Towards-Global-Aerobic-Activity-Monitoring with PySpark

Gabriele Cialone, gabriele.cialone@studio.unibo.it

Abstract

Application of **classification algorithms** to raw data of the time series and to some high level features (frequency features) extracted from the raw data for human activity recognition, as suggested by the papers:

- Introducing a New Benchmarked Dataset for Activity Monitoring (in relevant papers of the UCI dataset link)
- Towards global aerobic activity monitoring (cited by the first one)

relying on **PySpark** and **HDFS**.

The dataset used for the experiment is: PAMAP2 Physical Activity Monitoring

1 Architecture

The **cluster** is made up of **3 Vagrant VMs**. Each VM is created with its own specific Vagrantfile that specifies its ip address. The HADOOP cluster architecture is made up of:

- **node-master**: *NameNode-SecondaryNameNode-ResourceManager*

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://vagrantcloud.com/search.
  config.vm.box = "ubuntu/jammy64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # NOTE: This will enable public access to the opened port
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine and only allow access
  # via 127.0.0.1 to disable public access
  # config.vm.network "forwarded_port", guest: 8888, host: 8888, host_ip: "127.0.0.1"

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  config.vm.network "private_network", ip: "10.0.0.3"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

```

```
  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
    vb.memory = "4096"
    vb.customize ["modifyvm", :id, "--vram", "32"]
  end
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Enable provisioning with a shell script. Additional provisioners such as
  # Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
  # config.vm.provision :shell, path: "bootstrap.sh"
end
```

- **node1:** *DataNode-NodeManager*

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://vagrantcloud.com/search.
  config.vm.box = "ubuntu/jammy64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # NOTE: This will enable public access to the opened port
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine and only allow access
  # via 127.0.0.1 to disable public access
  # config.vm.network "forwarded_port", guest: 8888, host: 8888, host_ip: "127.0.0.1"

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  config.vm.network "private_network", ip: "10.0.0.4"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
```

```
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
    vb.memory = "4096"
    vb.customize ["modifyvm", :id, "--vram", "32"]
  end
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Enable provisioning with a shell script. Additional provisioners such as
  # Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
  # config.vm.provision :shell, path: "bootstrap.sh"
end
```

- **node2:** *DataNode-NodeManager*

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://vagrantcloud.com/search.
  config.vm.box = "ubuntu/jammy64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # NOTE: This will enable public access to the opened port
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine and only allow access
  # via 127.0.0.1 to disable public access
  # config.vm.network "forwarded_port", guest: 8888, host: 8888, host_ip: "127.0.0.1"

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  config.vm.network "private_network", ip: "10.0.0.5"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
```

```
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
    vb.memory = "4096"
    vb.customize ["modifyvm", :id, "--vram", "32"]
  end
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Enable provisioning with a shell script. Additional provisioners such as
  # Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
  # config.vm.provision :shell, path: "bootstrap.sh"
end
```

1.1 Configure the hadoop cluster

For each VM

1. **Configuration of the system**: a new user is created: *hadoop* user. Then it is added to the sudoers group.

```
sudo adduser hadoop
sudo visudo #for adding hadoop in sudoers
```

```
hadoop   ALL=(ALL:ALL) ALL #add to the sodoers
```

```
su - hadoop
mkdir .ssh
```

2. **Creation of the host file on each node**: the */etc/hosts* file is updated to let the VMs know each other replacing the raw ip with a name.

```
sudo nano /etc/hosts
```

```
#add the following rows:
10.0.0.3    node-master
10.0.0.4    node1
10.0.0.5    node2
```

3. **Distribution of authentication key-pairs for the Hadoop user**: SSH authentication key-pairs are distributed for the hadoop user. The *node-master* has to generate a public key that has to be added to its *authorized_keys* and the other nodes' ones as well.

   (a) Login to node-master as the Hadoop user, and generate an SSH key:

   ```
   ssh-keygen -b 4096
   ```

   (b) In each VM, make a new file master.pub in the /home/hadoop/.ssh directory. Paste the public key into this file and save the changes.

   ```
   cd .ssh
   cat id_rsa.pub
   nano master.pub
   ```

   (c) Copy the key file into the authorized key store

