Kyle Tranfaglia

COSC 311

Project 02

May 12, 2024

Activity Recognition Report

Task 1:

1) Final Window Size: 96

   I determined the size using a function I constructed to test window sizes in the range [16, 1024] such that the range increments by a value of 16. The function iterates over each window length in the range and then over each dataset. In the process, for each dataset and window length, the data is segmented using a sliding window, and features are extracted from each segment. These segments are then normalized and used to train a model, SVC, after being split into testing and training data. The window length that produces the highest accuracy is kept. This approach demonstrates the sensitivity of the model to different window sizes. When using the window length range [1, 1024], the best length was 1; I chose not to use all the data and forced a larger window size. When setting the range to [64, 1024] with an increment of 64, I found that a window size of 256 performed well; however, after some comparison, despite a window size of 64 producing more perfect accuracies amongst all the classifiers, the window size of 96 performed best overall for all classifiers considered.

2) Activity Sample Sizes (number of samples generated for each activity)

   COUGH: 34

   DRINK: 104

EAT: 555

READ: 521

SIT: 283

WALK: 523

Task 2:

1) Multiple features were extracted from the datasets. The features include mean, standard deviation, minimum value, maximum value, and root mean square of each segment from the datasets. I chose these features for numerous reasons. First, these features are computationally efficient and simple to implement. Second, they are relatively robust to noise and outliers compared to more complex or sensitive metrics. They provide a simple yet effective way to capture essential aspects of the data while minimizing the impact of noise or irregularities. Third, these statistical features are very interpretable. For example, a high standard deviation may suggest variability or fluctuations in sensor readings, while a low RMS value may indicate relatively stable or uniform motion. Next, they can capture different aspects of the underlying patterns or characteristics present in the data. For instance, variations in mean and standard deviation may indicate changes in activity intensity or frequency, while min and max values can highlight extreme movements or outliers. Finally, these features are also great statistical measures that provide a concise summary of the distribution of data within each segment. The mean gives an indication of the central tendency, the standard deviation reflects the spread or dispersion of the data, the min and max capture the range of values (more importantly, the extremes), and RMS provides a measure of the overall magnitude of the data.

Task 3:

1) I tested multiple normalization methods on the data, all of which are included in the sklearn package. These normalization methods include the following scalers: StandardScaler, MinMaxScaler, RobustScaler, MaxAbsScaler, and Normalizer. After testing each normalization method, I decided to use the StandardScaler. MinMaxScaler was another good choice, as it led to three models getting perfect accuracy for independent testing after training. Although this is a great performance, not all six of the tested models performed to that level of success. Thus, I chose the StandardScaler as it yielded the highest accuracies overall when considering all models, and it still led the best classifier to get a perfect accuracy in all tests and four classifiers got a perfect score using independent-test.
2) After conducting an experiment to compare the performance with and without feature normalization, the results were not surprising. Across the board, the models performed significantly better with normalized data, and, for some models, over three times better. This is expected as the data consists of a large number of numerical values with a big variance in size. However, it is important to note that the normalization did not impact the Random Forest model as for each test, it yielded an identical accuracy score. Despite this, the following results provide clear evidence that normalization significantly improves the performance of the models for this dataset:

   Normalization vs No Normalization Testing

   Test Classifier: SVC

   Average accuracy with feature normalization: 0.9920855706278554

   Average accuracy without feature normalization: 0.8934650393292826

Test Classifier: KNN

Average accuracy with feature normalization: 0.9846686449060336

Average accuracy without feature normalization: 0.8934650393292826

Test Classifier: Random Forest

Average accuracy with feature normalization: 0.9529505793415289

Average accuracy without feature normalization: 0.9529505793415289

Test Classifier: MLP

Average accuracy with feature normalization: 0.9653808110781404

Average accuracy without feature normalization: 0.25891090857708066

Test Classifier: Logistic Regression

Average accuracy with feature normalization: 0.9564699025010598

Average accuracy without feature normalization: 0.7424344119447976

Test Classifier: Polynomial LR

Average accuracy with feature normalization: 0.9787339268051434

Average accuracy without feature normalization: 0.5663401982949461

Test Classifier: Voting

Average accuracy with feature normalization: 0.9846671730017427

Average accuracy without feature normalization: 0.9385744901323537

Task 4:

1) Feature reduction was not used for the final product of this project. Feature reduction was not necessary as the accuracies for all the models were already beyond proficient.

