Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Klasifikacija ljudskih aktivnosti u pametnim satovima

Projekat iz predmeta Digitalno procesiranje signala

Student/ica:
Amina Hromić

Profesor/ica:
V. prof. dr Amila Akagić

Sarajevo, juni 2023.

Sadržaj

Uvod

Među najpopularnijim funkcionalnostima, kako pametnih satova, tako i drugih pametnih uređaja, je prikupljanje podataka o aktivnosti korisnika uređaja te analiza i klasifikacija istih. Prikupljajući i obrađujući signale sa mnoštva senzora kojima pametni sat raspolaže, poput žiroskopa, akcelerometra, mjerenja pulsa, itd., ovi uređaji su u stanju da prepoznaju šta njihov korisnik trenutno radi, donesu zaključke o korisničkim navikama i uzorcima ponašanja, koji se zatim mogu koristiti za različite svrhe.

Nebrojeno je mnogo istraživanja provedeno na ovu temu, predstavljajući različite algoritme, od kojih su neki obrađeni u seminarskom radu „Digitalno procesiranje signala u pametnim satovima“.

Neki od najpopularnijih i najefikasnijih algoritama za klasifikaciju ljudske aktivnosti spadaju u machine learning algoritme te će isti biti predstavljeni u nastavku (s izuzetkom neuralnih mreža).

Ovi algoritmi se zasnivaju na sličnom principu: izgradnji modela.

Najprije je potrebno učitati podatke i pripremiti iste za dalju upotrebu. Odnosno, potrebno je provjeriti da li su neke vrijednosti nulte vrijednosti, da li skup podataka sadrži duplikate, da li su iste mjerne jedinice primijenjene na sve podatke, i sl.

Zatim se model „trenira“, odnosno dio učitanih podataka se izdvoji za treniranje, a ostatak se iskoristi za testiranje, tj. za provjeru rezultata. Tipično, omjer je oko 70% - 80% podataka za treniranje, a ostatak predstavlja podatke za testiranje.

Naravno, pri treniranju i testiranju neophodno je prepoznati relevantne podatke, koji odgovaraju cilju testiranja, npr. odrediti koji senzori su relevantni te izdvojiti samo njihove podatke.

Nakon što je model kreiran i istreniran, vrijeme je za analizu dobijenih podataka i daljnje testiranje te bi u ovoj fazi bilo moguće iskoristiti isti model nad različitim skupovima podataka.

Ukoliko je korišteno više različitih algoritama ili formirano više raličitih modela, moguće je uporediti rezultate istih te izvršiti statističku analizu nad njima, kako bi se identificirao najefikasniji model.

Prikupljanje podataka

Korišten je skup podataka prikupljen na osnovu aktivnosti 30 ljudi, starosne dobi između 19 i 48 godina. Prikupljeni podaci su klasificirani u 6 skupina, tako da identificiraju 6 različitih aktivnosti: hodanje, hodanje uz stepenice, hodanje niz stepenice, sjedenje, stajanje i ležanje.

Podaci su prikupljeni sa žiroskopa i akcelerometra *Samsung Galaxy S II* uređaja, po *x, y* i *z* osi, frekvencijom uzorkovanja od 50 Hz.

Nad prikupljenim podacima su zatim primijenjene različite metode digitalnog procesiranja, kako bi se dobio skup podataka koji je javno dostupan (i iskorišten u ovoj implementaciji).

Konkretno, korišten je low-pass Butterworth filter sa cutoff frekvencijom od 20Hz kako bi se eliminisala buka, šumovi i bilo kakve smetnje koje bi uticale na vjerodostojnost podataka. Podaci s akcelerometra su podijeljeni u dvije skupine: gravitacione komponente i komponente koje oslikavaju kretanje tijela, a svi podaci su analizirani u prozorima od 2.56 sekundi.

Od prikupljenih podataka, 70% je iskorišteno za treniranje (*train*), a 30% za testiranje, odnosno, podaci prikupljeni od 21 osobe su se koristili za treniranje modela, a podaci od 9 nasumično odabranih osoba za testiranje.

Podaci su javno dostupni upravo u ovom formatu i zaista, nakon skidanja odgovarajućeg foldera, vidljivo je da je tačno 70% podataka iskorišteno za treniranje, tj. nalazi se u „*train*“ folderu, a ostatak u „*test*“ folderu.

Pretprocesiranje podataka

Za implementaciju sam odlučila iskoristiti *Python* programski jezik, kao jedan od najčešće korištenih jezika za ovakve analize, i sa dosta bibliotečnih funkcija koje podržavaju istu. Za okruženje sam koristila *Jupyter Notebook*.

Učitavanje podataka

Sljedeći isječak predstavlja učitavanje skinutog foldera sa podacima, te se vidi da se skup podataka sastoji od ukupno 7352 reda sa 128 uzoraka u svakom redu (prozoru):

In [1]:
```
# 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)
```

```
(7352, 128)
```

In [ ]:

Provjera *null* vrijednosti

Neophodno je provjeriti da li svi podaci postoje u skupu podataka, odnosno, da li su neki podaci null vrijednosti. Ovo se može postići primjenom *isNull* funkcije (bilo je potrebno prethodno pretvoriti *NumPy array* u *DataFrame*).

In [39]:
```
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)
```

```
Null values: 0
0      float64
1      float64
2      float64
3      float64
4      float64
        ...
123    float64
124    float64
125    float64
126    float64
127    float64
Length: 128, dtype: object
```

Zaključak je da nema null vrijednosti te da su podaci tipa *float*.
Isto je potrebno uraditi i za podatke iz *test* file-a te se dobija isti rezultat, 0 *null* vrijednosti.

Provjera duplih vrijednosti

Iako će se u nastavku koristiti funkcija *unique* kako bi eliminisali eventualne duplikate, radi potpunosti postupka, bit će izvršena i provjera nad podacima za duple podatke.

In [40]:
```
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())))
```

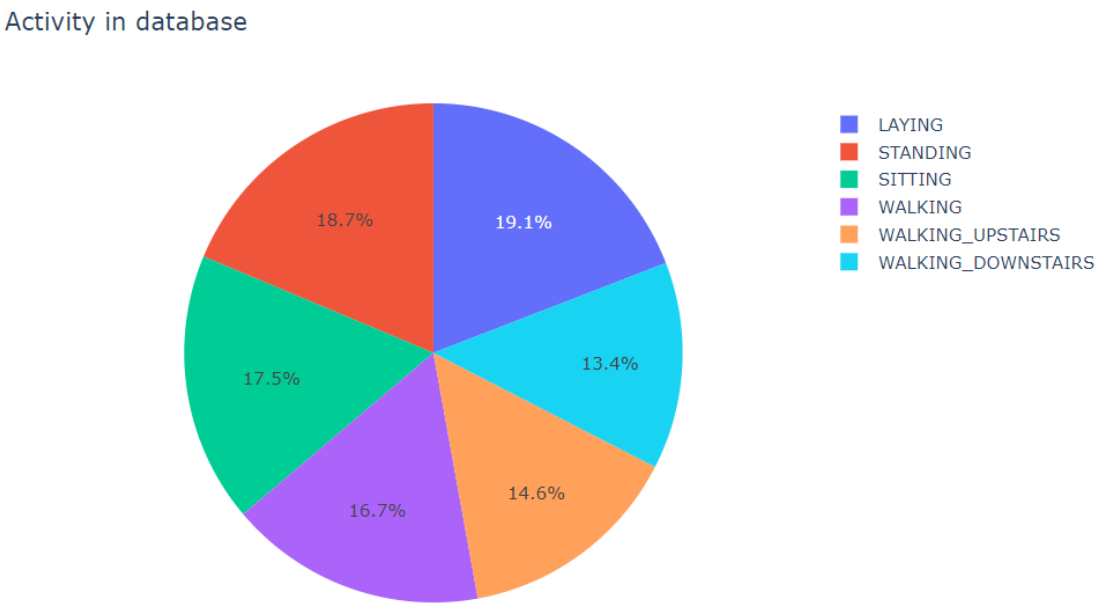
```
Null values: 0
0      float64
1      float64
2      float64
3      float64
4      float64
        ...
123    float64
124    float64
125    float64
126    float64
127    float64
Length: 128, dtype: object
Number of duplicates in train set:0
```

Iz ažuriranog prikaza je očigledno da nema duplikata u skupu podataka. Ekvivalentan rezultat se dobije i za test podatke.

U svrhu što boljeg upoznavanja sa skupom podataka, isti je korisno sagledati grafički, npr. grupisanog po aktivnostima. Ovo se može postići na sljedeći način:

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

S obzirom na to da se imena aktivnosti nalaze u zadnjoj koloni baze, upravo je ta kolona iskorištena za labele u pie chart grafu.



Može se zaključiti da su sve mjerene aktivnosti prilično ravnomjerno zastupljene, što je odlika dobrog skupa podataka.

Grupisanje podataka i redukcija dimenzionalnosti

Idući korak pri učitavanju podataka jeste da podatke grupišemo na način pogodniji za obradu istih. Prilikom primjene machine learning algoritama, podaci se nerijetko grupišu u trojke (posebno ukoliko bi na ovaj primjer htjeli primijeniti neku neuralnu mrežu), pri čemu bi u ovom slučaju prva vrijednost prikazivala prozore, druga broj uzoraka u svakom prozoru i treća broj osa (kako su za signale pri mjerenju uzete vrijednosti sa *x*, *y* i *z* ose, to će biti predstavljeno brojem 3).

