Realtime Anomaly Detection for Content Delivery Proxy Logs

Inuwa Amir Usman, Aizaz Ali Qureshi, Meskine Mouloud, Tosayeva Arzu

Eotvos Lorand University,Budapest,Pázmány Péter stny. 1/C, 1117

Abstract. A content delivery network (CDN) refers to a geographically distributed group of servers which work together to provide fast delivery of Internet content. A CDN allows for the quick transfer of assets needed for loading Internet content including HTML pages, javascript files, stylesheets, images, and videos. The popularity of CDN services continues to grow, and today the majority of web traffic is served through CDNs, including traffic from major sites like Facebook, Netflix, and Amazon.

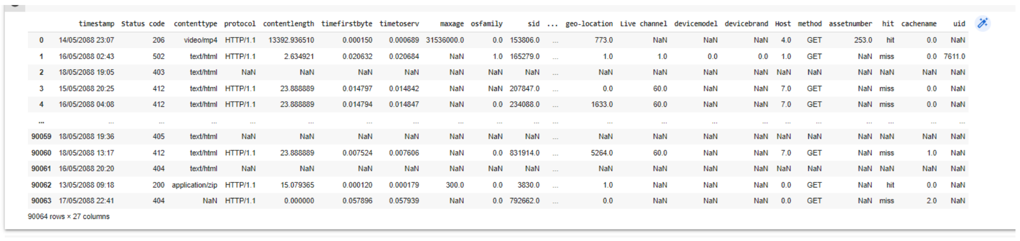
Keywords: Anomaly Detection · Real-Time · Content Delivery Networks.

1 Introduction

1.1 Aims and Objectives

The main aim of this project is to identify anomalies using unsupervised machine learning algorithms in content delivery networks from real life-CDN proxy logs. We have started from understanding the dataset and reading related papers, choosing proper unsupervised algorithms to apply, using variety of open source technologies, identifying anomalies from real-life proxy logs. AT the end, we built an anomaly detection system.

Data Preprocessing Only two levels of This dataset has 27 different features with both numerical and categorical data.In order to process data, we have first handled its null values for both categorical and numerical values. In order to handle with them, we have filled nan with their mode values.



	timestamp	Status code	contenttype	protocol	contentlength	timefirstbyte	timetoserv	maxage	osfamily	sid	...	geo-location	Live channel	devicemodel	devicebrand	Host	method	assetnumber	hit	cachename	uid
0	14/05/2088 23:07	206	video/mp4	HTTP/1.1	13392.936510	0.000150	0.000689	31536000.0	0.0	153806.0	...	773.0	NaN	NaN	NaN	4.0	GET	253.0	hit	0.0	NaN
1	16/05/2088 02:43	502	text/html	HTTP/1.1	2.634921	0.020632	0.020684	NaN	1.0	165279.0	...	1.0	1.0	0.0	0.0	1.0	GET	NaN	miss	0.0	7611.0
2	18/05/2088 19:05	403	text/html	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	15/05/2088 20:25	412	text/html	HTTP/1.1	23.888889	0.014797	0.014842	NaN	NaN	207847.0	...	0.0	60.0	NaN	NaN	7.0	GET	NaN	miss	0.0	NaN
4	16/05/2088 04:08	412	text/html	HTTP/1.1	23.888889	0.014794	0.014847	NaN	0.0	234088.0	...	1633.0	60.0	NaN	NaN	7.0	GET	NaN	miss	0.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
90059	18/05/2088 19:36	405	text/html	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
90060	18/05/2088 13:17	412	text/html	HTTP/1.1	23.888889	0.007524	0.007606	NaN	0.0	831914.0	...	5264.0	60.0	NaN	NaN	7.0	GET	NaN	miss	1.0	NaN
90061	16/05/2088 20:20	404	text/html	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
90062	13/05/2088 09:18	200	application/zip	HTTP/1.1	15.079365	0.000120	0.000179	300.0	0.0	3830.0	...	1.0	NaN	NaN	NaN	0.0	GET	NaN	hit	0.0	NaN
90063	17/05/2088 22:41	404	NaN	HTTP/1.1	0.000000	0.057896	0.057939	NaN	0.0	792662.0	...	0.0	NaN	NaN	NaN	0.0	GET	NaN	miss	2.0	NaN

90064 rows × 27 columns

Fig. 1. All features coming with the dataset.