   The overall accuracies are as follows (average of self-test, independent-test, and CVT):

1. Voting: 1.0
2. SVC: 0.9998349834983499
3. Polynomial LR: 0.9998349834983499
4. MLP: 0.9988448844884489
5. KNN: 0.9986798679867986
6. Random Forest: 0.9933993399339934
7. Logistic Regression: 0.9805280528052805

Additionally, the independent test yielded 4 perfect scores. Despite this, I still implemented a feature reduction function, and upon limiting the features to 1, 2, 3, and 4, as opposed to the 5 extracted features, the overall performance, when considering all the models and the best model performance, was lower.

2) The performance comparison (accuracy) for all the classifiers using each test, self-test, independent-test, and CVT, is as follows:

Model Testing using self-test, independent-test, and cross-validation

Test Classifier: SVC

Average accuracy with self-test: 1.0

Average accuracy with independent-test: 1.0

Average accuracy with cross-validation: 0.9995049504950495

Average accuracy of all tests 0.9998349834983499

Test Classifier: KNN

Average accuracy with self-test: 0.999009900990099

Average accuracy with independent-test: 0.9975247524752475

Average accuracy with cross-validation: 0.9995049504950495

Average accuracy of all tests 0.9986798679867986

Test Classifier: Random Forest

Average accuracy with self-test: 1.0

Average accuracy with independent-test: 1.0

Average accuracy with cross-validation: 0.9801980198019802

Average accuracy of all tests 0.9933993399339934

Test Classifier: MLP

Average accuracy with self-test: 1.0

Average accuracy with independent-test: 0.9975247524752475

Average accuracy with cross-validation: 0.9990099009900991

Average accuracy of all tests 0.9988448844884489

Test Classifier: Logistic Regression

Average accuracy with self-test: 0.9876237623762376

Average accuracy with independent-test: 0.9801980198019802

Average accuracy with cross-validation: 0.9737623762376237

Average accuracy of all tests 0.9805280528052805

Test Classifier: Polynomial LR

Average accuracy with self-test: 1.0

Average accuracy with independent-test: 1.0

Average accuracy with cross-validation: 0.9995049504950495

Average accuracy of all tests 0.9998349834983499

Test Classifier: Voting

Average accuracy with self-test: 1.0

Average accuracy with independent-test: 1.0

Average accuracy with cross-validation: 1.0

Average accuracy of all tests 1.0

Classifier Rankings

1. Voting: 1.0
2. SVC: 0.9998349834983499
3. Polynomial LR: 0.9998349834983499
4. MLP: 0.9988448844884489
5. KNN: 0.9986798679867986
6. Random Forest: 0.9933993399339934
7. Logistic Regression: 0.9805280528052805

3) Upon analysis of the above accuracy results, the voting classifier is best as it is the only classifier to obtain a perfect accuracy for all tests, yielding an average accuracy score of 1.0. Other classifiers obtained perfect accuracy scores for individual tests, but none other than the voting classifier scored perfect on all three, making it the best classifier. The voting model is a bit computationally heavy as it requires the remaining classifiers to train and predict the true label and then vote using a soft voting system. However, the extra computation is justified by the perfect accuracy amongst all tests.

The classification report for the voting classifier is as follows:

Classification reports for best classifier: Voting

Classification report with self-test:

	precision	recall	f1-score	support
COUGH	1.00	1.00	1.00	34
DRINK	1.00	1.00	1.00	104
EAT	1.00	1.00	1.00	555
READ	1.00	1.00	1.00	521
SIT	1.00	1.00	1.00	283
WALK	1.00	1.00	1.00	523
accuracy			1.00	2020
macro avg	1.00	1.00	1.00	2020
weighted avg	1.00	1.00	1.00	2020

Classification report with independent-test:

	precision	recall	f1-score	support
COUGH	1.00	1.00	1.00	8
DRINK	1.00	1.00	1.00	18
EAT	1.00	1.00	1.00	116
READ	1.00	1.00	1.00	111
SIT	1.00	1.00	1.00	55
WALK	1.00	1.00	1.00	96

