Data Streams and Online Machine Learning in Python

Manikandan Jeeva Follow

Jan 19 · 11 min read


Data has a better idea

Photo by Franki Chamaki on Unsplash

Data has become more than a set of binary digits in everyone's day-to-day decision-making processes.

The power of data and the insights derived from it was once limited to large business corporations. **Now, the power of data is available to anyone who is willing to trade their data for a piece of information** (example: one has to turn on the location sensor in the mobile phone to share his/her location data to find the minimal traffic route to a destination).

Data generation has seen exponential growth in the last decade along with the growth in infrastructure to handle it. Every application in our smartphone generates tons of data.

IoT (Internet of things) generates data every second about a state of a mechanical device, Server and application logs along with actual user interaction events like clicks and transaction data only grows over time as the user base grows for the application.

> ***In 2018 for a single minute in a day****—YouTube users watched 4.3 million videos, Amazon shipped 1,111 packages, Twitter had close to half a million tweets, Instagram inducted 50 thousand photos, Google performed 3.8 million searches, and Reddit received 1,944 user comments. An estimate quotes that by 2020 for every person on earth, 1.7 MB of data will be generated for every second.*

It was in 2005, Roger Mougalas from O'Reilly Media coined Big Data as a term. It was used to denote the large amount of data which cannot be managed by the traditional business intelligence tools or data processing applications (mainstream relational). It was in the same year Yahoo! built Hadoop on top of Google's MapReduce with an objective of indexing the entire world wide web. The core concept of distributed computing and the key-value pair data representation (instead of the tabular data representation) has been adopted by most of the new age data integration and business intelligence tools over time.

Any organization which has to deal with massive amount of data (structured, semi-structured, and unstructured) for their business development must have crossed path with one of the tools which had its roots in Hadoop core concepts.

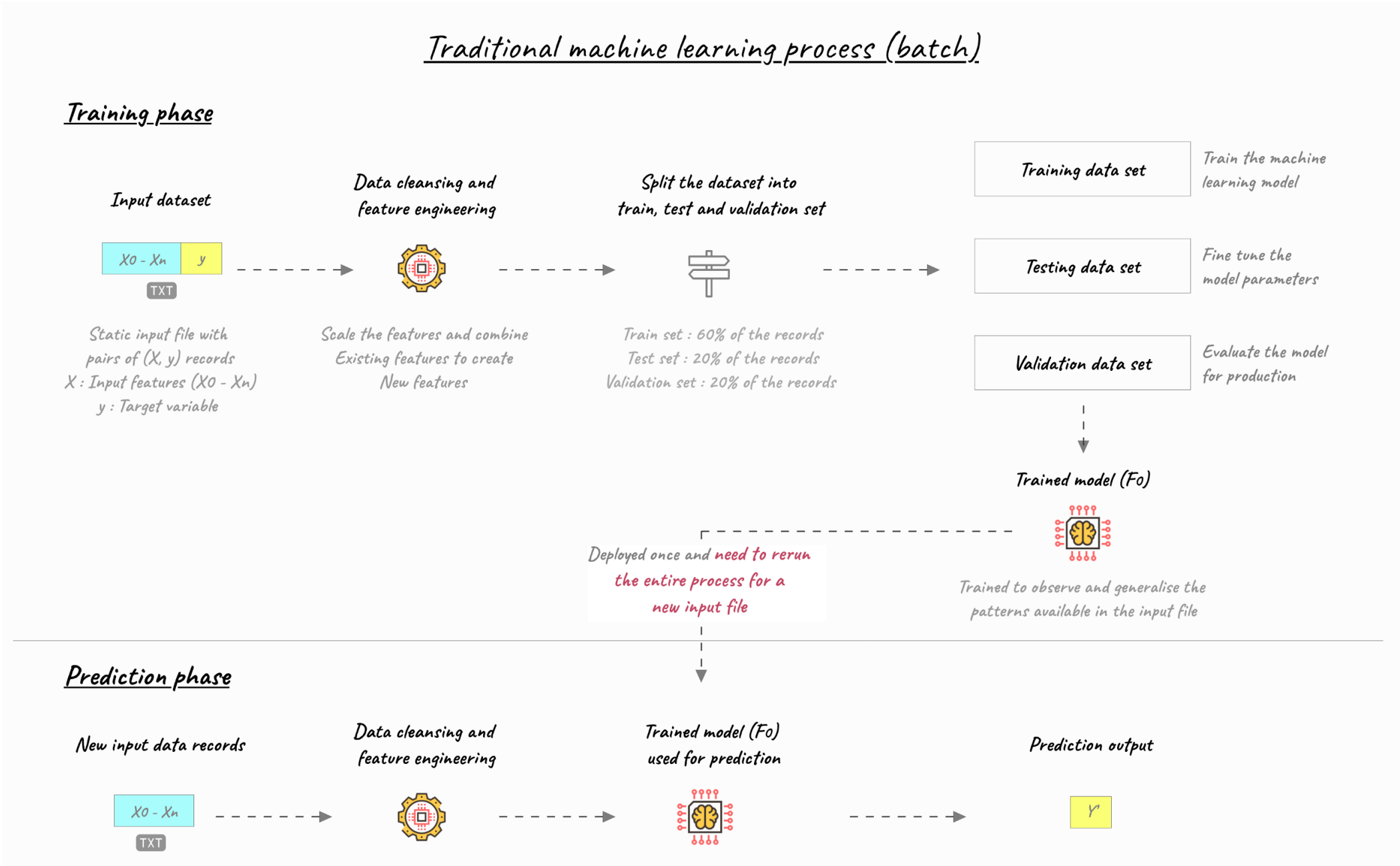
Big Data is an evolving term. It currently describes the large volume of structured or unstructured data that has the potential to be analyzed by various machine learning algorithms for patterns/insights.
Big Data is characterized by