In [5]:
```
# 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)
```

```
(7352, 128, 3)
```

Isti postupak je sada potrebno ponoviti za svih 9 fajlova sa podacima, preuzetih sa interneta. Radi uštede vremena, kao i smanjenja vjerovatnoće za grešku, u nastavku se nalazi funkcija napisana za istovremeno učitavanje svih fajlova (kao i njen poziv), a zasniva se na jednostavnoj upotrebi *stringova*.

In [7]:
```
# 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)
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
```

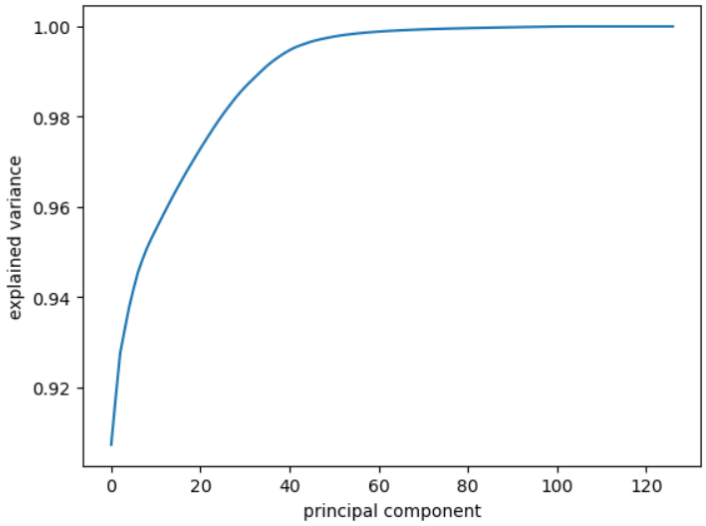
Moguće je primijetiti da su sada učitani svi podaci, te razdvojeni u dvije grupe: „*train*“ i „*test*“.

Korisno je napomenuti da se dimenzionalnost mogla smanjiti koristeći neke od provjerenih metoda za redukciju dimenzionalnosti, poput PCA.

PCA (*Principal Component Analysis*) pronalazi smjerove na kojima se podaci najviše mijenjaju te je čest algoritam koji se primjenjuje u fazi pretprocesiranja podataka.
Primjer korištenja PCA je dat u nastavku.

```
In [58]: 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)

X matrix size: (7352, 127)
```

```
In [60]: 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')

Out[60]: Text(0, 0.5, 'explained variance')
```



Posmatrajući varijansu, može se zamijetiti da više od 90% varijanse otpada na prvih 50 komponenti te bi se ove komponente trebale koristiti za dalju klasifikaciju podataka.

```
In [61]: principal_component=PCA(n_components=127).fit_transform(X)
X_test_pca=pca.transform(X_t)
#,columns=['Comp1','Comp2','Comp3']
pca_df=pd.DataFrame(data=principal_component)
pca_test_d=pd.DataFrame(data=X_test_pca)
pca_df.head()
pca_df.shape

Out[61]: (7352, 127)
```

Ukoliko bi postojala potrebe za vizualizacijom ovako obrađenih podataka, mogao bi se koristiti t-SNE algoritam koji predstavlja nelinearnu tehniku mapiranja strukture podataka u prostor niže dimenzionalnosti, bez da mijenja originalnu strukturu.
U implementaciji, koristila bi se TSNE biblioteka.

Provjera ispravnosti podataka

U skladu sa koracima predstavljenim u uvodu, sada je potrebno da se uvjerimo u ispravnost podataka. U ovom slučaju, s obzirom na to da se radi o kreiranju jednog modela, potrebno se uvjeriti da podaci za treniranje i testove podjednako dobro razvijaju i testiraju sve mogućnosti model. Odnosno, neophodno je provjeriti kakva je distribucija podataka za svih 6 aktivnosti, kako se ne bi desilo da neka aktivnost nije zastupljena u oba skupa podataka.

In [8]:
```
# 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)
```

```
Train Dataset
Class=1, total=1226, percentage=16.676
Class=2, total=1073, percentage=14.595
Class=3, total=986, percentage=13.411
Class=4, total=1286, percentage=17.492
Class=5, total=1374, percentage=18.689
Class=6, total=1407, percentage=19.138
Test Dataset
Class=1, total=496, percentage=16.831
Class=2, total=471, percentage=15.982
Class=3, total=420, percentage=14.252
Class=4, total=491, percentage=16.661
Class=5, total=532, percentage=18.052
Class=6, total=537, percentage=18.222
Both
Class=1, total=1722, percentage=16.720
Class=2, total=1544, percentage=14.992
Class=3, total=1406, percentage=13.652
Class=4, total=1777, percentage=17.254
Class=5, total=1906, percentage=18.507
Class=6, total=1944, percentage=18.876
```

Upoređujući ispisane rezultate, može se primijetiti da su razlike među njima zanemarive te je moguće preći na idući korak u razvoju modela.

Analiza podataka i kreiranje modela

Nakon pretprocesiranja ulaznih podataka, moguće ih je grafički obraditi kako bi izvukli odgovarajuće zaključke. Prvo je potrebno proći kroz fazu treniranja modela te će biti učitani podaci za treniranje. Također, da bi se eliminisale duplicirane vrijednosti, bit će korištena „unique“ bibliotečna funkcija (istu je bilo potrebno prvo importovati iz numpy Python biblioteke).

```
In [11]: # 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)

         [ 1  3  5  6  7  8 11 14 15 16 17 19 21 22 23 25 26 27 28 29 30]
```

Na ovaj način su učitani podaci svih učesnika pri prikupljanju podataka. Iduće je potrebno izabrati jednog učesnika, recimo učesnika broj 10.

Nad ovim subjektom će biti primijenjen niz funkcija te će dosadašnji postupak biti objedinjen. Prvo će biti potrebno dohvatiti podatke za odgovarajućeg subjekta. Također, s obzirom na to da je pri prikupljanju i preprocesiranju ulaznih podataka došlo do određenog preklapanja (*overlapping* od 50%), isti je potrebno ukloniti kako bi rezultati bili vjerodostojni.

I naposlijetku, moguće je grafički prikazati podatke žiroskopa i akceleratora, ovisne o vremenu.

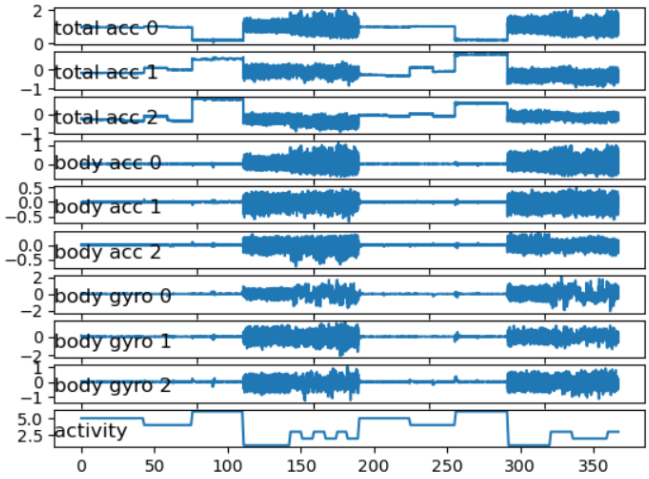
```
In [19]: # plot all vars for one subject
         from numpy import array
         from numpy import dstack
         from numpy import unique
         from pandas import read_csv
         from matplotlib import pyplot

         # 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[10]
subX, suby = data_for_subject(trainX, trainy, sub_map, sub_id)
print(subX.shape, suby.shape)
# plot data for subject
plot_subject(subX, suby)
```

```
[ 1  3  5  6  7  8 11 14 15 16 17 19 21 22 23 25 26 27 28 29 30]
(368, 128, 9) (368, 1)
```



Kao rezultat, dobija se uzorak za subject 10, numerički i grafički predstavljen.

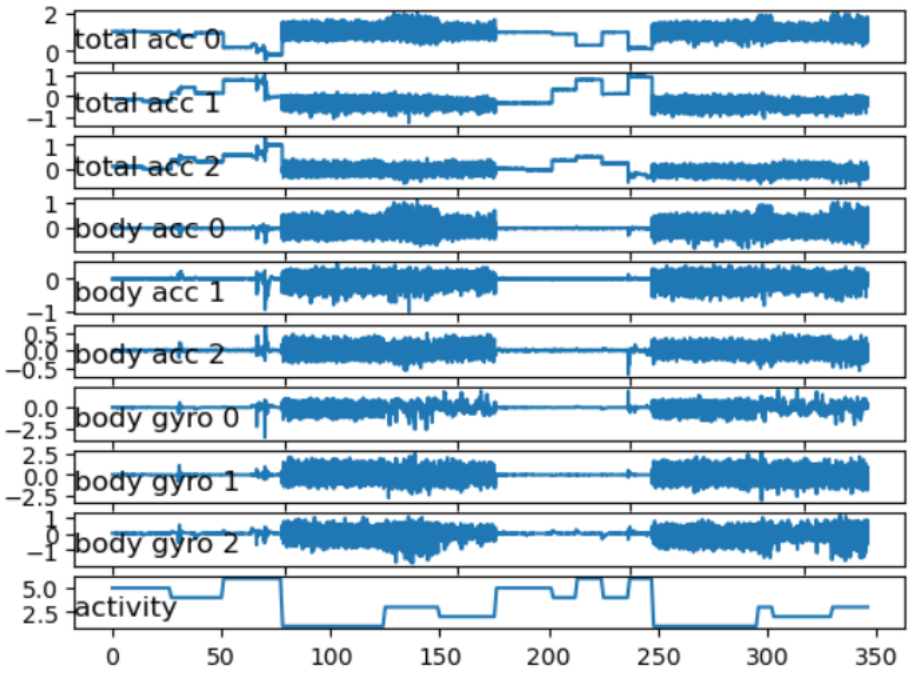
Analizirajući dobijene grafike, vidljivo je da zona sa višom frekvencijom, odnosno gustim linijama s mnogo varijacija, odgovara aktivnostima koje uključuju hodanje. S druge strane, ravnim linijama sa manjim skokovima u amplitude odgovaraju aktivnosti sjedenja, stajanja i ležanja. Međutim, moguće je primijetiti da su skokovi u amplitude ipak prilično primjetni. Ovo bi se moglo potkrijepiti činjenicom da je rijetko koji korisnik potpuno miran, čak i kada se radi o ležanju ili sjedenju (npr., moguće je da mijenja pozicije). Ove fluktuacije je moguće eliminisati primjenom filtera.

