Adaptive Sliding Window (ADWIN)

Adaptive sliding window algorithm is automatically maintaining the window of variable size when date change with rigorous guarantees of it performance.

> Algorithm Setting

Input:

- Confidence value δ belong to (0, 1)
- Sequence of real vaues $x_1 , x_2 , x_2 , . . . , x_t , . . .$
- The value of x_t is available only at time t

Assume:

- x_t is always in [0, 1]
- an Interval [a, b] such that $a <= x_t <= b$ with probability 1
- Nothing else is known about the sequence of distributions D_t
- μ_t and σ_t^2 are unknown for all t

The adapting sliding window algorithm called ADWIN suitable for data streams with sudden drift. The algorithm keeps a sliding window *W* with the most recently read examples. The main idea of ADWIN is as follows: whenever two "large enough" subwindows of *W* exhibit "distinct enough" averages, one can conclude that the corresponding expected values are different, and the older portion of the window is dropped. This involves answering a statistical hypothesis: "Has the average μ_t remained constant in *W* with confidence δ "? The pseudo-code of ADWIN is listed below.

> **ADWIN0:** ADAPTIVE WINDOWING ALGORITHM

```
1: Initialize Window $W$
2: for each $t > 0$
3:     do $\{x_t\} \bigcup W \rightarrow W$ (i.e., add $x_t$ to the head of W)
4:         repeat Drop elements from the tail of W
5:             until | $\hat{\mu}_{W_0}$ - $\hat{\mu}_{W_1}$ | < $\epsilon_{cut}$ holds
6:                 for every spilt of $W$ into $W$ = $W_0 W_1$
7:         output $\hat{\mu}_W$
```

The key part of the algorithm lies in the definition of ϵ_{cut} and the test it is used for. The different statistical tests can be used for this purpose, but propose only one specific implementation. Let n denote the size of W, and n_0 and n_1 the sizes of W_0 and W_1 consequently, so that $n = n_0 + n_1$. Let $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ be the averages of the values in W_0 and W_1 , and $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ their expected values. The value of ϵ_{cut} is proposed as follows:

$$\epsilon_{cut} = \sqrt{\frac{1}{2m}\frac{4}{\delta'}}$$

where

$$m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_1}}, and, \delta' = \frac{\delta}{n}$$

The statistical test in line 6 of the pseudo-code checks if the observed average in both subwindows differs by more than threshold ϵ_{cut}. The threshold is calculated using the Hoeffding bound, thus gives formal guarantees of the base classifiers performance. The phrase "holds for every split of W into $W = W_0W_1$" means that we need to check all pairs of subwindows W0 and W1 created by splitting W into two. The verification of all subwindows is very costly due to the number of possible split points. That is why the authors proposed an improvement to the algorithm that allows to find a good cut point quickly. The originally proposed ADWIN algorithms are also lossless learners, thus the window size W can grow infinitely if no drift occurs. This can be easily improved by adding a parameter that would limit the windows maximal size. In its original form, proposed by Bifet, ADWIN works only for 1-dimensional data, e.g., the running error. For this method to be used for n-dimensional raw data, a separate window should be maintained for each dimension. Such a modified model, although costly, reflects the fact that the importance of each feature may change at different pace.

References:

> - A. Bifet, R. Gavalda. (2007). "Learning from Time-Changing Data with Adaptive Windowing". Proceedings of the 2007 SIAM International Conference on Data Mining 443-448.
> - A. Bifet, J. Read, B.Pfahringer.G. Holmes, I. Zliobaite. (2013). "CD-MOA: Change Detection Framework for Massive Online Analysis". Springer Berlin Heidelberg 8207(9): 443-448.

Drift Detection Method (DDM)

Drift Detection Method (DDM) model the number of classification errors with a Binomial distribution. The idea of this method is that in each iteration an online classifier predicts the decision class of an example. That prediction can be either *true* or *false*, thus for a set of examples the error is a random variable from Bernoulli trials.

Let us denote p_i as the probability of a *false* prediction and s_i as its standard deviation calculated as given by Equation (1):

$$s_i = \sqrt{\frac{p_i(1-pi)}{i}} \tag{1}$$

For a sufficiently large number of examples ($n > 30$), the Binomial distribution is closely approximated by a Normal distribution with the same mean and variance. For each example in the data stream the error rate is tracked updating two registers: p_{min} and s_{min}. These values are used to calculate a *warning level* condition presented in Equation 2 and an *alarm level* condition presented in Equation 3. Each time a warning level is reached, examples are remembered in a separate window. If afterwards the error rate falls below the warning threshold, the warning is treated as a false alarm and the separate window is dropped. However, if the alarm level is reached, the previously taught base learner is dropped and a new one is created, but only from the examples stored in the separate warning" window.

$$p_i + s_i \geq p_{min} + \alpha s_{min} \tag{2}$$

$$p_i + s_i \geq p_{min} + \beta s_{min} \tag{3}$$

The value α and β in the above conditions decide about the confidence levels at which the warning and alarm signals are triggered.

