Detecting Concept Drift in Fully Distributed Environments

István Hegedűs, Nyers Lehel and Róbert Ormándi

University of Szeged, Szeged, Hungary and Subotica Tech, Subotica, Serbia

Motivation

- We are in the middle of big data era.
- Our devices and applications accumulate more and more data.
- Paradoxically while getting access to more and more data, we can find less information by applying the original retrieval techniques.
- This phenomena increases the importance of emerging fully distributed, large-scale data mining algorithms that can work in unreliable networks, take the privacy of users into account and be adaptive to changes in the data.

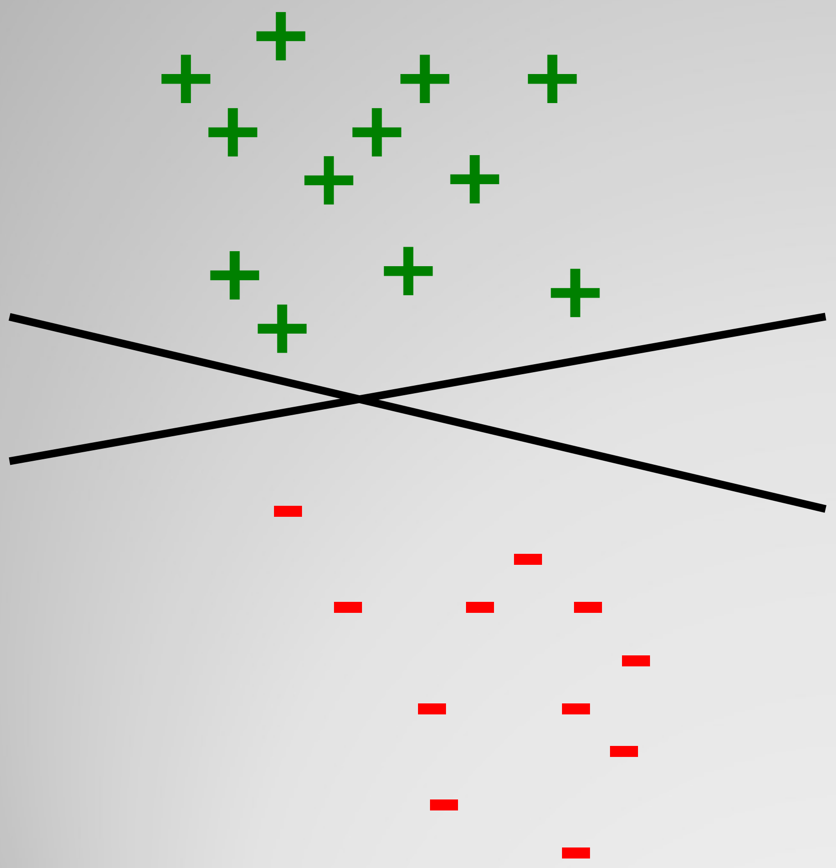
Supervised Learning

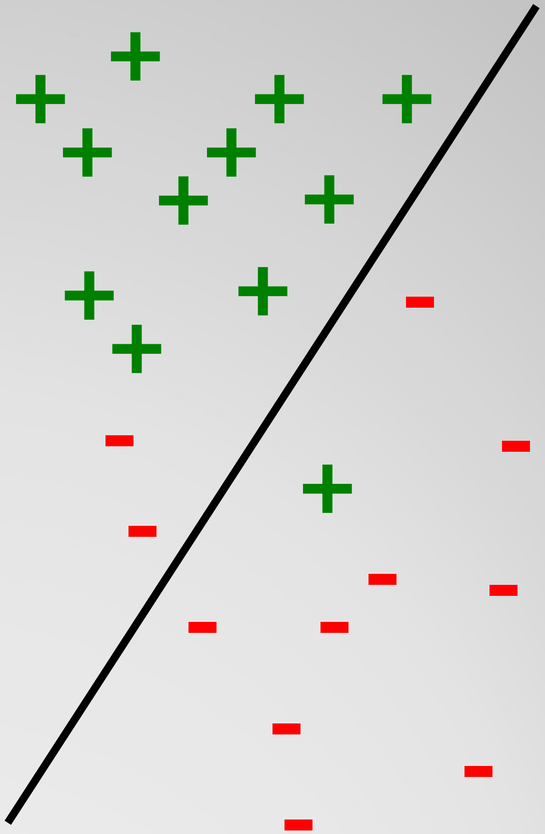
- We have a manually labeled training database