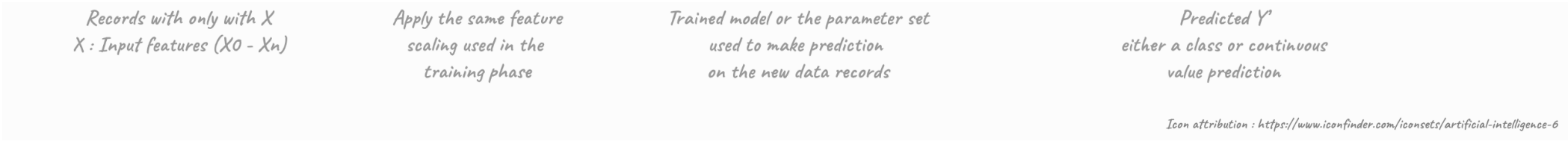
- **Volume** (size of the data)

- **Velocity** (the speed at which data gets generated or shared over a data pipeline. i.e., Climate or traffic-related data has to be real-time without delay)
- **Variety** (the data does not conform to a said layout. i.e., Relational databases expects data inputs in a said layout but cannot work with input data files with varying layout structure)

The infinite amount of cloud storage is available to battle massive data volume, and the combination of natural language processing and natural language understanding makes it possible to counter the data variety along with the NoSQL databases to store the same. **In this article, we will focus on the velocity attribute of the big data—how to handle data streams in python and also how to train a machine learning model using online learning techniques in python to adapt to the incoming data streams.**

Traditional machine learning processes will start with a static input data file.

