Effort Matters

Clara Wong

wongc148

400372818

IBEHS 4QZ3

Submitted on: November 23rd, 2025

Academic Integrity Statement

As a future member of the engineering profession, I am responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario.

Introduction

Today's smartphones and smartwatches combine accelerometers and gyroscopes to help detect motion and orientation. The UCI Human Activity Recognition dataset leverages these sensors to record 30 subjects performing five activities: sitting, standing, walking, walking upstairs, and walking downstairs. These five activities were then further characterized as low and high-intensity activities. To distinguish between low and high-intensity activity windows, a Support Vector Machine (SVM) was trained on statistical features derived from the accelerometer. An RBF SVM was selected for this activity due to its ability to classify non-linearly separable data; an appropriate C and gamma value were chosen based on trial and error and scikit learn documentation. Then, the model was trained on five features extracted from the X_train file: mean, standard deviation, signal magnitude area (SMA), mean jerk, and peak-to-peak. After training the model, the model was evaluated on data from the X_test file. The SVM model achieved a high classification accuracy of 96%, as well as scores of 90%+ on other metrics such as precision, recall, and f1-score, indicating that the SVM was effective at classifying low and high-intensity activities. However, the SVM's offline performance does not reflect the additional challenges encountered if deployed in a real-time environment, highlighting the constraints of a real-time cyber-physical system.

Methods

Preprocessing

The raw x-, y-, and z-axis accelerometer data, along with selected features from the X_train file, were extracted from the dataset. Although both sensors (accelerometer and gyroscope) provide valuable information to help distinguish between low- and high-intensity activities, only accelerometer data was used, as the low- and high-intensity activities experience a significant enough difference in body acceleration to differentiate solely on acceleration. For example, greater, rapid changes in acceleration values are expected for all activities where the body is in motion (e.g. walking, walking upstairs, and walking downstairs), and a sharper increase in acceleration between walking up and down stairs, as opposed to walking. The SVM model was trained on five features; four features were extracted from the X_train dataset:

- 201 - tBodyAccMag-mean()
- 202 - tBodyAccMag-std()
- 16 - tBodyAcc-sma()
- 227 - tBodyAccJerkMag-mean())

One custom feature (peak-to-peak magnitude of acceleration) was calculated from the raw body accelerometry data. Therefore, the X_train and y_train data files, along with all three raw body accelerometry data files, were loaded. Then, as mentioned in the assignment instructions, all samples that had an outcome of 'laying' were excluded. After removing 'laying' samples, the activity distribution graph was as shown:

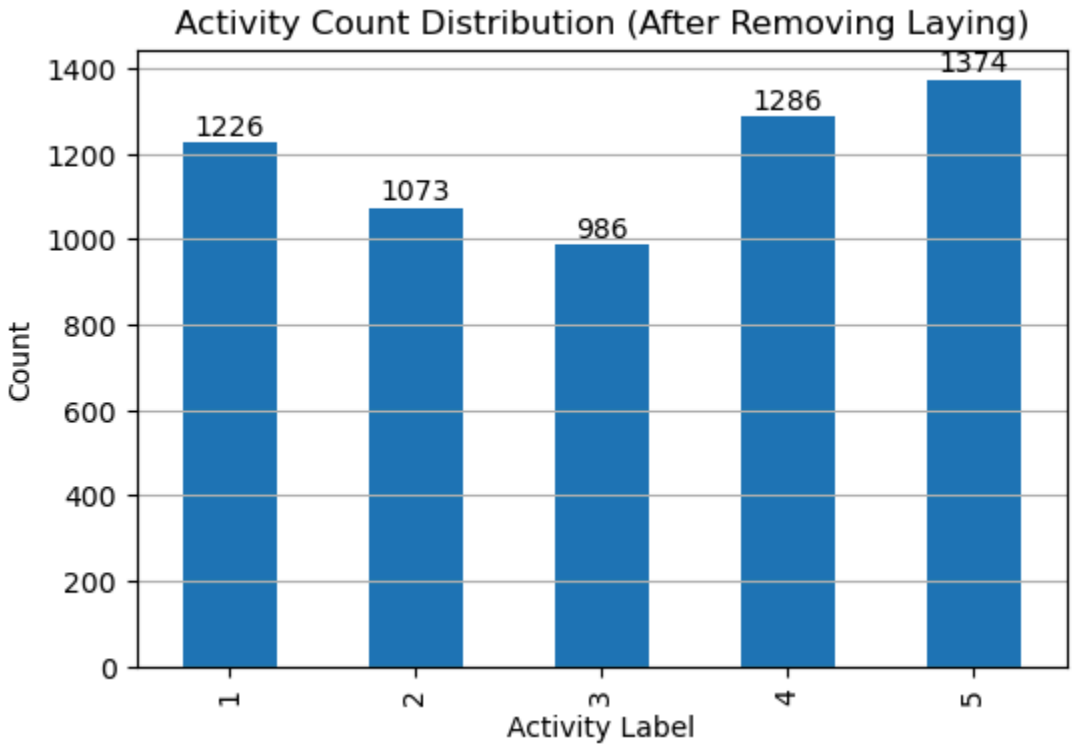

Figure 1. Data distribution of activity count, after removing 'laying' samples

After, the activities were mapped to an activity intensity class, with 0 indicating low-intensity and 1 indicating high-intensity. From this, a class balance graph was created, which showed that there was a higher low-intensity activity count:

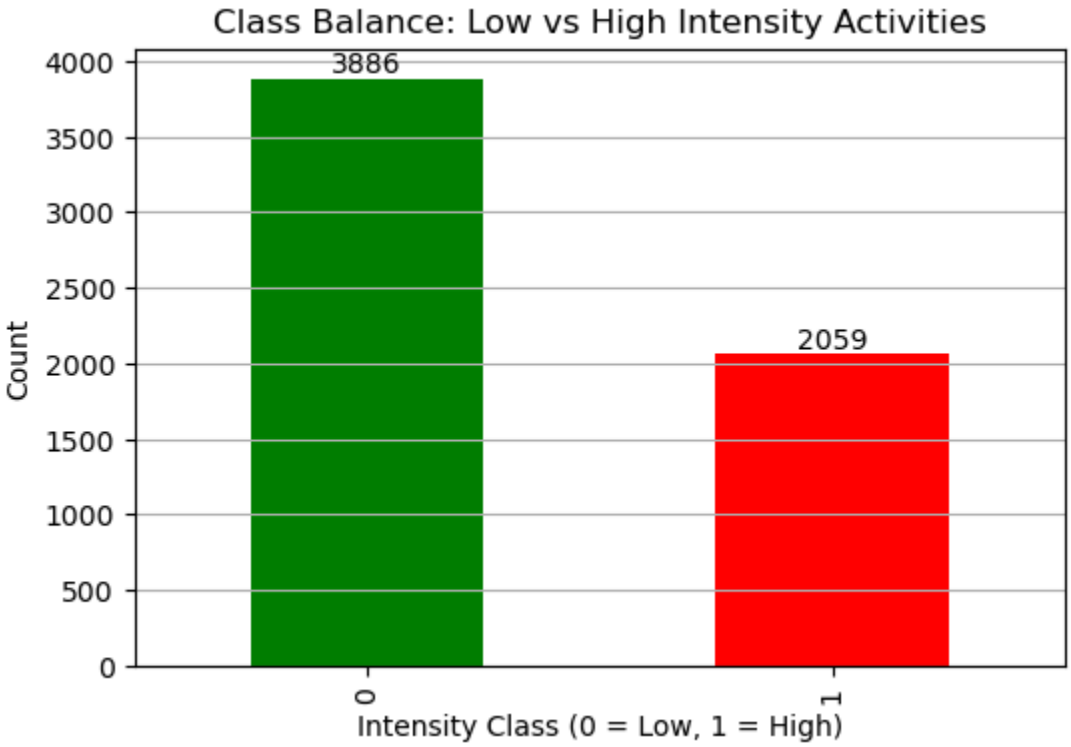

Figure 2. Class balance of low vs. high intensity activities

Features

The following five features were chosen to train the SVM model on:

1. Mean acceleration: captures the overall intensity level of a movement within a given window. The low-intensity activities will have small, constant mean acceleration values, while high-intensity activities will have higher mean acceleration values.
2. Standard deviation of acceleration: captures variability of movement. Low-intensity activities will have low standard deviation (low variability) since the signal would be steady with little fluctuation. High-intensity activities will have a high standard deviation (high variability) since stair walking has more abrupt movements.
3. Signal Magnitude Area (SMA) of acceleration: gets the absolute acceleration across the x, y, and z axes to represent the overall magnitude of movement, regardless of direction. High-intensity activities will have a higher SMA value.
4. Mean jerk acceleration signals: since ‘jerk’ refers to the time derivative of acceleration, this feature represents how quickly the acceleration changes. The high-intensity activities will have faster and greater changes in acceleration, whereas low-intensity activities have more gradual changes. Jerk can be used to identify sharp body movements, which are typical in stair walking [1].