$$S = \{(x_1, y_1), \ldots, (x_l, y_l)\} \in \mathbb{R}^d \times \{-1, +1\}.$$

- Here xi called feature vector that describes an object of real world (email) as a real-valued vector
- yi is the class label which assigns the feature vector to a well-defined class (e.g. spam).
- The goal—through the learning phase—of the classification problem is to find a model
 f : Rd→ {−1,+1}
 which can classify any sample coming from the same probability distribution (D).

Illustration of classification with a linear model





Fully distributed data

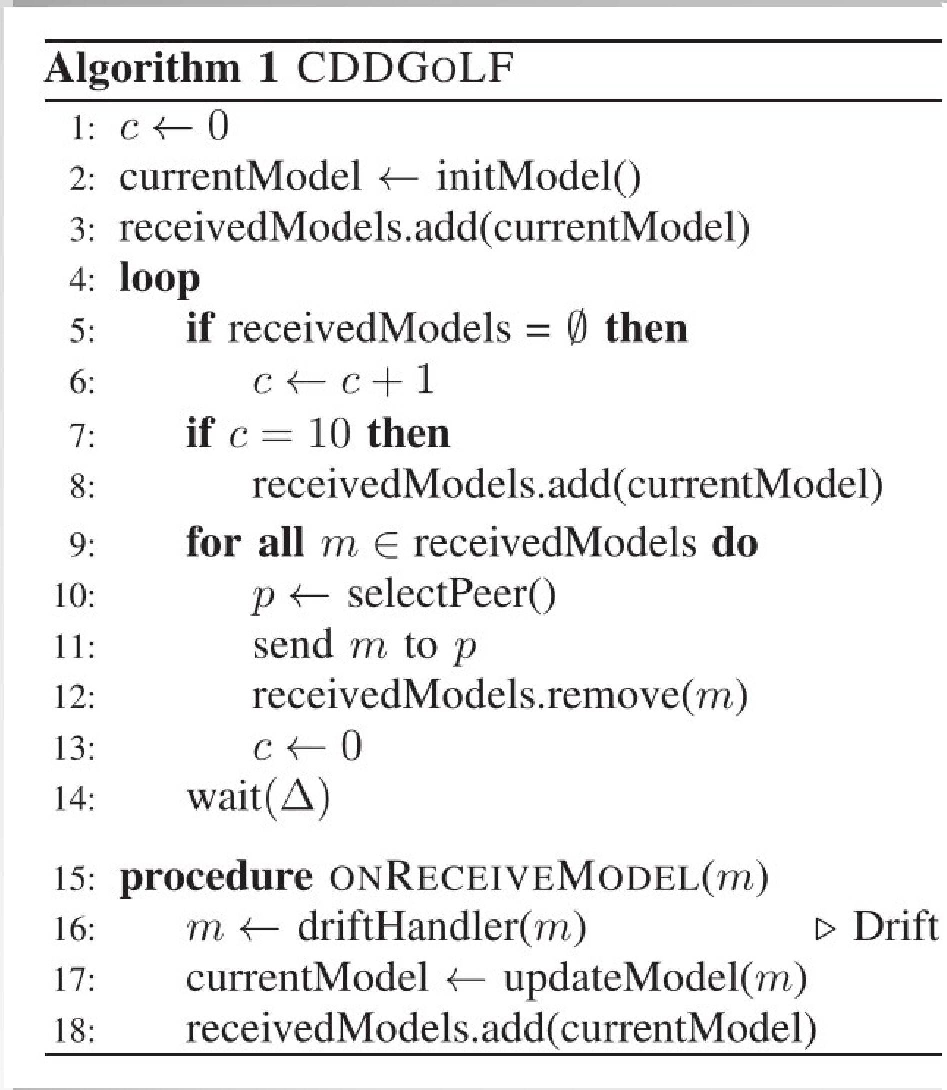
- Horizontal data distribution
- Every node has very few samples, we assume they have only one
- We do not allow for moving data, only local processing (privacy preservation)
- Drifting: data distribution can change