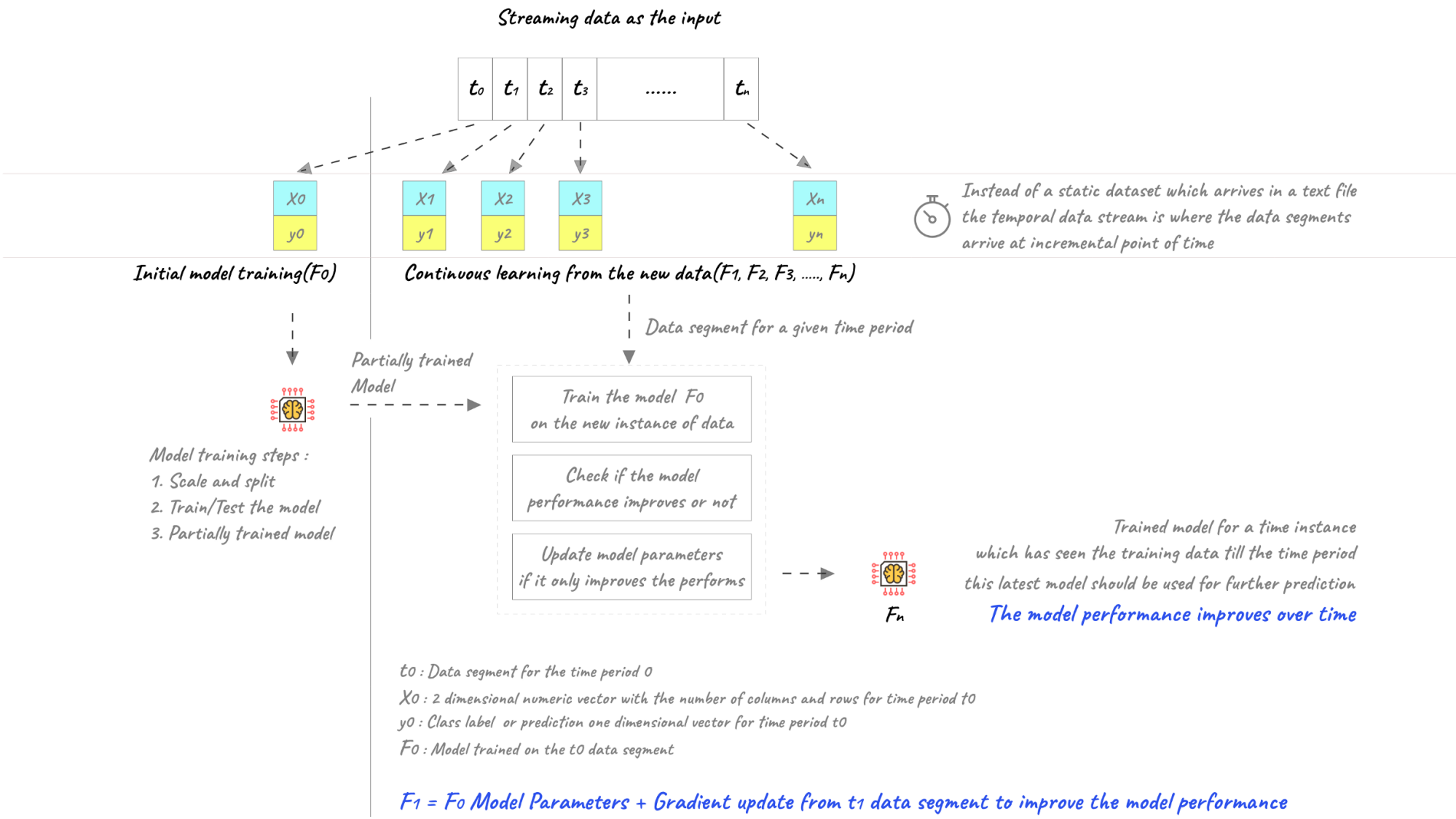

Traditional machine learning process (batch)
Training phase
Input dataset
X0 - Xn
y
TXT
Static input file with pairs of (X, y) records
X : Input features (X0 - Xn)
y : Target variable
Data cleansing and feature engineering
Scale the features and combine Existing features to create New features
Split the dataset into train, test and validation set
Train set : 60% of the records
Test set : 20% of the records
Validation set : 20% of the records
Training data set
Train the machine learning model
Testing data set
Fine tune the model parameters
Validation data set
Evaluate the model for production
Trained model (F0)
Trained to observe and generalise the patterns available in the input file
Deployed once and need to rerun the entire process for a new input file
Prediction phase
New input data records
X0 - Xn
TXT
Data cleansing and feature engineering
Trained model (F0) used for prediction
Prediction output
Y'