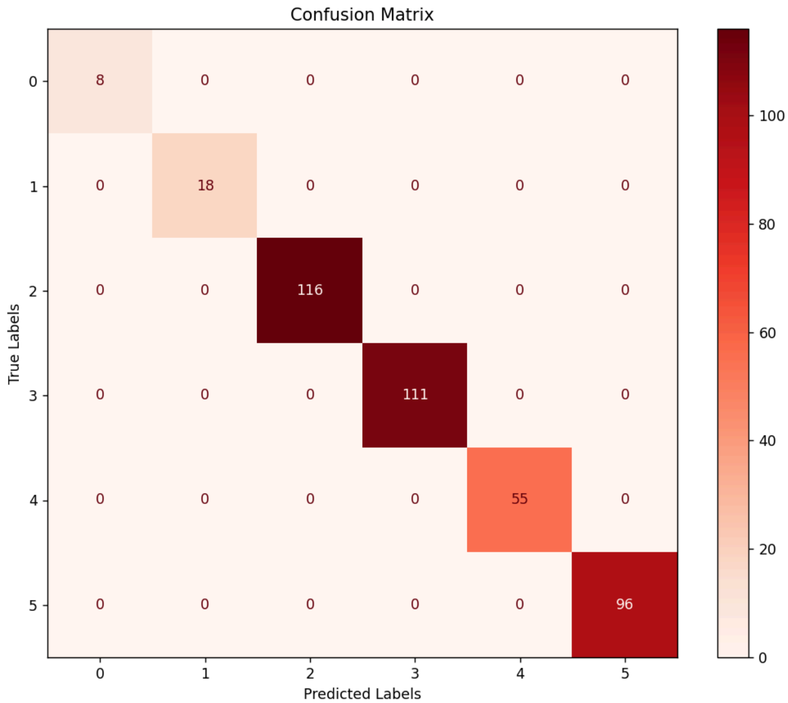
	precision	recall	f1-score	support
accuracy			1.00	404
macro avg	1.00	1.00	1.00	404
weighted avg	1.00	1.00	1.00	404

Classification report with cross-validation:

	precision	recall	f1-score	support
COUGH	1.00	1.00	1.00	34
DRINK	0.78	1.00	0.87	104
EAT	1.00	1.00	1.00	555
READ	1.00	0.94	0.97	521
SIT	1.00	1.00	1.00	283
WALK	1.00	1.00	1.00	523
accuracy			0.98	2020
macro avg	0.96	0.99	0.97	2020
weighted avg	0.99	0.98	0.99	2020

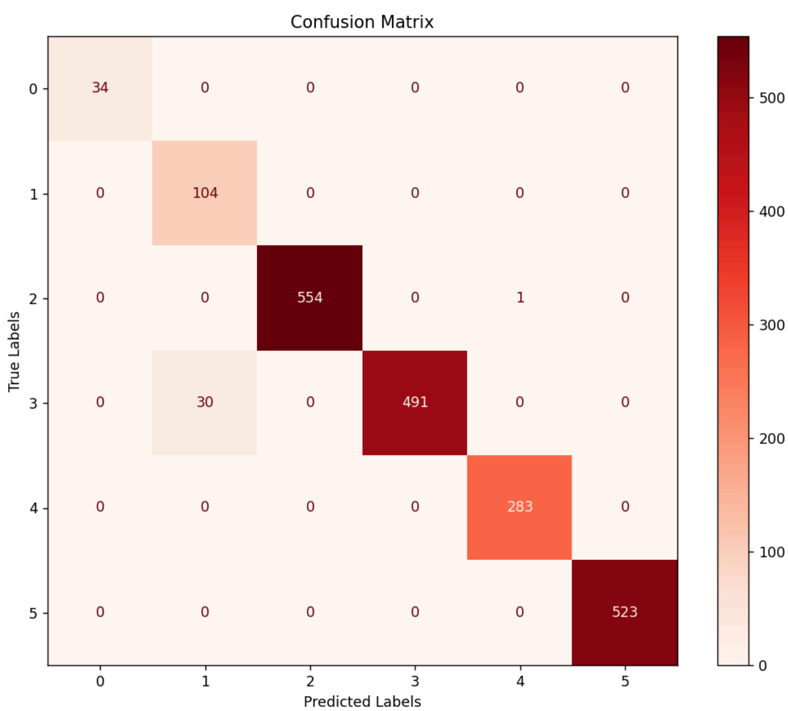
Confusion matrices for voting classifier:

Independent-test:



Confusion Matrix
True Labels
Predicted Labels

CVT:



Confusion Matrix
True Labels
Predicted Labels

For the voting classifier, there were no cases of misclassification, as indicated by the confusion matrix, which displays an all-zero matrix except for the main diagonal, which is all the true labels that were correctly identified. Additionally, this is represented by the perfect accuracy score for all tests on the voting classifier. Therefore, misclassification does not often happen for any activity, as it never happens for this model.

