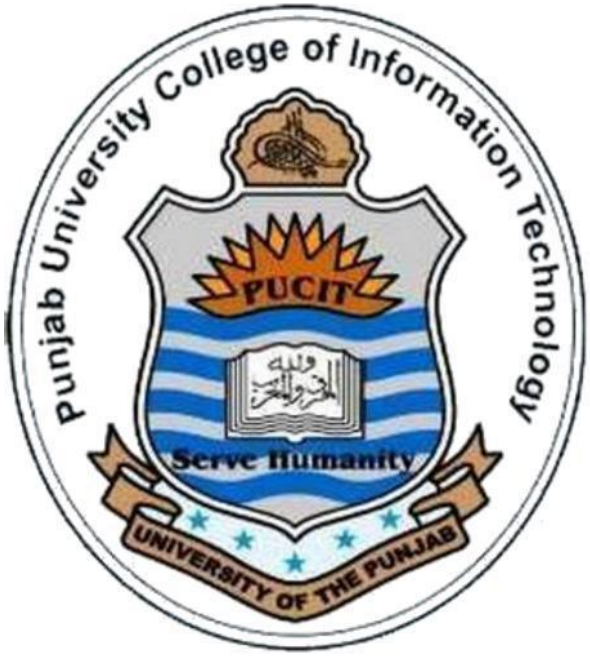
PUNJAB UNIVERSITY COLLEGE OF INFORMATION AND TECHNOLOGY



Submitted To:

DR. MUHAMMMAD ADEEL NISAR

Submitted By:

BCSF20M001	MUNAZZA SHAHZAD MUGHAL
BCSF20M0013	HAFIZA LAIBA QASIM
BCSF20M028	AYESHA ARSHAD
BCSF20M038	TOOBA ATIF
BCSF20M061	FATIMA HABIB

ACTVITY RECOGNITION FOR COGNITIVE VILLAGE DATASET

- Firstly, we mounted Google Drive to access the project files.
- Then we imported all the libraries i.e., NumPy, pandas, sklearn to implement all the functions and classifiers.
- We load the labels and training data one by one from NumPy files into numpy arrays data structures.
- The shapes of the training data (9 arrays of sensor data and one of labels) are then displayed which can be seen in the screenshot below.

```
[ ]
    (2284, 800, 3)
    (2284, 800, 3)
    (2284, 268, 3)
    (2284, 80, 3)
    (2284, 80, 3)
    (2284, 800, 3)
    (2284, 200, 3)
    (2284, 268, 3)
    (2284, 268, 3)
    (2284,)
```

- Test labels and test data from NumPy files are loaded in array data structures and then their shape is displayed.

```
]
    (2284, 800, 3)
    (2284, 800, 3)
    (2284, 268, 3)
    (2284, 80, 3)
    (2284, 80, 3)
    (2284, 800, 3)
    (2284, 200, 3)
    (2284, 268, 3)
    (2284, 268, 3)
    (2284,)
```

- Then we reshape the **training** data and then **normalize** the data by using fit_transform method that calculates the mean and standard deviation of the features and uses them to perform the scaling operation. Then, reshape them back to original shape.
- The above step is repeated for the **testing** data.
- Six different features are computed for each **training** example of every file and then stored in the array. The specific features calculated are the mean, maximum, minimum, square root of mean squared values, number of zero crossings, and standard deviation.
- The above step is repeated for the **testing** data.

- Extracted features are then concatenated to have a suitable shape for classification algorithms.
- Now, we have successfully prepared the preprocessed and feature-extracted data for classification. The next steps would involve applying classification algorithms, evaluating the performance using metrics like accuracy and F1 score, and comparing different classification methods.

This report aims to compare the results of various classifiers based on the data provided. The classifiers evaluated in this analysis include DNN, Random Forest, Support Vector Machine (SVM), and KNN. The evaluation metrics used for comparison are confusion matrix, accuracy and weighted F1 score.