5. Peak-to-peak acceleration magnitude: captures the intensity of motion and dynamic range of motion by finding the difference between the minimum and maximum magnitude in a given window. This feature is useful for distinguishing between activities like walking vs stair walking. The peak-to-peak acceleration magnitude would be small for walking. However, walking downstairs would result in high downward acceleration peaks, whereas walking upstairs would result in high upward acceleration peaks.

SVM setup

An SVM with a Radial Basis Function (RBF) kernel, C-value of 100, and a gamma value of 'scale' was used. The RBF kernel is used because it can model complex, non-linear relationships in the data by mapping samples into a high-dimensional feature space. Unlike linear or polynomial kernels, the RBF kernel does not depend on a fixed polynomial degree, making it optimal for HAR data that is not linearly separable [2]. The C parameter controls the penalty for misclassified data points during training. A small C value achieves a larger margin, but often results in more misclassifications in the training data. A large C value focuses on minimizing the training error by fitting the training data as accurately as possible, but runs the risk of overfitting. C is 1 by default and was used to train this model. A higher C value of 10 was tested, but it did not improve prediction results; therefore, the default was used. A gamma value of 'scale' (default) was used. The SVM automatically chooses a value for gamma based on the dataset's feature variance $\frac{1}{n_{features} \cdot Var(X)}$; this is a good value to set it to, since the gamma value adapts to the scale of the dataset [3].

Results

The metrics table and confusion matrix below demonstrate that the model performs well overall. It can identify low- vs. high-intensity activities with an accuracy of 96% (overall percentage of correct predictions). Additionally, the precision, recall, and f1-score for each class, as well as the overall macro and weighted scores, are high (90%+), indicating that the classifier is reliable at identifying activity intensity levels. The high precision score shows that the model predicts high-intensity activities ("positive" case) with few false positives (high-intensity predictions that are actually low-intensity). The high recall score demonstrates that the model can correctly identify samples that truly belong to the high or low-intensity class, with few false negatives. The high f1-score, as the harmonic mean of the two, reflects the high precision and recall scores. The confusion matrix also demonstrates that relatively few samples were misclassified, with the majority of predictions included in the true positive or true negative boxes. Finally, the PCA visualization transforms the high-dimensional data into two directions that capture the most variance in the data. From *Figure 4*, the wider spread of the orange cluster (high-intensity) demonstrates the larger variation in accelerometer and jerk signals. Conversely, the blue cluster (low-intensity) is more compact, with less spread, and exhibits more consistent movement. Overall, the partial separation between the two clusters proves that the chosen features have some discriminative ability. However, the clusters have some overlap, indicating that perfect linear separation is not possible [4].

Table 1: Classification Report for the test data

	precision	**recall**	**f1-score**	**support**
0	0.95	0.99	0.97	1519
1	0.98	0.91	0.94	891
accuracy			0.96	2410
Macro avg	0.96	0.95	0.96	2410
Weighted avg	0.96	0.96	0.96	2410

