# Detailed analysis of the code

Initially, we install Kafka python to access Kafka functions from the python environment.

```
[1] !pip install kafka-python

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kafka-python in /usr/local/lib/python3.7/dist-packages (2.0.2)
```

Then we install TensorFlow and TensorFlow-io packages, which are used to create deep learning models.

```
[2] !pip install tensorflow_io==0.17.1

    !pip install tensorflow==2.4.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow io==0.17.1 in /usr/local/lib/python3.7/dist-packages (0.17.1)
```

We use the below command to extract the Kafka file that we have downloaded from canvas.

```
[3] !tar -xzf kafka_2.13-2.7.2.tgz
```

Now, we need to set up both Kafka and Zookeeper instances by using the below commands. Kafka used the localhost:9092 port by default and zookeeper uses port 2181.

```
[4] !./kafka_2.13-2.7.2/bin/zookeeper-server-start.sh -daemon ./kafka_2.13-2.7.2/config/zookeeper.properties
    !./kafka_2.13-2.7.2/bin/kafka-server-start.sh -daemon ./kafka_2.13-2.7.2/config/server.properties
    !echo "Waiting for 10 secs until kafka and zookeeper services are up and running"
    !sleep 10

Waiting for 10 secs until kafka and zookeeper services are up and running
```

Once, the Kafka and zookeeper servers are started, we create the Kafka topics with the below specifications:

- susy-train: partitions=1, replication-factor=1
- susy-test: partitions=2, replication-factor=1

```
[ ] !./kafka_2.13-2.7.2/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 1 --topic susy-train
    !./kafka_2.13-2.7.2/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 2 --topic susy-test

    Created topic susy-train.
    Created topic susy-test.
```

```
[ ] !./kafka_2.13-2.7.2/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic susy-train
    !./kafka_2.13-2.7.2/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic susy-test

    Topic: susy-train       PartitionCount: 1       ReplicationFactor: 1    Configs: segment.bytes=1073741824
            Topic: susy-train       Partition: 0    Leader: 0       Replicas: 0     Isr: 0
    Topic: susy-test        PartitionCount: 2       ReplicationFactor: 1    Configs: segment.bytes=1073741824
            Topic: susy-test        Partition: 0    Leader: 0       Replicas: 0     Isr: 0
            Topic: susy-test        Partition: 1    Leader: 0       Replicas: 0     Isr: 0
```

Now we import all the required libraries needed, mainly the Kafka Producer, pandas, and TensorFlow libraries.

```
[5] import os
    from datetime import datetime
    import time
    import threading
    import json
    from kafka import KafkaProducer
    from kafka.errors import KafkaError
    from sklearn.model_selection import train_test_split
    import pandas as pd
    import tensorflow as tf
    import tensorflow_io as tfio
```

We print the TensorFlow versions.

```
[6] print("tensorflow-io version: {}".format(tfio.__version__))
    print("tensorflow version: {}".format(tf.__version__))

    tensorflow-io version: 0.17.1
    tensorflow version: 2.4.0
```

Here, below are the column names of SUSY data, that we shall use for classification. It consists of 19 attributes.

```
[7] COLUMNS = [
              #  labels
               'class',
              #  low-level features
               'lepton_1_pT',
               'lepton_1_eta',
               'lepton_1_phi',
               'lepton_2_pT',
               'lepton_2_eta',
               'lepton_2_phi',
               'missing_energy_magnitude',
               'missing_energy_phi',
              #  high-level derived features
               'MET_rel',
               'axial_MET',
               'M_R',
               'M_TR_2',
               'R',
               'MT2',
               'S_R',
               'M_Delta_R',
               'dPhi_r_b',
               'cos(theta_r1)'
               ]
```