```
cat ~/.ssh/master.pub >> ~/.ssh/authorized_keys
```

4. **Downloading and unpacking hadoop binaries**: the hadoop binaries are downloaded and unpacked

   (a) Log into node-master as the hadoop user, download the Hadoop tarball from Hadoop project page, and unzip it:

```
cd
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz
tar -xzf hadoop-3.3.4.tar.gz
mv hadoop-3.3.4 hadoop
```

5. **Installing Java**: Java is installed and the *JAVA_HOME* is added to the *PATH* and to the *ENVIRONMENT VARIABLES*. On all VMs:

   (a) Installing Java:

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

   (b) Setting JAVA_HOME

      i. Edit ~/hadoop/etc/hadoop/hadoop-env.sh and replace this line:

```
export JAVA_HOME=${JAVA_HOME}
```

      ii. This has to be done for all the VMs and for the following file: [/etc/profile, .bashrc, .profile]

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
export PATH=${PATH}:${JAVA_HOME}

export HADOOP_HOME=/home/hadoop/hadoop
export PATH=${PATH}:${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin
```

6. **Setting NameNode location**: the *NameNode* location is set in the *core-site.xml* configuration file:

   (a) Update ~/hadoop/etc/hadoop/core-site.xml file to set the NameNode location to node-master on port 9000:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://node-master:9000</value>
    </property>
</configuration>
```

7. **Setting the path for HDFS**: the path for HDFS is set in the *hdfs-site.xml* configuration file

   (a) Edit ~/hadoop/etc/hadoop/hdfs-site.xml to resemble the following configuration:

```
<configuration>
    <property>
            <name>dfs.namenode.name.dir</name>
            <value>/home/hadoop/data/nameNode</value>
    </property>

    <property>
            <name>dfs.datanode.data.dir</name>
            <value>/home/hadoop/data/dataNode</value>
    </property>

    <property>
            <name>dfs.replication</name>
            <value>1</value>
    </property>
</configuration>
```

8. **Setting and configuring YARN as job scheduler**: YARN job scheduler is set and configured in *yarn-site.xml*.

   (a) Edit the ~/hadoop/etc/hadoop/mapred-site.xml file, setting YARN as the default framework for MapReduce operations:

```
<configuration>
    <property>
            <name>mapreduce.framework.name</name>
            <value>yarn</value>
    </property>
    <property>
            <name>yarn.app.mapreduce.am.env</name>
            <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
            <name>mapreduce.map.env</name>
            <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
            <name>mapreduce.reduce.env</name>
            <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
</configuration>
```

   (b) Edit ~/hadoop/etc/hadoop/yarn-site.xml, which contains the configuration options for YARN

```
<configuration>
    <property>
            <name>yarn.acl.enable</name>
            <value>0</value>
    </property>

    <property>
            <name>yarn.resourcemanager.hostname</name>
            <value>node-master</value>
    </property>

    <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

9. **Configuring workers**: the *workers* file is used by startup to start required daemons on all nodes. For this purpose this file is edited to include both the nodes. Edit ~/hadoop/etc/hadoop/workers to include both of the nodes:

```
node1
node2
```

10. **Configuring memory allocation**: the memory allocation properties are configured in *yarn-site.xml*

    (a) Edit ~/hadoop/etc/hadoop/yarn-site.xml and add the following lines:

```
<property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>1536</value>
</property>

<property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>1536</value>
</property>

<property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>128</value>
</property>

<property>
        <name>yarn.nodemanager.vmem-check-enabled</name>
        <value>false</value>
</property>
```

11. **Configuration of map-reduce**: the configuration for map reduce is set in the *mapred-site.xml*

    (a) Edit ~/hadoop/etc/hadoop/mapred-site.xml and add the following lines:

```
<property>
        <name>yarn.app.mapreduce.am.resource.mb</name>
        <value>512</value>
</property>

<property>
        <name>mapreduce.map.memory.mb</name>
        <value>256</value>
</property>

<property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>256</value>
</property>
```

12. **Duplicating configuration files on each node**: it is necessary to duplicate configuration files on each node

    (a) Copy the Hadoop binaries to worker nodes:

```
 cd /home/hadoop/
 scp hadoop-*.tar.gz node1:/home/hadoop
 scp hadoop-*.tar.gz node2:/home/hadoop
```