Categorical Features For categorical features we used one-hot-encoder. The one-hot encoding technique was used to transform the categorical features into numerical values.

Feature Selection For categorical features we used one-hot-encoder. The one-hot encoding technique was used to transform the categorical features into numerical values.

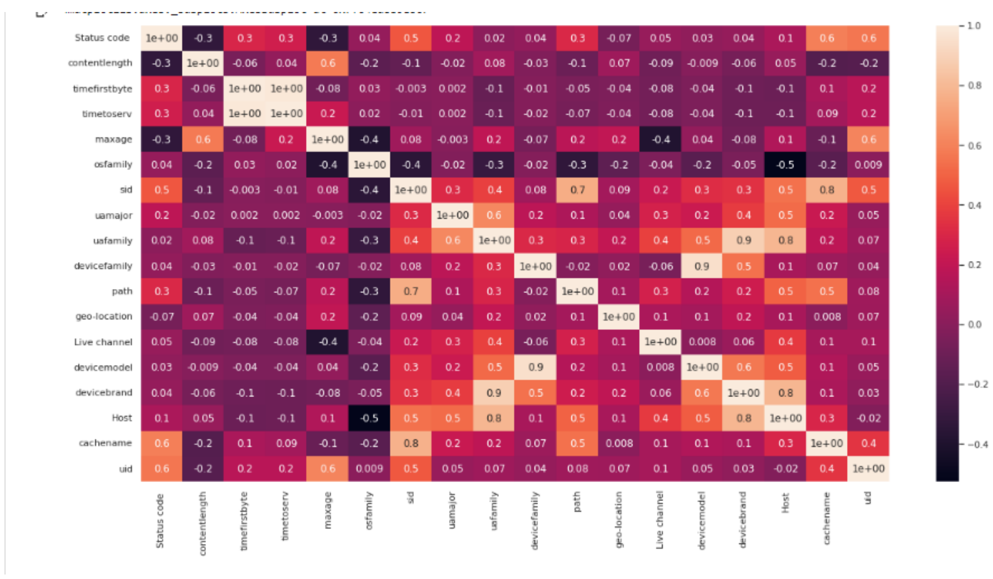


Fig. 2. Correlation for Features

As a result we got 14 features which we then used for different implementations.

2 Unsupervised Algorithm Implementation

We implemented LSTM Autoencoder in terms of detecting anomalies. LSTMs can provide better results than parametric models and standard RNNs when dealing with complex autocorrelation sequences. Train an LSTM autoencoder on data from 2088-05-13 to 2088-05-19 .We assume that there were no anomalies and they were normal.Using the LSTM autoencoder to reconstruct the error on the test data from 2088-05-13 to 2088-05-19. If the reconstruction error for the test data is above the threshold, we label the data point as an anomaly. We used standard scaler and created sequences in terms of time steps. We want our network to have memory of 30 days, so we set TIME STEPS=30.Dropout value,learning rate,number of epochs, batch size have been used. After all we found MAE loss on the training data and make loss value in the training data as the reconstruction error threshold. If the reconstruction loss for a data point in the test set is greater than this threshold value we label this as anomaly.