Now we load the data from the SUSY file into the pandas data structure.

```
[8] susy_iterator = pd.read_csv('SUSY.csv.gz', header=None, names=COLUMNS, chunksize=100000)
    susy_df = next(susy_iterator)
    susy_df.head()
```

| | class | lepton_1_pT | lepton_1_eta | lepton_1_phi | lepton_2_pT | lepton_2_eta | lepton_2_phi | missing_energy_magnitude | missing_energy_phi | MET_rel |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 0.972861 | 0.653855 | 1.176225 | 1.157156 | -1.739873 | -0.874309 | 0.567765 | -0.175000 | 0.810061 |
| 1 | 1.0 | 1.667973 | 0.064191 | -1.225171 | 0.506102 | -0.338939 | 1.672543 | 3.475464 | -1.219136 | 0.012955 |
| 2 | 1.0 | 0.444840 | -0.134298 | -0.709972 | 0.451719 | -1.613871 | -0.768661 | 1.219918 | 0.504026 | 1.831248 |
| 3 | 1.0 | 0.381256 | -0.976145 | 0.693152 | 0.448959 | 0.891753 | -0.677328 | 2.033060 | 1.533041 | 3.046260 |
| 4 | 1.0 | 1.309996 | -0.690089 | -0.676259 | 1.589283 | -0.693326 | 0.622907 | 1.087562 | -0.381742 | 0.589204 |

We print the number of rows and columns of SUSY data.

```
[9] # Number of datapoints and columns
    len(susy_df), len(susy_df.columns)

(100000, 19)
```

```
[10] # Number of datapoints belonging to each class (0: background noise, 1: signal)
     len(susy_df[susy_df["class"]==0]), len(susy_df[susy_df["class"]==1])

(54025, 45975)
```

Now we split the dataset and drop the column that we predict. Now the data is stored in partitions which can later be used by the consumer groups for retrieving data efficiently.

```
# Split the dataset

train_df, test_df = train_test_split(susy_df, test_size=0.4, shuffle=True)
print("Number of training samples: ",len(train_df))
print("Number of testing sample: ",len(test_df))

x_train_df = train_df.drop(["class"], axis=1)
y_train_df = train_df["class"]

x_test_df = test_df.drop(["class"], axis=1)
y_test_df = test_df["class"]

# The labels are set as the kafka message keys so as to store data
# in multiple-partitions. Thus, enabling efficient data retrieval
# using the consumer groups.
x_train = list(filter(None, x_train_df.to_csv(index=False).split("\n")[1:]))
y_train = list(filter(None, y_train_df.to_csv(index=False).split("\n")[1:]))

x_test = list(filter(None, x_test_df.to_csv(index=False).split("\n")[1:]))
y_test = list(filter(None, y_test_df.to_csv(index=False).split("\n")[1:]))

Number of training samples:  60000
Number of testing sample:  40000
```

Now we print the size of the training and test data.

```
[12] NUM_COLUMNS = len(x_train_df.columns)
     len(x_train), len(y_train), len(x_test), len(y_test)

(60000, 60000, 40000, 40000)
```

Now we store the training and testing data in Kafka. Storing the data in Kafka simulates an environment for continuous remote data retrieval for training and inference purposes using the producer.

```
[13] # Store the train and test data in kafka

def error_callback(exc):
    raise Exception('Error while sendig data to kafka: {0}'.format(str(exc)))

def write_to_kafka(topic_name, items):
  count=0
  producer = KafkaProducer(bootstrap_servers=['127.0.0.1:9092'])
  for message, key in items:
    producer.send(topic_name, key=key.encode('utf-8'), value=message.encode('utf-8')).add_errback(error_callback)
    count+=1
  producer.flush()
  print("Wrote {0} messages into topic: {1}".format(count, topic_name))

write_to_kafka("susy-train", zip(x_train, y_train))
write_to_kafka("susy-test", zip(x_test, y_test))
```

```
Wrote 60000 messages into topic: susy-train
Wrote 40000 messages into topic: susy-test
```

