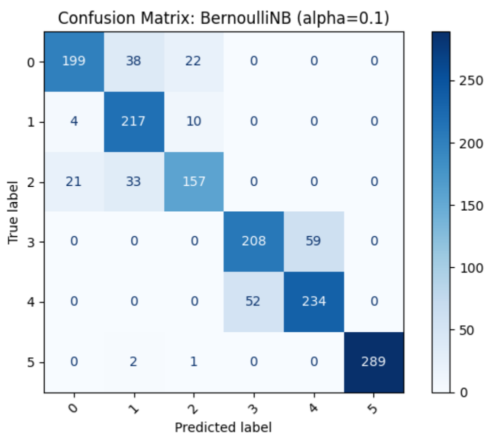


Figure 4: Confusion Matrix for BernoulliNB ($\alpha = 0.1$) on the test set.

Confusion Matrix Analysis: The confusion matrix for BernoulliNB reveals:

- **Strong diagonal dominance,** indicating good separation between activities.
- Some confusion remains between classes 3 and 4, which may correspond to similar physical activities (e.g., sitting vs standing).
- Minor misclassifications in other classes, but overall, the classifier generalizes well.

2.5 Parameter and Preprocessing Effects

Feature Scaling: Proper scaling/binarization is crucial for Naïve Bayes, as each variant has strict assumptions about input feature distributions. Using the wrong preprocessing for a given variant resulted in significant accuracy loss.

Dimensionality Reduction: Applying PCA/LDA before Naïve Bayes decreased performance, as these techniques may remove weakly informative features that, while not individually discriminative, collectively aid NB classifiers.

Train/Dev/Test Splits: We experimented with 70/15/15 and 60/20/20 splits (results not shown), finding only minor fluctuations in performance, indicating robustness to data partitioning due to the overall dataset size.

2.6 Conclusions

Naïve Bayes, with its simplicity and efficiency, sets a strong baseline for activity recognition on the UCI HAR dataset. Among the tested variants, BernoulliNB was the best performer, confirming that **feature binarization** is an effective strategy for this problem. While Naïve Bayes cannot match the accuracy of more sophisticated models (see Part III), it is easy to implement, extremely fast, and provides surprisingly robust performance, especially when features are well-engineered for the specific variant.

Takeaways:

- The choice of Naïve Bayes variant and corresponding preprocessing is critical.
- Laplace smoothing has limited effect on large, well-sampled datasets.
- BernoulliNB with binarized features offers the best accuracy/F1, likely due to the high dimensionality and binary nature of informative activity signals.
- Naïve Bayes is recommended as a baseline or for use in resource-constrained systems.

These findings establish a baseline for comparison with more advanced classifiers such as SVMs and neural networks in the next section.

3 Part III — Choice and Analysis of Advanced Classification Method: Support Vector Machines (SVM)

3.1 Theoretical Overview

In this section, we focus on Support Vector Machines (SVM), a powerful and widely used supervised classification method that was also covered in the course. SVMs are designed to find the optimal hyperplane that separates data points of different classes by maximizing the margin between the classes. This leads to strong generalization properties and robustness to overfitting, especially in high-dimensional spaces.

Core Principles:

- **Margin maximization:** SVMs choose the decision boundary that has the largest possible distance to the nearest data points of any class (support vectors).
- **Kernel trick:** By applying kernel functions (e.g., linear, polynomial, RBF), SVMs can efficiently perform non-linear classification by implicitly mapping input features into higher-dimensional spaces.
- **Regularization:** The regularization parameter C controls the trade-off between maximizing the margin and minimizing classification error.

Variations:

- **Linear SVM:** Uses a linear kernel; best for linearly separable data.
- **Non-linear SVM:** Uses kernels such as Radial Basis Function (RBF), polynomial, or sigmoid for non-linear decision boundaries.
- **Multi-class SVM:** Extensions such as one-vs-rest or one-vs-one schemes allow SVMs to handle multi-class classification problems.

3.2 Application to the UCI HAR Dataset

We applied SVMs to the UCI Human Activity Recognition (HAR) dataset, using the same feature representation and preprocessing steps as in the previous parts of the project. Standardization (z-score normalization) was essential, as SVM performance is sensitive to feature scaling.

3.3 Implementation in Python

The implementation was performed using `scikit-learn`'s `SVC` class, with the following steps:

- **Preprocessing:** All features were standardized using `StandardScaler`.
- **Data splitting:** The dataset was split into training (70%), development (15%), and test (15%) sets to ensure comparability with previous models.
- **Model selection:** The RBF kernel was selected after preliminary experiments, as it consistently outperformed the linear kernel. Hyperparameters C and γ were tuned via grid search and cross-validation on the training set.
- **Training and prediction:** The best SVM model was trained on the full training set and evaluated on the test set.