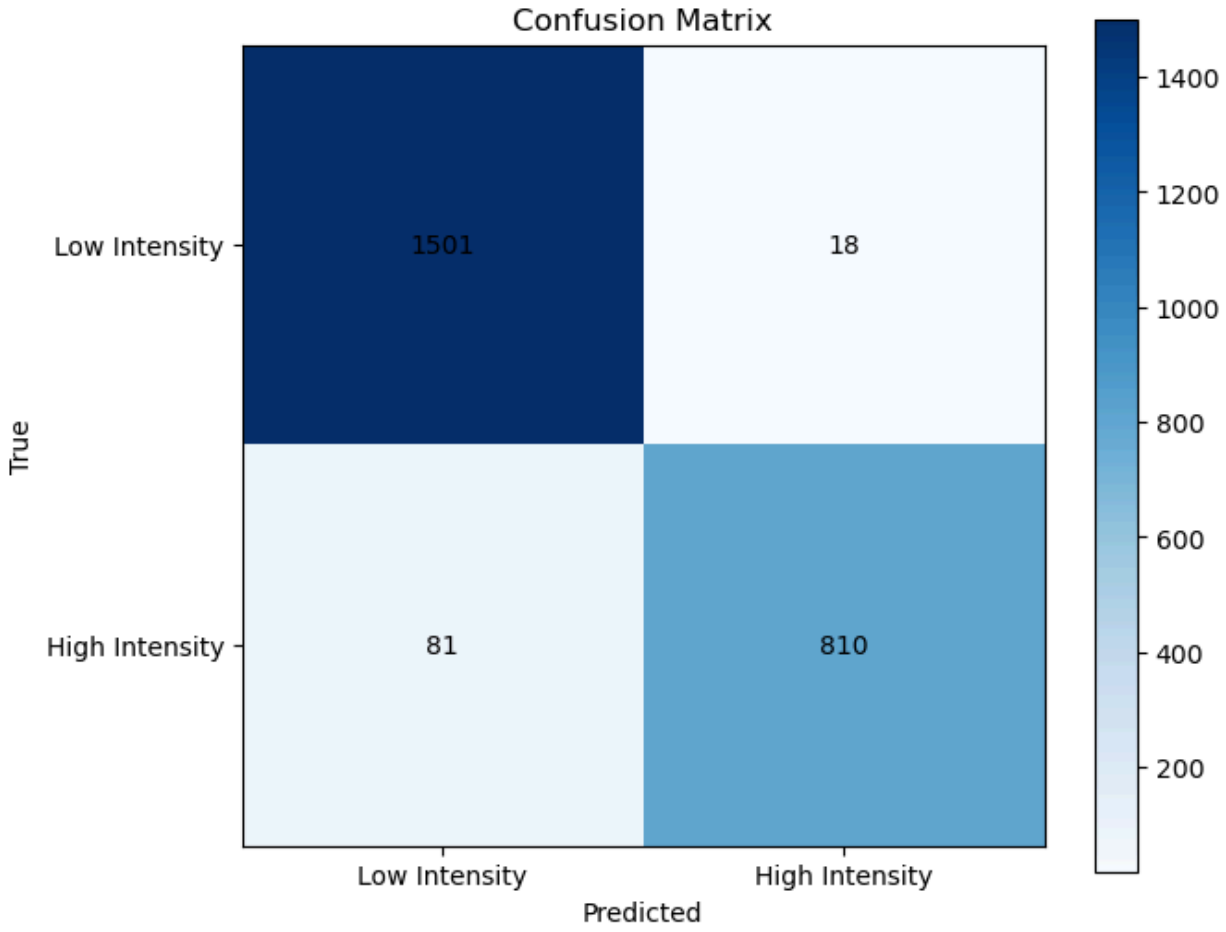

Figure 3. Confusion matrix of the high-intensity vs low-intensity test results

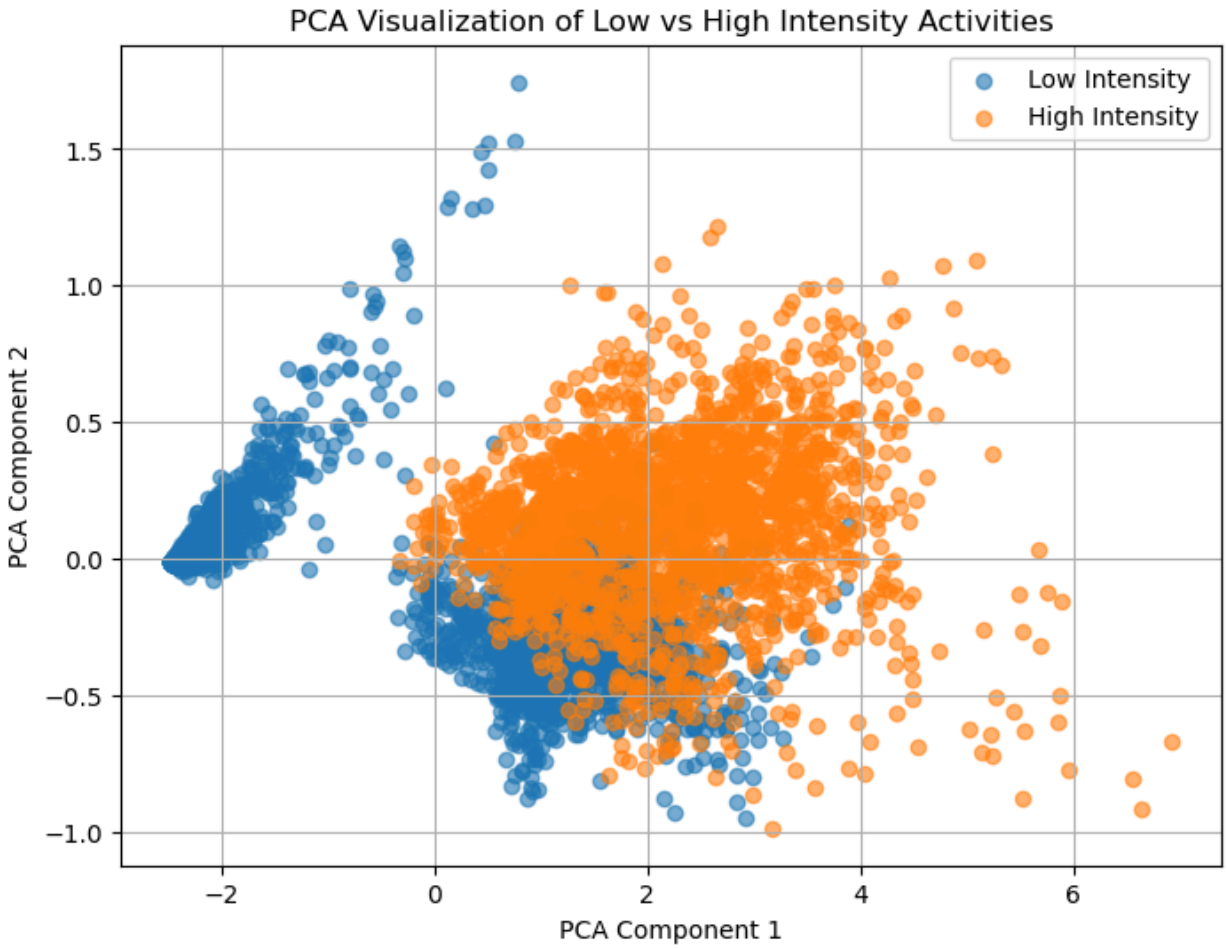

Figure 4. PCA visualization of Low vs. High intensity activities

Discussion

As mentioned in the results section, the PCA plot in *Figure 4* shows that perfect linear separation is not possible with the chosen features. This justifies the use of a non-linear classifier, such as the RBF kernel SVM, over an SVM with a linear or polynomial kernel. Additionally, the data was scaled before training the SVM on it. Without standardization, the data SVM heavily biases features with a large range, such as peak-to-peak acceleration, leading to poor generalization. With scaling, the model treats all features uniformly, improving the accuracy of the model. As noted above in the *Methods* section, the five chosen features were the mean, standard deviation, SMA of body acceleration, mean jerk magnitude, and peak-to-peak magnitude of the body acceleration. Time-domain accelerometer features were chosen because acceleration signals can differentiate movement intensity through signal magnitude and variability. Although gyroscope data provides information regarding activity type, it is used for distinguishing orientation changes, which do not necessarily indicate activity intensity.

Reflection

For this assignment, the SVM performed well in an offline environment where the model was trained and then tested on pre-collected data. However, several limitations can arise if it were deployed in a real-time cyber-physical system. For one, the SVM model was trained on data that was collected from volunteers in a controlled test environment. The data was also pre-processed by applying noise filters, sampling in fixed-width sliding windows, and filtered again through a Butterworth low-pass filter before the model was trained on it. In real-time, the classifier may not be as accurate because the real-time data may not get processed through robust, real-time filtering. Moreover, predicting activity intensity using the same SVM model across different devices may not yield accurate results due to sensor variability (e.g. smartphones and smartwatches use varying types of accelerometer and gyroscope sensors) and environmental noise (e.g. differences in device placement). Additionally, scalability may be an issue due to high prediction latency and slow training time. Although the RBF SVM is ideal because it is a non-linear classifier, it also has a higher training time due to higher algorithm complexity. This makes training computationally expensive and slow on large datasets typical of real-time CPSs. Aside from training, an RBF SVM is more computationally heavy than a linear model, which could affect the speed of real-time predictions due to the large influx of data [5].