Task 5:

1) During this project, I acquired extensive experience in machine learning. First, I learned about sliding windows and data segmentation, how to optimize a sliding window by finding an ideal window size, and overall, how to handle large datasets in preparation for feature extraction. Second, I gained experience in feature extraction, such that I had to intuitively decide on feature extraction methods depending on the dataset, and experiment with the extraction to fine-tune the process. Third, I learned how vast an impact normalization can have on a classifier's accuracy, depending on the dataset. During this process, I was exposed to a large library of scalers that are available for normalizing data. This project also provided me with much more experience in model training and testing, as it required me to test feature reduction methods and determine if it was necessary or beneficial for the project. This project also increased my understanding of CVT and using multiple testing methods to evaluate the performance of a model and to use that information to classify the best model for the data.
2) Overall, the model performance is excellent. However, like all things, it is not perfect. To improve the performance of the models, I could spend more time fine-tuning the parameters of the models not obtaining 1.0 accuracy, attempt feature reduction again,

restructure the feature extraction method (extract different / more / less features), or experiment with more window sizes. Each of these things greatly impacts the performance of a model and is very flexible, such that many variations and applications are available for testing, especially with data extraction.

3) The most challenging task I encountered in the project at first was figuring out how to make a sliding window, as this was something that I had never done before. After some struggle, I got my sliding window function operating correctly. However, once I realized I needed to find a way to determine the best window size, I encountered a more challenging task. I eventually figured out how to compute this, but it took a lot of thought, failed attempts, and many run-time errors. The best window size computation is not only one of my longest functions, but it took me the longest to write and was the hardest for me to grasp. Therefore, the most challenging task, especially since it involves the sliding window function, was the best sliding window computation.
4) After finishing the project, I still have the following questions:
    a) What is the benefit of doing a self-test? Isn't it not as helpful as an independent test in most cases?
    b) Is normalization always beneficial as long as a stronger feature is not losing weight in the process, or if many bad features are not gaining weight? It seems to be useful as long as all the features are at least decent.
    c) Since KNN is simple, computationally efficient, and a "lazy algorithm," should this always be the first model to test with since it may perform well despite being simplistic? In this project, it performs very well, even with varying neighbor amounts, proving that not all datasets require a complex model.

Code:

"""

Kyle Tranfaglia

COSC311 - Project02

Last updated 05/08/24

Task 1: Data Segmentation - segmentation of data with sliding window.

Task 2: Feature Extraction - Each segment is used to extract multiple features to represent an activity.

Task 3: Dataset Generation - Combination of all features and corresponding activity labels to generate sample. Features

normalized before model train and test

Task 4: Model Training and testing - Experiment to compare classifiers and find the best classifier for the dataset.

The best classifier is denoted by best overall performance in modeling data (accurate labeling)

Task 5: Experience and Potential Improvements - Reflection of project

"""