Dakle, subjekt 10 je hodao barem dva puta.

Mijenjajući kod kako bi odgovarao različitim subjektima, moguće je dobiti grafičku reprezentaciju aktivnosti za još učesnika:

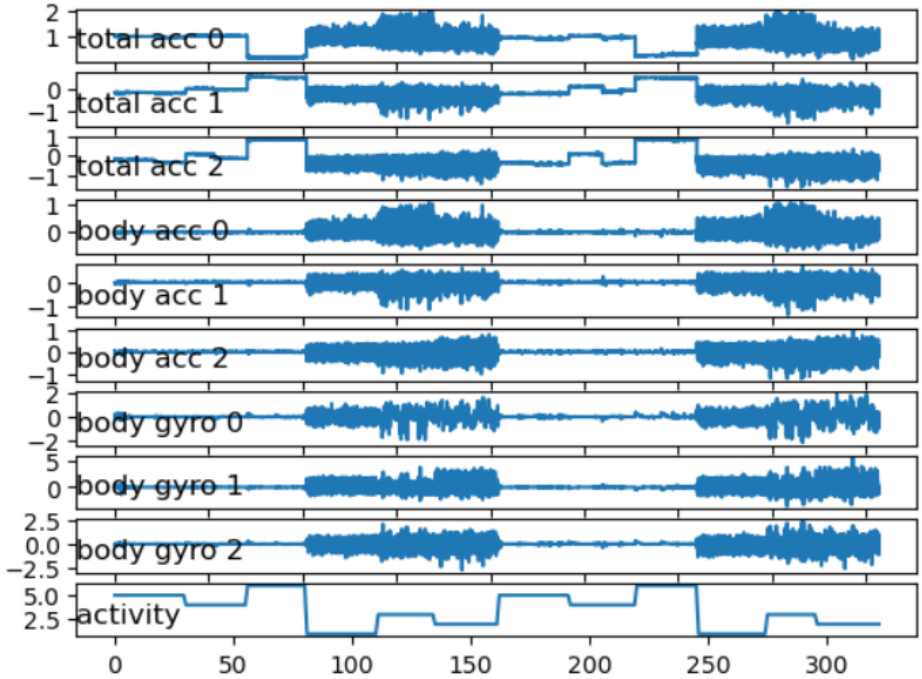
Subjekt 0:

```
[ 1  3  5  6  7  8 11 14 15 16 17 19 21 22 23 25 26 27 28 29 30]
(347, 128, 9) (347, 1)
```



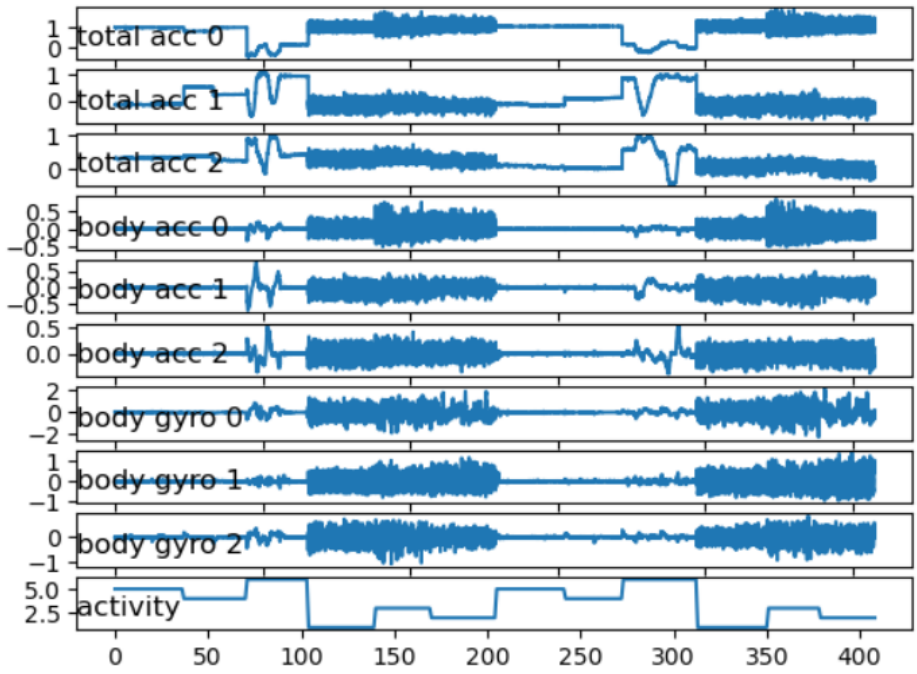
Subjekt 7:

```
[ 1  3  5  6  7  8 11 14 15 16 17 19 21 22 23 25 26 27 28 29 30]
(323, 128, 9) (323, 1)
```



Subjekt 15:

```
[ 1  3  5  6  7  8 11 14 15 16 17 19 21 22 23 25 26 27 28 29 30]
(409, 128, 9) (409, 1)
```



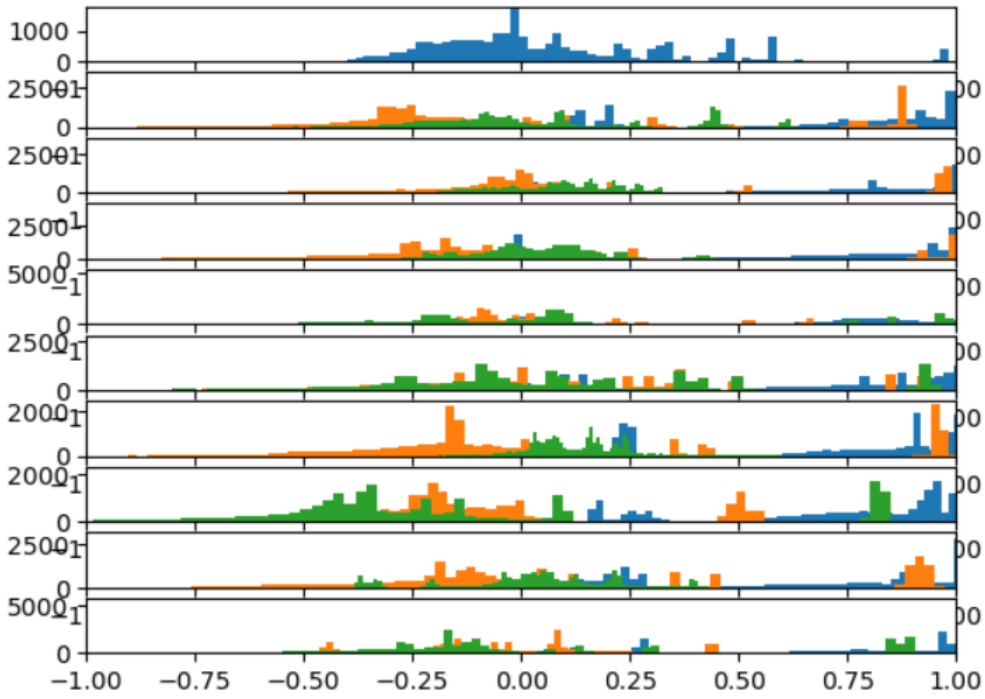
Poredeći ove grafove sa grafovima subjekta 10, mogu se izvesti slični zaključci.

Radi bolje analize, ove grafove je moguće pretvoriti u histograme.

Konstruisat će se po jedan histogram za svaki subjekt, na kojem će biti moguće očitati vrijednost sa sve tri ose, koristeći funkciju *plot_subject_histograms()*. Dodatno, ova funkcije ograničava broj subjekata na jednom histogramu.

```
# 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)
        # total 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[:,:,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)
```

```
C:\Users\Amina\AppData\Local\Temp\ipykernel_17016\1866615651.py:70: MatplotlibDeprecationWarning: Auto-removal of overlapping a
xes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  ax = pyplot.subplot(n, 1, k+1, sharex=xaxis)
```



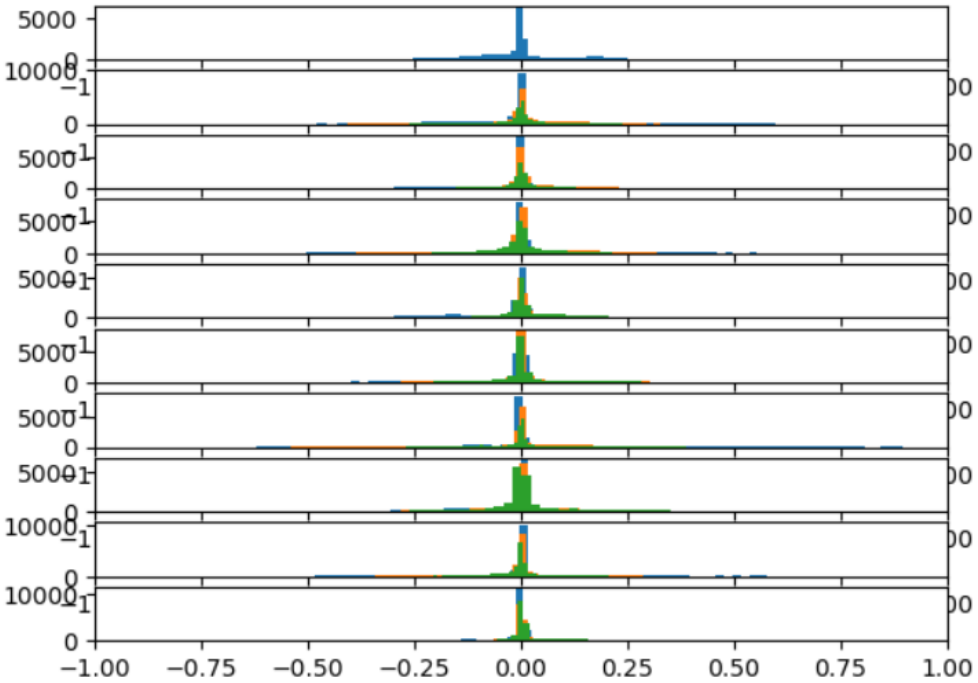
Tri boje predstavljaju različite ose: *x*, *y* i *z* su predstavljene plavom, narančastom i zelenom respektivno. Prikazano je prvih 10 subjekata te ukupne vrijednosti akceleratora za njih.

