Univerzitet u Sarajevu

Elektrotehnički fakultet u Sarajevu

Odsjek za računarstvo i informatiku

Klasifikacija ljudskih aktivnosti u pametnim satovima

Kod korišten u projektu iz predmeta Digitalno procesiranje signala

Student/ica:

Amina Hromić

Profesor:

V. prof. dr Amila Akagić

Sarajevo, juni 2023.

```
################################################################################
                                #Preprocesiranje podataka

################################## Load and check #####################################
# load dataset
from pandas import read_csv

# load a single file as a numpy array
def load_file(filepath):
        dataframe = read_csv(filepath, header=None, delim_whitespace=True)
        return dataframe.values

data = load_file('train/Inertial Signals/total_acc_y_train.txt')
print(data.shape)

################################## Add a third feature #################################
# load dataset
from numpy import dstack
from pandas import read_csv

# load a single file as a numpy array
def load_file(filepath):
        dataframe = read_csv(filepath, header=None, delim_whitespace=True)
        return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
        loaded = list()
        for name in filenames:
                data = load_file(prefix + name)
                loaded.append(data)
        # stack group so that features are the 3rd dimension
```

```
        loaded = dstack(loaded)
        return loaded

# load the total acc data
filenames = ['total_acc_x_train.txt', 'total_acc_y_train.txt', 'total_acc_z_train.txt']
total_acc = load_group(filenames, prefix='train/Inertial Signals/')
print(total_acc.shape)

############################## Load everything and check ##############################
# load a dataset group, such as train or test
def load_dataset(group):
 filepath = group + '/Inertial Signals/'
 # load all 9 files as a single array
 filenames = list()
 # total acceleration
 filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
 # body acceleration
 filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
 # body gyroscope
 filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
 # load input data
 X = load_group(filenames, filepath)
 # load class output
 y = load_file(group + '/y_'+group+'.txt')
 return X, y

# load all train
trainX, trainy = load_dataset('train')
print(trainX.shape, trainy.shape)
# load all test
testX, testy = load_dataset('test')
print(testX.shape, testy.shape)
```

```
############################## Check for null and duplicate values ########################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
# Load the data into a NumPy array
data = np.loadtxt('train/Inertial Signals/total_acc_y_train.txt')

# Convert the NumPy array to a DataFrame
train_df = pd.DataFrame(data)

# Check for null values
print('Null values:', train_df.isnull().values.sum())

# Print the data types of the DataFrame columns
print(train_df.dtypes)

print('Number of duplicates in train set:{}'.format(sum(train_df.duplicated())))
print(train_df.columns)

#print(train_df.columns)
#last_column = train_df.iloc[:, -1]
#px.pie(train_df, names=last_column, title='Activity in database')

##############################Calculate and visualise variance ############################
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
pca=PCA(n_components=127).fit(X)   # number of components
principal_component=pca.transform(X)
```

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('principal component')
plt.ylabel('explained variance')

###################### Example of reducing dimensionality using PCA ####################
from sklearn import preprocessing
X=train_df.drop(train_df.columns[20],axis=1)
Y=train_df[train_df.columns[20]]
print('X matrix size:',X.shape)
X=preprocessing.StandardScaler().fit(X).transform(X)
#test
test_df=pd.read_csv('test/Inertial Signals/total_acc_y_test.txt')
# Convert the NumPy array to a DataFrame
test_df = pd.DataFrame(data)
X_test=test_df.drop(test_df.columns[0],axis=1)
X_t=preprocessing.StandardScaler().fit(X_test).transform(X_test)

principal_component=PCA(n_components=127).fit_transform(X)
X_test_pca=pca.transform(X_t)
pca_df=pd.DataFrame(data=principal_component)
pca_test_d=pd.DataFrame(data=X_test_pca)
pca_df.head()
pca_df.shape
################################### Check the data ####################################
# summarize class balance
from numpy import array
from numpy import vstack
from pandas import read_csv
from pandas import DataFrame

# load a single file as a numpy array
def load_file(filepath):
```

```
        dataframe = read_csv(filepath, header=None, delim_whitespace=True)
        return dataframe.values
# summarize the balance of classes in an output variable column
def class_breakdown(data):
        # convert the numpy array into a dataframe
        df = DataFrame(data)
        # group data by the class value and calculate the number of rows
        counts = df.groupby(0).size()
        # retrieve raw rows
        counts = counts.values
        # summarize
        for i in range(len(counts)):
                percent = counts[i] / len(df) * 100
                print('Class=%d, total=%d, percentage=%.3f' % (i+1, counts[i], percent))

# load train file
trainy = load_file('train/y_train.txt')
# summarize class breakdown
print('Train Dataset')
class_breakdown(trainy)

# load test file
testy = load_file('test/y_test.txt')
# summarize class breakdown
print('Test Dataset')
class_breakdown(testy)