import numpy as np

import pandas as pd

import seaborn as sb

import matplotlib.pyplot as plt

from sklearn.model_selection import cross_val_score, cross_val_predict

from sklearn.svm import SVC

```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report,
accuracy_score
from sklearn.preprocessing import (StandardScaler, PolynomialFeatures, MinMaxScaler,
RobustScaler,
                               MaxAbsScaler, PowerTransformer, QuantileTransformer, Normalizer)


# Sliding window function to segment data
def sliding_window(df, win_len):
    segments = []
    num_samples = len(df)
    # Iterate over the data with a step size of window length
    for i in range(0, num_samples - win_len + 1, win_len):
        segment = df.iloc[i:i + win_len]  # Get segment of data including 'window_length'
consecutive samples
```

```
        segments.append(segment)  # Add the segment to the list of segments
    return segments



# Determine the best window length for segmentation
def best_window_length(act_data, all_labels):
    window_lengths = range(32, 1025, 32)  # Define window length range to run test
    best_accuracy = 0
    best_window_length = 0
    # Iterate over each window length in range
    for win_len in window_lengths:
        samples = []
        labels = []
        # Iterate over each dataset
        for i, df in enumerate(act_data):
            segments = sliding_window(df, win_len)  # Segment the data using sliding window
            # Extract features from each segment
            for segment in segments:
                features = segment.values.flatten()  # Flatten feature (segment) values to 1D list
                samples.append(features)  # Append to sample lis
                labels.append(all_labels[i])  # Append corresponding label to a list to track true label

        # Normalize the features
```

```
        scaler = StandardScaler()
        samples_normalized = scaler.fit_transform(samples)

        # Split the data into training and testing sets
        x_train, x_test, y_train, y_test = train_test_split(samples_normalized, labels, test_size=0.2,
random_state=7)

        # Train a classifier
        svc = SVC(kernel='linear', C=6, gamma='scale', random_state=7)
        svc.fit(x_train, y_train)

        # Evaluate the classifier
        prediction = svc.predict(x_test)
        accuracy = accuracy_score(y_test, prediction)

        # Update the best window length if the current one has higher accuracy
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_window_length = win_len

    return best_window_length
```

```
# Extract the features
def extract_features(segment):
    features = []

    # Statistical features
    features.extend(segment.mean(axis=0))  # Mean along each axis
    features.extend(segment.std(axis=0))  # Standard deviation along each axis
    features.extend(segment.max(axis=0))  # Maximum value along each axis
    features.extend(segment.min(axis=0))  # Minimum value along each axis
    features.extend(np.sqrt(np.mean(segment ** 2, axis=0)))  # Root Mean Square (RMS) along
each axis

    return features



# Display confusion matrix using matplot (Display as heatmap)
def display_confusion_matrix(cm):
    # Display a heatmap / confusion matrix using matplotlib and the sklearn toolset
    matrix_display = ConfusionMatrixDisplay(confusion_matrix=cm)
    fig, ax = plt.subplots(figsize=(10, 8))  # Create layout and structure figure
    matrix_display.plot(ax=ax, cmap='Reds')  # Create Plot
    # Plot labels
    plt.xlabel('Predicted Labels')
```

```
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    # plt.savefig('confusion_matrix.png')  # Save plot as png
    plt.show()  # Display plot


# Evaluate classifiers using self test
def evaluate_self_test(classifier, features, labels, display=0):
    classifier.fit(features, labels)  # Fit the classifier on the entire dataset
    predictions = classifier.predict(features)  # Make predictions
    if display == 0:
        return accuracy_score(labels, predictions)  # Calculate accuracy
    elif display == 1:
        return classification_report(labels, predictions)  # Create classification report
    else:
        cm = confusion_matrix(labels, predictions)  # Create confusion matrix
        display_confusion_matrix(cm)


# Evaluate classifiers using independent test
def evaluate_independent_test(classifier, features_train, labels_train, features_test, labels_test,
display=0):
    classifier.fit(features_train, labels_train)  # Fit the classifier on the training data
```

```
    predictions = classifier.predict(features_test)  # Make predictions on the test data
    if display == 0:
        return accuracy_score(labels_test, predictions)  # Calculate accuracy
    elif display == 1:
        return classification_report(labels_test, predictions)  # Create classification report
    else:
        cm = confusion_matrix(labels_test, predictions)  # Create confusion matrix
        display_confusion_matrix(cm)


# Evaluate classifiers using cross-validation test
def evaluate_cross_validation(classifier, features, labels, display=0):
    scores = cross_val_score(classifier, features, labels, cv=10)  # Use cross_val_score to perform
cross-validation
    predictions = cross_val_predict(classifier, features, labels, cv=6)  # Predict true labels
    if display == 0:
        return np.mean(scores)  # Calculate the average accuracy
    elif display == 1:
        return classification_report(labels, predictions)  # Create classification report
    else:
        cm = confusion_matrix(labels, predictions)  # Create confusion matrix
        display_confusion_matrix(cm)
```

```
# Condense features by eliminating them based on importance
def condenseFeatures(attributeTrain, targetTrain, numFeatures):