> Algorithm Setting

Input:

- S: a data stream of examples
- C: classifier

Output: *W*: a window with examples selected to train classifier C

> DDM: Drift Detection Method

```
1: Initialize $(i, p_i, s_i, ps_{min}, p_{min}, s_{min})$;
2: $newDrift \leftarrow false$;
3: $W \leftarrow \emptyset$;
4: $W' \leftarrow \emptyset$;
5: **for all** examples $x_i \in S$ **do**
6:     **if** prediction $C(x_i)$ is incorrect **then**
7:         $p_i \leftarrow p_i + (1.0 - p_i)/i$;
8:     **else**
9:         $p_i \leftarrow p_i - (p_i)/i$;
10:    compute $s_i$ using (1);
11:    $i \leftarrow i + 1$;
12:    **if** $i > 30$ (approximated normal distribution) **then**
13:        **if** $p_i + p_s \leq ps_{min}$ **then**
14:            $p_{min} \leftarrow p_i$;
15:            $s_{min} \leftarrow s_i$;
16:            $ps_{min} \leftarrow p_i + s_i$;
17:        **if** drift detected (3) **then**
```

```
        Initalize $(i, p_i, s_i, ps_{min}, p_{min}, s_{min}$;
        $W \leftarrow W'$;
        $W' \leftarrow \emptyset$;
    else if warning level reached (2) then
        if $newDrift = true$ then
            $W' \leftarrow \emptyset$;
            $newDrift \leftarrow false$;
        $W' \leftarrow W' \cup x_i$;
    else
        $newDrift \leftarrow true$;
$W \leftarrow W \cup x_i$;
```

References:

- Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004)

Stream Volatility

Current drift detection techniques detect a change in distribution within a stream. However, there are no current techniques that analyze the change in the rate of these detected changes. We coin the term stream volatility, to describe the rate of changes in a stream. A stream has a high volatility if changes are detected frequently and has a low volatility if changes are detected infrequently.

Volatility detection focus on the rate de changes of detected concept drifts in the data stream. Volatility detection works usually in parallel with drift detectors and the two are designed to run concurrently.

To illustrate the differences between drift detection and volatility detection. Firstly, we will formally define these two notions.

Drift Detection: Let $S_1 = (x_1, x_2, \ldots, x_m)$ and $S_2 = (x_{m+1}, x_{m+2}, \ldots, x_n)$ with $0 < m < n$ represent two samples of instances from a stream with population means μ_1 and μ_2 respectively. The drift detection problem is testing the null hypothesis H_0 that $\mu_1 = \mu_2$ against the alternate hypothesis H_1 that they are from different distributions with $\mu_1 \neq \mu_2$. In practice the underlying data distribution is unknown and a test statistic based on sample means needs to be constructed by the drift detector. A false negative occurs when the null hypothesis is accepted incorrectly and a false positive occurs when the alternate hypothesis is accepted incorrectly. The hypothesis test is as follows: we accept hypothesis H_1 whenever $Pr(|\hat{\mu}_1 - \hat{\mu}_2| > \epsilon) > \delta$, where δ lies in the interval (0,1) and is a parameter that controls the maximum allowable false positive rate, while ϵ is is a function of δ and the test statistic used to model the difference between the sample means.

Volatility Detection: Let $C_1 = (c_1, c_2, \ldots, c_k)$ and $C_2 = (c_{k+1}, c_{k+2}, \ldots, c_t)$ represent a sample of cut points detected from a stream. p_i represent the distance intervals (periods) between two consecutive cut points c_i and c_{i-1}. We are able to derive volatility windows $P_1 = (p_1, p_2, \ldots, p_k)$ and $P_2 = (p_{k+1}, p_{k+2}, \ldots, p_{t-1})$ with sample variance of σ_1 and σ_2. The volatility detection problem can be expressed as testing $\frac{\sigma_1}{\sigma_2} \lessgtr 1.0 \pm \beta$, where β is a user-expressed tolerance threshold. If the test holds true we say that there is a shift in volatility between the two samples.

> Algorithm Setting

Input

- A sequence of real values $p_1, p_2, \ldots, p_t$ representing the distance intervals between cut points discovered by drift detection techniques.

Output

- Shift points of stream volatility

> Stream Volatility Detector

```
1: Initialize buffer $B$ and Reservoir $R$;
2: Boolean: volatilityShift $\leftarrow$ false
3: **for each** $t > 0$ **do**
4:     $j \leftarrow$ addToBuffer($x_t, B$);
5:     addToReservoir($j, R$);
6:     RelativeVariance $\leftarrow \frac{\sigma_B}{\sigma_R}$;
7:     **if** Relative Variance $\lessgtr 1.0 \pm \beta$ **then**
8:         volatilityShift $\leftarrow$ True;
9:     **end**
10: **end**
11: **Funtion** addToBuffer(item $k$, Buffer $B$)
12:     add $k$ as tail of $B$;
13:     return head of $B$;
14: **end**
15: **Function** addToReservoir(iterm $k$, Reservoir $R$)
16:     rPos $\leftarrow random()$;
17:     R[rPos] $\leftarrow k$;
18: **end**
```

There are two main components in the volatility detector: a buffer and a reservoir. The buffer is a sliding window that keeps the most recent samples of drift intervals acquired from a drift detection technique. The reservoir is a pool that stores previous samples which ideally represent the overall state of the stream.

The progression of the volatility detector when inputs come in is as follows: First, when input x_t arrives at position t, it is first moved into the buffer where a sliding window keeps recent samples. As the sliding window slides, the oldest entry in the buffer is dropped from the buffer and moved into the reservoir, then the reservoir stores the dropped entry by randomly replacing one of its stored samples. Lastly, the detector then compares the samples in the buffer to the samples in the reservoir to analyze for differences. The primary difference between the buffer and the reservoir is that the buffer always keeps the most recent contents of the stream whereas the reservoir keeps an overall view of the stream. For a change in relative volatility to be detected, we use the Relative Variance measure. Relative Variance at a particular point is calculated as $\frac{\sigma_B}{\sigma_R}$, where σ_B is the variance calculated using the samples in the buffer and σ_R is the variance calculated using the samples in the reservoir.

Reference

> - Huang, D.T.J., Koh, Y.S., Dobbie, G., Pears, R.: Detecting volatility shift in data streams. In: 2014 IEEE International Conference on Data Mining (ICDM), pp. 863–868 (2014)