Let's take a supervised learning process. **The process starts with receiving a static data file with labels in it as the input file**, perform exploratory data analysis, scale and perform feature engineering, split the data records into train, test and validation set. Train the model using the training data records, fine tune the model parameters using the test data records and perform model selection based on the performance metrics on the validation data records. The well-trained model is then deployed to production to make a prediction/classification on unknown data records. **The model is limited to the patterns it has observed in the static input file and cannot adapt to the real-time behavioural changes.** Every time there is a new training data is made available, the entire process of training the model has to start from scratch. Training a whole model could be a resource-intensive and time-consuming process, which the business applications cannot afford.

Online learning could solve this problem to a greater extent. The more the availability of training data the better the model performance on the new set of records

> ***“It’s not who has the best algorithm that wins; It’s who has the most data.”*** *by Andrew Ng.*


Online machine learning process (simplified flow)
Streaming data as the input
t0 t1 t2 t3 ..... tn
X0 y0
X1 y1
X2 y2
X3 y3
Xn yn
Instead of a static dataset which arrives in a text file the temporal data stream is where the data segments arrive at incremental point of time
Initial model training(F0)
Continuous learning from the new data(F1, F2, F3, ..., Fn)
Data segment for a given time period
Partially trained Model
Train the model F0 on the new instance of data
Check if the model performance improves or not
Update model parameters if it only improves the performs
Model training steps :
1. Scale and split
2. Train/Test the model
3. Partially trained model
Fn
Trained model for a time instance which has seen the training data till the time period this latest model should be used for further prediction
The model performance improves over time
t0 : Data segment for the time period 0
X0 : 2 dimensional numeric vector with the number of columns and rows for time period t0
y0 : Class label or prediction one dimensional vector for time period t0
F0 : Model trained on the t0 data segment
F1 = F0 Model Parameters + Gradient update from t1 data segment to improve the model performance

> *Fn = F(n-1) Model Parameters + Gradient update from t(n-1) data segment to improve the model performance*

Icon attribution : https://www.iconfinder.com/iconsets/artificial-intelligence-6

An incremental/online learning algorithm is one that generates the model based on a given stream of training data *t0, t1, t2, …., tn* a sequence of models *f0, f1, …., fn* trained incrementally. ti is labeled training data *ti = (xi, yi)* and ***fi* is the model dependent on parameters from *f(i-1)* and recent parameter update from data segment ti.**

In other words, an online training algorithm can take the new input data records and learn from it seamlessly without going through the entire training process from scratch. Online training is helpful in the following scenarios.

1. Where the model training and adaption has to happen on the device and the training data is too sensitive (due to data privacy) that it cannot move to an offline environment. i.e., Health devices like smartwatch and applications. Not a single generalized model can handle all possible user behaviour

2. Where the new training data is not transferable to an offline process environment due to the network traffic/barrier or the device has to work in an offline mode completely. i.e., Navigation systems guiding

transport vessels in the sea might not have a stable network connection to load the data from their IoT devices to home office

3. Huge training data sets which cannot fit into the memory of a single machine at a given point of time

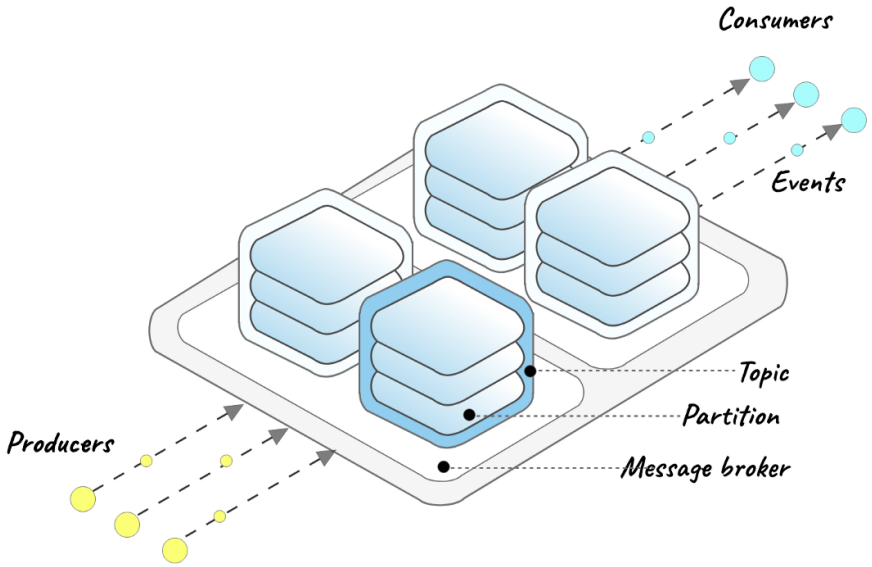
4. Retraining and distribution of the new model across all devices might be a costly endeavour