GoLF

- Gossip Learning Framework (GoLF) - is a learning framework designed for performing fully distributed learning in large scale networks.
- The basic idea behind GoLF is that large number of online models take random walks in the network while improving themselves using the training samples contained at the nodes and getting combined by applying ensemble learning techniques.

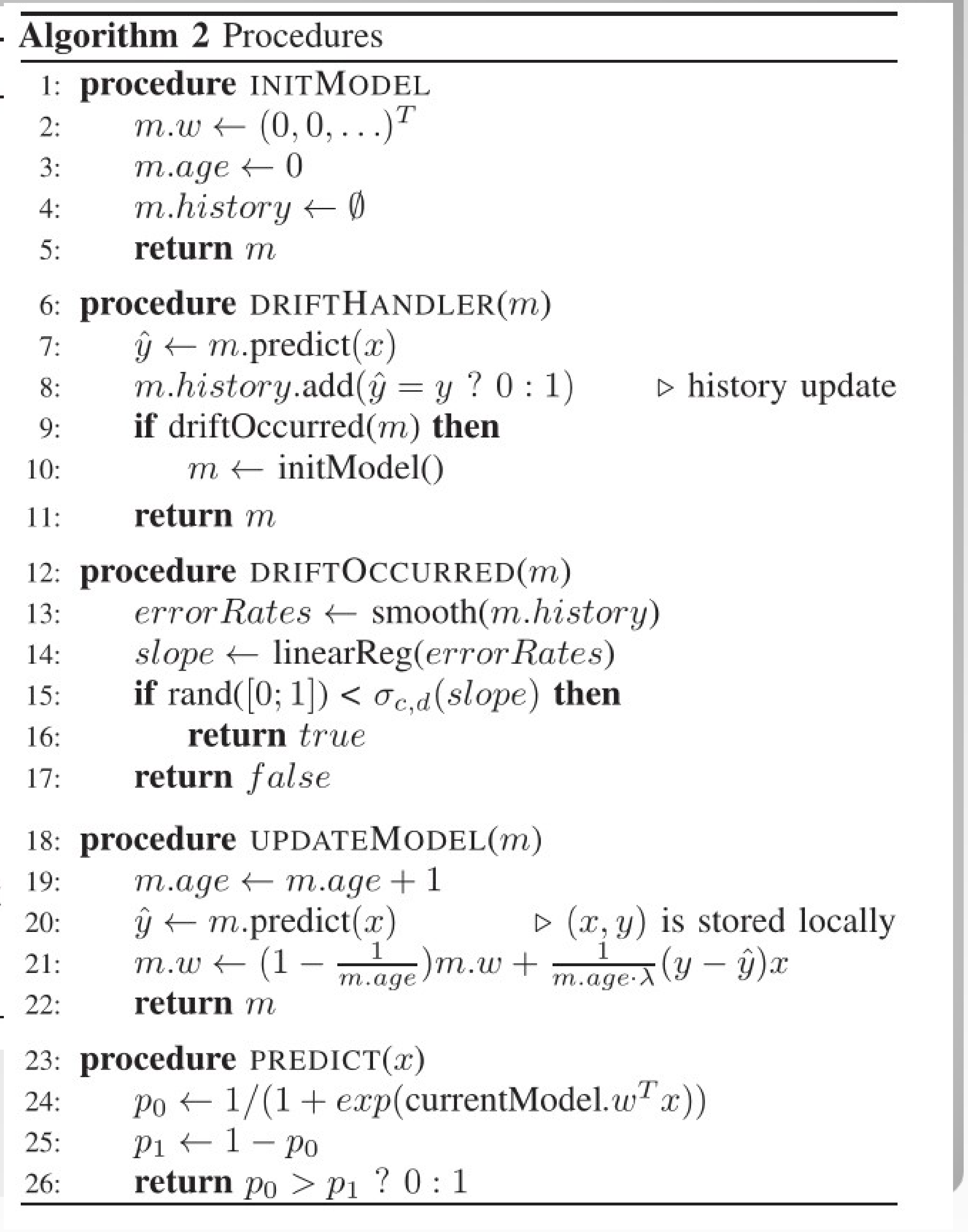
Concept Drift

- The distribution that generates the training databases (D) may change over the time. This change—especially when it is significant and sudden—makes the already trained models inaccurate. This phenomena is called concept drift.
- we have to identify the time moment t when (sudden) drift occurred while we also handle it. This problem is known as the drift detection problem.



