**Project for real-time anomaly detection using kafka and python**

We assumed that zookeeper and kafka are running in the localhost, it follows this process:

STEP 1: Train an unsupervised machine learning model for anomalies detection task

STEP 2: Pickle/Save the model to be used in real-time predictions

STEP 3: Generate fake streaming data and send it to a kafka topic

STEP 4: Read the topic data with several subscribers to be analyzed by the model

STEP 5: Predict if the data is an anomaly, if so, send the data to another kafka topic

STEP 6: Subscribe a slack bot to the last topic to send a message in slack channel if an anomaly arrives

# Usage:

**First train the anomaly detection model, run the file:**

```bash

model/train.py

```



**Create the required topics**

```bash

kafka-topics.sh --zookeeper localhost:2181 --topic transactions --create --partitions 3 --replication-factor 1

kafka-topics.sh --zookeeper localhost:2181 --topic anomalies --create --partitions 3 --replication-factor 1

```

**Check the topics are created**

```bash

kafka-topics.sh --zookeeper localhost:2181 --list

```

\* Check file \*\*settings.py\*\* and edit the variables if needed

\* Start the producer, run the file

```bash

streaming/producer.py

```

\* Start the anomalies detector, run the file

``bash

streaming/anomalies\_detector.py

```

\* Start sending alerts to Slack, make sure to register the env variable SLACK\_API\_TOKEN,

then run

```bash

streaming/bot\_alerts.py

```

The idea is to:

* Train an anomaly detection algorithm using unsupervised machine learning.
* Create a new data producer that sends the transactions to a Kafka topic.
* Read the data from the Kafka topic to make the prediction using the trained ml model.
* If the model detects that the transaction is not an inlier, send it to another Kafka topic.
* Create the last consumer that reads the anomalies and sends an alert to a Slack channel.

The article assumes that you know the basics of Apache Kafka, machine learning, and Python.

The transactions could represent any relevant information to analyze in real-time and predict if there could be something out of the ordinary, such as credit card transactions, GPS logs, system consumption metrics, etc.

**1. STEPWISE PROJECT STRUCTURE: METHODOLOGY**

Our project structure has four code bases. Check the complete codebases in the zipped folder.

First, check the settings.py; it has some variables we need to set, like the Kafka broker host and port; you can leave the ones by default (listening on localhost and default ports of Kafka and zookeeper).

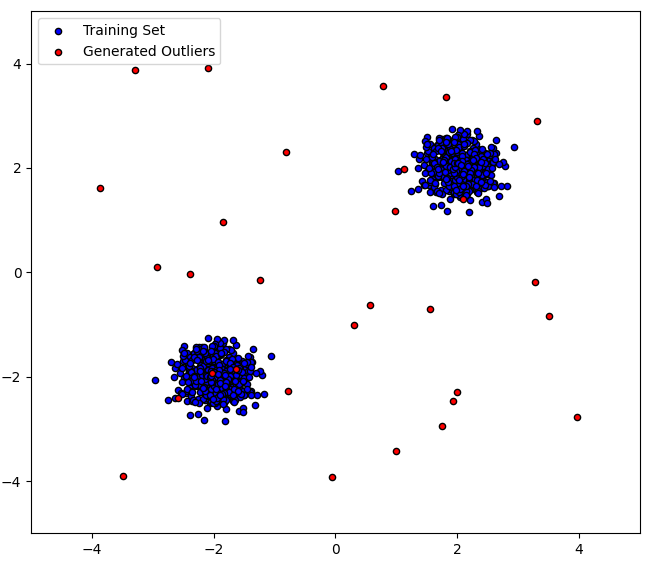
The streaming/utils.py file contains the configurations to create Kafka consumers and producers; it has some default options that you can also change if needed.

Now install the requirements:

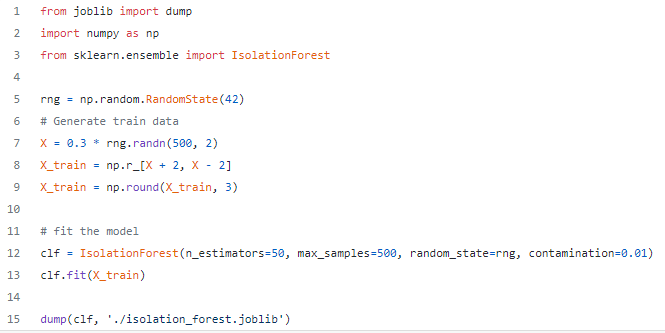
pip install -r requirements.txt

**2. Train the model**

To illustrate how to set up this solution, we are going to generate random data; it will have two variables, they look like this:



Next, we are going to use an Isolation Forest model to detect the outliers; in simple words, this model will try to isolate the data points by tracing random lines over one of the (sampled) variables' axes and, after several iterations, measure how "hard" was to isolate each observation, so in the train.py file we have:



After running this, the isolation\_forest.joblib file should be created; this is the trained model.

**3. Create the Topics**

We will use two topics; the first one is called "transactions," where the producer will send new transaction records. Let's create it from the terminal with:

kafka-topics.sh --zookeeper localhost:2181 --topic transactions --create --partitions 3 --replication-factor 1

The second topic is going to be called "anomalies," here is where the module that detects anomalies will send the data, and the last consumer will read it to send a slack notification:

kafka-topics.sh --zookeeper localhost:2181 --topic anomalies --create --partitions 3 --replication-factor 1

**4. Transactions Producer:**

Now we are going to generate the first producer that will send new data to the Kafka topic "transactions"; we'll use the confluent-Kafka package; in the file streaming/producer.py, we have:

With this code, a producer will send data to a Kafka topic, with a probability of OUTLIERS\_GENERATION\_PROBABILITY; the data will come from an "outlier generator," will send an auto-incremental id, the data needed for the machine learning model and the current time in UTC.

Let's check everything is correct so far, run the producer.py file, and log to the topic as a consumer from the terminal:

kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic transactions

You should see the incoming messages like this:

**5. Outlier Detector Consumer:**

The data is coming. Now we must read it from a consumer, pass it to the machine learning model to make the predictions, and filter the outliers. This is done in the streaming/anomalies\_detector.py file that looks like this:

The consumer read messages from the "transactions" topic and a consumer sent outliers to the "anomalies" topic; besides the data we already had, it will enrich the record with the score given by the model, a measure of "how much" the data is considered an outlier.

Notice that the only messages that go to the new topic are those whose prediction output is -1; this is how this model categorizes that the data is an inlier.

Also, notice that the topic has three partitions, so at the end, I'm using multiprocessing to simulate three independent consumers and speed up the process; they all belong to the same group\_id. In production probably those consumers will run on different servers.

Let's check this step, make sure the producer is running and run the anomalies\_detector.py file, now in a terminal, let's open the anomalies topic, we should see the incoming transactions that the model predicted as outlier, it should look like this:

Anomalies detection. Gif by the AUthor.

Here there is a visualization of how the transactions producer and outlier detection run simultaneously; the top window is the transaction producer topic, and the bottom one is the outliers sent to the anomalies topic.

Real-time anomalies detection. Gif by the Author.

**6. Slack notification:**

As the last step, we want to take some actions with these detected outliers; in a real-life scenario, it could block a transaction, scale a server, generate a recommendation, send an alert to an administrative user, etc.

Here we are going to send an alert to a Slack channel; for this, make sure to create a slack app, add the app to the slack channel and register an environment variable called SLACK\_API\_TOKEN to authenticate Slack.