Here, we are streaming the data from Kafka into TensorFlow.

```
[14] def decode_kafka_item(item):
  message = tf.io.decode_csv(item.message, [[0.0] for i in range(NUM_COLUMNS)])
  key = tf.strings.to_number(item.key)
  return (message, key)

BATCH_SIZE=64
SHUFFLE_BUFFER_SIZE=64
train_ds = tfio.IODataset.from_kafka('susy-train', partition=0, offset=0)
train_ds = train_ds.shuffle(buffer_size=SHUFFLE_BUFFER_SIZE)
train_ds = train_ds.map(decode_kafka_item)
train_ds = train_ds.batch(BATCH_SIZE)
```

We now build and model to train. We are using adam optimizer here with 10 epochs.

```
# Set the parameters

OPTIMIZER="adam"
LOSS=tf.keras.losses.BinaryCrossentropy(from_logits=True)
METRICS=['accuracy']
EPOCHS=10
```

A less complex neural network has been used with 4 layers, 3 with activation function relu and the last layer with activation function sigmoid, the complexity of the model can be increased by modifying the learning strategy, tuning hyper-parameters, etc.

```
[16]
# design/build the model
model = tf.keras.Sequential([
  tf.keras.layers.Input(shape=(NUM_COLUMNS,)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               2432
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 68,481
Trainable params: 68,481
Non-trainable params: 0
_________________________________________________________________
None
```

The model is now compiled. For losses, binarycorsenthropy is used and metrics are ‘accuracy’.

```
[17]
# compile the model
model.compile(optimizer=OPTIMIZER, loss=LOSS, metrics=METRICS)
```

```
[18]
# fit the model
model.fit(train_ds, epochs=EPOCHS)
```

```
Epoch 1/10
1875/1875 [==============================] - 55s 29ms/step - loss: 0.4953 - accuracy: 0.7570
Epoch 2/10
1875/1875 [==============================] - 52s 27ms/step - loss: 0.4485 - accuracy: 0.7915
Epoch 3/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.4451 - accuracy: 0.7953
Epoch 4/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.4418 - accuracy: 0.7959
Epoch 5/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.4410 - accuracy: 0.7983
Epoch 6/10
1875/1875 [==============================] - 52s 28ms/step - loss: 0.4398 - accuracy: 0.7977
Epoch 7/10
1875/1875 [==============================] - 55s 29ms/step - loss: 0.4389 - accuracy: 0.7981
Epoch 8/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.4392 - accuracy: 0.7978
Epoch 9/10
1875/1875 [==============================] - 54s 28ms/step - loss: 0.4378 - accuracy: 0.7980
Epoch 10/10
1875/1875 [==============================] - 56s 30ms/step - loss: 0.4369 - accuracy: 0.7997
<tensorflow.python.keras.callbacks.History at 0x7f51faf9af50>
```

To infer the test data streaming. The kafkaFroupIODataset function is used. Once all the messages are read from Kafka and the latest offsets are committed, the consumer does not read the messages from the beginning. It is possible to train for a single epoch only with data inputs during the training phase, once a data point is consumed by the model it can be discarded.