Algorithm 1 CDDGoLF

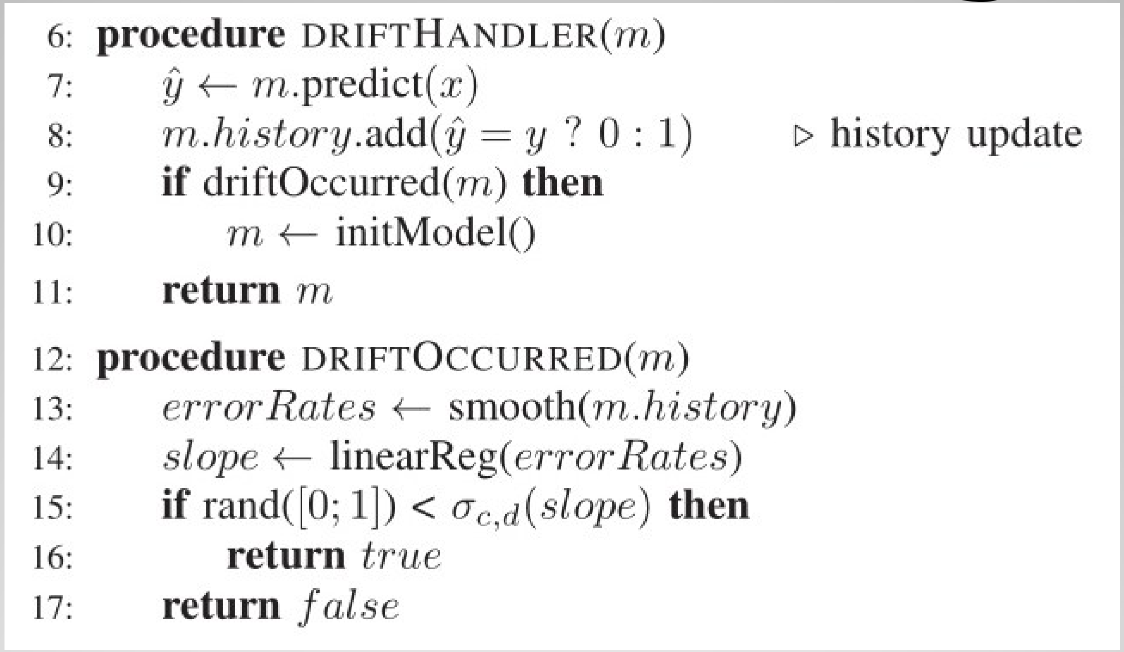
```
$c \leftarrow 0$
currentModel $\leftarrow$ initModel()
receivedModels.add(currentModel)
**loop**
    **if** receivedModels = $\emptyset$ **then**
        $c \leftarrow c + 1$
    **if** $c = 10$ **then**
        receivedModels.add(currentModel)
    **for all** $m \in$ receivedModels **do**
        $p \leftarrow$ selectPeer()
        send $m$ to $p$
        receivedModels.remove($m$)
        $c \leftarrow 0$
    wait($\Delta$)
**procedure** onReceiveModel($m$)
    $m \leftarrow$ driftHandler($m$)                    ▷ Drift
    currentModel $\leftarrow$ updateModel($m$)
    receivedModels.add(currentModel)
```