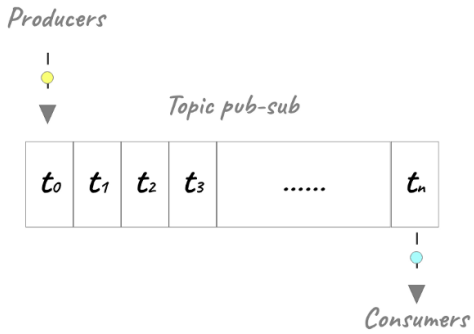
Let's sidetrack a bit to understand what is a data stream and look at a brief overview of Kafka.

A data stream is where the data is available instantly as and when an event occurs. It can be real-time or near real-time. Instead of collecting and storing data from events overtime and then shipping the file to an application, the data is available to the application as and when the event occurs. i.e., a credit card transaction has to be instant —the available credit has to be updated instantly after a transaction. It cannot wait for an overnight batch rerun as the available credit displayed to the consumer might not be accurate and leads to a lot of confusion due to transaction duplication.

Data streams Kafka with an example

Kafka distributed system





Youtube example for Kafka streams

Kafka is a distributed publish-subscribe messaging system (incubated at Linkedin). It is used to handle high-throughput messaging applications like clicks on web pages, log aggregations, and stream processing. Three important keywords in Kafka will be a producer (publisher), consumer (subscribers), and topic. **The system is comparable to an infinite magnetic tape spool. An application can write to Kafka real-time as and when the events occur (publish messages to stream), and another application can read from the tape from a given offset (consume messages from the stream).**

Multiple data streams can co-exist in Kafka messaging environment they are distinguished between each other by their topic name. i.e., Youtube will be the best example. Viewers can subscribe to a channel (topic). Channel owner (producer) can push new content (event) to a channel. Subscribers (consumers) receives the notification of new content. A channel owner can run multiple channels. A channel can have a lot of subscribers. A subscriber can subscribe to various channels.

RabbitMQ, Google PubSub, Amazon Kinesis Stream, Wallaroo, and Redis Streams are few alternatives to Kafka. In any distributed

system there has to be a process which needs to take the responsibility of task coordination, state-management and configuration management and Kafka depend upon zookeeper to perform these necessary tasks. Download the Kafka tgz file from https://kafka.apache.org/downloads and unzip the same into a folder.

Run the following two commands after moving into the extracted Kafka folder (run this in two separate command windows or terminal)

```
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

Now with both zookeeper and Kafka servers up and running, we need a way to connect the Kafka server through python. We will use Kafka-python package. run the following command to install the package

```
pip install kafka-python
```

After the package installation download the codes from Github

```
01.Blog_04_KafkaProducer_Code_V1.0.py
01.Blog_04_KafkaConsumer_Code_V2.0.py
```

The producer code is the one which generates a clickstream classification data (ten dependent variable and one target variable) and publishes the data segments into a specific topic in the message broker.

In the code make sure to update the Kafka folder path variable and install dependencies like sklearn.