Sa histograma je vidljivo da su neke vrijednosti raspoređene oko nule, što upućuje na kontinualne distribucije podataka za prvih 10 uzoraka.

Može se primijetiti da amplituda sada djeluje kao zajednička horizontalna osa (vrijednost joj je i dalje između -1 i 1) te da histogrami predstavljaju vrijednosti akcelerometra, tj. ukupno ubrzanje.

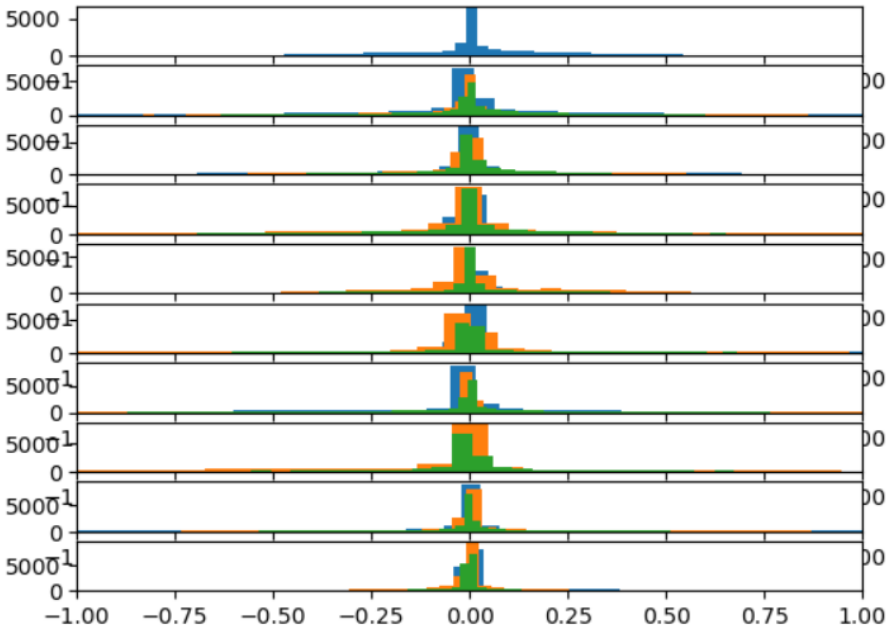
Međutim, bilo bi korisno prikazati ubrzanje tijela, umjesto ukupnog ubrzanja. Ovo se postiže tako što se izmijeni *subX* u kodu:
pyplot.hist(to_series(subX[:,:,3+i]), bins=100).

U ovom slučaju, uzimanje vrijednosti u petlji unutar *subX* niza (prva vrijednost predstavlja vremenske intervale, druga signale senzora, a treća x, y i z komponente) je pomjereno za 3, tako da se ne uzima ukupno ubrzanje već ubrzanje tijela. Očekivano, sada se grafovi mijenjaju kako bi predstavili vrijednosti ubrzanja tijela za 10 subjekata.



Ovaj put vrijednosti su centrirane oko nule, što može značiti i da je srednja vrijednost blizu nule.

Ukoliko se vrijednosti u petlji pomjere za još 3, moguće je dohvatiti i prikazati podatke za mjerenja žiroskopa (promjena pozicije tijela): *pyplot.hist(to_series(subX[:,:,6+i]), bins=100)*

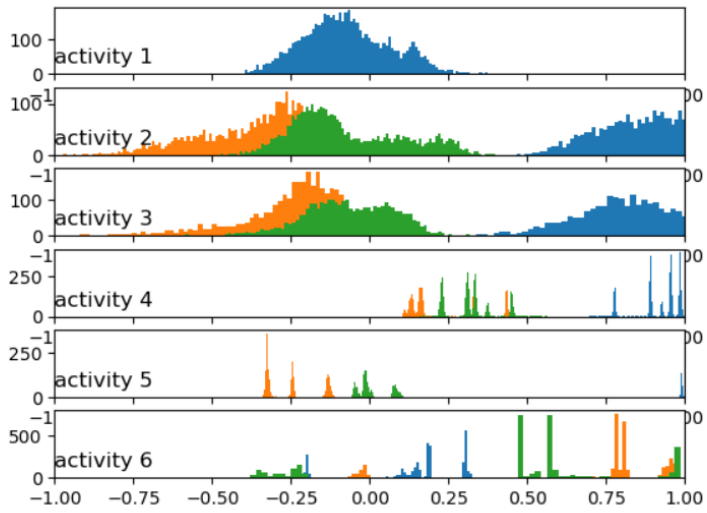


Sada kada su podaci istraženi u konekstu različitih subjekata, potrebno je slično uraditi i za svih 6 aktivnosti. U nastavku će biti predstavljeni grafovi koji predstavljaju vrijednosti ubrzanja i pozicije tijela za svaku aktivnost.

```
# 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][:,:,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)
```

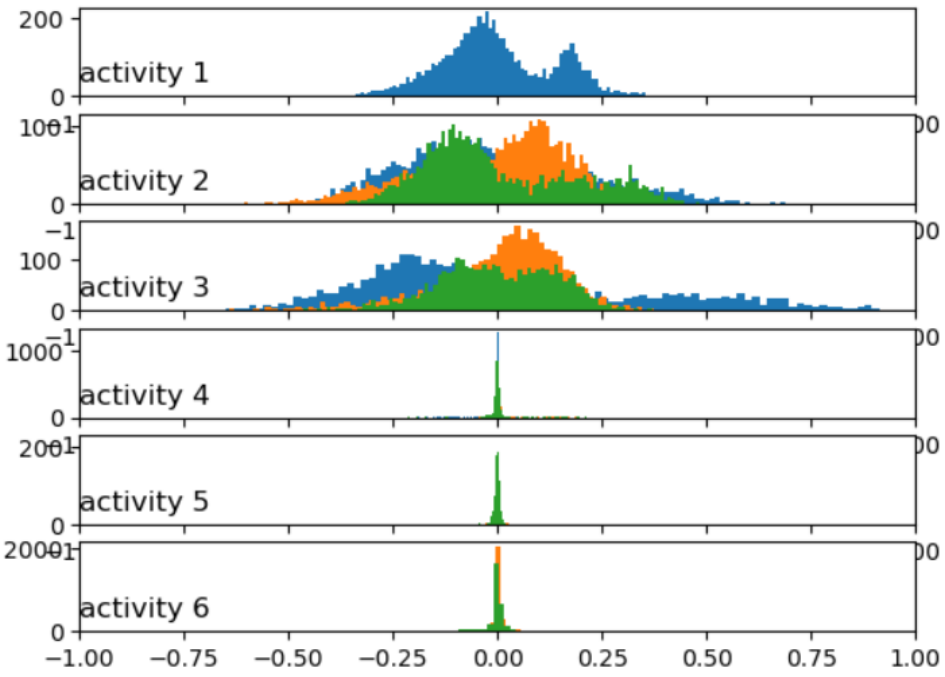
```
C:\Users\Amina\AppData\Local\Temp\ipykernel_17016\1250655591.py:75: MatplotlibDeprecationWarning: Auto-removal of overlapping a
xes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  ax = pyplot.subplot(len(activity_ids), 1, k+1, sharex=xaxis)
```



x, *y* i *z* osa su predstavljene plavom, narančastom i zelenom, respektivno. Jasno je vidljivo da svaka aktivnost ima različitu distribuciju, što su dobre vijesti, jer će ih biti prilično lagano raspoznati. Prve tri aktivnosti imaju Gaussovu raspodjelu, uz manja odstupanja, dok posljednje tri imaju multimodalne distribucije (imaju više od jednog vrha.)