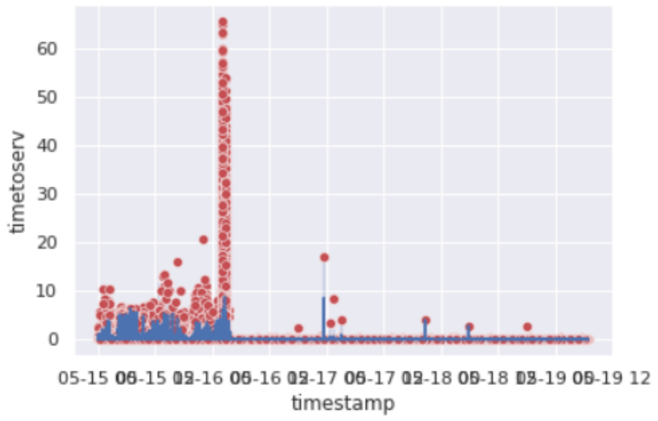


Fig. 3. Anomalies for the timetoserv feature

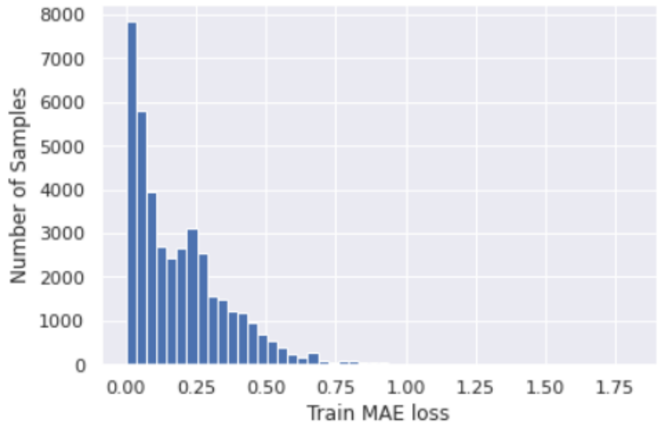


Fig. 4. LSTM Train Loss

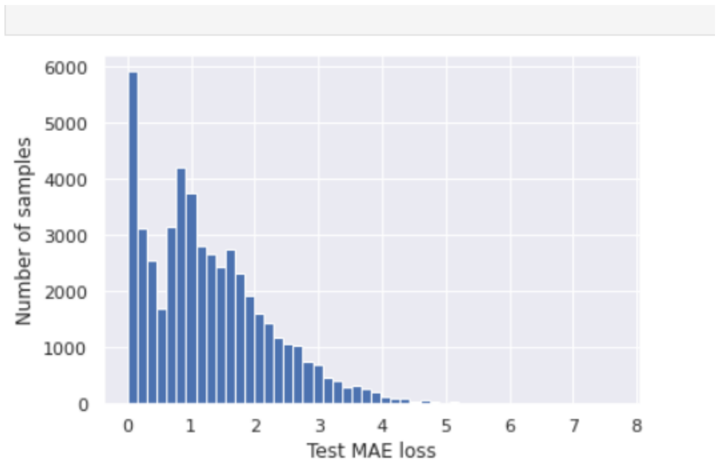


Fig. 5. LSTM Test Loss

3 Architecture

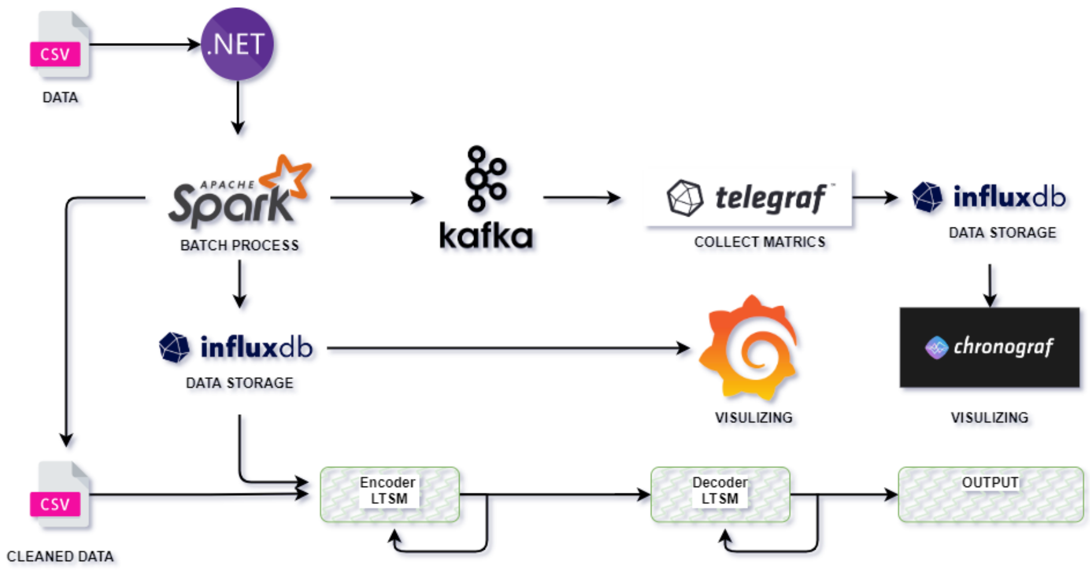


Fig. 6. Architecture

4 Batch processing with .NET for Apache Spark

Batch processing is the transformation of data at rest, meaning that the source data has already been loaded into data storage, Batch processing is generally performed over large, flat datasets that need to be prepared for further analysis. Log processing and data warehousing are common batch-processing scenarios. In this scenario, we must see how close the server we placed, and finds the anomaly in the data the three main key features @timeStamp and, Time for the first Byte, Time to Server. First we have done: 1.Creating and run a .NET for Apache Spark application, 2.Reading data into a DataFrame and prepare it for analysis, 3.Processing the data using Spark SQL.

4.1 Preparing Data