    estimator = LogisticRegression(max_iter=1000)  # Instantiate logistic regression as the estimator

    # Instantiate RFE with logistic regression estimator and recursively select top _ features, then fit RFE to the data
    rfe = RFE(estimator, n_features_to_select=numFeatures)
    rfe.fit(attributeTrain, targetTrain)

    return rfe.support_  # Return selected features


# Main

# Task 1: Data Segmentation

# Load CSV files into DataFrames
files = ["COUGH.csv", "DRINK.csv", "EAT.csv", "READ.csv", "SIT.csv", "WALK.csv"]
activity_data = [pd.read_csv(file, skiprows=1) for file in files]
```

```
all_labels = ["COUGH", "DRINK", "EAT", "READ", "SIT", "WALK"]  # Define the list of all
activity labels

# Compute and display the best window length (one that yields the highest model accuracy)
window_length = best_window_length(activity_data, all_labels)
print("Best Window Length Found:", window_length, "\n")

# window_length = 512  # Define window length

# Segment data into a list (of lists)
segmented_data = [sliding_window(dataset, window_length) for dataset in activity_data]

print("Activity Sample Sizes")
for i, segment in enumerate(segmented_data):
    print(all_labels[i] + ":", len(segment))

# Task 2: Feature Extraction

# Store features and labels for each segment
features_per_segment = []
labels_per_segment = []

# Iterate over each dataset (list of segments) in segmented_data
```

```
for i, dataset_segments in enumerate(segmented_data):
    # Iterate over each segment in the current dataset
    for segment in dataset_segments:
        features = extract_features(segment)  # Extract features for the current segment
        features_per_segment.append(features)  # Append the extracted features to the
features_per_segment list
        labels_per_segment.append(all_labels[i])  # Append the corresponding label to the
labels_per_segment list

# Task 3: Dataset Generate

# Combine features and labels for each segment to generate samples
samples = np.array(features_per_segment)
labels = np.array(labels_per_segment)

# Normalize the features
scaler = StandardScaler()
samples_normalized = scaler.fit_transform(samples)

# Define classifiers
classifiers = {
    "SVC": SVC(kernel='linear', C=6, probability=True, gamma='scale', random_state=7),
    "KNN": KNeighborsClassifier(n_neighbors=3),
```

```
    "Random Forest": RandomForestClassifier(n_estimators=315, criterion='gini', max_depth=14,
min_samples_split=3,
                                            min_samples_leaf=3, max_features='sqrt', random_state=7),
    "MLP": MLPClassifier(hidden_layer_sizes=100, activation='tanh', solver='adam', alpha=1e-5,
batch_size=36, tol=1e-6,
                          learning_rate_init=0.01, learning_rate='constant', max_iter=10000,
random_state=7),
    "Logistic Regression": LogisticRegression(solver='liblinear', random_state=7),
    "Polynomial LR": make_pipeline(PolynomialFeatures(2),
LogisticRegression(solver='liblinear', random_state=7))
}

# Add VotingClassifier separately in order to use all classifiers in dict for voting
voting_classifier = VotingClassifier(estimators=list(classifiers.items()), voting='soft')
classifiers["Voting"] = voting_classifier

print("\nNormalization vs No Normalization Testing")

# Evaluate classifiers with and without normalization to compare accuracy results
for classifier_name, classifier_obj in classifiers.items():
    # Experiment 1: With feature normalization
    scores_with_normalization = cross_val_score(classifier_obj, samples_normalized, labels,
cv=6)
```

```
    # Experiment 2: Without feature normalization
    scores_without_normalization = cross_val_score(classifier_obj, samples, labels, cv=6)

    # Compare and display the average performance metrics
    print("Test Classifier:", classifier_name)
    print("Average accuracy with feature normalization:", scores_with_normalization.mean())
    print("Average accuracy without feature normalization:",
scores_without_normalization.mean())

# Task 4: Model Training and Testing

# Split the data into training and testing subsets for independent testing
x_train, x_test, y_train, y_test = train_test_split(samples_normalized, labels, test_size=0.2,
random_state=7)

print("\nModel Testing using self-test, independent-test, and cross-validation")

classifiers_results = {}
# Evaluate classifiers using different evaluation methods
for classifier_name, classifier_obj in classifiers.items():
    # Call functions to test the models using a specific evaluation method
    self_test_results = evaluate_self_test(classifier_obj, samples_normalized, labels)
```

```
    independent_test_results = evaluate_independent_test(classifier_obj, x_train, y_train, x_test,
y_test)
    cross_validation_results = evaluate_cross_validation(classifier_obj, samples_normalized,
labels)
    result_average = (self_test_results + independent_test_results + cross_validation_results) / 3
    classifiers_results[classifier_name] = result_average  # Append to dictionary of average
accuracy results

    # Compare and display the average performance metrics
    print("Test Classifier:", classifier_name)
    print("Average accuracy with self-test:", self_test_results)
    print("Average accuracy with independent-test:", independent_test_results)
    print("Average accuracy with cross-validation:", cross_validation_results)
    print("Average accuracy of all tests", result_average)

# Sort the dictionary containing the average accuracy results
sorted_classifier_results = dict(sorted(classifiers_results.items(), key=lambda item: item[1],
reverse=True))

# Display ranking of all classifiers, ordered from highest to lowest average accuracy
print("\nClassifier Rankings")
for i, (key, value) in enumerate(sorted_classifier_results.items()):
    print(str(i + 1) + ". ", key + ":", value)
```