# summarize combined class breakdown
print('Both')
combined = vstack((trainy, testy))
class_breakdown(combined)
```

```
# Load the necessary libraries
from numpy import unique

# Load data
sub_map = load_file('train/subject_train.txt')
train_subjects = unique(sub_map)
print(train_subjects)

################################################################################################
                                                #Data analysis

################################### Total acceleration ###################################
# plot all vars for one subject
from numpy import array
from numpy import dstack
from numpy import unique
from pandas import read_csv
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
        dataframe = read_csv(filepath, header=None, delim_whitespace=True)
        return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
        loaded = list()
        for name in filenames:
                data = load_file(prefix + name)
                loaded.append(data)
        # stack group so that features are the 3rd dimension
        loaded = dstack(loaded)
```

```
        return loaded


# load a dataset group, such as train or test
def load_dataset(group):
        filepath = group + '/Inertial Signals/'
        # load all 9 files as a single array
        filenames = list()
        # total acceleration
        filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
        # body acceleration
        filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
        # body gyroscope
        filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
        # load input data
        X = load_group(filenames, filepath)
        # load class output
        y = load_file(group + '/y_'+group+'.txt')
        return X, y


# get all data for one subject
def data_for_subject(X, y, sub_map, sub_id):
        # get row indexes for the subject id
        ix = [i for i in range(len(sub_map)) if sub_map[i]==sub_id]
        # return the selected samples
        return X[ix, :, :], y[ix]


# convert a series of windows to a 1D list
def to_series(windows):
        series = list()
        for window in windows:
                # remove the overlap from the window
```

```
                half = int(len(window) / 2) - 1
                for value in window[-half:]:
                        series.append(value)
        return series


# plot the data for one subject
def plot_subject(X, y):
        pyplot.figure()
        # determine the total number of plots
        n, off = X.shape[2] + 1, 0
        # plot total acc
        for i in range(3):
                pyplot.subplot(n, 1, off+1)
                pyplot.plot(to_series(X[:, :, off]))
                pyplot.title('total acc '+str(i), y=0, loc='left')
                off += 1
        # plot body acc
        for i in range(3):
                pyplot.subplot(n, 1, off+1)
                pyplot.plot(to_series(X[:, :, off]))
                pyplot.title('body acc '+str(i), y=0, loc='left')
                off += 1
        # plot body gyro
        for i in range(3):
                pyplot.subplot(n, 1, off+1)
                pyplot.plot(to_series(X[:, :, off]))
                pyplot.title('body gyro '+str(i), y=0, loc='left')
                off += 1
        # plot activities
        pyplot.subplot(n, 1, n)
        pyplot.plot(y)
        pyplot.title('activity', y=0, loc='left')
```

```
        pyplot.show()

# load data
trainX, trainy = load_dataset('train')
# load mapping of rows to subjects
sub_map = load_file('train/subject_train.txt')
train_subjects = unique(sub_map)
print(train_subjects)
# get the data for one subject
sub_id = train_subjects[15]
subX, suby = data_for_subject(trainX, trainy, sub_map, sub_id)
print(subX.shape, suby.shape)
# plot data for subject
plot_subject(subX, suby)

################################ Total acceleration histogram ############################
# plot histograms for multiple subjects
from numpy import array
from numpy import unique
from numpy import dstack
from pandas import read_csv
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
        dataframe = read_csv(filepath, header=None, delim_whitespace=True)
        return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
        loaded = list()
        for name in filenames:
```

```
            data = load_file(prefix + name)
            loaded.append(data)
    # stack group so that features are the 3rd dimension
    loaded = dstack(loaded)
    return loaded

# load a dataset group, such as train or test
def load_dataset(group):
    filepath = group + '/Inertial Signals/'
    # load all 9 files as a single array
    filenames = list()
    # total acceleration
    filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
    # body acceleration
    filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
    # body gyroscope
    filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
    # load input data
    X = load_group(filenames, filepath)
    # load class output
    y = load_file(group + '/y_'+group+'.txt')
    return X, y

# get all data for one subject
def data_for_subject(X, y, sub_map, sub_id):
    # get row indexes for the subject id
    ix = [i for i in range(len(sub_map)) if sub_map[i]==sub_id]
    # return the selected samples
    return X[ix, :, :], y[ix]

# convert a series of windows to a 1D list
```

```
def to_series(windows):
	series = list()
	for window in windows:
		# remove the overlap from the window
		half = int(len(window) / 2) - 1
		for value in window[-half:]:
			series.append(value)
	return series