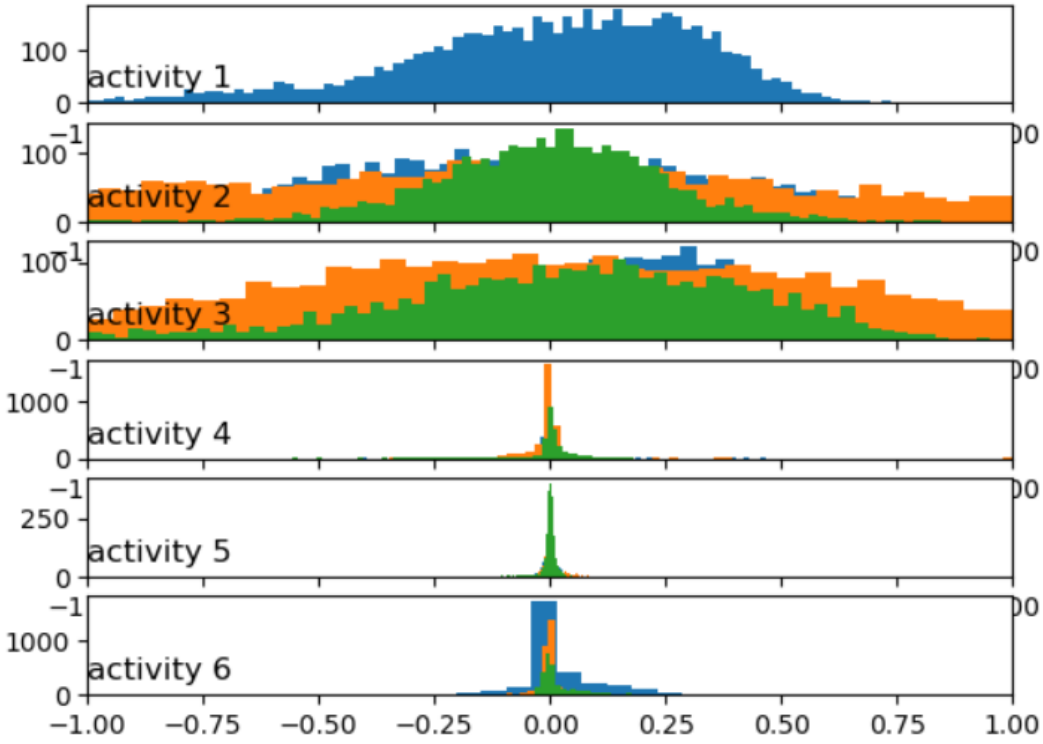
Podaci su najprije grupisani po aktivnosti, koristeći slične funkcije kao i ranije, a zatim su konstruisani histogrami ukupnog ubrzanja za svaku od šest mjerenih aktivnosti. Kao i dosada, ukupno ubrzanje predstavlja ubrzanje koje subjekt ima zbog svog kretanja, kao i zbog gravitacionog ubrzanja, dok se ubrzanje tijela odnosi samo na ono ubrzanje koje subjekt ima zbog kretanja tijela.

Histogram ubrzanja tijela se može dobiti na isti način kao i ranije, pomjeranjem uzorkovanja za 3.



Izdvojivši samo ubrzanje tijela, moguće je primijetiti da je distribucija stacionarnih aktivnosti (posljednja tri histograma) i nestacionarnih dosta sličnija. U ovom slučaju, prva tri histograma imaju bimodalne raspodjele, dok posljednja tri histograma imaju Gaussovu raspodjelu.

Što se tiče vrijednosti sa žiroskopa za svaku od šest aktivnosti, rezultati su slični onima za ubrzanje tijela, s izuzetkom što prva tri histograma također imaju Gaussovu distribuciju, koja je manje centrirana.



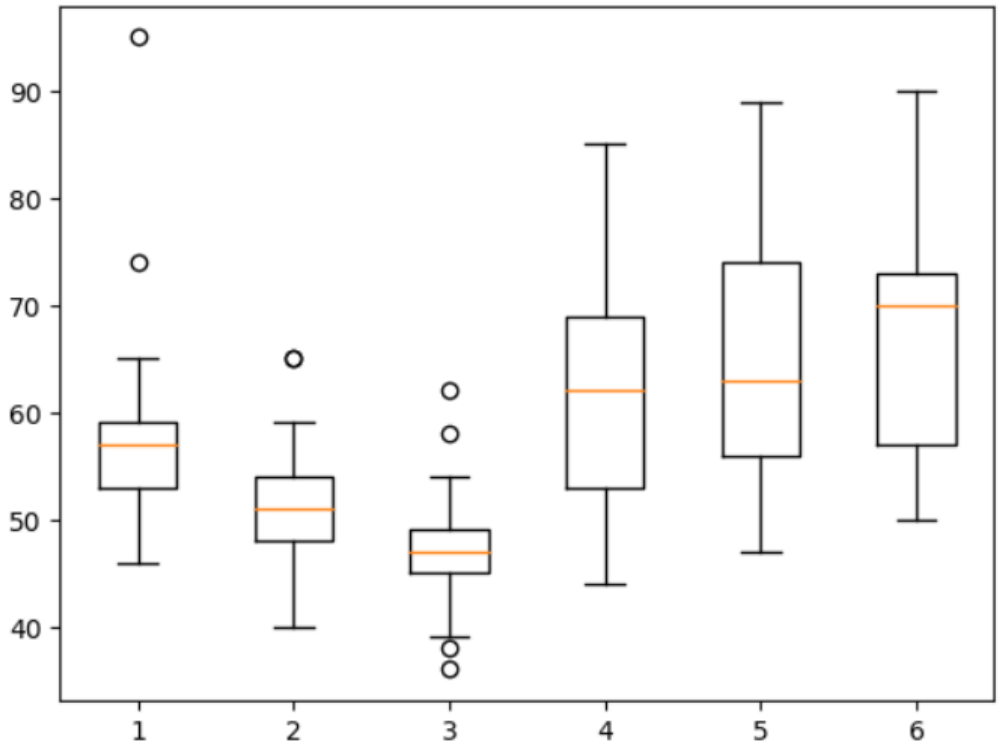
Dodatno, za klasifikaciju aktivnosti bi moglo biti korisno znati i koliko je koja aktivnost trajala. Za grafički prikaz bi se mogli koristiti takozvani „*box and whiskers*“ grafovi. Sa njih je moguće očitati medijanu (50%, ili drugi kvartil Q2), te prvi i treći kvartil, Q1 i Q3 (25% i 75%), kao i ekstreme (minimum i maksimum). Za iscrtavanje ovog grafa u *Pythonu* koristit će se *boxplot* funkcija.

Prije toga, podaci će biti organizovani tako što će se podijeliti po subjektima, zatim po aktivnostima na osnovu kojih će biti nacrtani odgovarajući grafovi (prebrojat će se redovi, odnosno prozori, za svaku aktivnost). Naravno, kao prvi korak bit će izvršeni slični koraci kao i dosad – učitavanje podataka, uklanjanje preklapanja (*overlapping*), i sl.

In [32]:
```
# 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 training dataset
X, y = load_dataset('train')
# load mapping of rows to subjects
sub_map = load_file('train/subject_train.txt')
# plot durations
plot_activity_durations_by_subject(X, y, sub_map)
```



Kao rezultat se dobije vrijeme (predstavljeno brojem prozora) koje su subjekti proveli radeći svaku od šest aktivnosti. Moguće je primijetiti da su subjekti više vremena proveli radeći stacionarne aktivnosti (aktivnost 4, 5 i 6, tj. sjedenje, stajanje i ležanje), pri čemu je najviše vremena utrošeno na aktivnost 5 (stajanje). Najmanje vremena je provedeno na aktivnosti 3, koja predstavlja hodanje niz stepenice.

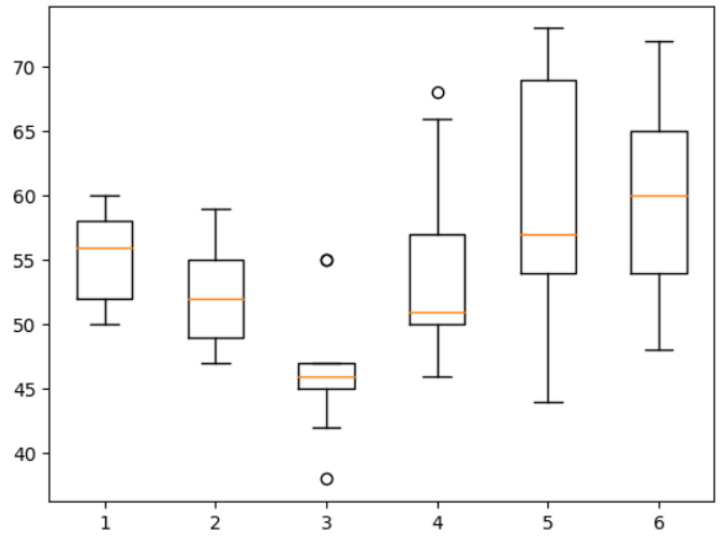
Zaključak je da ova skupina podataka jako dobro modelira svakodnevno ljudsko ponašanje (npr., većina ljudi zaista brže siđe niz stepenice no što se popne uz njih).

Naravno, treba imati na umu da je najstariji subjekt imao 48 godina, kao i da informacije o zdravlju i fizičkoj kondiciji subjekata nisu javno dostupne. Ovo bi trebalo imati na umu pri daljnjem modeliranju. Ukoliko pretpostavimo da su svi subjekti dobrog zdravlja i fizičke spreme, moguće je da se, poredeći rezultate ovih aktivnosti sa rezultatima osobe lošijeg zdravlja, ili u kasnim godinama, koja se kreće sporije, dobiju pogrešni rezultati.

Iz ovog razloga većina pametnih uređaja ima personalizovani profil za svoje korisnike, gdje modele pravi na osnovu njihovih podataka.

Svaki od ovih grafova se mogao nacrtati i za „*test*“ skupinu podataka, kako bi provjerili da li postoje neka značajna odstupanja. U nastavku se nalazi boxplot za testne podatke:

```
# 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)
```



Moguće je primijetiti da su rezultantni grafovi slični te da nije došlo do značajnije promjene (najduža aktivnost je i dalje aktivnost broj 5, najkraća broj 3, i sl.).