Bonus

The ROC-AUC curve provides an additional performance metric. The model achieves a high ROC-AUC score of 0.9932; this is further demonstrated visually by the high area under the curve in *Figure 5*. The high ROC-AUC score indicates strong class separability and ability to rank performance accurately, as it performs significantly better than random guessing (which would have a score of 0.50).

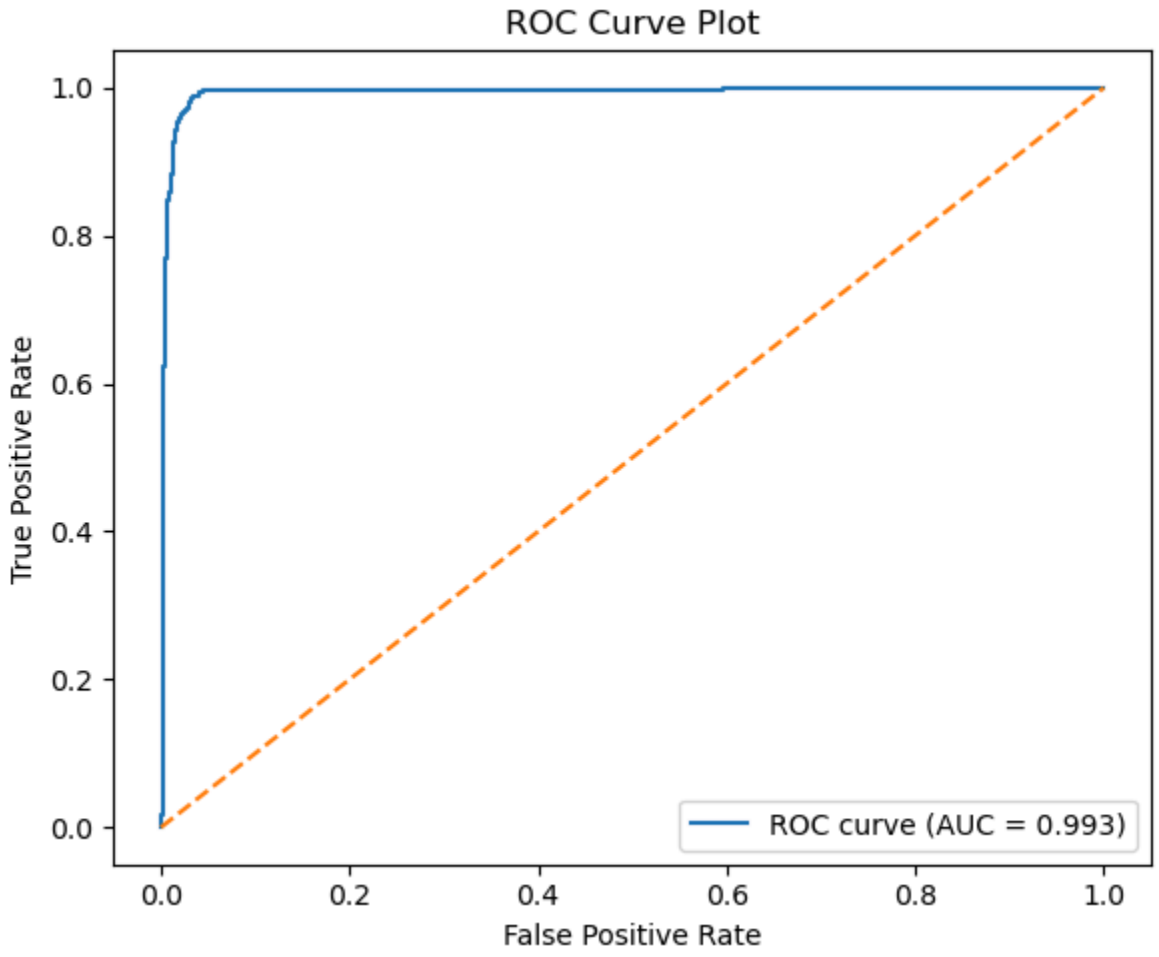

Figure 5. ROC Curve Plot

References

[1] H. Zhou *et al.*, “Towards Real-Time Detection of Gait Events on Different Terrains Using Time-Frequency Analysis and Peak Heuristics Algorithm,” *Sensors (Basel, Switzerland)*, vol. 16, no. 10, p. 1634, Oct. 2016, doi: 10.3390/s16101634.