# plot histograms for multiple subjects
def plot_subject_histograms(X, y, sub_map, n=10):
	pyplot.figure()
	# get unique subjects
	subject_ids = unique(sub_map[:,0])
	# enumerate subjects
	xaxis = None
	for k in range(n):
		sub_id = subject_ids[k]
		# get data for one subject
		subX, _ = data_for_subject(X, y, sub_map, sub_id)
		# body acc
		for i in range(3):
			ax = pyplot.subplot(n, 1, k+1, sharex=xaxis)
			ax.set_xlim(-1,1)
			if k == 0:
				xaxis = ax
			pyplot.hist(to_series(subX[:,:,6+i]), bins=100)
	pyplot.show()

# load training dataset
X, y = load_dataset('train')
# load mapping of rows to subjects
```

```
sub_map = load_file('train/subject_train.txt')
# plot histograms for subjects
plot_subject_histograms(X, y, sub_map)

################################ Histogram per activity ###################################
# plot histograms per activity for a subject
from numpy import array
from numpy import dstack
from numpy import unique
from pandas import read_csv
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
	dataframe = read_csv(filepath, header=None, delim_whitespace=True)
	return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
	loaded = list()
	for name in filenames:
		data = load_file(prefix + name)
		loaded.append(data)
	# stack group so that features are the 3rd dimension
	loaded = dstack(loaded)
	return loaded

# load a dataset group, such as train or test
def load_dataset(group):
	filepath = group + '/Inertial Signals/'
	# load all 9 files as a single array
	filenames = list()
```

```
	# total acceleration
	filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
	# body acceleration
	filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
	# body gyroscope
	filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt', 'body_gyro_z_'+group+'.txt']
	# load input data
	X = load_group(filenames, filepath)
	# load class output
	y = load_file(group + '/y_'+group+'.txt')
	return X, y

# get all data for one subject
def data_for_subject(X, y, sub_map, sub_id):
	# get row indexes for the subject id
	ix = [i for i in range(len(sub_map)) if sub_map[i]==sub_id]
	# return the selected samples
	return X[ix, :, :], y[ix]

# convert a series of windows to a 1D list
def to_series(windows):
	series = list()
	for window in windows:
		# remove the overlap from the window
		half = int(len(window) / 2) - 1
		for value in window[-half:]:
			series.append(value)
	return series

# group data by activity
def data_by_activity(X, y, activities):
```

```
	# group windows by activity
	return {a:X[y[:,0]==a, :, :] for a in activities}

# plot histograms for each activity for a subject
def plot_activity_histograms(X, y):
	# get a list of unique activities for the subject
	activity_ids = unique(y[:,0])
	# group windows by activity
	grouped = data_by_activity(X, y, activity_ids)
	# plot per activity, histograms for each axis
	pyplot.figure()
	xaxis = None
	for k in range(len(activity_ids)):
		act_id = activity_ids[k]
		# total acceleration
		for i in range(3):
			ax = pyplot.subplot(len(activity_ids), 1, k+1, sharex=xaxis)
			ax.set_xlim(-1,1)
			if k == 0:
				xaxis = ax
			pyplot.hist(to_series(grouped[act_id][:,:,6+i]), bins=100)
			pyplot.title('activity '+str(act_id), y=0, loc='left')
	pyplot.show()

# load data
trainX, trainy = load_dataset('train')
# load mapping of rows to subjects
sub_map = load_file('train/subject_train.txt')
train_subjects = unique(sub_map)
# get the data for one subject
sub_id = train_subjects[0]
subX, suby = data_for_subject(trainX, trainy, sub_map, sub_id)
```

```
# plot data for subject
plot_activity_histograms(subX, suby)

################################### Plot durations ######################################
# plot durations of each activity by subject
from numpy import array
from numpy import dstack
from numpy import unique
from pandas import read_csv
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
	dataframe = read_csv(filepath, header=None, delim_whitespace=True)
	return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
	loaded = list()
	for name in filenames:
		data = load_file(prefix + name)
		loaded.append(data)
	# stack group so that features are the 3rd dimension
	loaded = dstack(loaded)
	return loaded

# load a dataset group, such as train or test
def load_dataset(group):
	filepath = group + '/Inertial Signals/'
	# load all 9 files as a single array
	filenames = list()
	# total acceleration
```

```
        filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
        # body acceleration
        filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
        # body gyroscope
        filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
        # load input data
        X = load_group(filenames, filepath)
        # load class output
        y = load_file(group + '/y_'+group+'.txt')
        return X, y

# get all data for one subject
def data_for_subject(X, y, sub_map, sub_id):
        # get row indexes for the subject id
        ix = [i for i in range(len(sub_map)) if sub_map[i]==sub_id]
        # return the selected samples
        return X[ix, :, :], y[ix]

# convert a series of windows to a 1D list
def to_series(windows):
        series = list()
        for window in windows:
                # remove the overlap from the window
                half = int(len(window) / 2) - 1
                for value in window[-half:]:
                        series.append(value)
        return series