```
def create_topic(logger=None, kafka_path=None, topic=None):

     ‘’’ Routine will create a new topic; assuming
delete_all_topics
         will run before this routine ‘’’;

     cmd_string = f’{kafka_path}bin/kafka-topics.sh — create
—
                    zookeeper localhost:2181 — replication-
factor 1
                    — partitions 1 — topic {topic.lower()}’;

     cmd_status = subprocess.check_output(cmd_string,
                   stderr=subprocess.STDOUT, shell=True);


     cmd_output = cmd_status.decode(‘utf-8’).split(‘\n’)[0];

     logger.info(f’’);
     logger.info(f’{cmd_output}’);
     logger.info(f’’);

return None;
```

The above function takes the kafka_path and topic_name as input and creates a topic in the Kafka system. The subprocess module is used to submit the command in the command line and returns the response. The code also checks whether an active zookeeper process is available before creating the topic.

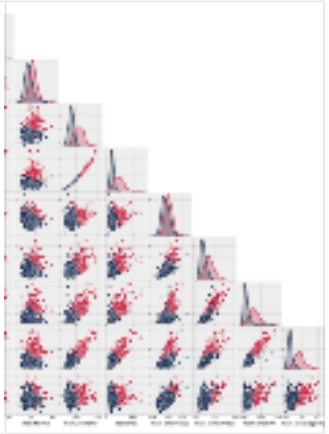
```
def run_producer(logger=None, topic=None):

    ''' Run a producer to put messages into a topic ''';

    # The function simulates the clickstream data with pass
through
    # X0 - X9   : (Dependent) Numerical features which
detail the
                  attributes of an advertisement
    # y         : (Target) Whether the advertisement
resulted in a
                  user click or not
    # for every instance a random choice is made to generate
the
                  number of records

    producer = KafkaProducer(bootstrap_servers=
['localhost:9092'],
               value_serializer=lambda x:
dumps(x).encode('utf-8'));

    logger.info(f'Publishing messages to the topic :
{topic}');

    random_choice_seq = [ i for i in range(100,110) ];
```

```
    record_count = 0;

    for i in range(10000):
        number_of_records =
random.choice(random_choice_seq);
        record_count += number_of_records;
        X, y = generate_sample(logger=logger,
                number_of_records=number_of_records);

        if i == 0 :
            X_scalar = MinMaxScaler();
            X_scalar.fit(X);
            X = X_scalar.transform(X);
        else:
            X = X_scalar.transform(X);

        data = {'X' : X.tolist(), 'y' : y.tolist()};

        producer.send(f'{topic}', value=data);

        sleep(0.05);

    logger.info(f'Closing producer process;
                Total records generated is
                {format(record_count, "09,")}');