We also evaluate the performance of the test data. Because it is an 'exactly-once' semantics, test data cannot be reused, to run inference on test data new consumer groups should be used every time.

```
[19] test_ds = tfio.experimental.streaming.KafkaGroupIODataset(
         topics=["susy-test"],
         group_id="testcg",
         servers="127.0.0.1:9092",
         stream_timeout=10000,
         configuration=[
             "session.timeout.ms=7000",
             "max.poll.interval.ms=8000",
             "auto.offset.reset=earliest"
         ],
     )

     def decode_kafka_test_item(raw_message, raw_key):
       message = tf.io.decode_csv(raw_message, [[0.0] for i in range(NUM_COLUMNS)])
       key = tf.strings.to_number(raw_key)
       return (message, key)

     test_ds = test_ds.map(decode_kafka_test_item)
     test_ds = test_ds.batch(BATCH_SIZE)
```

```
[20] res = model.evaluate(test_ds)
     print("test loss, test acc:", res)

     1250/1250 [==============================] - 22s 17ms/step - loss: 0.4340 - accuracy: 0.7975
     test loss, test acc: [0.43404877185821533, 0.7975249886512756]
```

```
[ ] !./kafka_2.13-2.7.2/bin/kafka-consumer-groups.sh --bootstrap-server 127.0.0.1:9092 --describe --group testcg

    GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                      HOST            CLIENT-ID
    testcg          susy-test       0          21664           21664           0               rdkafka-a8c3c780-04d1-4176-bf74-3df709c2e775 /172.28.0.2     rdkafka
    testcg          susy-test       1          18336           18336           0               rdkafka-a8c3c780-04d1-4176-bf74-3df709c2e775 /172.28.0.2     rdkafka
```

In online learning, the data once consumed by the model may not be available for training again.

When all of the messages are consumed from the topics, the dataset disconnects from Kafka after waiting for 10sec, within these 10 sec time frames if any new data is received, the processes of training and data consumption continues.

```
[21] online_train_ds = tfio.experimental.streaming.KafkaBatchIODataset(
         topics=["susy-train"],
         group_id="cgonline",
         servers="127.0.0.1:9092",
         stream_timeout=10000, # in milliseconds, to block indefinitely, set it to -1.
         configuration=[
             "session.timeout.ms=7000",
             "max.poll.interval.ms=8000",
             "auto.offset.reset=earliest"
         ],
     )
```

The incrementally trained model can be saved in a periodic fashion and can be utilized to infer the test data in either online or offline modes.

```
[22] def decode_kafka_online_item(raw_message, raw_key):
       message = tf.io.decode_csv(raw_message, [[0.0] for i in range(NUM_COLUMNS)])
       key = tf.strings.to_number(raw_key)
       return (message, key)

     for mini_ds in online_train_ds:
       mini_ds = mini_ds.shuffle(buffer_size=32)
       mini_ds = mini_ds.map(decode_kafka_online_item)
       mini_ds = mini_ds.batch(32)
       if len(mini_ds) > 0:
         model.fit(mini_ds, epochs=3)
```

```
Epoch 1/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4444 - accuracy: 0.8008
Epoch 2/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4271 - accuracy: 0.8076
Epoch 3/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4226 - accuracy: 0.8008
Epoch 1/3
32/32 [==============================] - 0s 4ms/step - loss: 0.3886 - accuracy: 0.8320
Epoch 2/3
32/32 [==============================] - 0s 4ms/step - loss: 0.3720 - accuracy: 0.8369
Epoch 3/3
32/32 [==============================] - 0s 4ms/step - loss: 0.3597 - accuracy: 0.8496
Epoch 1/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4461 - accuracy: 0.7910
Epoch 2/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4286 - accuracy: 0.7891
Epoch 3/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4124 - accuracy: 0.8086
Epoch 1/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4405 - accuracy: 0.7920
Epoch 2/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4310 - accuracy: 0.8018
Epoch 3/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4250 - accuracy: 0.7959
```

# How did you execute the task using Kafka, and why is Kafka important in this machine-learning model?

We use the below command to extract the Kafka file that we have downloaded from canvas.

```
[3] !tar -xzf kafka_2.13-2.7.2.tgz
```

Now, we need to set up both Kafka and Zookeeper instances by using the below commands. Kafka used the localhost:9092 port by default and zookeeper uses port 2181.

```
[4] !./kafka_2.13-2.7.2/bin/zookeeper-server-start.sh -daemon ./kafka_2.13-2.7.2/config/zookeeper.properties
    !./kafka_2.13-2.7.2/bin/kafka-server-start.sh -daemon ./kafka_2.13-2.7.2/config/server.properties
    !echo "Waiting for 10 secs until kafka and zookeeper services are up and running"
    !sleep 10
```

```
Waiting for 10 secs until kafka and zookeeper services are up and running
```