Nakon ovog postupka, moguće je na više različitih načina redizajnirati model koji bi u budućnosti prepoznavao o kojoj se aktivnosti radi.

Ponovo bi se prikupljali podaci o ubrzanju i kretanju tijela te bi se njihove raspodjele mogle uporediti sa raspodjelama dobijenim u prethodnoj analizi, koristeći statističke formule za Gaussovu (normalnu) distribuciju (raspodjelu), binomijalnu raspodjelu, multinomijalnu, i sl.

Naravno, svaki novi skup podataka bi morao biti obrađen na isti način kao što je prikazano ovdje.

Zasnivajući se na ovoj analizi, moguće je napraviti i predviđajući model. Ovo se najčešće postiže korsiteći neuralne mreže te stoga neće biti detaljnije obrađeno u ovom izlaganju.

Efikasnost modela je moguće testirati na više načina. Jedan od njih su takozvane *confusion* matrice.

Pri dizajnu jednog modela česta je pojava da se podaci klasificiraju na više različitih načina, kako bi se utvrdio najefikasniji model. U nastavku će biti istraženi različiti načini klasifikacije, a njihov uspjeh testiran konstrukcijom *confusion* matrica.

Klasifikatori i testiranje modela

Logistička regresija

Logistička regresija predstavlja algoritam korišten za binarnu klasifikaciju tako što modelira vezu između ulaznih podataka (nezavisnih varijabli) i vjerovatnoće da određeni podatak pripada specifičnoj klasi (zavisne varijable). Statistički model koji se dobije ovom metodom se naziva „*logit*“.

Za predviđanja, ovaj model učenja koristi logistički *sigmoid* funkciju nad linearnom kombinacijom ulaznog skupa podataka.

Mana ovog algoritma je da podrazumijeva da postoji linearna veza između varijabli i modela, što znači da neće davati pouzdane rezultate u slučaju kompleksnijeg skupa podataka. U tom slučaju, može se koristiti *Decision Tree* algoritam ili LDA, koji će također biti implementirani u nastavku.

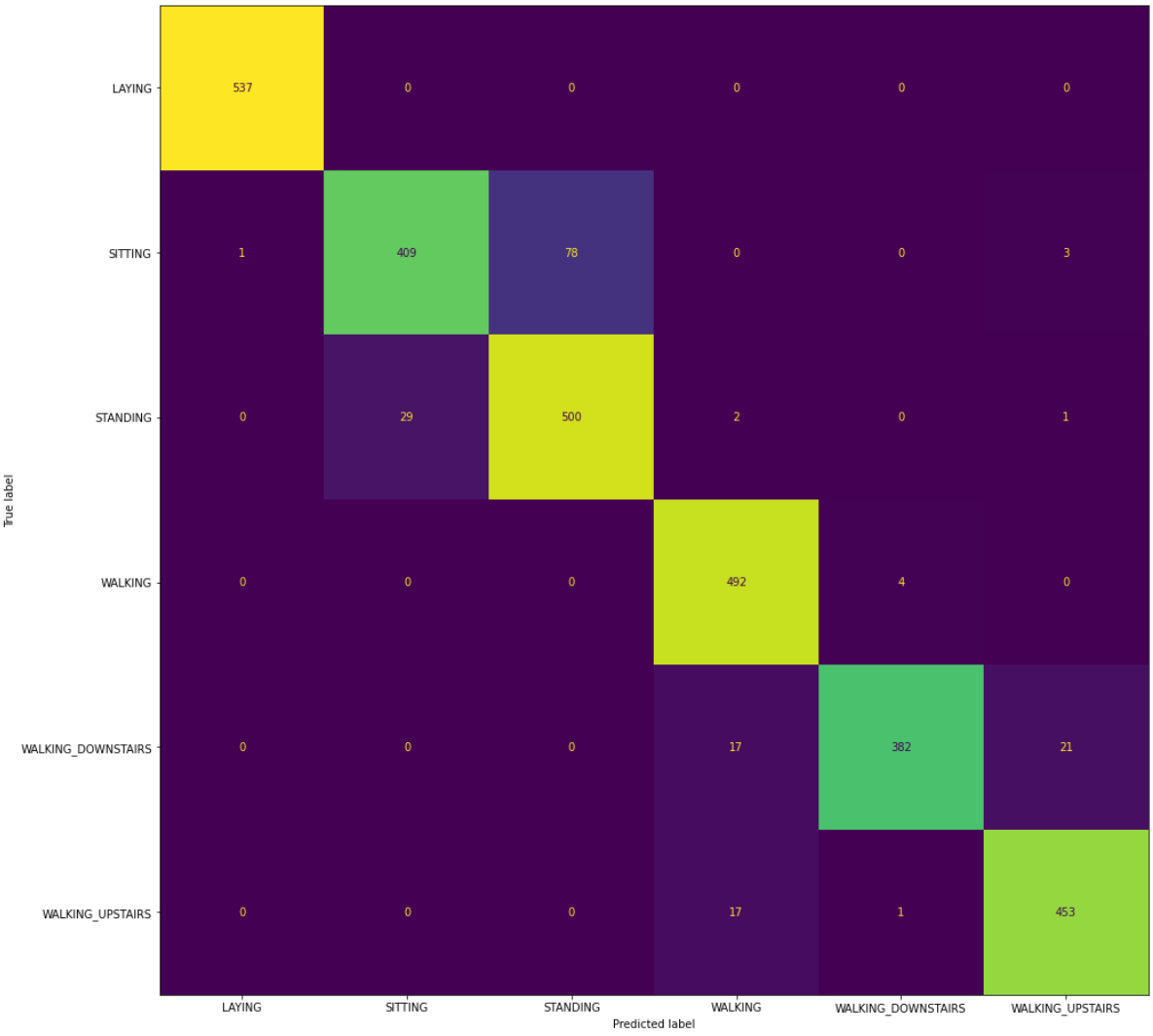
U nastavku se nalazi primjer implementacije logičke regresije, gdje *X__train* i *Y_train* nisu ništa drugo do ranije učitani podaci pri demonstraciji rada PCA.

```
from sklearn.linear_model import LogisticRegression
#logistic regression
X_train=train_df.drop(train_df.columns[20],axis=1)
Y_train=train_df[train_df.columns[20]]
X_test=test_df.drop(train_df.columns[20],axis=1)
Y_test=test_df[train_df.columns[20]]

lr_clf=LogisticRegression(C=0.01,solver='liblinear')
lr_clf.fit(X_train,Y_train)
```

```
from sklearn.metrics import classification_report,accuracy_score
y_hat=lr_clf.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score,confusion_matrix,classification_report,ConfusionMatrixDisplay
cm = confusion_matrix(Y_test,y_hat, labels=lr_clf.classes_)
disp= ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=lr_clf.classes_)
fig, ax = plt.subplots(figsize=(20,20))
disp.plot(ax=ax)
```

Kao rezultat, dobija se sljedeća confusion matrica:


LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
True label
537 0 0 0 0 0
1 409 78 0 0 3
0 29 500 2 0 1
0 0 0 492 4 0
0 0 0 17 382 21
0 0 0 17 1 453
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
Predicted label