    return None;
```

KafkaProducer function is used to connect to the Kafka server running on port number 9092 and publish messages to a specific topic. In the above function, we use the scikit-learn toy data generation module to generate data samples for the classification problem. We will publish close to 10 thousand messages with 100 to 110 data records in each

message. At the end of the procedure code, the stream will have ~1M data records and 10,000 messages.

Check the below blog-post to learn more about the scikit-learn module and the series of functions used to build a basic machine learning process pipeline



Wait for the producer code to complete as the consumer code uses the message offset to close the training loop.

The consumer code contains the module to consume the messages from the beginning of the topic until the last published message. (it is necessary to execute and complete the producer code before submitting the consumer code else the consumer will not read the entire topic. it will probably read only a partial set of messages) and also the online model training

Getting back to online learning we need a learning algorithm which can take a new set of data records and update the trained model parameters to improve the model performance metrics.

1. Scikit-learn contains few classifier models with the parital_fit method. The parital_fit allows fitting a trained model to a smaller subset of new data instead of the entire training dataset

2. Warm start a Neural Nets. For new training iteration, the weights can come from the previously trained models. TensorFlow has checkpoints to handle this case

3. Bayesian methods will be very well suited for online learning since the belief about the parameter distribution is updated as the new training data records come into perspective

We will use the Scikit-learn SGDclassfier with the partial fit.

Cost function, gradient and gradient update

Cost function :

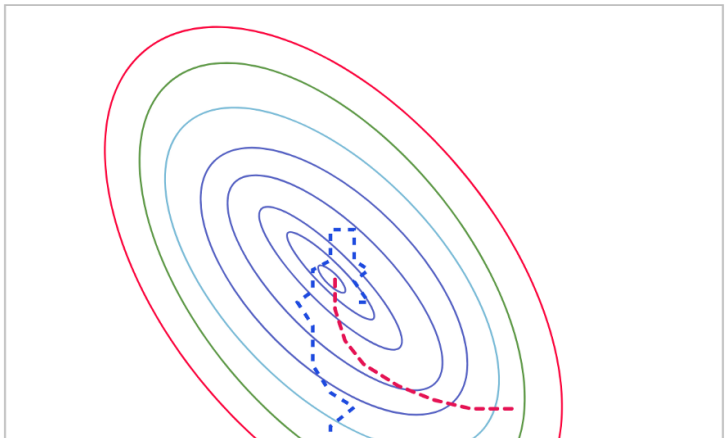
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(f(\theta)^{(i)} - y^{(i)}\right)^2$$

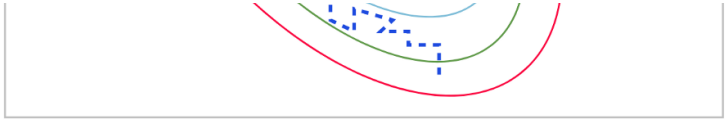
Gradient :

$$\frac{d\,J(\theta)}{d\theta j} = \frac{1}{m}\sum_{i=1}^{m}(f(\theta)^{(i)} - y^{(i)})\,.\,X_j^{(i)}$$

Gradient update:

$$\theta i = \theta i - \alpha\,.\left[\frac{1}{m}\sum_{i=1}^{m}(f(\theta)^{(i)} - y^{(i)})\,.\,X_j^{(i)}\right]$$





Red stripped line is the batch learning and blue stripped is the stochastic gradient descent; SGD might stagger around but eventually reaches the global minimum

Since the cost function is convex for SGD, a gradient can be used to reach the global minimum. The complete batch training might reach the global minimum smoothly versus the smaller set of input data records, which will stagger around and eventually converge to the global minimum.

The model parameters are updated based on the gradient from the new set of training data and also the alpha which controls the learning rate (how much of information from the new training set should be used to influence the trained model parameters)

```
def initial_model(logger=None):

        ''' Simulation of the initial model setup in a
traditional
            ML training ''';
```

```
        clf = SGDClassifier(

              loss='log',
              # log as a loss gives the logistic regression
```

```
                penalty='none',
                # l2 as a default for the linear SVM;
                fit_intercept=True,
                shuffle=True,
                # shuffle after each epoch
                eta0=0.001,
                learning_rate='constant',
                average=False,
                # computes the averaged SGD weights
                random_state=1623,
                verbose=0,
                max_iter=1000,
                warm_start=False
               );

          return clf;
```

The consumer code will use KafkaConsumer to connect to an active Kafka server and start consuming the messages from the beginning of the topic pipeline. The numpy data element is not serializable, so it is converted into a python list in the producer code and dumped into a JSON. The consumer will reverse the process to get the numpy array back. The scikit-learn parital_fit method plays well with numpy array/pandas data object.