[2] S. GN, “The RBF kernel in SVM: A Complete Guide,” Quark Machine Learning. Accessed: Nov. 22, 2025. [Online]. Available: https://www.quarkml.com/2022/10/the-rbf-kernel-in-svm-complete-guide.html

[3] “SVC,” scikit-learn. Accessed: Nov. 22, 2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[4] IBM, “PCA,” IBM. Accessed: Nov. 22, 2025. [Online]. Available: https://www.ibm.com/think/topics/principal-component-analysis

[5] GeeksforGeeks, “Why is scikitlearn SVM.SVC() extremely slow?,” *GeeksforGeeks*, Aug. 26, 2024. Accessed: Nov. 22, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/why-is-scikit-learn-svmsvc-extremely-slow/

Appendix

Code for data loading

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import matplotlib.pyplot as plt

# Load training data
train_path = 'assignment4_data/train/'

# Load activity data and labels
y_train = pd.read_csv(train_path + 'y_train.txt', delim_whitespace=True,
header=None)
subject_train = pd.read_csv(train_path + 'subject_train.txt',
delim_whitespace=True, header=None)
X_train = pd.read_csv(train_path + 'X_train.txt', delim_whitespace=True,
header=None)

# Load raw accelerometer data
acc_x = pd.read_csv(train_path + 'Inertial Signals/body_acc_x_train.txt',
delim_whitespace=True, header=None)
acc_y = pd.read_csv(train_path + 'Inertial Signals/body_acc_y_train.txt',
delim_whitespace=True, header=None)
acc_z = pd.read_csv(train_path + 'Inertial Signals/body_acc_z_train.txt',
delim_whitespace=True, header=None)

# Exclude laying for simplicity
exclude_laying = y_train[0] != 6

# Get the indices of the rows to keep
indices_to_keep = exclude_laying[exclude_laying].index

# Filter both y_test and X_test
y_train = y_train.loc[indices_to_keep].reset_index(drop=True)
X_train = X_train.loc[indices_to_keep].reset_index(drop=True)

acc_x = acc_x.loc[indices_to_keep].reset_index(drop=True)
```

```
acc_y = acc_y.loc[indices_to_keep].reset_index(drop=True)
acc_z = acc_z.loc[indices_to_keep].reset_index(drop=True)

# Show data distribution after removing laying
plt.figure(figsize=(6,4))
counts_raw = y_train[0].value_counts().sort_index()
bars = counts_raw.plot(kind='bar')

# Add number labels on top of bars
for i, value in enumerate(counts_raw):
    plt.text(i, value + 5, str(value), ha='center', va='bottom', fontsize=10)
plt.title("Activity Count Distribution (After Removing Laying)")
plt.xlabel("Activity Label")
plt.ylabel("Count")
plt.grid(True, axis='y')
plt.show()

print("Raw activity counts:")
print(y_train[0].value_counts().sort_index())
```

Code for feature engineering

```
def compute_features(acc_x, acc_y, acc_z, x_data):
    ## Features extracted: 201 tBodyAccMag-mean(), 202 tBodyAccMag-std(), 16
tBodyAcc-sma(), 227 tBodyAccJerkMag-mean()
    features = pd.DataFrame()

    # Columns you want to extract (Python is 0-indexed)
    columns_to_extract = [15, 200, 201, 226]

    selected_X_data = x_data.iloc[:, columns_to_extract].values

    # Extract selected features
    features['mean_acc'] = selected_X_data[:, 0]
    features['std_acc'] = selected_X_data[:, 1]
    features['sma_acc'] = selected_X_data[:, 2]
    features['mean_jerk_acc'] = selected_X_data[:, 3]

    mag_acc = np.sqrt(acc_x**2 + acc_y**2 + acc_z**2)
```

```
    # Custom Feature: Peak to peak magnitude
    features['peak_to_peak_acc'] = np.max(mag_acc, axis=1) - np.min(mag_acc,
axis=1)

    selected_features = features[['mean_acc', 'std_acc', 'sma_acc',
'mean_jerk_acc', 'peak_to_peak_acc']]

    return selected_features
```

Code for model training

```
def map_intensity(outcome):
    # 0 = low-intensity, 1 = high-intensity
    return outcome.replace({1: 0, 2: 1, 3: 1, 4: 0, 5: 0})

y_train = map_intensity(y_train)