Preparing the data sample for implementing batch processing operations for Apache Spark Batch Processing. Initially, we must change the input file into a Data Frame, which is a distributed collection of data structured into named columns. Then, define the columns of your data through the Schema parameter. Before describing the schema, we select the delimiter and select the features that are most relevant to find any abnormalities in data e.g., TIMESTAMP, CONTENTYPE, TIME TO FIRST BYTE, TIME TO SEVER, MAXAGE, GEOLOCATION, ETC. Further, to delete rows with NA (null) values, use the Na method, and to remove specific columns from your data, use the Drop method. This helps you avoid errors when attempting to examine null data or columns that are irrelevant for the final uploading or pipeline with the database.

4.2 Data to InfluxDB

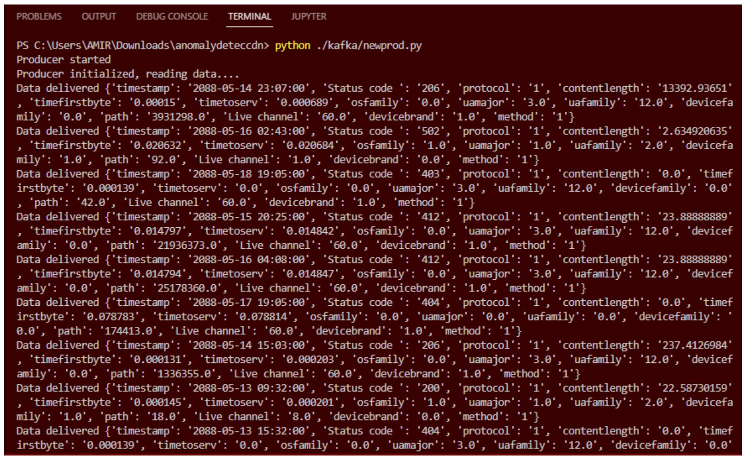
For analyzing the cleaned data after the batch processing, we create a Python Program that changes the time stamp help of pandas as pd, and for uploading to influx dB we can import influx dB DataFrameClient but must mention the following environment that connects the localhost:8086 and dump the data into influx dB bucket project1db.

