Review on IEEE BigData 2019 Real-time Machine Learning Competition on Data Streams

Seojeong Yu

Inha University
Big Data Lab

Contents

1. Introduction

1.0 Motivation

IEEE Big Data Cup Challenge 2019 - Real-time Machine Learning Competition on Data Streams is a novel type of the machine learning competition. The main idea is to build incremental classifiers and provide the predictions for the data instances that are arriving in stream. Processing data streams is in high demand in recent years due to the rapid development of IoT and many other technologies that act as data sources. Data are generated in real-time from a high number of various data sources: sensors, IoT devices, social networks, applications, bank and market transactions. Building prediction models for data streams is an essential task due to its importance in real-time decision making process. However, this is very challenging because of the dynamic nature of the data where instances arrive in high speed and the stream is often subject to changes. This competition represents the typical scenario in Data Stream Mining that has different requirements compared to the traditional batch learning setting. The most significant ones would be

1. Process an example at a time and inspect it only once at most
2. Use a limited amount of memory
3. Work in a limited amount of time
4. Be ready to predict at any time

1.1 Concept

Data

- Telecommunication company and network providers

Predicting use cases :

- Capacity planning and activity prediction
- Anomaly detection

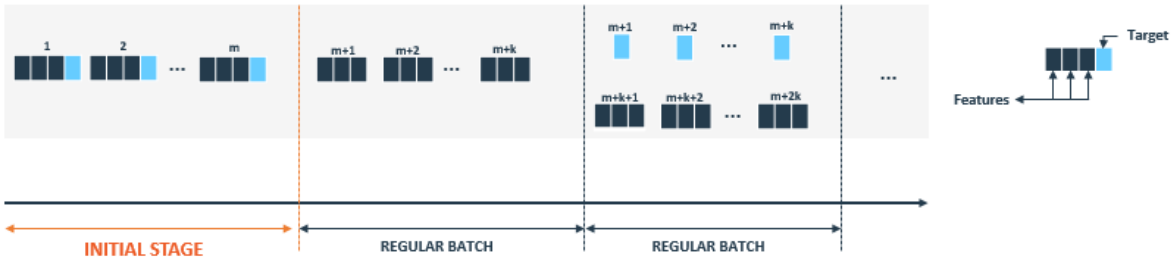
Main idea

- to build incremental classifier
- provide the predictions for the data instance arriving in the stream

Goal

- to build models able to **predict the future values** with high accuracy while taking into account **evolutionary nature of these data**
- **incrementally** learning model
- **detect changes** in order to adapt to them as soon as they occur

1.2 Data Stream Flow



step 1: dataset(csv file)이 주어지고 모델을 미리 학습시킨다.
step 2: online으로 server에 5일치의 예측을 보낸다. → **test 단계**
step 3: online으로 server에서 직전에 예측한 것의 실제 값 5개를 보내주면 모델을 학습시킨다. → **train 단계**
step 2, 3이 반복된다.

1.3 Metrics

- **RRCF(Robust Random Cut Forest)**

- provides anomaly detection on streaming data
- uses an ensemble of random forests of tree graphs
- constructing a tree of 10-1000 vertices from a random sampling of the pool
- creates more trees of the same size 1-1000 times which creates the forest
- define the normal for a new data point by injecting it into the trees

  → see how much that changes the makeup of the forest(depth, width)

```
#check anomaly with RRCF
def anomaly_detection(self, data):
    for tree in self.forest:
        if len(tree.leaves) > self.tree_size:
            tree.forget_point(self.idx-self.tree_size)

        tree.insert_point(data, index=self.idx)

        if not self.idx in self.avg_codisp:
            self.avg_codisp[self.idx] = 0
        self.avg_codisp[self.idx] += tree.codisp(self.idx) / self.num_trees
    # avg_codisp은 (각 tree 이 point를 anomaly로 생각하는 정도)의 평균
    mean = np.array(list(self.avg_codisp.values())).mean()
    std = np.array(list(self.avg_codisp.values())).std()

    z = (self.avg_codisp[self.idx] - mean)/std
    self.idx += 1
    if z > 3.0 or z < -3.0 :
        return self.previous_train_batch.mean()
        # if abs(z-score) is over 3.0
        # replace the value with the mean of whole data we met

    else :
        return data
    #if not over 3.0, then no need to replace the value
```

- **Mondrian Forest**

  Training

- **split** is decided independently of the target
- same scale & important → extremely randomized tree
  - split feature index f → probability proportional to **u_b[f]-l_b[f]**

- u_b , l_b : upper & lower bounds of all the features

- after fixing the feature index, the split threshold is then drawn from a uniform distribution with limits l_b, u_b