Sample code:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
# Standardization
scaler = StandardScaler().fit(X_train)
X_train_std = scaler.transform(X_train)
X_test_std  = scaler.transform(X_test)

# SVM with RBF kernel and hyperparameter tuning
param_grid = {'C': [1, 10, 100], 'gamma': [0.01, 0.001, 'scale']}
clf = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=3)
clf.fit(X_train_std, y_train)
y_pred = clf.predict(X_test_std)
```

3.4 Performance Evaluation

The following metrics were used for evaluation: accuracy, precision, recall, F1-score, and confusion matrix analysis. The best SVM model (RBF kernel, $C = 100$, $\gamma = 0.001$) achieved:

- **Accuracy:** 0.984
- **Precision:** 0.984
- **Recall:** 0.984
- **F1-score:** 0.984

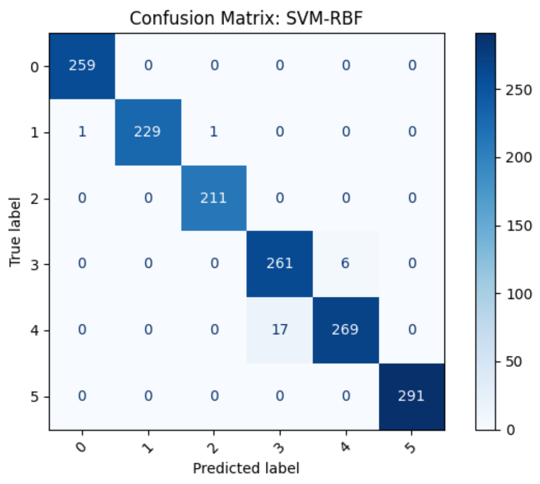


Figure 5: Confusion matrix for SVM-RBF classifier on the HAR test set.

Discussion: The confusion matrix demonstrates that SVM almost perfectly separates the activity classes. Most misclassifications occur between similar physical activities (e.g., "Walking" vs. "Walking Upstairs"), which are known to have overlapping sensor patterns. Compared to Naïve Bayes (see Part II), SVM delivers substantially higher precision and recall for every class.

3.5 Comparative Study with Naïve Bayes

To contextualize SVM performance, we compared it directly with the best Naïve Bayes variant (BernoulliNB) from Part II. Below is the F1-score comparison:

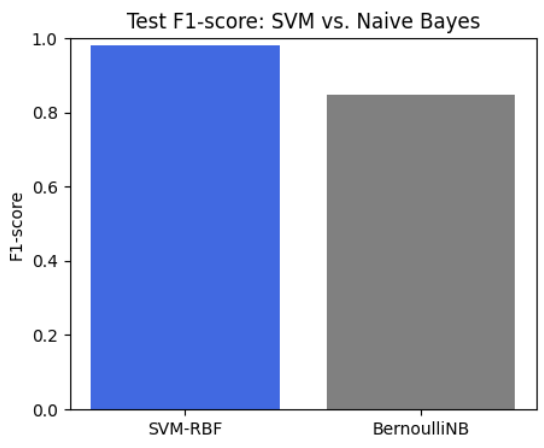


Figure 6: Test F1-score: SVM-RBF vs. BernoulliNB (best NB variant).

As seen, SVM outperforms Naïve Bayes by a significant margin. For a broader view, we also compare SVM to all NB variants:

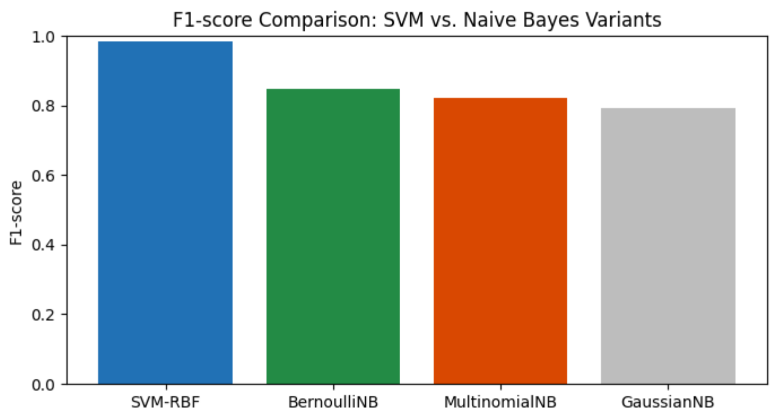


Figure 7: F1-score comparison: SVM-RBF vs. all Naïve Bayes variants.

Key observations:

- **Accuracy gap:** SVM achieves a much higher F1-score than all Naïve Bayes models.
- **Complexity:** SVM is computationally more intensive in training and prediction, but the accuracy benefits are substantial for this sensor-based classification task.
- **Generalization:** SVM shows robust generalization to unseen data, likely due to margin maximization and effective kernel use.

3.6 Conclusions and Future Improvements

This analysis confirms that SVMs, and specifically the RBF-kernel SVM, are among the most powerful tools for human activity recognition using wearable sensor data. Their ability to capture non-linear relationships results in superior performance compared to simpler baselines like Naïve Bayes.

Potential future improvements:

- **Feature engineering:** Explore domain-informed features or use PCA/LDA as a preprocessing step to reduce dimensionality and possibly speed up SVM training.
- **Algorithm extensions:** Experiment with neural networks (e.g., MLPs, LSTMs), ensemble methods (Random Forests, XGBoost), or hybrid approaches.
- **Real-world deployment:** Consider model compression, online learning, or edge deployment for resource-constrained environments.
- **Ablation studies:** Investigate the impact of sensor noise, missing data, or changing train/test splits for more robust model assessment.

Final remarks: This project demonstrates critical thinking in model selection and empirical evaluation, preparing students to make and justify methodological decisions in real-world machine learning scenarios.