    (b) Unzip the binaries, rename the directory:

```
tar -xzf hadoop-3.3.4.tar.gz
mv hadoop-3.3.4 hadoop
```

    (c) Run on the master to copy the Hadoop configuration files to the worker nodes:

```
for node in node1 node2; do
scp ~/hadoop/etc/hadoop/* $node:/home/hadoop/hadoop/etc/hadoop/;
done
```

13. **Formatting HDFS**: on the *node-master*:

    (a) HDFS needs to be formatted like any classical file system. For this purpose, this command has to be run on the *node-master*.

```
hdfs namenode -format
```

    (b) The HDFS has to be started and stopped with

```
start-dfs.sh
```

    and

```
stop-dfs.sh
```

    (c) It is necessary to run the command

```
start-yarn.sh
```

    to start a *ResourceManager* on *node-master* and *NodeManager* on *node1* and *node2*

    (d) Check that every process is running with the jps command on each node. On node-master, you should see the following (the PID number will be different)

```
jps
```

    The expected output on *node-master* is something like that

```
xxxxx Jps
yyyyy NameNode
zzzzz SecondaryNameNode
iiiii ResourceManager
```

    The expected output on **data nodes** is something like that:

```
jjjjj Jps
kkkkk DataNode
lllll NodeManager
```

14. **Monitoring YARN and Hadoop**:

    - This has to be run to monitor hadoop:

```
hdfs dfsadmin -report
hdfs dfsadmin -help
http://localhost:9870 after vagrant ssh -- -L 9870:localhost:9870
```

    - This has to be run to minitor yarn:

```
hdfs dfsadmin -report
hdfs dfsadmin -help
http://localhost:9870 after vagrant ssh -- -L 9870:localhost:9870

yarn node -list
yarn application -list
http://localhost:8088
```

15. **File loading**: for the file loading step, it is necessary to run on the *node-master*:

```
hdfs dfs -mkdir -p /user/hadoop
hdfs dfs -mkdir dataset
hdfs dfs -put PAMAP2_Dataset dataset
```

1.2 Setting up and configuring Spark on top of the Hadoop YARN cluster

1. **Downloading and installing Spark Binaries**: the Spark binaries are downloaded and unpacked. Log on node-master as the hadoop user, and run:

```
cd /home/hadoop
wget https://dlcdn.apache.org/spark/spark-3.4.0/spark-3.4.0-bin-hadoop3.tgz
tar -xvf spark-3.4.0-bin-hadoop3.tgz
rm spark-3.4.0-bin-hadoop3.tgz
mv spark-3.4.0-bin-hadoop3 spark
```

2. **Adding the Spark binaries directory to PATH**: Edit /home/hadoop/.profile and add the following line:

```
PATH=/home/hadoop/spark/bin:$PATH
```

3. **Integration of Spark with YARN**: to communicate with the YARN Resource Manager, Spark needs to be aware of your Hadoop configuration. This is done via the HADOOP_CONF_DIR environment variable. The SPARK_HOME variable is not mandatory, but is useful when submitting Spark jobs from the command line.

   (a) Edit the hadoop user profile /home/hadoop/.profile and add the following lines:

```
export HADOOP_CONF_DIR=/home/hadoop/hadoop/etc/hadoop
export SPARK_HOME=/home/hadoop/spark
export LD_LIBRARY_PATH=/home/hadoop/hadoop/lib/native:$LD_LIBRARY_PATH
```

   (b) Restart the session by logging out and logging in again.

   (c) Rename the spark default template config file:

```
mv $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-
defaults.conf
```

   (d) Edit $SPARK_HOME/conf/spark-defaults.conf and set spark.master to yarn:

```
spark.master    yarn
```

4. **Configuring memory allocation**: it is important to give YARN Containers the maximum allowed memory. If the memory requested is above the maximum allowed, YARN will reject creation of the container, and the Spark application won't start. Get the value of yarn.scheduler.maximum-allocation-mb in $HADOOP_CONF_DIR/yarn-site.xml. This is the maximum allowed value, in MB, for a single container. Make sure that values for Spark memory allocation, configured in the following section, are below the maximum.

