```
timestamp, temperature, humidity, illuminance, dust, thi
2022/07/01 22:17:53, 27.50, 64.60, 26.85, 0.79, 76.92
2022/07/01 22:18:51, 27.55, 62.92, 26.75, 9.68, 76.78
2022/07/01 22:19:56, 27.55, 62.45, 27.08, 2.34, 76.72
2022/07/01 22:20:53, 27.60, 63.18, 22.18, 14.93, 76.89
2022/07/01 22:21:51, 27.59, 62.95, 26.79, 3.78, 76.84
・・・・・・・
```

リスト1　今回扱うデータはセンサで計測した値が格納されているcsvファイル

```
cqpub-ml-example-{yourname}/
  sensordata/
    cqpub-sensordata-example01.csv
```

リスト2　バケットとデータの構成関係

```
import pandas as pd
import os

# IoTデータの登録先
s3_bucket = 'cqpub-ml-example-yourname'
data_file = 'sensordata/cqpub-sensordata-example01.csv'

# IoTデータのCSVカラム
columns = [ 'temperature','humidity', 'illuminance', 'dust', 'thi']

# 移動平均の計算間隔（分）
freq = 30

# S3バケットからCSVファイルを読み込む
df = pd.read_csv(os.path.join('s3://', s3_bucket, data_file), parse_dates=True)
df['timestamp'] = pd.to_datetime(df['timestamp'])

df = df.set_index('timestamp')
df = df.dropna()
```

リスト3　データの読み込み

```
# スパイク除去等に利用する集計設定
agg_config = {}
for column in columns:
    agg_config[column] = 'median'

# 移動平均を算出 (freqのWindowで中央値を取得)
df = df.resample(str(freq) + 'min').agg(agg_config)
```
リスト4　移動平均の計算

```
from io import StringIO
import boto3

movingavg_file = 'mldata/cqpub-sensordata-example01-movingavg.csv'

csv_buffer = StringIO()
df.to_csv(csv_buffer)

s3 = boto3.resource('s3')
s3.Object(s3_bucket, movingavg_file).put(Body=csv_buffer.getvalue())
```
リスト5　移動平均の計算結果を保存

```
import copy, json

df_len = len(df)
train_ds = {}
test_ds = {}
ds_item = { 'start': None, 'target': [] }

# 学習用とテスト用のデータを 8:2 で分割する
train_df = df[:int(df_len*0.8)]
test_df = df[int(df_len*0.8):]


# データを以下の形式に変換する。
# {
#   "start": "2016-01-01 00:00:00",
```

```
#   "target": [4.9, 5.3, 3.4, 5.9, ...]
# }
for column in columns:
    train_ds[column] = copy.deepcopy(ds_item)
    train_ds[column]['start'] = str(train_df.index.values[0])
    test_ds[column] = copy.deepcopy(ds_item)
    test_ds[column]['start'] = str(test_df.index.values[0])

for index, row in train_df.iterrows():
    for column in columns:
        train_ds[column]['target'].append(str(row[column]))

for index, row in test_df.iterrows():
    for column in columns:
        test_ds[column]['target'].append(str(row[column]))


# jsonlineの形式で 一時ファイルに書き出し
train_path = 'train.jsonl'
test_path = 'test.jsonl'

def save_to_file(path, dataset):
    with open(path, 'wb') as fp:
        for column in columns:
            fp.write(json.dumps(dataset[column]).encode("utf-8"))
            fp.write("¥n".encode('utf-8'))

# ファイルに保存
save_to_file(train_path, train_ds)
save_to_file(test_path, test_ds)

# s3にアップロード
deepar_prefix = 'deepar'
s3 = boto3.resource('s3')
bucket = s3.Bucket(s3_bucket)

with open(train_path, 'rb') as data:
    bucket.put_object(Key=f'{deepar_prefix}/{train_path}', Body=data)

with open(test_path, 'rb') as data:
    bucket.put_object(Key=f'{deepar_prefix}/{test_path}', Body=data)
```