4.3 Fields and Data Tags

The selection of field values in metadata is the selection of fields same as batch process data but must configure the data tags and fixed tags in python, Besides the timestamp field, each data element consists of various tags (optional, indexed metadata fields), fields (key and value), and measurement (container for tags, fields, and timestamps).

5 Apache Kafka

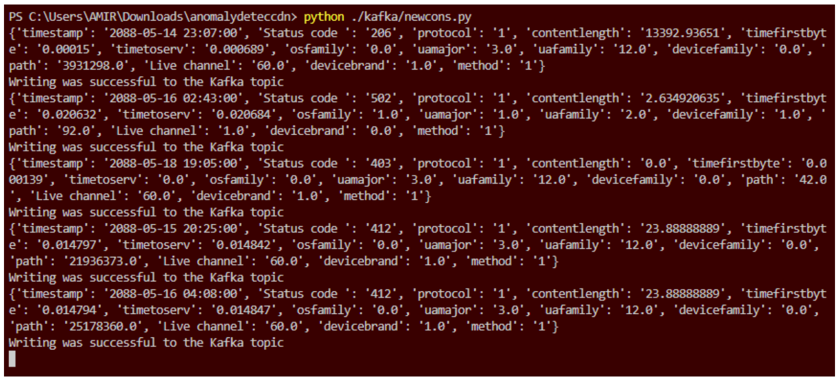
A Kafka Console Producer (kafka-console-producer) is one of the utilities that comes with Kafka packages. It is used to write data to a Kafka Topic using standard input or the command line. When we type anything into the console, kafka-console-producer writes it to the cluster. Topics are made up of Partitions where the data is written by the Producers. As we can see from the figure7 Producer running and sending to a kafka topic waiting to be consumed.



```
PS C:\Users\AMIR\Downloads\anomalydeteccdn> python ./kafka/newprod.py
Producer started
Producer initialized, reading data....
Data delivered {'timestamp': '2088-05-14 23:07:00', 'Status code ': '206', 'protocol': '1', 'contentlength': '13392.93651'
, 'timefirstbyte': '0.00015', 'timetoserv': '0.000689', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefa
mily': '0.0', 'path': '3931298.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-16 02:43:00', 'Status code ': '502', 'protocol': '1', 'contentlength': '2.634920635'
, 'timefirstbyte': '0.020632', 'timetoserv': '0.020684', 'osfamily': '1.0', 'uamajor': '1.0', 'uafamily': '2.0', 'devicefa
mily': '1.0', 'path': '92.0', 'Live channel': '1.0', 'devicebrand': '0.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-18 19:05:00', 'Status code ': '403', 'protocol': '1', 'contentlength': '0.0', 'timef
irstbyte': '0.000139', 'timetoserv': '0.0', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefamily': '0.0'
, 'path': '42.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-15 20:25:00', 'Status code ': '412', 'protocol': '1', 'contentlength': '23.88888889'
, 'timefirstbyte': '0.014797', 'timetoserv': '0.014842', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicef
amily': '0.0', 'path': '21936373.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-16 04:08:00', 'Status code ': '412', 'protocol': '1', 'contentlength': '23.88888889'
, 'timefirstbyte': '0.014794', 'timetoserv': '0.014847', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicef
amily': '0.0', 'path': '25178360.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-17 19:05:00', 'Status code ': '404', 'protocol': '1', 'contentlength': '0.0', 'timef
irstbyte': '0.078783', 'timetoserv': '0.078814', 'osfamily': '0.0', 'uamajor': '0.0', 'uafamily': '0.0', 'devicefamily': '
0.0', 'path': '174413.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-14 15:03:00', 'Status code ': '206', 'protocol': '1', 'contentlength': '237.4126984'
, 'timefirstbyte': '0.000131', 'timetoserv': '0.000203', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicef
amily': '0.0', 'path': '1336355.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-13 09:32:00', 'Status code ': '200', 'protocol': '1', 'contentlength': '22.58730159'
, 'timefirstbyte': '0.000145', 'timetoserv': '0.000201', 'osfamily': '1.0', 'uamajor': '1.0', 'uafamily': '2.0', 'devicefa
mily': '1.0', 'path': '18.0', 'Live channel': '8.0', 'devicebrand': '0.0', 'method': '1'}
Data delivered {'timestamp': '2088-05-13 15:32:00', 'Status code ': '404', 'protocol': '1', 'contentlength': '0.0', 'timef
irstbyte': '0.000139', 'timetoserv': '0.0', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefamily': '0.0'
```