5. **Configuring the Spark driver memory allocation in cluster mode**: In order to configure the Spark Driver Memory Allocation in cluster mode, it is necessary to edit the file $SPARK_HOME/conf/spark-defaults.conf:

```
spark.driver.memory    512m
```

In cluster mode, the Spark Driver runs inside YARN Application Master. The amount of memory requested by Spark at initialization is configured either in spark-defaults.conf, or through the command line.

6. **Configuring the Spark executors' memory allocation**: the Spark Executors' memory allocation is calculated based on two parameters inside $SPARK_HOME/conf/spark-defaults.conf:

   - spark.executor.memory: sets the base memory used in calculation
   - spark.yarn.executor.memoryOverhead: is added to the base memory. It defaults to 7% of base memory, with a minimum of 384MB

   To set executor memory to 512MB, edit $SPARK_HOME/conf/spark-defaults.conf and add the following line:

```
spark.executor.memory          512m
```

2 Data description

- 3 Colibri wireless IMUs
  - sampling frequency: 100H
    * 1 IMU over the wrist on the dominant arm
    * 1 IMU on the chest
    * 1 IMU on the dominant side’s ankle
- 54 columns per row
  - 1 timestamp (s)
  - 2 activityID (class)
  - 3 heart rate (bpm)
  - 4-20 IMU hand
  - 21-37 IMU chest
  - 38-54 IMU ankle
- The IMU sensory data contains the following columns:
  - 1 temperature (°C)
  - 2-4 3D-acceleration data (ms-2), scale: ±16g, resolution: 13-bit
  - 5-7 3D-acceleration data (ms-2), scale: ±6g, resolution: 13-bit
  - 8-10 3D-gyroscope data (rad/s)
  - 11-13 3D-magnetometer data (μT)
  - 14-17 orientation

3 Workflows

- **Dropping rows** with **NaN values** ∼ *time features*
- **Axis feature aggregation** and **feature selection** ∼ *time features*
- **Axis feature aggregation**, **feature selection** and **body parts aggregation** ∼ *time features*
- **Dropping heart rate column** ∼ *time features*
- **Values prediction** ∼ *time features*
- **Time domain** and **frequency domain features extraction** ∼ *time and frequency features*

3.1 Dropping rows with at least one NaN value

1. **Dropping** the **rows** with at least one NaN value: it results in a great reduction of the rows, as well as a possible loss of useful information. The number of rows is dropped down from almost 3.000.000 to nearly 300.000
2. **Creation of train-test sets**: assembly and normalise or not the features
3. **Train results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.7028/0.6378	0.7945/0.7582	0.1298/0.1372
normalised	0.6886/0.6426	0.8519/0.8252	0.0822/0.0797

4. **Test results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.7039/0.6387	0.7950/0.7583	0.1302/0.1382
normalised	0.6863/0.6406	0.8511/0.8240	0.0826/0.0797

3.2 Axis feature aggregation and feature selection

This method is used for the selection of the columns that have a variance greater than a certain threshold. If the variance of a column is very low then it will have a negligible role for the discrimination. In this case, the higher the variance, the better the opportunity to discriminate between different classes. It applies:

1. **Union of dataframes**
2. **Rows with NaNs removal**
3. **Axis aggregation of the same body part**
4. **Feature selection**: features with a variance greater than threshold are selected
5. **Creation** of **train-test** datasets: assembly and normalise or not the features
6. **Train results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.6882/0.6423	0.9023/0.8739	0.0597/0.0672
normalised	0.6882/0.6423	0.9023/0.8739	0.0833/0.1085

7. **Test results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.6868/0.6412	0.9014/0.8722	0.0591/0.0666
normalised	0.6868/0.6412	0.9013/0.8720	0.0822/0.1065

3.3 Axis feature aggregation, feature selection and body parts aggregation

This method not only aggregates the measuraments of different axis of the same body part but it also aggregates some processed features of different body parts

1. **Dropping rows** with **NaN values**
2. **Union of DataFrames**
3. **Axis aggregation** of the **same body part**
4. **Time-features computation**
5. **Body parts weighted accumulation**
6. **Feature selection**: features with a variance greater than threshold are selected
7. **Creation** of **train-test** datasets: assembly and normalise or not the features
8. **Train results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.8476/0.8034	0.9979/0.9979	0.1645/0.1597
normalised	0.8476/0.8034	0.9979/0.9979	0.3130/0.3130

