Appendix 1

Classification tasks

In [ ]:
```
""" List of libraries used in this project"""

import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
from sklearn.model_selection import cross_val_predict
from sklearn.pipeline import Pipeline
from sklearn.decomposition import IncrementalPCA
from sklearn.metrics import ConfusionMatrixDisplay
```

In [ ]:
```
""" Load dataset, loop through each folder,
and create features and labels datasets"""

folder_path = r"C:\Users\aleks\OneDrive - Coventry University\Desktop\Machine_Le

# Create empty list to store processed data
dataset = []
labels = []
 # List all 19 activities
activities = ['a01', 'a02', 'a03', 'a04', 'a05',
              'a06', 'a07', 'a08', 'a09', 'a10',
              'a11','a12','a13', 'a14', 'a15',
              'a16', 'a17', 'a18', 'a19']
 # List all 8 participants
participants = ['p1', 'p2', 'p3', 'p4',
                'p5', 'p6', 'p7', 'p8']

 # Loop through each activity
for activity_number, activity in enumerate(activities, start=1):
    activity_label = f'activity_{activity_number:02d}'
    activity_folder_path = os.path.join(folder_path,
                                        activity)
    # Loop through each participant
    for participant in participants:
        participant_folder = os.path.join(
```

```
                activity_folder_path,
                participant)
        for filename in os.listdir(participant_folder):
            file_path = os.path.join(
                participant_folder,
                filename)
            if os.path.isfile(file_path):
                # Single file shape is 125 (time-steps) x 45 (sensors features)
                segment = pd.read_csv(file_path,
                                      header=None)
                flattened_dataset = segment.values.flatten().tolist()
                # Append all segments into the dataste and create 2D array
                dataset.append(flattened_dataset)
                # Append labels for every segment
                labels.append(activity_label)
dataset_array = np.array(dataset)
labels_array = np.array(labels)
```

In [ ]:

```
""" Create dataframes"""
df = pd.DataFrame(dataset_array)
df['label'] = labels_array
```

In [ ]:

```
"""Confirm number of features"""
df.shape
```

In [ ]:

```
""" Check the first five rows"""
df.head(5)
```

In [ ]:

```
""" Check label's column"""
print(df['label'])
```

In [ ]:

```
"""Create variable X (features) and y (target labels)"""
X = df.drop(columns=['label'])
y = df['label']
```

In [ ]:

```
"""Logistic regression Pipeline"""

pipeline_logreg = Pipeline([
    ('scaler_lr', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('lr', LogisticRegression(
         max_iter=4000,
    random_state=42))
     ])
```

In [ ]:

```
"""Support Vector Machine Pipeline"""

pipeline_svm = Pipeline([
    ('scaler_svm', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('svm', SVC(random_state=42))
     ])
```

Gaussian Naive Bayes no random state - deterministic by default

In [ ]:
```
"""Gaussian NB Pipeline"""

pipeline_gnb = Pipeline([
    ('scaler_gnb', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('gnb', GaussianNB()),
  ])
```

In [ ]:
```
"""Decision Tree Pipeline"""

pipeline_decisiontree = Pipeline([
    ('scaler_dt', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('decision_tree', DecisionTreeClassifier(
         random_state=42))
     ])
```

In [ ]:
```
"""Random Forest Pipeline"""

pipeline_RandomForest = Pipeline([
    ('scaler_dt', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('random_forest', RandomForestClassifier(
         random_state=42))
     ])
```

In [ ]:
```
"""k-NN Pipeline"""

pipeline_knn = Pipeline([
    ('scaler_dt', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('knn', KNeighborsClassifier(n_neighbors=3))
     ])
```

In [ ]:
```
"""Scoring used for initial cross-validation"""

scoring = {
    'accuracy': 'accuracy',
    'f1_weighted': 'f1_weighted',
    'precision_macro': 'precision_macro',
    'recall_macro': 'recall_macro'
}
```

In [ ]:
```
"""3-cross fold validation on Pipeline using Logistic Regression"""

scores_logreg = cross_validate(
    pipeline_logreg,
    X, y,
    cv=3,
```

```python
    scoring=scoring,
    n_jobs=-1)
```

In [ ]:
```python
"""Calculate and print the mean of each evaluation metric"""

for key in sorted(scores_logreg.keys()):
    print(f"{key}: {scores_logreg[key]}")
    print(f"Mean {key}: {np.mean(scores_logreg[key]):.4f}\n")
```

In [ ]:
```python
"""3-cross validation on Pipeline using SVM"""

scores_svm = cross_validate(
    pipeline_svm,
    X, y,
    cv=3,
    scoring=scoring,
    n_jobs=-1)
```