Prediction

- takes account all the nodes in the path of a new point from the root to the leaf, weigh the nodes on the basis of how sure/unsure we are about the prediction in that particular node

```
def partial_train(self, X_test, y_test):
    y_pred, y_std = self.mfr.predict(X_test, return_std=True)
    self.mfr.partial_fit(X_test, y_test)
    #print('pred : %f, std: %f, y: %f'%(y_pred, y_std, y_test))
    return y_pred, y_std
```

1.4 API

Table: List of all functions in the API

__init__(self, batch_size)	Puts credentials and the address of the server with whom the communication will be established
create_metadata(user_id, code, token)	Returns the metadata of user_id, code and token
create_forest(num_trees)	Returns the forest of Robust Random Cut Trees
generate_predictions(self)	Yields prediction
anomaly_detection(self, data, index)	Replaces the value with the mean of the whole data if detected as anomaly. Else, returns the data
loop_messages(self)	generate prediction and get prediction one by one and send to server

2. Process

2.0 Preprocess

Data stream이 들어오는 형태 :

```
syntax = "proto3";
option java_package = "ex.grpc";
option objc_class_prefix = "HSW";
package file;

// The data service definition.
service DataStreamer {
  // Sends multiple greetings
  rpc sendData (stream Prediction) returns (stream Message) {}
}

message Message{
    int32  rowID =1;
    string date = 2;
    float  target = 3;
    string Deadline = 4;
    string Released = 5;
    string tag= 6;
}

message Prediction{
    int32 rowID = 1;
    float target = 2;
}
```

test batch는 Message의 target값이 존재하지 않는 형태로 들어온다.

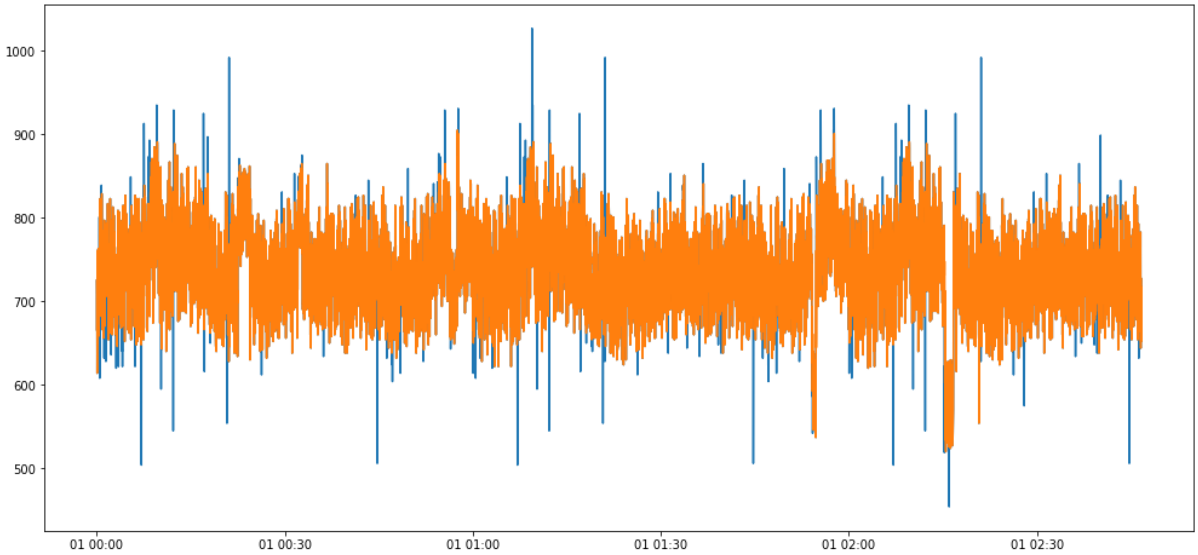
2.1 RRCF + Mondrian Forests

우선 데이터가 stream을 통해 들어오는 시간이 되기 전에, mondrian forest와 robust random cut forest를 미리 메일을 통해 주어진 training data로 학습시키기 위해서 .csv file을 Pandas dataframe format으로 받아서 학습시켰다. 이렇게 학습을 시키는 과정에서 Mondrian Forest는 feature와 target값이 필요하지만 주어진 데이터는 univariate time series이기 때문에 feature로 과거 3개의 값을 할당해주었다. - 실제 stream에서는 데이터가 5개씩 들어오므로 바로 이전 3개의 값을 참조할 수는 없어서, 5, 6, 7일 전의 값을 feature로 이용하였다 -