The following table summarizes the performance metrics of each classifier:

Classifier	**Accuracy**	**F1 Score**
DNN	18%	0.15
Random Forest	45 %	0.44
SVM	20%	0.15
KNN	16%	0.15

Based on the provided results, the **Random Forest** classifier outperforms the other classifiers in terms of overall accuracy, and F1-score. It achieved an accuracy of 45%, which indicates that it correctly classified 45% of the instances in the dataset.

```
[ ]  # Random Forest Classifier
     rf = RandomForestClassifier()
     rf.fit(X_train, y_train)
     rf_predictions = rf.predict(X_test)
     rf_accuracy = accuracy_score(y_test, rf_predictions)
     rf_f1_score = f1_score(y_test, rf_predictions, average='weighted')
     rf_confusion_matrix = confusion_matrix(y_test, rf_predictions)

     print("Random Forest Accuracy:", rf_accuracy*100,"%")
     print("Random Forest F1 Score:", rf_f1_score)
     print("Random Forest Confusion Matrix:")
     print(rf_confusion_matrix)
```

```
Random Forest Accuracy: 45.45454545454545 %
Random Forest F1 Score: 0.4465083073173811
Random Forest Confusion Matrix:
[[32  1  0 ...  0  0  0]
 [ 0 39  1 ...  1  0  0]
 [ 0  7 10 ...  0  1  1]
 ...
 [ 0  3  0 ... 15  0  0]
 [ 0  0  0 ...  0 32  0]
 [ 0  0  0 ...  0  0 28]]
```

The **DNN** achieved an accuracy of 18%. However, its F1-score is lower compared to the Random Forest classifier. This indicates that the DNN had a higher number of false positives and false negatives.

```
dnn = MLPClassifier(hidden_layer_sizes=(256, 128, 64), activation='relu',learning_rate_init=0.03,max_iter=2000)
dnn.fit(X_train, y_train)
dnn_predictions = dnn.predict(X_test)
dnn_accuracy = accuracy_score(y_test, dnn_predictions)
dnn_f1_score = f1_score(y_test, dnn_predictions, average='weighted')
dnn_confusion_matrix = confusion_matrix(y_test, dnn_predictions)

# Print the evaluation metrics
print("DNN Accuracy:", dnn_accuracy*100)
print("DNN F1 Score:", dnn_f1_score)
print("DNN Confusion Matrix:")
print(dnn_confusion_matrix)
```

```
DNN Accuracy: 18.96853146853147
DNN F1 Score: 0.1549968198059873
DNN Confusion Matrix:
[[12  1  0 ...  0  1  0]
 [ 0  5  1 ...  2  0  0]
 [ 0  2  0 ...  2  1  0]
 ...
 [ 0  0  0 ...  1  0  0]
 [ 0  0  0 ...  0  2  0]
 [ 0  0  0 ...  1  0  0]]
```

The **Support Vector Machine (SVM)** classifier achieved an accuracy of 20%. SVMs are effective in handling high-dimensional data, but in this scenario, the Random Forest classifier performed well.

```
classfier = SVC(C=1.0, kernel='linear')
classfier.fit(X_train,y_train)
estimatedLabels = classfier.predict(X_test)

svm_accuracy = accuracy_score(y_test,estimatedLabels)
svm_f1_score = f1_score(y_test,estimatedLabels,average='weighted')
svm_confusion_matrix = confusion_matrix(y_test,estimatedLabels)

# Print the evaluation metrics
print("SVM Accuracy:", svm_accuracy*100,"%")
print("SVM F1 Score:", svm_f1_score)
print("SVM Confusion Matrix:")
print(svm_confusion_matrix)
```

```
SVM Accuracy: 20.06118881118881 %
SVM F1 Score: 0.15737104875619132
SVM Confusion Matrix:
[[16  0  0 ...  0  0  0]
 [ 0 30  0 ...  0  0  0]
 [ 0  9  0 ...  0  0  0]
 ...
 [ 0  2  1 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]]
```

