This is the third assignment for the Coursera course "Advanced Machine Learning and Signal Processing"

Just execute all cells one after the other and you are done - just note that in the last one you must update your email address (the one you've used for coursera) and obtain a submission token, you get this from the programming assignment directly on coursera.

Please fill in the sections labelled with "###YOUR_CODE_GOES_HERE###"

This notebook is designed to run in a IBM Watson Studio default runtime (NOT the Watson Studio Apache Spark Runtime as the default runtime with 1 vCPU is free of charge). Therefore, we install Apache Spark in local mode for test purposes only. Please don't use it in production.

In case you are facing issues, please read the following two documents first:

https://github.com/IBM/skillsnetwork/wiki/Environment-Setup

https://github.com/IBM/skillsnetwork/wiki/FAQ

Then, please feel free to ask:

https://coursera.org/learn/machine-learning-big-data-apache-spark/discussions/all

Please make sure to follow the guidelines before asking a question:

https://github.com/IBM/skillsnetwork/wiki/FAQ#im-feeling-lost-and-confused-please-help-me

If running outside Watson Studio, this should work as well. In case you are running in an Apache Spark context outside Watson Studio, please remove the Apache Spark setup in the first notebook cells.

In [1]:
```
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown('# <span style="color:red">'+string+'</span>'))


if ('sc' in locals() or 'sc' in globals()):
    printmd('<<<<<!!!!! It seems that you are running in a IBM Watson Studio Apache Spark Notebook. Please run it in an IBM Watson Studio Default Runtime (without Apache Spark) !!
```

In [2]:
```
!pip install pyspark==2.4.5
```

```
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes i
tead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/util.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting pyspark==2.4.5
  Downloading pyspark-2.4.5.tar.gz (217.8 MB)
     |████████████████████████████████| 217.8 MB 11 kB/s s eta 0:00:01MB/s eta 0:00:02
Collecting py4j==0.10.7
  Downloading py4j-0.10.7-py2.py3-none-any.whl (197 kB)
     |████████████████████████████████| 197 kB 25.2 MB/s eta 0:00:01
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-2.4.5-py2.py3-none-any.whl size=218257927 sha256=db57ccc2818c097f6e3cc35f1b2edf2208a7c7b29394643eaae16d5894db1c89
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/01/c0/03/1c241c9c482b647d4d99412a98a5c7f87472728ad41ae55e1e
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.7 pyspark-2.4.5
```

In [3]:
```
try:
    from pyspark import SparkContext, SparkConf
    from pyspark.sql import SparkSession
except ImportError as e:
    printmd('<<<<<!!!!! Please restart your kernel after installing Apache Spark !!!!!>>>>>')
```

In [4]:
```
sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))

spark = SparkSession \
    .builder \
    .getOrCreate()
```

In [5]:
```
!wget https://github.com/IBM/coursera/raw/master/coursera_ml/a2.parquet
```

```
--2021-04-11 15:35:46--  https://github.com/IBM/coursera/raw/master/coursera_ml/a2.parquet
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/IBM/skillsnetwork/raw/master/coursera_ml/a2.parquet [following]
--2021-04-11 15:35:46--  https://github.com/IBM/skillsnetwork/raw/master/coursera_ml/a2.parquet
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/IBM/skillsnetwork/master/coursera_ml/a2.parquet [following]
--2021-04-11 15:35:46--  https://raw.githubusercontent.com/IBM/skillsnetwork/master/coursera_ml/a2.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 59032 (58K) [application/octet-stream]
Saving to: ‘a2.parquet’

a2.parquet          100%[===================>]  57.65K  --.-KB/s    in 0.002s

2021-04-11 15:35:46 (27.1 MB/s) - ‘a2.parquet’ saved [59032/59032]
```

Now it's time to have a look at the recorded sensor data. You should see data similar to the one exemplified below....