Algorithm 2 Procedures

```
**procedure** initModel
    $m.w \leftarrow (0, 0, \ldots)^T$
    $m.age \leftarrow 0$
    $m.history \leftarrow \emptyset$
    **return** $m$
**procedure** driftHandler($m$)
    $\hat{y} \leftarrow m$.predict($x$)
    $m.history$.add($\hat{y} = y \ ? \ 0 : 1$)          ▷ history update
    **if** driftOccurred($m$) **then**
        $m \leftarrow$ initModel()
    **return** $m$
**procedure** driftOccurred($m$)
    $errorRates \leftarrow$ smooth($m.history$)
    $slope \leftarrow$ linearReg($errorRates$)
    **if** rand([0; 1]) < $\sigma_{c,d}(slope)$ **then**
        **return** $true$
    **return** $false$
**procedure** updateModel($m$)
    $m.age \leftarrow m.age + 1$
    $\hat{y} \leftarrow m$.predict($x$)              ▷ $(x, y)$ is stored locally
    $m.w \leftarrow (1 - \frac{1}{m.age})m.w + \frac{1}{m.age \cdot \lambda}(y - \hat{y})x$
    **return** $m$
**procedure** predict($x$)
    $p_0 \leftarrow 1/(1 + exp($currentModel$.w^T x))$
    $p_1 \leftarrow 1 - p_0$
    **return** $p_0 > p_1 \ ? \ 0 : 1$
```

The solution – detecting drift



```
procedure DRIFTHANDLER($m$)
    $\hat{y} \leftarrow m$.predict($x$)
    $m.history$.add($\hat{y} = y$ ? 0 : 1)        ▷ history update
    if driftOccurred($m$) then
        $m \leftarrow$ initModel()
    return $m$
procedure DRIFTOCCURRED($m$)
    $errorRates \leftarrow$ smooth($m.history$)
    $slope \leftarrow$ linearReg($errorRates$)
    if rand([0; 1]) < $\sigma_{c,d}(slope)$ then
        return $true$
    return $false$
```

- if the slope is negative or 0, the learning is probably in the convergence or in the already converged phase, respectively. Otherwise, if the slope is
- positive, drift probably occurred.

Experimental results:Drift Handling

- For both databases we modeled the drift by changing the labeling over the time.
- Algorithms without concept drift handling capabilities show burn in effect, i.e. after a certain time they cannot adapt to the changing concepts.