KNN didn't perform well as compared to other classifiers and it has the lowest accuracy i.e. 16%.

```
# KNN Classifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
knn_f1_score = f1_score(y_test, knn_predictions, average='weighted')
knn_confusion_matrix = confusion_matrix(y_test, knn_predictions)
print("KNN Accuracy:", knn_accuracy*100, "%")
print("KNN F1 Score:", knn_f1_score)
print("KNN Confusion Matrix:")
print(knn_confusion_matrix)
```

```
KNN Accuracy: 16.695804195804197 %
KNN F1 Score: 0.1522517275316911
KNN Confusion Matrix:
[[19  2  0 ...  0  0  0]
 [ 0 22  4 ...  0  0  0]
 [ 1  7  0 ...  1  0  0]
 ...
 [ 0  5  2 ...  4  0  0]
 [ 2  6  2 ...  2  0  0]
 [ 0  1  3 ...  1  0  0]]
```

The following are some of the reasons why which classifier performs good/bad.

RANDOM FOREST CLASSIFIER :

Robust to Noisy Data: Sensor data can be prone to noise and outliers. Random Forest is relatively robust to such noise because it averages the predictions of multiple decision trees, reducing the impact of individual outliers and noisy instances.

Captures Nonlinear Relationships: Random Forest can capture complex and nonlinear relationships between sensor inputs and outputs. Each decision tree in the forest can model a different aspect of the data, and their collective predictions provide a comprehensive representation of the underlying patterns.

Implicit Feature Selection: Random Forest automatically performs feature selection by evaluating the importance of each feature based on how much it contributes to the predictive performance of the ensemble. This information can help identify the most relevant sensor measurements for the task at hand.

Generalization Ability: Random Forest tends to have good generalization ability, meaning it can effectively generalize from the training data to unseen test data. This property is crucial when dealing with sensor data, where the model needs to perform well on new instances collected in real-world scenarios.

Overall, the combination of ensemble learning, decision trees, random feature selection, and bagging makes Random Forest a powerful algorithm for analyzing sensor data, enabling accurate predictions and robust performance even in the presence of noise and high-dimensional input spaces.

KNN :

Sensitive to Noisy Data and Outliers: KNN is sensitive to noisy data and outliers in the training set. Since it determines the class of a new instance based on the majority class among its nearest neighbors, the presence of noisy or mislabeled instances can significantly impact the classification results. Outliers can also distort the distances, leading to incorrect neighbor selection and erroneous classification.

Lack of Local Structure Learning: KNN does not explicitly learn the underlying structure or decision boundaries of the data. It relies solely on the local neighborhood information. In cases where the sensor data exhibits complex or non-linear decision boundaries, KNN may struggle to capture and generalize from such patterns, resulting in poor classification performance.

SVM :

Nonlinear Relationships: Random Forests are capable of capturing complex and nonlinear relationships between input features and target variables. They achieve this by constructing an ensemble of decision trees that can collectively model intricate patterns in the data. SVMs, on the other hand, primarily rely on kernel functions to transform the data into a higher-dimensional space where linear separation is possible. If the sensor data exhibits highly nonlinear relationships, Random Forests might be more adept at learning and capturing these patterns, potentially outperforming SVMs.

DNN :

Insufficient Data: DNNs typically require a large amount of labeled training data to generalize well. If the available sensor data is limited, training a DNN may lead to overfitting, where the model memorizes the training examples rather than learning meaningful patterns. Overfitting can result in poor generalization to new, unseen sensor data. In contrast, Random Forests and SVMs can be more effective with smaller datasets as they are less prone to overfitting.

CONCLUSION

In conclusion, based on the provided results, the Random Forest classifier emerged as the best performer among the evaluated classifiers. It achieved the highest overall accuracy, while also demonstrating a good F1-score. Therefore, the Random Forest classifier is recommended for the given dataset and task.