In [6]:
```
df=spark.read.load('a2.parquet')

df.createOrReplaceTempView("df")
spark.sql("SELECT * from df").show()
```

```
+-----+-----------+------------------+------------------+------------------+
|CLASS|   SENSORID|                 X|                 Y|                 Z|
+-----+-----------+------------------+------------------+------------------+
|    0|         26| 380.66434005495194| -139.3470983812975|-247.93697521077704|
|    0|         29| 104.74324299209692| -32.27421440203938|-25.105013725863852|
|    0| 8589934658| 118.11469236129976| 45.916682927433534| -87.97203782706572|
|    0|34359738398| 246.55394030642543|-0.6122810693132044|-398.18662513951506|
```

```
|    0|17179869241|-190.32584900181487|  234.7849657520335|-206.34483804019288|
|    0|25769803830| 178.62396382387422| -47.07529438881511|  84.38310769821979|
|    0|25769803831|  85.03128805189493|-4.3024316644854546|-1.1841857567516714|
|    0|34359738411| 26.786262674736566| -46.33193951911338| 20.880756008396055|
|    0| 8589934592|-16.203752396859194| 51.080957032176954| -96.80526656416971|
|    0|25769803852|   47.2048142440404|  -78.2950899652916| 181.99604091494786|
|    0|34359738369| 15.608872398939273| -79.90322809181754|  69.62150711098005|
|    0|         19|-4.8281721129789315| -67.38050508399905| 221.24876396496404|
|    0|         54| -98.40725712852762|-19.989364074314732|  -302.695196085276|
|    0|17179869313| 22.835845394816594|   17.1633660118843| 32.877914832011385|
|    0|34359738454|  84.20178070080324| -32.81572075916947| -48.63517643958031|
|    0|          0|  56.54732521345129| -7.980106018032676|  95.05162719436447|
|    0|17179869201|  -57.6008655247749|  5.135393798773895| 236.99158698947267|
|    0|17179869308| -65.59264738389012| -48.92660057215126| -61.58970715383383|
|    0|25769803790|  34.82337351291005|  9.483542084393937|  197.6066372962772|
|    0|25769803825|  39.80573823439121|-0.7955236412785212| -79.66652640650325|
+-----+-----------+-------------------+-------------------+-------------------+
only showing top 20 rows
```

Let's check if we have balanced classes – this means that we have roughly the same number of examples for each class we want to predict. This is important for classification but also helpful for clustering

In [7]:
```
spark.sql("SELECT count(class), class from df group by class").show()
```

```
+------------+-----+
|count(class)|class|
+------------+-----+
|        1416|    1|
|        1626|    0|
+------------+-----+
```

Let's create a VectorAssembler which consumes columns X, Y and Z and produces a column "features"

In [8]:
```
from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols=["X","Y","Z"],
                                  outputCol="features")
```

Please insatiate a clustering algorithm from the SparkML package and assign it to the clust variable. Here we don't need to take care of the "CLASS" column since we are in unsupervised learning mode – so let's pretend to not even have the "CLASS" column for now – but it will become very handy later in assessing the clustering performance. PLEASE NOTE – IN REAL-WORLD SCENARIOS THERE IS NO CLASS COLUMN – THEREFORE YOU CAN'T ASSESS CLASSIFICATION PERFORMANCE USING THIS COLUMN

In [9]:
```
from pyspark.ml.clustering import KMeans

clust = KMeans().setK(13).setSeed(1)
```

Let's train...

In [10]:
```
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[vectorAssembler, clust])
model = pipeline.fit(df)
```

...and evaluate...

In [11]:
```
prediction = model.transform(df)
prediction.show()
```

```
+-----+-----------+-------------------+-------------------+-------------------+--------------------+----------+
|CLASS|   SENSORID|                  X|                  Y|                  Z|            features|prediction|
+-----+-----------+-------------------+-------------------+-------------------+--------------------+----------+
|    0|         26| 380.66434005495194| -139.3470983812975|-247.93697521077704|[380.664340054951...|        12|
|    0|         29| 104.74324299209692| -32.27421440203938|-25.105013725863852|[104.743242992096...|        11|
|    0| 8589934658| 118.11469236129976| 45.916682927433534| -87.97203782706572|[118.114692361299...|        11|
|    0|34359738398| 246.55394030642543|-0.6122810693132044|-398.18662513951506|[246.553940306425...|        12|
|    0|17179869241|-190.32584900181487|  234.7849657520335|-206.34483804019288|[-190.32584900181...|         5|
|    0|25769803830| 178.62396382387422| -47.07529438881511|  84.38310769821979|[178.623963823874...|         2|
|    0|25769803831|  85.03128805189493|-4.3024316644854546|-1.1841857567516714|[85.0312880518949...|        11|
|    0|34359738411| 26.786262674736566| -46.33193951911338| 20.880756008396055|[26.7862626747365...|         0|
|    0| 8589934592|-16.203752396859194| 51.080957032176954| -96.80526656416971|[-16.203752396859...|         1|
|    0|25769803852|   47.2048142440404|  -78.2950899652916| 181.99604091494786|[47.2048142440404...|         7|
|    0|34359738369| 15.608872398939273| -79.90322809181754|  69.62150711098005|[15.6088723989392...|         2|
|    0|         19|-4.8281721129789315| -67.38050508399905| 221.24876396496404|[-4.8281721129789...|         7|
|    0|         54| -98.40725712852762|-19.989364074314732|  -302.695196085276|[-98.407257128527...|         8|
|    0|17179869313| 22.835845394816594|   17.1633660118843| 32.877914832011385|[22.8358453948165...|         0|
|    0|34359738454|  84.20178070080324| -32.81572075916947| -48.63517643958031|[84.2017807008032...|        11|
|    0|          0|  56.54732521345129| -7.980106018032676|  95.05162719436447|[56.5473252134512...|         2|
|    0|17179869201|  -57.6008655247749|  5.135393798773895| 236.99158698947267|[-57.600865524774...|        10|
|    0|17179869308| -65.59264738389012| -48.92660057215126| -61.58970715383383|[-65.592647383890...|         0|
|    0|25769803790|  34.82337351291005|  9.483542084393937|  197.6066372962772|[34.8233735129100...|         7|
|    0|25769803825|  39.80573823439121|-0.7955236412785212| -79.66652640650325|[39.8057382343912...|        11|
+-----+-----------+-------------------+-------------------+-------------------+--------------------+----------+
only showing top 20 rows
```