リスト6　学習用データと評価用データの作成

```
import sagemaker
sagemaker_session = sagemaker.Session()
region = sagemaker_session.boto_region_name
role = sagemaker.get_execution_role()
image_name = sagemaker.image_uris.retrieve("forecasting-deepar", region)

output_path = "s3://{}/{}/output".format(s3_bucket, deepar_prefix)

# 学習期間を14日、予測期間を4日とする。
# ポイント数でデータ長を指定する必要があるので、freq の幅に応じて指定する。
train_len = int(14 * 24 * (60/freq)) # 14d * 24h * 60min/30min
pred_len = int(4 * 24 * (60/freq)) # 4d * 24h * 60min/30min

estimator = sagemaker.estimator.Estimator(
    image_name,
    sagemaker_session=sagemaker_session,
    role=role,
    instance_count=1,
    instance_type='ml.c4.xlarge',
    base_job_name='cqpub-deepar',
    output_path=output_path
)

# 学習のパラメータ設定
hyperparameters = {
    "time_freq": str(freq) + 'min',
    "epochs": "500",
    "early_stopping_patience": "10",
    "mini_batch_size": "64",
    "learning_rate": "1E-4",
    "context_length": str(train_len),
    "prediction_length": str(pred_len)
}
estimator.set_hyperparameters(**hyperparameters)

# 先ほど一時保存したデータを指定
data_channels = {
    "train": f's3://{s3_bucket}/{deepar_prefix}/{train_path}',
    "test": f's3://{s3_bucket}/{deepar_prefix}/{test_path}'
}

# 学習の実行
```

```
43: estimator.fit(inputs=data_channels, wait=True)
```
リスト7　時系列予測の学習処理

```
# 推論プロセスを立ち上げる
predictor = estimator.deploy(
  instance_type = 'ml.t2.large',
  initial_instance_count = 1
)

pred_data = {}

# 推論時には，学習用とテスト用で分けたデータを結合したデータ（元データ全体）を用いる
for column in columns:
    pred_data[column] = copy.deepcopy(train_ds[column])
    pred_data[column]['target'].extend(test_ds[column]['target'])

# Sagemaker の predictor用にインプット形式を加工し，　predict()で推論結果を取得
pred_res = {}
for column in columns:
    pred_req = {
        'instances': [pred_data[column]]
    }
    pred_res[column] = predictor.predict(
        json.dumps(pred_req).encode('utf-8'),
        initial_args={'ContentType': 'application/json'}
    )

# 推論プロセスを削除する
predictor.delete_predictor()
```
リスト8　時系列予測の推論処理

```
from sagemaker import RandomCutForest
from sagemaker.amazon.common import RecordSerializer
import numpy as np
import io

rcf_session = sagemaker.Session()
image_name = sagemaker.image_uris.retrieve("randomcutforest", region)

rcf_prefix = 'rcf'
s3_output_path = "s3://{}/{}/output".format(s3_bucket, rcf_prefix)

rcf_estimator = sagemaker.estimator.Estimator(
    image_name,
    sagemaker_session=sagemaker_session,
    role=role,
    instance_count=1,
    instance_type='ml.c4.xlarge',
    base_job_name='cqpub-randomcutforest',
    output_path=s3_output_path
)

hyperparameters = {
    "num_samples_per_tree": 2,
    "num_trees": 100,
    "feature_dim": 120
}
rcf_estimator.set_hyperparameters(**hyperparameters)

# シングリング処理によって特徴量の次元を増やす
def shingle(data, shingle_size):
    num_data = len(data)
    shingled_data = np.zeros((num_data-shingle_size, shingle_size))

    for n in range(num_data - shingle_size):
        shingled_data[n] = data[n:(n+shingle_size)]
    return shingled_data

shingle_size = 5 * 24
train_data = shingle(df['dust'], shingle_size)

# RandomCutForestで学習するにあたり、RecordIOと呼ばれる形式でS3にアップロードする
def convert_and_upload_training_data(
    ndarray, bucket, prefix, filename='rcf_train.pbr'):
```

```
44:
45:     import boto3
46:     import os
47:
48:     serializer = RecordSerializer()
49:     buffer = serializer.serialize(ndarray)
50:
51:     s3 = boto3.resource('s3')
52:     s3.Bucket(bucket).put_object(Body=buffer,
53:                                  Key='rcf/train/rcf_train.pbr',
54:                                  ContentType='application/x-recordio-protobuf')
55:
56:     s3_object = os.path.join(prefix, 'train', filename)
57:     s3_path = 's3://{}/{}'.format(bucket, s3_object)
58:     return s3_path
59:
60: rcf_train_data = convert_and_upload_training_data(train_data, s3_bucket, rcf_prefix)
61:
62: # S3にアップロードしたRecordIO形式のデータを学習時のInputとする
63: rcf_train_input = sagemaker.inputs.TrainingInput(
64:     rcf_train_data,
65:     distribution='ShardedByS3Key',
66:     content_type='application/x-recordio-protobuf')
67:
68: rcf_estimator.fit({'train': rcf_train_input})
```

リスト9　異常検知の学習処理

```
# 推論プロセスを立ち上げる
rcf_predictor = rcf_estimator.deploy(
  instance_type = 'ml.t2.large',
  initial_instance_count = 1
)

from sagemaker.serializers import CSVSerializer
from sagemaker.deserializers import JSONDeserializer


# train_dataはCSV形式, 推論のレスポンスはJSON形式となるため,
# それぞれの変換設定をおこない推論する。
rcf_predictor.serializer = CSVSerializer()
```

```
rcf_predictor.deserializer = JSONDeserializer()
rcf_pred_res = rcf_predictor.predict(train_data)

# 推論のレスポンスを描画用に加工
scores = [datum['score'] for datum in rcf_pred_res['scores']]

# 推論プロセスを削除する
rcf_predictor.delete_predictor()
```

リスト10　異常検知の推論処理