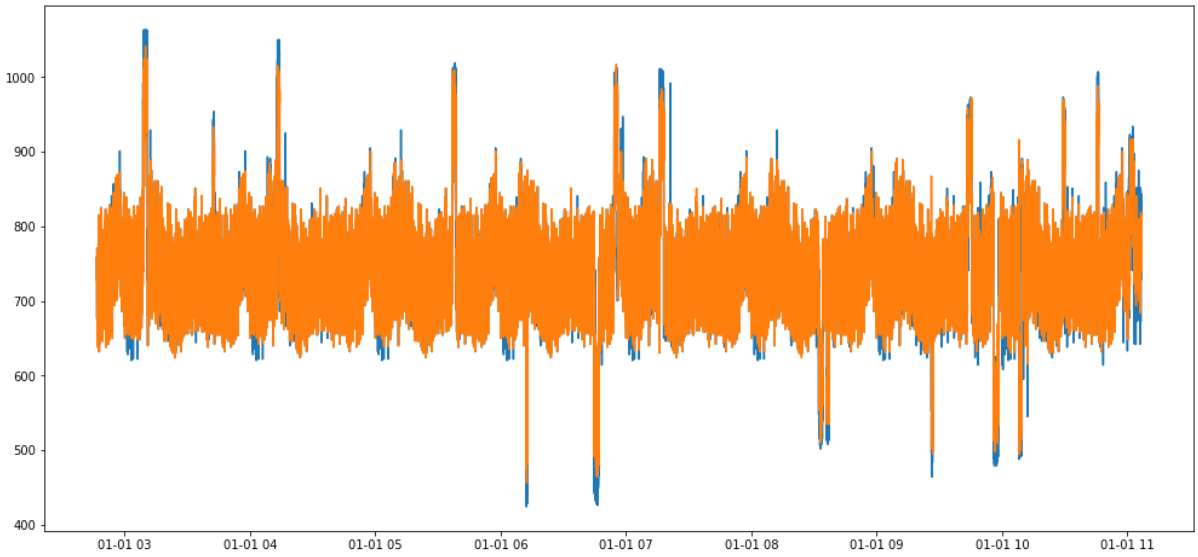
date	target	prev1	prev2	prev3
2018-01-02 00:00:04	361.0	341.0	388.0	397.0
2018-01-02 00:00:05	359.0	361.0	341.0	388.0
2018-01-02 00:00:06	369.0	359.0	361.0	341.0
2018-01-02 00:00:07	387.0	369.0	359.0	361.0
2018-01-02 00:00:08	415.0	387.0	369.0	359.0

test과정 : mondrian forest regressor를 사용하여 prediction하였다.
train과정 : target값을 받아 Robust Random Cut Forest를 사용해 anomaly 여부를 지금까지의 평균과 표준편차를 통해 z-score를 판단하고 z값이 절댓값 3.0을 넘는 경우 anomaly라고 판단해서 과거 batch 5개 값의 평균으로 그 값을 대체하여 model을 update 하는데 사용했다.

파란색 : 원래 데이터
주황색 : anomaly detection 후 이상치는 평균값으로 대체한 데이터



local환경에서 5, 6, 7일전의 데이터를 feature로, mfr는 tree 100개, rrcf는 tree 40개로 실험한 결과 (10000개의 training data, 30000개의 incremental test & training data) 아래 그래프와 같은 결과가 나왔다. 주황색 : 예측값, 파란색 : data의 정제된 값



이렇게 나온 30000개의 예측값과 정제된 값을 가지고 계산한 r2_score는 88.5% 이다.

3. Results

Leader Board

초반에 에러율이 높았지만 빠르게 baseline을 따라잡은 것을 볼 수 있다. 중간에 data에 큰 Drift가 존재했던 걸로 보인다.

최종 순위: 7위

4. Bibliography

[1] Balaji Lakshminarayanan, Daniel M. Roy, Yee Whye Teh. “Mondrian Forests: Efficient Online Random Forests”, Advances in Neural Information Processing Systems 27 (NIPS), 2014, pages 3140-3148, 2014
[2] Protocol Buffers. [Online]. Available: https://developers.google.com/protocol-buffers
[3] gRPC. [Online]. Available: https://grpc.io/
[4] Python. [Online]. Available: https://www.python.org/doc/
[5] Robust Random Cut Forest [Online]. Available : https://klabum.github.io/rrcf/
[6] Scikit-learn. Mondrian Forest [Online]. Available : https://scikit-garden.github.io
[7] S. Guha “Robust Random Cut Forest Based Anomaly Detection on Streams” in Proceedings of The 33rd International Conference on Machine Learning, PMLR 48:2712-2721, 2016.

소스 코드: https://github.com/hovy1994/Machine-learning/blob/master/challenge_final_code.py