In [12]:
```
prediction.createOrReplaceTempView('prediction')
spark.sql('''
select max(correct)/max(total) as accuracy from (

    select sum(correct) as correct, count(correct) as total from (
        select case when class != prediction then 1 else 0 end as correct from prediction
    )

    union

    select sum(correct) as correct, count(correct) as total from (
        select case when class = prediction then 1 else 0 end as correct from prediction
    )
)
''').rdd.map(lambda row: row.accuracy).collect()[0]
```

Out[12]: 0.9329388560157791

If you reached at least 55% of accuracy you are fine to submit your predictions to the grader. Otherwise please experiment with parameters setting to your clustering algorithm, use a different algorithm or just re-record your data and try to obtain. In case you are stuck, please use the Coursera Discussion Forum. Please note again – in a real-world scenario there is no way in doing this – since there is no class label in your data. Please have a look at this further reading on clustering performance evaluation https://en.wikipedia.org/wiki/Cluster_analysis#Evaluation_and_assessment

In [13]:
```
!rm -f rklib.py
!wget https://raw.githubusercontent.com/IBM/coursera/master/rklib.py
```

```
--2021-04-11 15:37:37--  https://raw.githubusercontent.com/IBM/coursera/master/rklib.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2540 (2.5K) [text/plain]
Saving to: 'rklib.py'
```

```
rklib.py            100%[===================>]   2.48K  --.-KB/s    in 0s

2021-04-11 15:37:37 (23.0 MB/s) - ‘rklib.py’ saved [2540/2540]
```

In [14]:
```
!rm -Rf a2_m3.json
```

In [15]:
```
prediction= prediction.repartition(1)
prediction.write.json('a2_m3.json')
```

In [16]:
```
import zipfile
import os

def zipdir(path, ziph):
    for root, dirs, files in os.walk(path):
        for file in files:
            ziph.write(os.path.join(root, file))

zipf = zipfile.ZipFile('a2_m3.json.zip', 'w', zipfile.ZIP_DEFLATED)
zipdir('a2_m3.json', zipf)
zipf.close()
```

In [17]:
```
!base64 a2_m3.json.zip > a2_m3.json.zip.base64
```

In [18]:
```
from rklib import submit
key = "pPfm62VXEeiJOBL0dhxPkA"
part = "EOTMs"
email = "tjamesbu@gmail.com"
token = "5mH02vg4ssvVUHWD" # (have a Look here if you need more information on how to obtain the token https://youtu.be/GcDo0Rwe06U?t=276)


with open('a2_m3.json.zip.base64', 'r') as myfile:
    data=myfile.read()
submit(email, token, key, part, [part], data)
```

```
Submission successful, please check on the coursera grader page for the status
------------------------
{"elements":[{"itemId":"Cu6KW","id":"f_F-qCtuEei_fRLwaVDk3g~Cu6KW~9vuUvZrbEeuj8gofkbGnOQ","courseId":"f_F-qCtuEei_fRLwaVDk3g"}],"paging":{},"linked":{}}
------------------------
```

In [ ]:
```

```