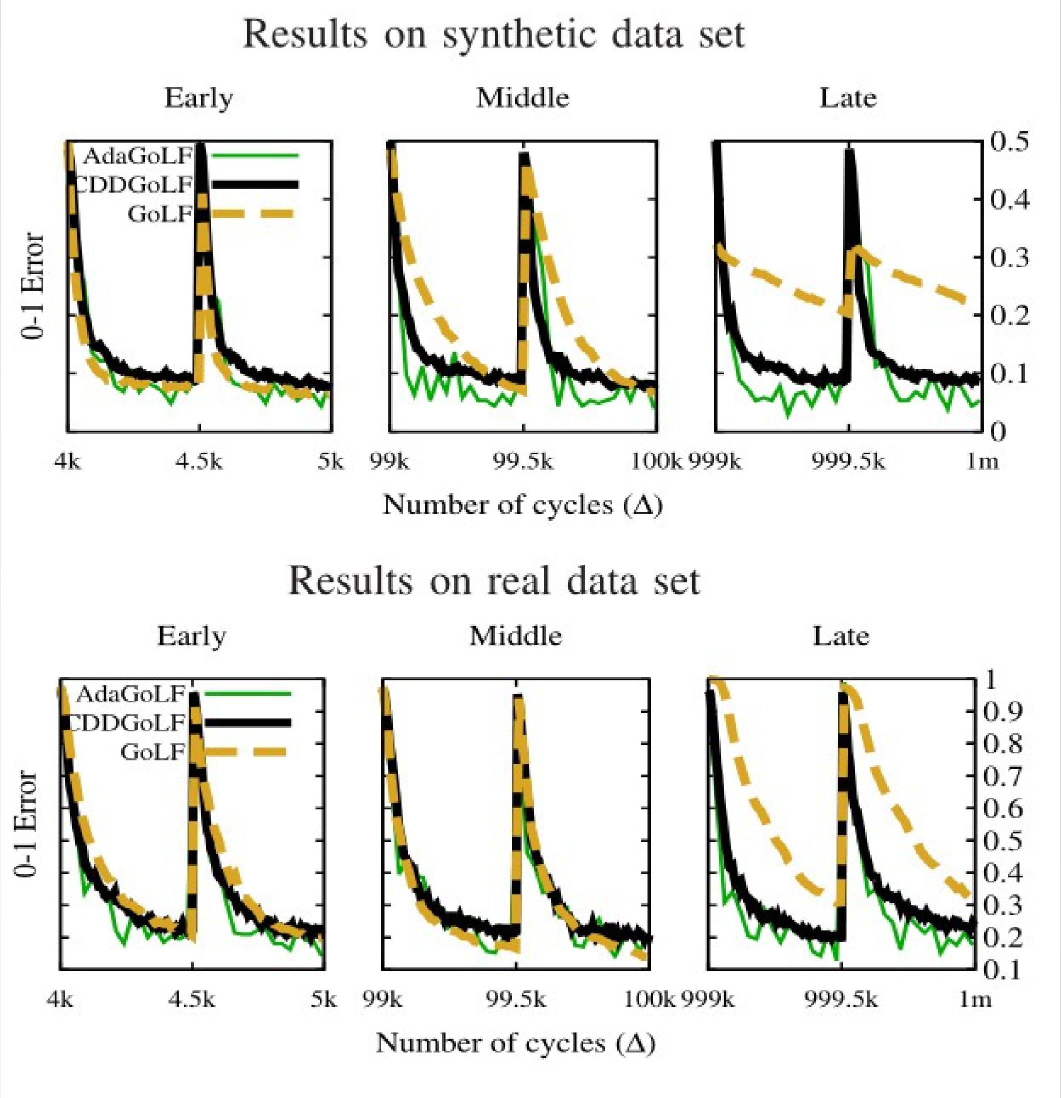


Figure 1. Adaptive methods avoid the burn in effect.