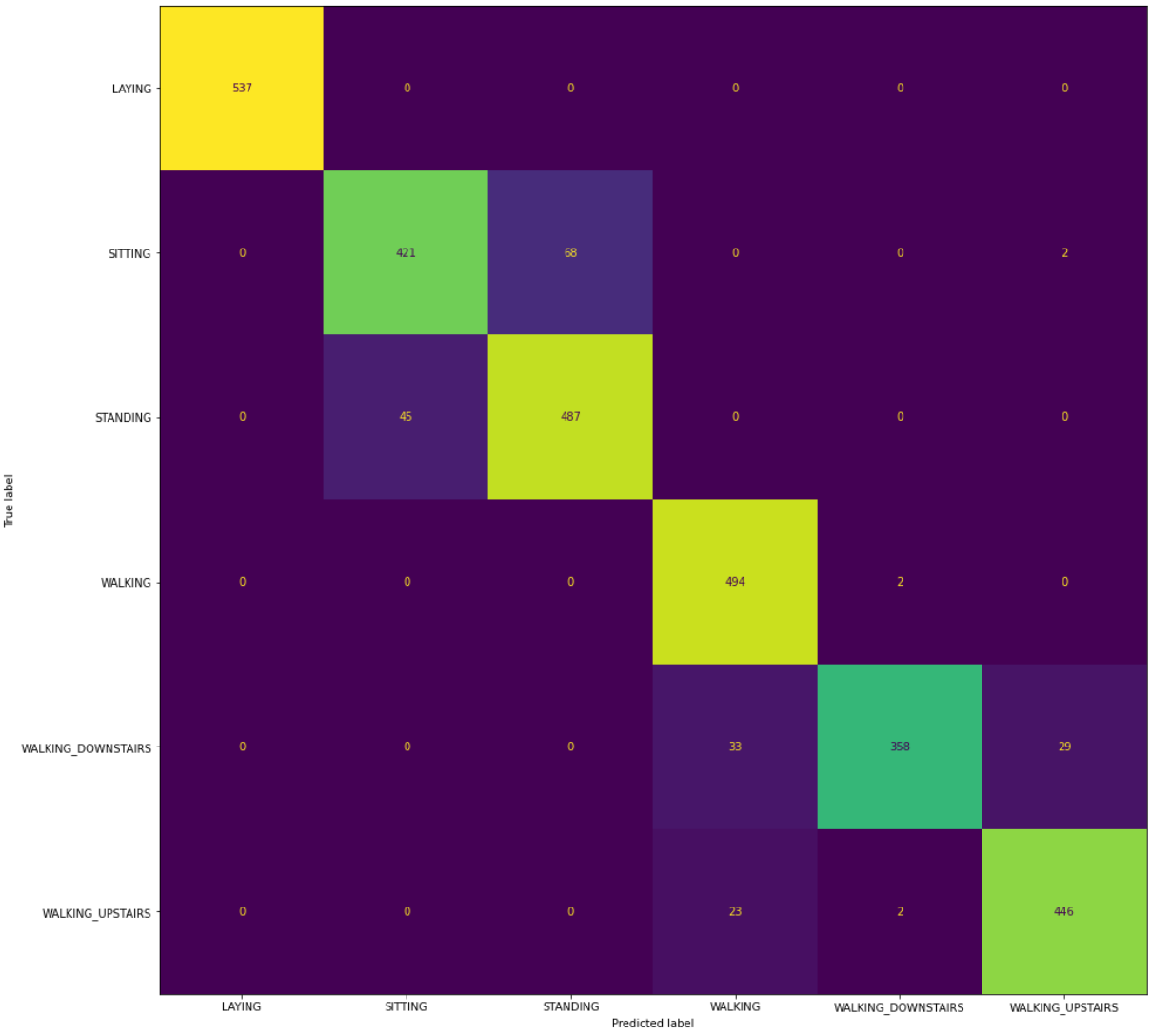
Support Vector Machine

Support Vector Machine (SVM), slično kao i logistička regresija, spada u modele učenja, i jedan je od često primijenjenih algoritama, naročito kod mašinskog učenja.

Princip na kojem počiva način rada SVM jeste podjela podataka u dvije klase koristeći *kernel* funkciju.

In [75]:
```
from sklearn import svm

svm_clf=svm.SVC(kernel='rbf')
svm_clf.fit(X_train,Y_train)
y_hat=svm_clf.predict(X_test)
```

In [76]:
```
cm = confusion_matrix(Y_test,y_hat, labels=svm_clf.classes_)
disp= ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=svm_clf.classes_)
fig, ax = plt.subplots(figsize=(20,20))
disp.plot(ax=ax)
```

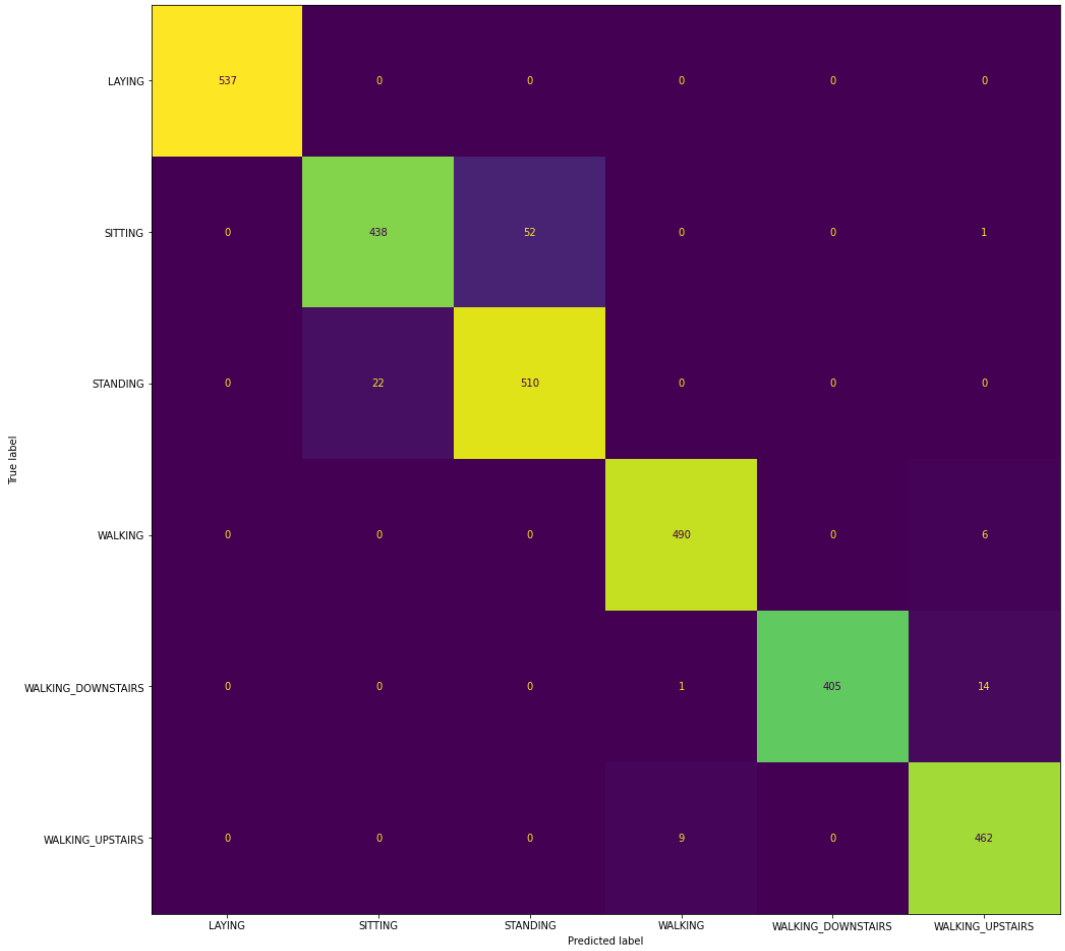

LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
True label
537 0 0 0 0 0
0 421 68 0 0 2
0 45 487 0 0 0
0 0 0 494 2 0
0 0 0 33 358 29
0 0 0 23 2 446
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
Predicted label

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) je algoritam učenja koji se koristi za redukciju dimenzionalnosti (dakle, primjeren je za ovaj skup podataka) i klasifikaciju, i to najčešće multiklasnu klasifikaciju, iako daje dobre rezultate i za binarnu klasifikaciju.

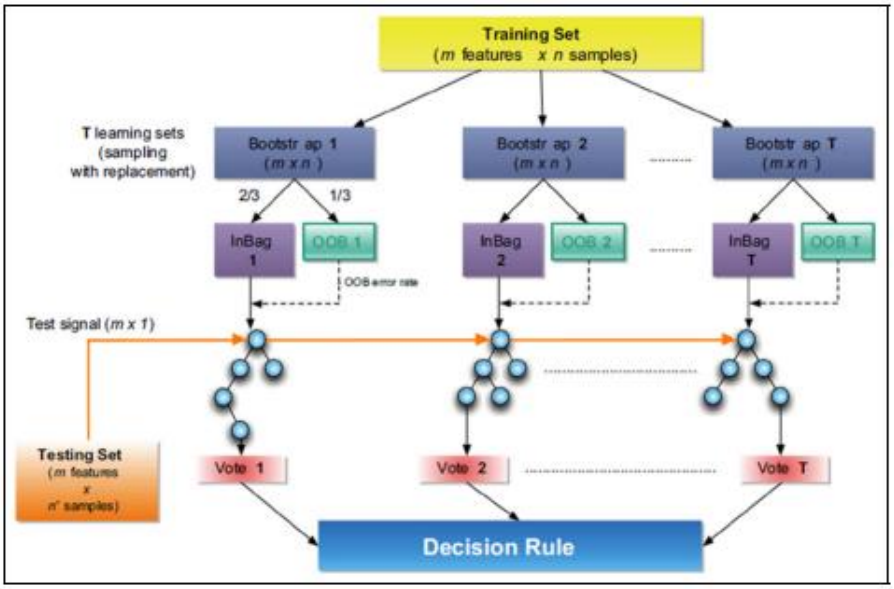
LDA radi na način da pronalazi linearnu kombinaciju koja maksimizira udaljenost među klasama, a minimizira varijaciju u klasama. Ovo postiže projektovanjem podataka u prostor niže dimenzionalnosti, čuvajući pri tom što je više moguće informacije o klasi podataka.

```
In [77]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         LDA_clf = LinearDiscriminantAnalysis()
         LDA_clf.fit(X_train,Y_train)
         y_hat=LDA_clf.predict(X_test)
```



Random Forest Algorithm

Random Forest algoritam (RFA) je poznati algoritam, usko vezan za mašinsko učenje, i korišten za klasifikaciju podataka. RFA bira (potpuno nasumično ili primjenom neke sistematične metode) skupove podataka i njihovih osobina, i za svaki odabrani skup konstruiše po jedno (nasumično donoseći odluke i birajući pravac kretanja) stablo odlučivanja (decision tree), tzv. estimator, od kojih će svako kao rezultat dati svoju procjenu. Procjena koja se najviše puta ponovila je uzeta kao krajnji rezultat.



Dijagram toka za *random forest algorithm*
(Balli, Sagbas and Peker, 2018, p. 40)

U nastavku se nalazi implementacija ovog algoritma sa promjenom dimenzionalnosti podataka, kako bi instanca DTA klasifikatora iste mogla prihvatiti (ranije je ista promijenjana na 3, u svrhe grafičkog prikaza podataka na tri ose).