# low vs high intensity class balance
plt.figure(figsize=(6,4))
counts_intensity = pd.Series(y_train[0]).value_counts().sort_index()
bars = counts_intensity.plot(kind='bar', color=['green', 'red'])

# Add number labels on bars
for i, value in enumerate(counts_intensity):
    plt.text(i, value + 5, str(value), ha='center', va='bottom', fontsize=10)

plt.title("Class Balance: Low vs High Intensity Activities")
plt.xlabel("Intensity Class (0 = Low, 1 = High)")
plt.ylabel("Count")
plt.grid(True, axis='y')
plt.show()

print("\nMapped intensity counts:")
print(pd.Series(y_train[0]).value_counts().sort_index())

y_train = y_train.values.ravel()

# Compute features for training
X_train_features = compute_features(acc_x.values, acc_y.values, acc_z.values,
X_train)
```

```
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_features)

# train SVM model
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train_scaled, y_train)
```

Code for evaluation

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA

# load testing data
test_path = 'assignment4_data/test/'

y_test = pd.read_csv(test_path + 'y_test.txt', delim_whitespace=True,
header=None)
X_test = pd.read_csv(test_path + 'X_test.txt', delim_whitespace=True,
header=None)

acc_x_test = pd.read_csv(test_path + 'Inertial Signals/body_acc_x_test.txt',
delim_whitespace=True, header=None)
acc_y_test = pd.read_csv(test_path + 'Inertial Signals/body_acc_y_test.txt',
delim_whitespace=True, header=None)
acc_z_test = pd.read_csv(test_path + 'Inertial Signals/body_acc_z_test.txt',
delim_whitespace=True, header=None)

# Exclude laying for simplicity
exclude_laying = y_test[0] != 6

# Get the indices of the rows to keep
indices_to_keep = exclude_laying[exclude_laying].index

acc_x_test = acc_x_test.loc[indices_to_keep].reset_index(drop=True)
acc_y_test = acc_y_test.loc[indices_to_keep].reset_index(drop=True)
acc_z_test = acc_z_test.loc[indices_to_keep].reset_index(drop=True)
y_test = y_test.loc[indices_to_keep].reset_index(drop=True)
X_test = X_test.loc[indices_to_keep].reset_index(drop=True)
```

```
y_test = map_intensity(y_test)

# Compute features for test set
X_test_features = compute_features(acc_x_test.values, acc_y_test.values,
acc_z_test.values, X_test)
X_test_scaled = scaler.transform(X_test_features)

# evaluate the model
y_pred = svm_model.predict(X_test_scaled)

print("Classification Report:\n", classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Visualize Confusion Matrix
plt.figure(figsize=(7,6))
plt.imshow(cm, cmap="Blues")
plt.colorbar()

classes = ["Low Intensity", "High Intensity"]

# Axis labels
plt.xticks(np.arange(2), classes)
plt.yticks(np.arange(2), classes)

# label squares
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha="center", va="center", color="black")

plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

# PCA on the feature-scaled data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)
```

```
plt.figure(figsize=(8,6))
plt.scatter(X_pca[y_train == 0, 0], X_pca[y_train == 0, 1], alpha=0.6, label='Low
Intensity')
plt.scatter(X_pca[y_train == 1, 0], X_pca[y_train == 1, 1], alpha=0.6,
label='High Intensity')

plt.title("PCA Visualization of Low vs High Intensity Activities")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.grid(True)
plt.show()
```

Code for Bonus

```
from sklearn.metrics import roc_curve, auc
y_score = svm_model.decision_function(X_test_scaled)

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_score)

# Compute AUC
roc_auc = auc(fpr, tpr)
print("ROC-AUC Score:", roc_auc)

# ROC curve plot
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--")  # chance line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Plot")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()
```