# group data by activity
def data_by_activity(X, y, activities):
        # group windows by activity
```

```
        return {a:X[y[:,0]==a, :, :] for a in activities}

# plot activity durations by subject
def plot_activity_durations_by_subject(X, y, sub_map):
        # get unique subjects and activities
        subject_ids = unique(sub_map[:,0])
        activity_ids = unique(y[:,0])
        # enumerate subjects
        activity_windows = {a:list() for a in activity_ids}
        for sub_id in subject_ids:
                # get data for one subject
                _, subj_y = data_for_subject(X, y, sub_map, sub_id)
                # count windows by activity
                for a in activity_ids:
                        activity_windows[a].append(len(subj_y[subj_y[:,0]==a]))
        # organize durations into a list of lists
        durations = [activity_windows[a] for a in activity_ids]
        pyplot.boxplot(durations, labels=activity_ids)
        pyplot.show()

# load test dataset
X, y = load_dataset('test')
# load mapping of rows to subjects
sub_map = load_file('test/subject_test.txt')
# plot durations
plot_activity_durations_by_subject(X, y, sub_map)

################################################################################
                        #Classifiers and model testing

############################## Random Forest Algorithm ##############################
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score

# load a dataset group, such as train or test
def load_dataset(group):
 filepath = group + '/Inertial Signals/'
 # load all 9 files as a single array
 filenames = list()
 # total acceleration
 filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
 # body acceleration
 filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
 # body gyroscope
 filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
 # load input data
 X = load_group(filenames, filepath)
 # load class output
 y = load_file(group + '/y_'+group+'.txt')
 return X, y

# load all train
trainX, trainy = load_dataset('train')

# load all test
testX, testy = load_dataset('test')



# Reshape trainX to 2-dimensional array
num_samples, num_timesteps, num_features = trainX.shape
trainX_reshaped = trainX.reshape((num_samples, num_timesteps * num_features))

# Reshape trainy to 1-dimensional array
```

```
trainy_reshaped = trainy.ravel()

# Create an instance of the Random Forest Classifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model to the reshaped training data
rf_clf.fit(trainX_reshaped, trainy_reshaped)



# Reshape testX to 2-dimensional array
num_samples_test, num_timesteps_test, num_features_test = testX.shape
testX_reshaped = testX.reshape((num_samples_test, num_timesteps_test * num_features_test))

# Reshape testy to 1-dimensional array
testy_reshaped = testy.ravel()

# Make predictions on the reshaped test data
predictions = rf_clf.predict(testX_reshaped)

accuracy = accuracy_score(testy, predictions)
print("Accuracy:", accuracy)

#Confusion matrix plain
from sklearn.metrics import confusion_matrix

# Make predictions on the reshaped test data
predictions = rf_clf.predict(testX_reshaped)

# Create a confusion matrix
cm = confusion_matrix(testy_reshaped, predictions)
print(cm)
```

```
#Confusion matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Create the confusion matrix

cm = confusion_matrix(testy_reshaped, predictions)
# Define the class labels
class_labels = ['WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS', 'SITTING', 'STANDING',
'LAYING']

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, square=True,
        xticklabels=class_labels, yticklabels=class_labels)

plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')

plt.show()

##################################### Decision Tree ###################################
import numpy as np

# Reshape trainX to 2-dimensional array
num_samples, num_timesteps, num_features = trainX.shape
trainX_reshaped = trainX.reshape((num_samples, num_timesteps * num_features))

# Create an instance of the DecisionTreeClassifier
dt_clf = DecisionTreeClassifier()
```

```
# Fit the classifier to the reshaped training data
dt_clf.fit(trainX_reshaped, trainy)

# Make predictions on the test data
testX_reshaped = testX.reshape((testX.shape[0], num_timesteps * num_features))
y_pred = dt_clf.predict(testX_reshaped)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(testy, y_pred)
print("Accuracy:", accuracy)

#Confusion matrix plain
from sklearn.metrics import confusion_matrix

# Reshape trainX and testX to 2-dimensional arrays
num_samples, num_timesteps, num_features = trainX.shape
trainX_reshaped = trainX.reshape((num_samples, num_timesteps * num_features))
testX_reshaped = testX.reshape((testX.shape[0], num_timesteps * num_features))

# Fit the Decision Tree Classifier to the reshaped training data
dt_clf = DecisionTreeClassifier()
dt_clf.fit(trainX_reshaped, trainy)

# Make predictions on the test data
y_pred = dt_clf.predict(testX_reshaped)

# Create the confusion matrix
cm = confusion_matrix(testy, y_pred)

print("Confusion Matrix:")
print(cm)
```

```
#Confusion matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Create the confusion matrix
cm = confusion_matrix(testy, y_pred)

# Define the class labels
class_labels = ['WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS', 'SITTING', 'STANDING',
'LAYING']

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, square=True,
        xticklabels=class_labels, yticklabels=class_labels)

plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')

plt.show()
```