In [ ]:
```python
"""Calculate and print the mean of each evaluation metric"""
for key in sorted(scores_svm.keys()):
    print(f"{key}: {scores_svm[key]}")
    print(f"Mean {key}: {np.mean(scores_svm[key]):.4f}\n")
```

In [ ]:
```python
"""3-cross validation on Pipeline using GaussianNavieBayes"""
scores_gnb = cross_validate(
    pipeline_gnb,
    X, y,
    cv=3,
    scoring=scoring,
    n_jobs=1)
```

In [ ]:
```python
"""Calculate and print the mean of each evaluation metric"""
for key in sorted(scores_gnb.keys()):
    print(f"{key}: {scores_gnb[key]}")
    print(f"Mean {key}: {np.mean(scores_gnb[key]):.4f}\n")
```

In [ ]:
```python
"""3-cross validation on Pipeline using Decision Tree"""
scores_decisiontree  = cross_validate(
    pipeline_decisiontree,
    X, y,
    cv=3,
    scoring=scoring,
    n_jobs=-1)
```

In [ ]:
```python
"""Calculate and print the mean of each evaluation metric"""
for key in sorted(scores_decisiontree.keys()):
    print(f"{key}: {scores_decisiontree[key]}")
    print(f"Mean {key}: {np.mean(scores_decisiontree[key]):.4f}\n")
```

In [ ]:
```python
"""3-cross validation on Pipeline using Random Forest"""
scores_random_forest = cross_validate(
    pipeline_RandomForest,
    X, y,
    cv=3,
    scoring=scoring,
    n_jobs=-1)
```

```
In [ ]: """Calculate and print the mean of each evaluation metric"""
for key in sorted(scores_random_forest.keys()):
    print(f"{key}: {scores_random_forest[key]}")
    print(f"Mean {key}: {np.mean(scores_random_forest[key]):.4f}\n")
```

```
In [ ]: """3-cross validation on Pipeline using k-NN"""
scores_knn = cross_validate(
    pipeline_knn,
    X, y,
    cv=3,
    scoring=scoring,
    n_jobs=-1)
```

```
In [ ]: """Calculate and print the mean of each evaluation metric"""
for key in sorted(scores_knn.keys()):
    print(f"{key}: {scores_knn[key]}")
    print(f"Mean {key}: {np.mean(scores_knn[key]):.4f}\n")
```

Split the data into 80:20 ratio

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Naive Bayes - search for best parameters using GridSearch

```
In [ ]: param_grid_GNB={
'gnb__var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]
}
```

```
In [ ]: estimator_GNB = GridSearchCV(
    pipeline_gnb,
    param_grid_GNB,
    scoring='accuracy',
    cv=3, n_jobs=-1)
```

```
In [ ]: estimator_GNB.fit(X_train, y_train)
```

Naive Bayes best parameters (default)

```
In [ ]: print("Best params:",
      estimator_GNB.best_params_)
```

SVM - search for best parameters using GridSearch

```
In [ ]: param_grid_svm={
    'svm__C':
    [0.01,0.1,1],
    'svm__kernel':
    ['linear', 'rbf', 'poly', 'sigmoid'],
}
```

```
In [ ]: estimator_svm = GridSearchCV(
    pipeline_svm,
    param_grid_svm,
```

```
    scoring='accuracy',
    cv=3, n_jobs=1)
```

```
estimator_svm.fit(X_train, y_train)
```

```
print("Best params:",
      estimator_svm.best_params_)
```

k-NN - search for best parameters using GridSearch

```
param_grid_knn={
    'knn__n_neighbors':
    [3, 5, 7],
    'knn__weights':
    ['uniform', 'distance'],
}
```

```
estimator_knn = GridSearchCV(
    pipeline_knn,
    param_grid_knn,
    scoring='accuracy',
    cv=3, n_jobs=1)
```

```
estimator_knn.fit(X_train, y_train)
```

```
print("Best params:", estimator_knn.best_params_)
```

Fit tuned models on training data and evaluate on test data.

SVM with the best parameters (C=1, kerner='rbf')

```
"""Support Vector Machine Pipeline"""

pipeline_svm_best = Pipeline([
    ('scaler_svm', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('svm', SVC(C=1, kernel='rbf',
         random_state=42))
     ])
```

Gaussian NB with the parameters (Default)

```
"""Gaussian NB Pipeline"""

pipeline_gnb_best = Pipeline([
    ('scaler_gnb', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('gnb', GaussianNB()),
  ])
```

k-NN with the best parameters (k=3, weights=distance)

```python
"""k-NN Pipeline"""

pipeline_knn_best = Pipeline([
    ('scaler_dt', StandardScaler()),
     ('pca', IncrementalPCA(
         n_components=60,
         batch_size=200)),
     ('knn', KNeighborsClassifier(n_neighbors=3,
                                  weights='distance'))
     ])
```