best_classifier = next(iter(sorted_classifier_results)) # Get the best classifier (first in sorted dictionary)

Display classification report for best classifier with self-test, independent-test, and CVT

The same functions for accuracy evaluations are used but with an end parameter of 1 to denote classification

self_test_results = evaluate_self_test(classifiers[best_classifier], samples_normalized, labels, 1)

independent_test_results = evaluate_independent_test(classifiers[best_classifier], x_train, y_train, x_test, y_test, 1)

cross_validation_results = evaluate_cross_validation(classifiers[best_classifier], samples_normalized, labels, 1)

print("\nClassification reports for best classifier:", best_classifier)

print("Classification report with self-test:\n", self_test_results)

print("Classification report with independent-test:\n", independent_test_results)

print("Classification report with cross-validation:\n", cross_validation_results)

Display the confusion matrix in a figure for best classifier with independent-test and CVT

The same functions for accuracy evaluations are used but with an end parameter of 2 to denote confusion matrix

evaluate_independent_test(classifiers[best_classifier], x_train, y_train, x_test, y_test, 2)

evaluate_cross_validation(classifiers[best_classifier], samples_normalized, labels, 2)

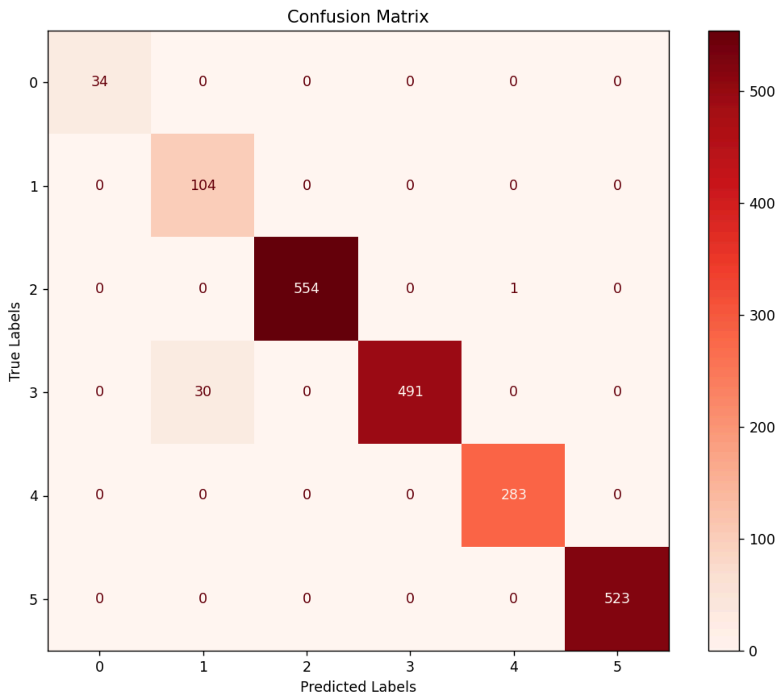
Results:

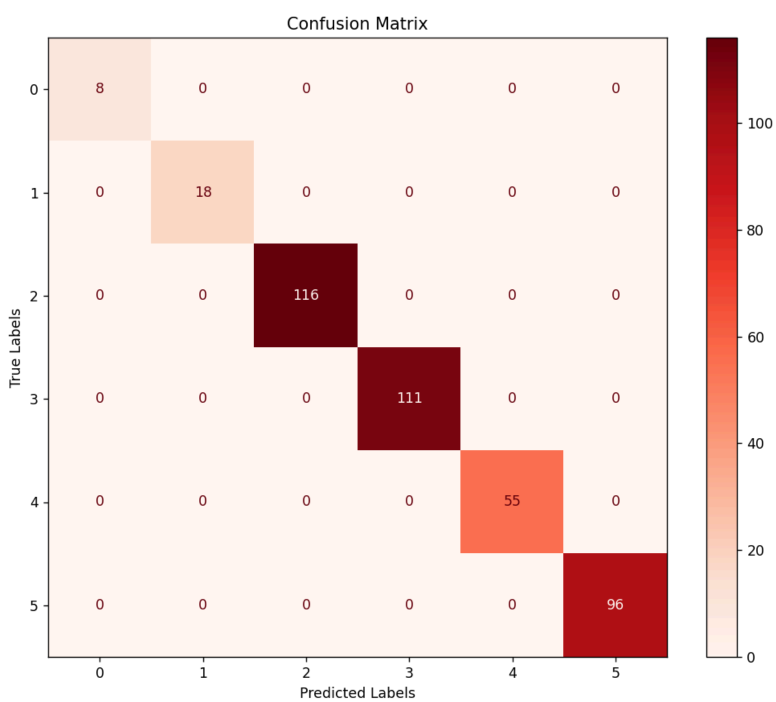
```
Classification report with independent-test:
              precision    recall  f1-score   support

       COUGH       1.00      1.00      1.00         8
       DRINK       1.00      1.00      1.00        18
         EAT       1.00      1.00      1.00       116
        READ       1.00      1.00      1.00       111
         SIT       1.00      1.00      1.00        55
        WALK       1.00      1.00      1.00        96

    accuracy                           1.00       404
   macro avg       1.00      1.00      1.00       404
weighted avg       1.00      1.00      1.00       404

Classification report with cross-validation:
              precision    recall  f1-score   support

       COUGH       1.00      1.00      1.00        34
       DRINK       0.78      1.00      0.87       104
         EAT       1.00      1.00      1.00       555
        READ       1.00      0.94      0.97       521
         SIT       1.00      1.00      1.00       283
        WALK       1.00      1.00      1.00       523

    accuracy                           0.98      2020
   macro avg       0.96      0.99      0.97      2020
weighted avg       0.99      0.98      0.99      2020
```

```
Classifier Rankings
1.  Voting: 1.0
2.  SVC: 0.9998349834983499
3.  Polynomial LR: 0.9998349834983499
4.  MLP: 0.9988448844884489
5.  KNN: 0.9986798679867986
6.  Random Forest: 0.9933993399339934
7.  Logistic Regression: 0.9805280528052805

Classification reports for best classifier: Voting
Classification report with self-test:
              precision    recall  f1-score   support

       COUGH       1.00      1.00      1.00        34
       DRINK       1.00      1.00      1.00       104
         EAT       1.00      1.00      1.00       555
        READ       1.00      1.00      1.00       521
         SIT       1.00      1.00      1.00       283
        WALK       1.00      1.00      1.00       523

    accuracy                           1.00      2020
   macro avg       1.00      1.00      1.00      2020
weighted avg       1.00      1.00      1.00      2020
```

```
Model Testing using self-test, independent-test, and cross-validation
Test Classifier: SVC
Average accuracy with self-test: 1.0
Average accuracy with independent-test: 1.0
Average accuracy with cross-validation: 0.9995049504950495
Average accuracy of all tests 0.9998349834983499
Test Classifier: KNN
Average accuracy with self-test: 0.999009900990099
Average accuracy with independent-test: 0.9975247524752475
Average accuracy with cross-validation: 0.9995049504950495
Average accuracy of all tests 0.9986798679867986
Test Classifier: Random Forest
Average accuracy with self-test: 1.0
Average accuracy with independent-test: 1.0
Average accuracy with cross-validation: 0.9801980198019802
Average accuracy of all tests 0.9933993399339934
Test Classifier: MLP
Average accuracy with self-test: 1.0
Average accuracy with independent-test: 0.9975247524752475
Average accuracy with cross-validation: 0.9990099009900991
Average accuracy of all tests 0.9988448844884489
Test Classifier: Logistic Regression
Average accuracy with self-test: 0.9876237623762376
Average accuracy with independent-test: 0.9801980198019802
Average accuracy with cross-validation: 0.9737623762376237
Average accuracy of all tests 0.9805280528052805
Test Classifier: Polynomial LR
Average accuracy with self-test: 1.0
Average accuracy with independent-test: 1.0
Average accuracy with cross-validation: 0.9995049504950495
Average accuracy of all tests 0.9998349834983499
Test Classifier: Voting
Average accuracy with self-test: 1.0
Average accuracy with independent-test: 1.0
Average accuracy with cross-validation: 1.0
Average accuracy of all tests 1.0
```

```
Best Window Length Found: 96

Activity Sample Sizes
COUGH: 34
DRINK: 104
EAT: 555
READ: 521
SIT: 283
WALK: 523

Normalization vs No Normalization Testing
Test Classifier: SVC
Average accuracy with feature normalization: 0.9920855706278554
Average accuracy without feature normalization: 0.8934650393292826
Test Classifier: KNN
Average accuracy with feature normalization: 0.9846686449060336
Average accuracy without feature normalization: 0.8934650393292826
Test Classifier: Random Forest
Average accuracy with feature normalization: 0.9529505793415289
Average accuracy without feature normalization: 0.9529505793415289
Test Classifier: MLP
Average accuracy with feature normalization: 0.9653808110781404
Average accuracy without feature normalization: 0.25891090857708066
Test Classifier: Logistic Regression
Average accuracy with feature normalization: 0.9564699025010598
Average accuracy without feature normalization: 0.7424344119447976
Test Classifier: Polynomial LR
Average accuracy with feature normalization: 0.9787339268051434
Average accuracy without feature normalization: 0.5663401982949461
Test Classifier: Voting
Average accuracy with feature normalization: 0.9846671730017427
Average accuracy without feature normalization: 0.9385744901323537
```


Confusion Matrix
True Labels
Predicted Labels
34 0 0 0 0 0
0 104 0 0 0 0
0 0 554 0 1 0
0 30 0 491 0 0
0 0 0 0 283 0
0 0 0 0 0 523
0 1 2 3 4 5
0 100 200 300 400 500


Confusion Matrix
True Labels
Predicted Labels
8 0 0 0 0 0
0 18 0 0 0 0
0 0 116 0 0 0
0 0 0 111 0 0
0 0 0 0 55 0
0 0 0 0 0 96
0 1 2 3 4 5
0 20 40 60 80 100