Experimental results:Drift Detection

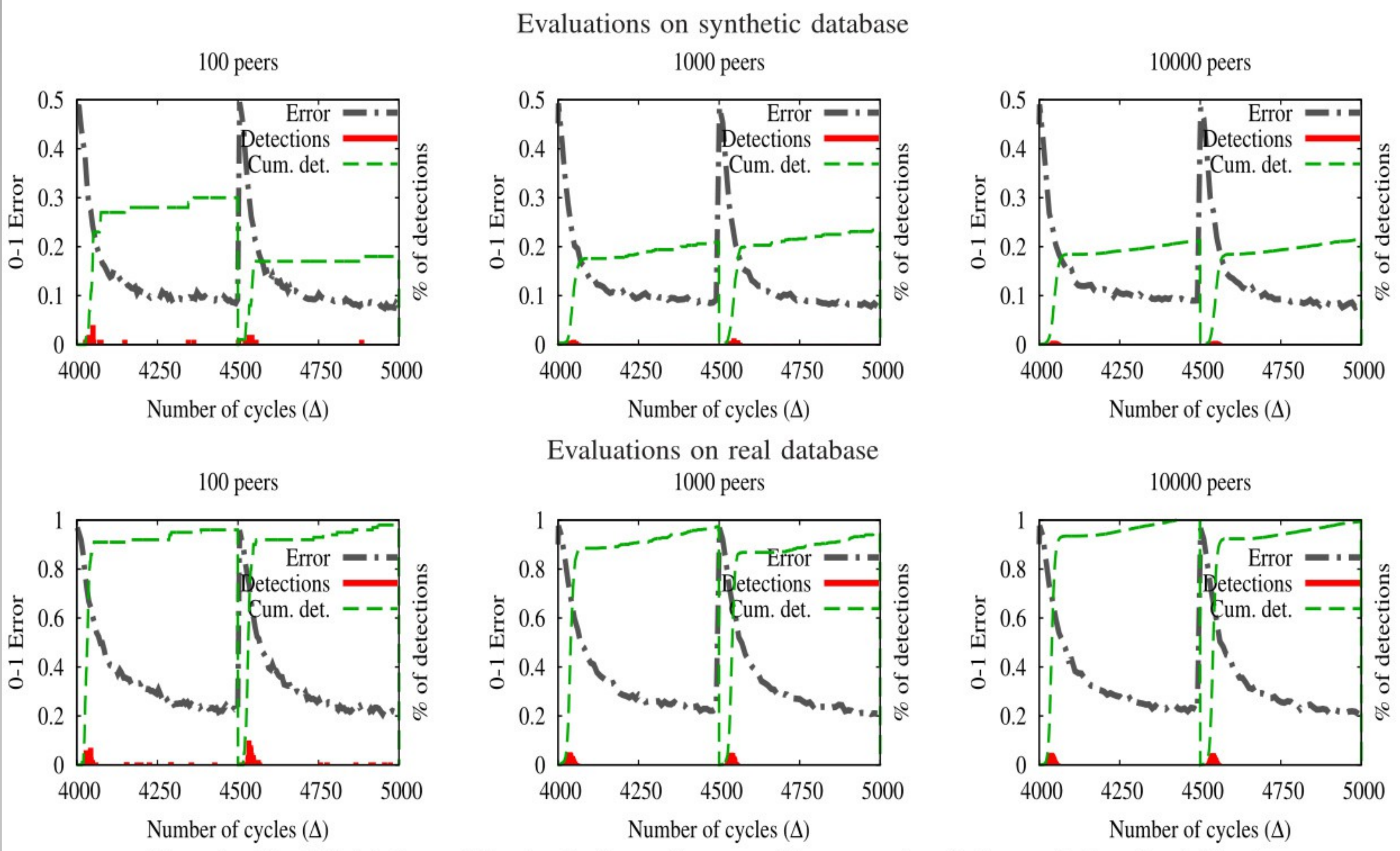


Figure 2. The drift detection and the classification performance of the proposed method on synthetic and real datasets.

CONCLUSION

- In this work we proposed a novel algorithm for detecting concept drifts in a fully distributed network on the top of our previously proposed learning framework (GoLF).
- The approach introduces a cache, called history, for collecting data about the performance of the models. Later in theprotocol a mechanism decides whether drift occurred.
- Through empirical evaluations we investigated the performance (both for drift handling and detection capabilities) and scalability of the method. Based on these, we pointed out that our current proposal is a suitable choice when we need the additional feature of drift detection, since the drift handling performance of our approach is similar to the state-of-the-art approaches.

Thank you for your time!