Train Naive Bayes

```python
pipeline_gnb_best.fit(X_train, y_train)
```

```python
Make prediction on unseen data
```

```python
y_pred_gnb = pipeline_gnb_best.predict(X_test)
```

Train SVMs

```python
pipeline_svm_best.fit(X_train, y_train)
```

Make prediction on unseen data

```python
y_pred_svm = pipeline_svm_best.predict(X_test)
```

```python
Train k-NN
```

```python
pipeline_knn_best.fit(X_train, y_train)
```

Make prediction on unseen data

```python
y_pred_knn = pipeline_knn_best.predict(X_test)
```

Evaluation of Gaussian Naive Bayes

```python
accuracy_gnb = accuracy_score(
    y_test, y_pred_gnb)
precision_gnb = precision_score(
    y_test, y_pred_gnb, average='macro')
recall_gnb = recall_score(
    y_test, y_pred_gnb, average='macro')
f1_gnb = f1_score(
    y_test, y_pred_gnb, average='macro')
```

```python
print(f"Accuracy: {accuracy_gnb:.2f}")
print(f"Precision: {precision_gnb:.2f}")
print(f"Recall: {recall_gnb:.2f}")
print(f"F1: {f1_gnb:.2f}")
```

Evaluation of Support Vector Machine

In [ ]:
```
accuracy_svm = accuracy_score(
    y_test, y_pred_svm)
precision_svm = precision_score(
    y_test, y_pred_svm, average='macro')
recall_svm = recall_score(
    y_test, y_pred_svm, average='macro')
f1_svm = f1_score(
    y_test, y_pred_svm, average='macro')
```

In [ ]:
```
print(f"Accuracy: {accuracy_svm:.2f}")
print(f"Precision: {precision_svm:.2f}")
print(f"Recall: {recall_svm:.2f}")
print(f"F1: {f1_svm:.2f}")
```

Evaluation of k-NN

In [ ]:
```
accuracy_knn = accuracy_score(
    y_test, y_pred_knn)
precision_knn = precision_score(
    y_test, y_pred_knn, average='macro')
recall_knn = recall_score(
    y_test, y_pred_knn, average='macro')
f1_knn = f1_score(
    y_test, y_pred_knn, average='macro')
```

In [ ]:
```
print(f"Accuracy: {accuracy_knn:.2f}")
print(f"Precision: {precision_knn:.2f}")
print(f"Recall: {recall_knn:.2f}")
print(f"F1: {f1_knn:.2f}")
```

Normalise each value so you can interpret each value as percentage. From Hands on Machine Learning.

In [ ]:
```
class_names = [
    'Activity 1', 'Activity 2', 'Activity 3', 'Activity 4',
    'Activity 5', 'Activity 6', 'Activity 7', 'Activity 8',
    'Activity 9', 'Activity 10', 'Activity 11', 'Activity 12',
    'Activity 13', 'Activity 14', 'Activity 15', 'Activity 16',
    'Activity 17', 'Activity 18', 'Activity 19'
]
```

In [ ]:
```
""" Plot Gaussain Naive Bayes Confusion Matrix"""

cm_gnb = confusion_matrix(
    y_test,
    y_pred_gnb,
    normalize='true')
plt.figure(figsize=(12, 10))
sns.heatmap(cm_gnb,
            annot=True,
            cmap="Blues",
            xticklabels=class_names,
            yticklabels=class_names)

plt.title("Confusion Matrix (Normalised) - GaussianNB",
          fontsize=14)
plt.xlabel("Predicted Label")
```

```
plt.ylabel("True Label")
plt.xticks(rotation=45,
           ha='right')
plt.tight_layout()
plt.show()
```

In [ ]:

```
"""Plot SVM Confusion Matrix"""
cm_svm = confusion_matrix(
    y_test,
    y_pred_svm,
    normalize='true')
plt.figure(figsize=(12, 10))
sns.heatmap(cm_svm,
            annot=True,
            cmap="Blues",
            xticklabels=class_names,
            yticklabels=class_names)

plt.title("Final Confusion Matrix (Normalised) – SVM",
          fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45,
           ha='right')
plt.tight_layout()
plt.show()
```

In [ ]:

```
""" Plot k-NN Confusion Matrix """

cm_knn = confusion_matrix(
    y_test,
    y_pred_knn,
    normalize='true')
plt.figure(figsize=(12, 10))
sns.heatmap(cm_knn,
            annot=True,
            cmap="Blues",
            xticklabels=class_names,
            yticklabels=class_names)

plt.title("Final Confusion Matrix (Normalised) – K-NN",
          fontsize=14)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45,
           ha='right')
plt.tight_layout()
plt.show()
```