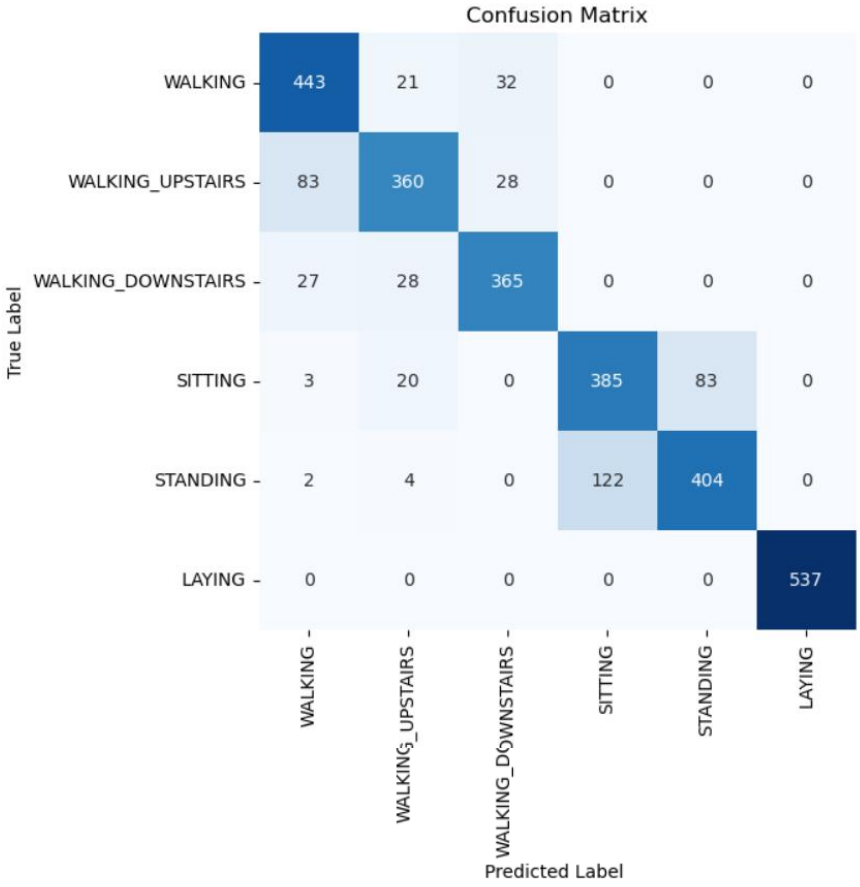
In [88]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

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

```
# 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)
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
Accuracy: 0.846284356973193
```

In [14]:

```
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 Algorithm

Decision tree algoritam (DTA) je još jedan algoritam koji se često koristi za klasifikaciju podataka.
Ovaj algoritam uči pravila koja mu omogućuju da konstruiše stablo odlučivanja te da se kreće kroz njega. Svaki roditelj čvor stabla odgovara varijabli ili nekom atributu, djeca čvorovi predstavljaju oznaku odgovarajuće klase ili pak predviđenu vrijednost, dok su grane odluke.

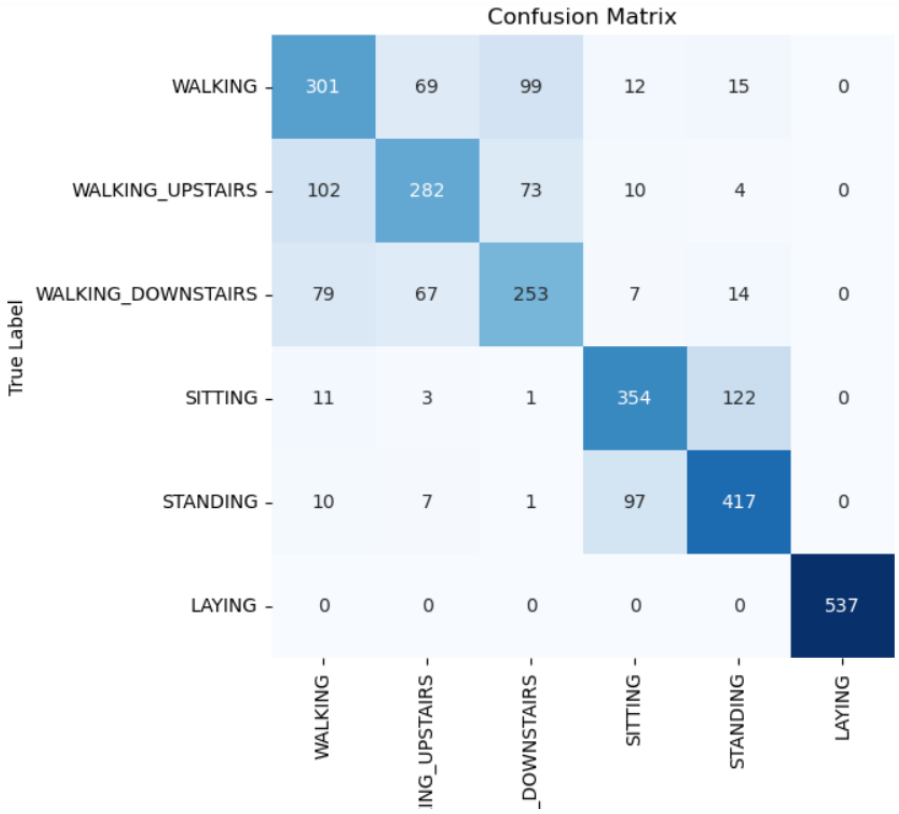
Kada je konstrukcija stabla odlučivanja završena, isto se može koristiti za predviđanja rezultata.

U nastavku se nalazi implementacija ovog algoritma sa promjenom dimenzionalnosti podataka, kako bi instanca DTA klasifikatora iste mogla prihvatiti.

In [92]:
```
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)
```

```
Accuracy: 0.7302341364099084
```

In [18]:
```
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()
```



Moguće je primijeitit da labele za klase, tj. za aktivnosti, imaju drukčiji raspored za RFA i DTA u odnosu na druge klasifikatore. Razlog ovome je što sam podatke učitala na malo drukčiji način pri kreiranju posljednje dvije matrice te je došlo do promjene u rasporedu.

Zaključak

Sada kada su dostupne *confusion* matrice za svaki algoritam klasifikacije, kao i njihove tačnosti, moguće je izvesti zaključke o ponašanju modela te najboljem klasifikatoru za isti.

Radi preglednosti, u nastavku su date vrijednosti tačnosti (*accuracy*) za svaki od ranije analiziranih klasifikatora, primijenjenih nad ovim skupom podataka:

1) Logistička regresija – 0.9409569053274517
2) SVM - 0.9307770614183916
3) LDA - 0.9643705463182898
4) RFA – 0.846284356973193
5) DTA – 0.7302341364099084

Može se izvesti zaključak da najbolju tačnost ima LDA, a zatim redom logistička regresija, SVM, RFA, DTA.

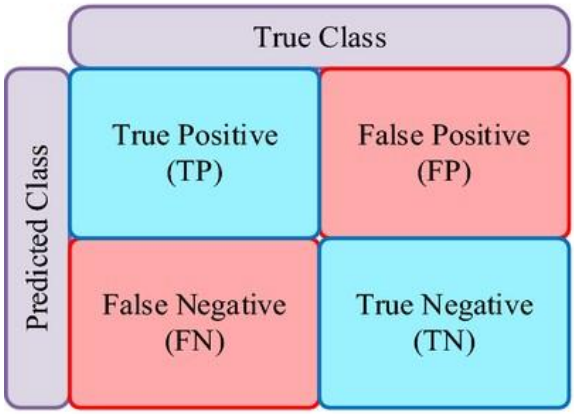
Što se tiče confusion matrica, one se tumače pomoću svojih predviđenih i pravih vrijednosti (redovi predstavljaju prave, „*true*“ vrijednosti, a kolone predviđene, „*predicted*“).
Dio matrice predstavlja pozitivne (positive) vrijednosti, a dio negativne (*negative*) vrijednosti.

Zatim se posmatra broj TP (True Positives), TN (True Negatives), FP (False Positives) te FN (False Negatives).

TP predstavlja broj instanci koje su ispravno predviđene kao pozitivne (i red i kolona su pozitivni).
TN predstavlja broj instanci koje su ispravno predviđene kao negativne (i red i kolona su negativni).
FP predstavlja broj instanci koje su pogrešno predviđene kao pozitivne (red je negativan, a kolona je pozitivna).
FN predstavlja broj instanci koje su pogrešno predviđene kao negativne (red je pozitivan, a kolona je negativna).

Postoji više različitih metrika koje se koriste pri radu s *confusion* matricama, poput tačnosti (*accuracy*), preciznosti (*precision*), osjetljivosti (*recall*), F1 rezultat (*F1 score*, kombinuje preciznost i osjetljivost).

Formule za dobijanje ovih mjera su prilično jednostavne, kao što se može i vidjeti u nastavku.



Function Name	Formula
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1 score	$\frac{2.Precision.Recall}{Precision+Recall}$

Formule za rad s *confusion* matricom (Raihan et al., 2023)

Iako su matrice predstavljene u ovom izvještaju multiklasne, iste je moguće pretvoriti u matricu za binarne klase te na taj način ručno izračunati vrijednosti za tačnost koje su već ranije dobijene kroz kod.

Literatura

Balli, S., Sagbas, E.A., and Peker, M. (2018) “Human activity recognition from smart watch sensor data using a hybrid of principal component analysis and random forest algorithm”, *Measurement and Control 2019, Vol. 52(1-2),* pp.37–45.

Brownlee, J. (2018). How to Model Human Activity From Smartphone Data. [Online] Available at: https://machinelearningmastery.com/how-to-model-human-activity-from-smartphone-data/ (Accessed: 25th June 2023).

Esmaeilpour, Z. (n.d.). Human Activity Recognition. [Online] Kaggle. Available at: https://www.kaggle.com/code/zeinabesmaeilpour/human-activity-recognition/notebook (Accessed: 29th June 2023).

Raihan, M.J., Khan, M.A.M., Kee, S.-H., Nahid, A. (2023). "Detection of the chronic kidney disease using XGBoost classifier and explaining the influence of the attributes on the model using SHAP." Scientific Reports, 13. doi: 10.1038/s41598-023-33525-0.