9. **Test results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.8430/0.7968	0.9989/0.9989	0.1525/0.1472
normalised	0.8430/0.7986	0.9524/0.9486	0.3122/0.3066

3.4 Dropping heart-rate column

This method removes the column with the greatest number of NaNs in order to avoid loosing too much data that could be useful for the classification. It looks for a trade-off between the loss of an important column for the discriminative task and avoiding loosing too many rows that might provide important information for the task.

1. **Union of Dataframes**
2. **Dropping heart-rate column**
3. **Dropping rows** with **NaN** values
4. **Creation** of **train-test datasets**: assembly and normalise or not the features
5. **Train results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.7031/0.6381	0.8190/0.7827	0.0720/0.0886
normalised	0.6880/0.6421	0.8529/0.8240	0.1055/0.1190

6. **Test results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.7029/0.6381	0.8197/0.7835	0.0718/0.0884
normalised	0.6877/0.6419	0.8538/0.8249	0.1053/0.1187

3.5 Values prediction

This method tries to avoid loosing too many rows by operating on the column with the greatest number of NaN values: the $heart\_rate$ column, by predicting the missing values with:

- **Nearest neighbour**: looks at the nearest non-NaN value
- **Linear interpolation**: fitting and interpolating a line between the nearest left and right non NaN values
- **Train results**:
  - Nearest neighbour:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.6902/0.6442	0.8455/0.8105	0.0623/0.7801
normalised	0.6903/0.6442	0.8572/0.8339	0.0956/0.1198

  - Linear interpolation;

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.7022/0.6372	0.8214/0.7874	0.0854/0.1072
normalised	0.6902/0.6442	0.8759/0.8508	0.0420/0.0367

- **Test results**:
  - Nearest neighbour

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.6901/0.6440	0.6440/0.8109	0.0621/0.0780
normalised	0.6901/0.6440	0.8565/0.8329	0.0958/0.1201

  - Linear interpolation

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.7019/0.6368	0.8213/0.7873	0.0848/0.1063
normalised	0.6901/0.6441	0.8759/0.8508	0.0425/0.0373

3.6 Time domain and frequency domain features extraction

- **Reduction of the train and test set** for computational reasons. This method relies on many more columns than the previous ones due to the fact that it is a **feature extraction** method. The reduction is performed by keeping the label distribution as equal as possible.
- **Dropping rows** with **NaN** values: this step can be avoided by running *nearest neighbour* and *linear interpolation*. This choice can be validated due to the fact that the data in question is a time series so the missing values could be predicted in this way.
- **Axis aggregation**: the measurements were taken for different body parts: hand, chest and ankle, and for each of them the sensors recordings were split into different axis: x,y and z. The purpose of the aggregation is to better understand the general behaviour of a body part
- **Time features computation**: these are computed with a sliding window of 512 samples. Each sample is recorded every 0.01 seconds. So it relies on a sliding window of 5 seconds of samples.
  - **mean**: it captures the general and overall behaviour of the samples in the window
  - **median**: it is more robust to outliers due to several causes. For example a wrong measurement can be caused by a broken sensor or an unusual behaviour of the person.
  - **standard deviation**: it is helpful for detecting outliers and some unusual behaviour
  - **peak acceleration**: peak acceleration is like a max-pooling. It summarises the most relevant contents of a window.
  - **absolute integral**: most of these time features are computed on the acceleration column, and the integral of the acceleration is the velocity of an object. Thus, it is useful for computing the velocity of the body part or each body part along the different axis in 5 seconds time interval.
- **Removing Direct Current** component: The DC component refers to the zero-frequency component of the signal in the frequency domain. In other words, it is the average value or the offset of the signal in the time domain. In signal processing, DC refers to a signal component that does not vary with time or changes very slowly, such as a constant or a slowly varying offset. In the context of the provided information, the DC component refers to the zero-frequency component of the signal in the frequency domain. Before calculating the power spectral density (PSD) in the frequency domain, the DC component is removed from the signal. This is a common pre-processing step in signal analysis to eliminate the effect of any offset or bias in the signal
- **Frequency-domain features extraction**:
  - **power spectral density (PSD)**: Spectral entropy encodes the spectral density (distribution of power in the frequency domain) into one value based on Shannon entropy. The spectral entropy (SE) of a signal is a measure of its spectral power distribution. The concept is based on the Shannon entropy, or information entropy, in information theory. The SE treats the signal's normalized power distribution in the frequency domain as a probability distribution, and calculates the Shannon entropy of it. The Shannon entropy in this context is the spectral entropy of the signal. This property can be useful for feature extraction in fault detection and diagnosis
  - **peak frequency of the PSD**
  - **power ratio of the frequency bands 0–2.75 Hz and 0–5 Hz**
  - **energy of the frequency band 0–10 Hz**
  - **spectral entropy of the normalized PSD on the frequency band 0-10 Hz**
- **Additional aggregations**: additional data is aggregated along the axis to provide additional information. This is done with: *gyroscope* and *magnetometer*.