Once, the Kafka and zookeeper servers are started, we create the Kafka topics with the below specifications:

- susy-train: partitions=1, replication-factor=1
- susy-test: partitions=2, replication-factor=1

```
[ ] !./kafka_2.13-2.7.2/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 1 --topic susy-train
    !./kafka_2.13-2.7.2/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 2 --topic susy-test

    Created topic susy-train.
    Created topic susy-test.
```

```
[ ] !./kafka_2.13-2.7.2/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic susy-train
    !./kafka_2.13-2.7.2/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic susy-test

    Topic: susy-train       PartitionCount: 1       ReplicationFactor: 1    Configs: segment.bytes=1073741824
            Topic: susy-train       Partition: 0    Leader: 0       Replicas: 0     Isr: 0
    Topic: susy-test        PartitionCount: 2       ReplicationFactor: 1    Configs: segment.bytes=1073741824
            Topic: susy-test        Partition: 0    Leader: 0       Replicas: 0     Isr: 0
            Topic: susy-test        Partition: 1    Leader: 0       Replicas: 0     Isr: 0
```

Kafka allows data from various sources to be written into it, for the above code we have used the SUSY dataset, we have used Kafka and created topics to store training and testing data, to infer the test data streaming and we have used Kafka to stream online. Online learning is different from the traditional training of the models where the model learns stepwise by repeating the process with fixed datasets and the model iterating over the same dataset multiple times. Whereas in online learning, the data once consumed by the model may not be available for training again.

The Kafka ecosystem helps in different ML use cases for model training, model serving, and model monitoring**.** Kafka is a middle layer between the datasets, the environment where the model fits, and the actual application that is used for real-time predictions.

```
[21] online_train_ds = tfio.experimental.streaming.KafkaBatchIODataset(
         topics=["susy-train"],
         group_id="cgonline",
         servers="127.0.0.1:9092",
         stream_timeout=10000, # in milliseconds, to block indefinitely, set it to -1.
         configuration=[
             "session.timeout.ms=7000",
             "max.poll.interval.ms=8000",
             "auto.offset.reset=earliest"
         ],
     )
```

```
[22] def decode_kafka_online_item(raw_message, raw_key):
       message = tf.io.decode_csv(raw_message, [[0.0] for i in range(NUM_COLUMNS)])
       key = tf.strings.to_number(raw_key)
       return (message, key)

     for mini_ds in online_train_ds:
       mini_ds = mini_ds.shuffle(buffer_size=32)
       mini_ds = mini_ds.map(decode_kafka_online_item)
       mini_ds = mini_ds.batch(32)
       if len(mini_ds) > 0:
         model.fit(mini_ds, epochs=3)
```

```
Epoch 1/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4444 - accuracy: 0.8008
Epoch 2/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4271 - accuracy: 0.8076
Epoch 3/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4226 - accuracy: 0.8008
Epoch 1/3
32/32 [==============================] - 0s 4ms/step - loss: 0.3886 - accuracy: 0.8320
Epoch 2/3
32/32 [==============================] - 0s 4ms/step - loss: 0.3720 - accuracy: 0.8369
Epoch 3/3
32/32 [==============================] - 0s 4ms/step - loss: 0.3597 - accuracy: 0.8496
Epoch 1/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4461 - accuracy: 0.7910
Epoch 2/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4286 - accuracy: 0.7891
Epoch 3/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4124 - accuracy: 0.8086
Epoch 1/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4405 - accuracy: 0.7920
Epoch 2/3
32/32 [==============================] - 0s 4ms/step - loss: 0.4310 - accuracy: 0.8018
Epoch 3/3
32/32 [==============================] - 0s 5ms/step - loss: 0.4250 - accuracy: 0.7959
```