Fig. 7. Architecture

From figure8 we can see consumer already reading messages from the stored kafka topic.Kafka consumers implement a "pull model". This means that Kafka consumers request data from Kafka brokers in order to get it in our case.This implementation was made so that consumers can control the speed at which the topics are being consumed.



```
PS C:\Users\AMIR\Downloads\anomalydeteccdn> python ./kafka/newcons.py
{'timestamp': '2088-05-14 23:07:00', 'Status code ': '206', 'protocol': '1', 'contentlength': '13392.93651', 'timefirstbyt
e': '0.00015', 'timetoserv': '0.000689', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefamily': '0.0', '
path': '3931298.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Writing was successful to the Kafka topic
{'timestamp': '2088-05-16 02:43:00', 'Status code ': '502', 'protocol': '1', 'contentlength': '2.634920635', 'timefirstbyt
e': '0.020632', 'timetoserv': '0.020684', 'osfamily': '1.0', 'uamajor': '1.0', 'uafamily': '2.0', 'devicefamily': '1.0', '
path': '92.0', 'Live channel': '1.0', 'devicebrand': '0.0', 'method': '1'}
Writing was successful to the Kafka topic
{'timestamp': '2088-05-18 19:05:00', 'Status code ': '403', 'protocol': '1', 'contentlength': '0.0', 'timefirstbyte': '0.0
00139', 'timetoserv': '0.0', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefamily': '0.0', 'path': '42.0
', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Writing was successful to the Kafka topic
{'timestamp': '2088-05-15 20:25:00', 'Status code ': '412', 'protocol': '1', 'contentlength': '23.88888889', 'timefirstbyt
e': '0.014797', 'timetoserv': '0.014842', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefamily': '0.0',
'path': '21936373.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Writing was successful to the Kafka topic
{'timestamp': '2088-05-16 04:08:00', 'Status code ': '412', 'protocol': '1', 'contentlength': '23.88888889', 'timefirstbyt
e': '0.014794', 'timetoserv': '0.014847', 'osfamily': '0.0', 'uamajor': '3.0', 'uafamily': '12.0', 'devicefamily': '0.0',
'path': '25178360.0', 'Live channel': '60.0', 'devicebrand': '1.0', 'method': '1'}
Writing was successful to the Kafka topic
```

Fig. 8. Architecture

6 Telegraf

Telegraf is InfluxData's data collection agent for collecting and reporting metrics. Its vast library of input plugins and "plug-and-play" architecture lets you quickly and easily collect metrics from many different sources. Telegraf has been implemented in our project in terms of collecting metrics.

7 Chronograf

Chronograf allowed us to quickly see the data that we have stored in InfluxDB we build robust queries and alerts. It is simple to use and includes templates and libraries to allowed us to rapidly build dashboards with real-time visualizations of our data.

7.1 Dashboards

Chronograf offered a complete dashboarding solution for visualizing our data. We created several dashboards with chronograf based on our data.

8 Summary

With the help of different technologies and algorithms we have built out Realtime Anomaly Detection System. As we have seen ,When training data contained both normal and abnormal observations, Our model identified outliers during the fitting process. It can be used when "outliers" are defined as points that exist in low-density regions of the dataset — any new observation which does not belong to high-density areas will be labeled as such automatically by the algorithm.