- **Heart-rate feature extraction**: From the heart rate data, the features mean and normalized mean (normalization is done on the interval defined by resting and maximum HR) are calculated. The resting HR of a test subject is extracted from the 3 minutes lying task of the data collection protocol, and is defined as the lowest HR value measured over this period. As for the maximum HR, a subject's age-predicted maximum HR (MHR = 220 − age) is used.
- **Train results**:

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.9650/0.9651	0.9805/0.9806	0.3590/0.2665
normalised	0.9685/0.9685	0.9815/0.9816	0.3590/0.2665

- **Test results**;

accuracy/f1-score	**classification tree**	**random forest**	**k-means**
non-normalised	0.6914/0.6356	0.7037/0.6153	0.2660/0.1207
normalised	0.6774/0.6180	0.6966/0.6110	0.2660/0.1207

4 Final considerations and results

4.1 General classification methods comparison

As it is possible to notice, the clustering **k-means** algorithm performs much worse than the **classification tree** and **random forest** due to the fact that **k-means** relies directly on the raw data. With it there is **no-learning**, only **memorization**. The main drawback of kNN is that it is a "lazy" algorithm, all computations is postponed to test time. If it is good for modelling complex data distributions, because it is a non-parametric instance-based algorithm, on the other hand it is only as good as the features and the distance in feature space are. For this reason it can suffer from the **curse of dimensionality**. In this case more than 40 features are used, and applying euclidean distance in such a huge dimensional space, will often result in bad results. On the contrary **classification tree** algorithm performs much better due to the fact that it looks for the feature that allows it to discriminate between classes in a better way. With the help of entropy or gini impurity indices, it can find the feature that allows it to reduce the cahos in labels' distribution. The most performing classification algorithm is **random forest** that uses ensemble techniques that are able to provide boostings in performance due to the fact that it can rely on more classifiers at the same time.

4.2 Pre-processing methods comparison

As it is possible to notice from the results, the worst pre-processing methods are the ones that **drop the rows with NaN values** and the **heart-rate column**. This is due to the fact that the first one just relies on raw time domain features that can be noisy, missing or with errors caused by sensors and the second one, even if it avoids loosing too many rows by dropping the column with the greatest number of NaNs. Nevertheless this is not enough because it gets rid of one of the most important columns for determining the activity class. A little bit better method is the one that **predicts the missing values** and tries to overcome the possible loss of information. Its application is validated by the fact that it is applied to a time series like the *heart_rate* measurements that it is suitable to a missing value prediction of this type. A significant boosting in performance is given by **axis feature aggregation and feature selection method** and by its improved variation **axis feature aggregation, feature selection and body parts aggregation** that also gives an estimate of the overall movement of the entire body and for this reason it can achieve almost 100% in both training and test set with random forest. Finally the best method is **time domain and frequency domain features extraction**, due to the fact that it relies on both time and frequency features. From the performance's results point of view it is not that good in the test: it seems to overfit. For computational capacity reasons (it relies

on many more features in input, more than 5 times of the previous ones), the method was applied on an extremely reduced part of the training set and for this reason it employs all its capacity on a reduced number of patterns. Nevertheless, it can give a significant boost in performance for k-means classifier.