```
consumer = KafkaConsumer(
               topic_name,
               bootstrap_servers=['localhost:9092'],
               auto_offset_reset='earliest',
               enable_auto_commit=True,
               value_deserializer=lambda x:
loads(x.decode('utf-8'))
               );
```

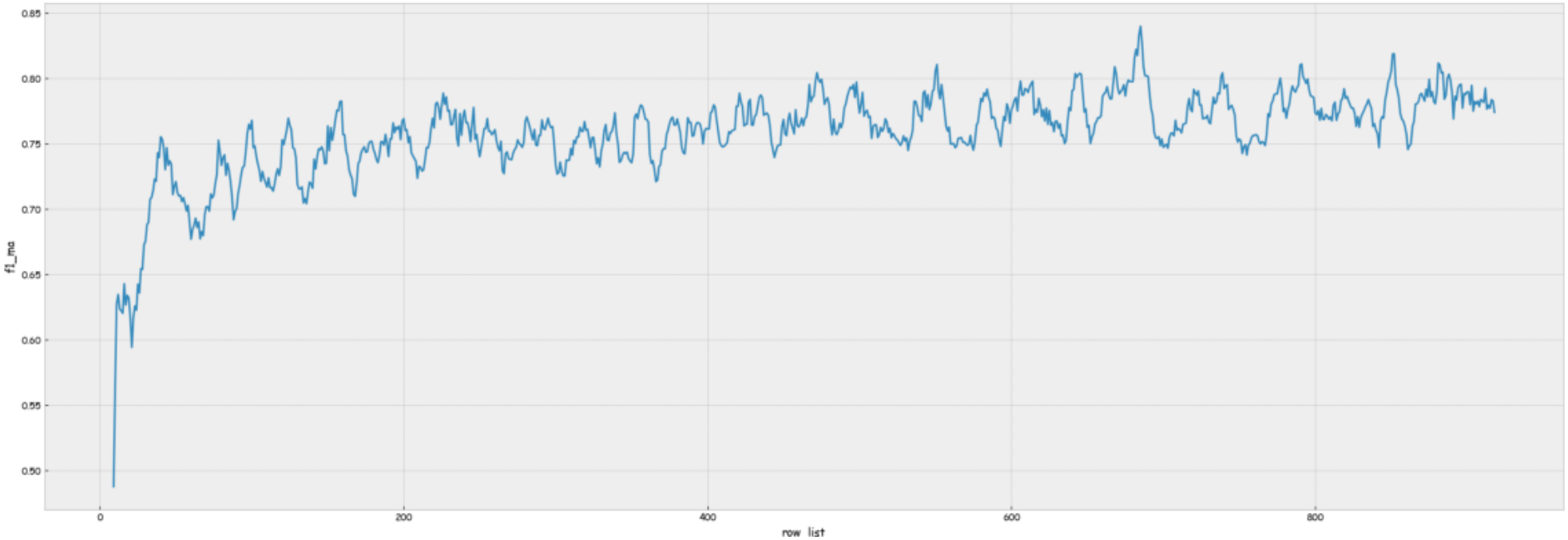
For the very first iteration, the model uses the entire data records from the message for training and testing. From the second iteration, we will only fit if the model performance improves.

```
if counter == 1:

            clf.partial_fit(X_train, y_train, classes=
[0,1]);
            y_test_predict = clf.predict(X_test);

            clf_log_loss = log_loss(y_test, y_test_predict,
                           labels=[0,1]);
            clf_acc_score = accuracy_score(y_test,
y_test_predict);
            clf_f1_score = f1_score(y_test, y_test_predict);

            row_list.append(selected_models);
            ll_list.append(clf_log_loss);
            accuracy_list.append(clf_acc_score);
            f1_list.append(clf_f1_score);

else:


            clf_temp = clf;

            clf_temp.partial_fit(X_train, y_train, classes=
[0,1]);
            y_test_predict = clf_temp.predict(X_test);
```

```
            clf_log_loss = log_loss(y_test, y_test_predict,
                           labels=[0,1]);
            clf_acc_score = accuracy_score(y_test,
y_test_predict);
            clf_f1_score = f1_score(y_test, y_test_predict);

            if clf_f1_score > (np.mean(f1_list) * 0.95) :

                clf = clf_temp;
                selected_models += 1;

                row_list.append(selected_models);
                ll_list.append(clf_log_loss);
                accuracy_list.append(clf_acc_score);
                f1_list.append(clf_f1_score);

counter += 1;

if counter == topic_offset:
             break;
```

The performance metrics were averaged over time, so the plots look smooth.

Model evaluation and results



F1 score for the classification task; The metric improves as the more number of data sample it passes through

We assumed data to be stationary, and the non-stationary will require alteration in the learning rate, model parameter averaging and the way we qualify the model for the learning task.

Happy coding and keep learning

> *Clap if you like this article and you can reach me on manikandan@datazymes.com for any further queries or suggestions*

Link to the *code* repo