Predicting Concept Drift Severity

Ruolin Jia
The University of Auckland,
New Zealand
rjia477@aucklanduni.ac.nz

Yun Sing Koh
The University of Auckland,
New Zealand
ykoh@cs.auckland.ac.nz

Gillian Dobbie
The University of Auckland,
New Zealand
gill@cs.auckland.ac.nz

Abstract

In data streams, concept drift severity refers to the amount of change of a concept. Low severity concept drifts are hard to detect using drift detectors, and in these cases, a high detection sensitivity is expected. However, high detection sensitivity comes with a higher false positive rate that degrades the performance of the drift detector. In this paper, we present PRESS (PREdictive Severity Seed) detector. PRESS learns the severity trends of a stream with recurrent volatility (the frequency of experiencing concept drifts) and predicts the severity of future concept drifts with a probability network. Experiments show that PRESS outperformed other existing drift detectors by reducing false positive rates while maintaining true positive rates. To the best of our knowledge, it is the first algorithm predicting the severity of concept drifts to improve the detection performance. In addition, we provide a simple yet effective severity measurement method, which does not require a detection warning period, for Seed and ADWIN detectors.

1 Introduction

In real-world cases, data streams experience frequent change of the underlying distribution, and this is known as concept drift. To mine a data stream, it is important to detect concept drifts to maintain the quality of the models. Various drift detection methods [Page, 1954; 1954; Gama *et al.*, 2004; Baena-Garcıa *et al.*, 2006; Bifet and Gavalda, 2007] have gained attention. Concept drifts have different properties that can characterise them. One property is concept drift volatility that represents the frequency of experiencing concept drifts. The other property is the amount or magnitude of a concept drift, known as drift severity [Kosina *et al.*, 2010]. Different levels of drift severity can influence the behaviour of a drift detector, thus tracking drift severity of a data stream is necessary. Although various methods have been proposed to track drift severity [Kosina *et al.*, 2010; Chen *et al.*, 2015; Webb *et al.*, 2016], until now, there are no existing methods for predicting the severity trends in a stream.

In this paper, we investigated how drift severity affects the behaviours of drift detectors. We found that it is difficult for current well-known drift detectors such as Seed [Huang *et al.*, 2014] and ADWIN [Bifet and Gavalda, 2007] to detect concept drifts if the drifts have low severity in the stream. In this case, a detector with high detection sensitivity is needed. The tradeoff of a current detector with high sensitivity is it may result in more false alarms. To overcome these challenges, we proposed the PRESS (PREdictive Severity Seed) drift detector. We assume that drift severity follows the same distribution if the frequency of occurrences (volatility) is constant. PRESS has two phases: training phase and testing phase. In the training phase, PRESS uses a probability network to learn the volatility changing patterns of a stream. Then, for each volatility pattern, it samples the corresponding severity values. In the testing phase, by using learned severity values, PRESS applies high detection sensitivity only when the severity is low. This technique has been shown, in most cases, to effectively reduce false positive rates compared with existing detectors while maintaining similar true positive rates and detection delays. In addition to PRESS, we also proposed a novel way of measuring drift severity for ADWIN and Seed detectors.

PRESS works under the assumption that many data streams concept drifts in the real-world follow behavioural patterns. This characteristic has been exploited by previous research in the area method to capture the regularity of these concept drifts [Kosina *et al.*, 2010; Chen *et al.*, 2015]. In such real-world cases with regular concept drifts, each concept drift is caused by some events. For example, in a manufacturing production line, concept drifts can arise due to the regular machine faults and these events are also regular based on the life span of the machine parts. This assumption exist in many different real world application that follows pattern regularity. Concept drift caused by the same events also share the same severity. In the case of regular concept drifts, the severity and volatility both follow some distribution patterns. In the event that no behavioural patterns exist, PRESS defaults to using a base state-of-the-art detector, where the accuracy of its results are no worse than the current base technique such as Seed.

In Section 2 we review data stream drift detection. In Section 4, we discuss background before presenting our methods. In Sections 5 and 6, we describe our PRESS algorithm and demonstrate its performance through experimentation. We then conclude our paper.

2 Related Work

In data stream mining, concept drift occurs when the underlying distribution of data streams changes. Formally, a concept drift is a change in the joint probability $P(\vec{x}, y) = P(y|\vec{x}) \times P(\vec{x})$ in which $\vec{x}$ is the input attributes vector and y is the dependent class label [Gama, 2010]. Drift detectors are applied to detect when a concept drift occurs. Detectors monitor the stream of some performance indicators, such as the prediction accuracy over time, of the predictor and alarm a drift when finding a change in the corresponding indicator [Klinkenberg and Renz, 1998]. Several drift detection algorithms [Page, 1954; Gama *et al.*, 2004; Baena-Garcıa *et al.*, 2006; Bifet and Gavalda, 2007] monitor such streams of indicators. In reality, these detectors often monitor the error stream of a predictor and signal a drift when the error increases significantly. Because drift detectors cannot make the perfect approximation of a stream and real-world streams are largely susceptible to noise, detectors may signal a false alarm, known as false positive, when in fact, there is no actual concept drift in the stream. To reduce false positive rates, Huang et al. [2015] use stream volatility, a measure of how frequent the concept drift occurrence is in a stream, to capture the trend of concept drifts. Based on these trends, their algorithms predicts the future drift positions and probability. Then it reduces the sensitivity of the drift detector when the concept drift is unlikely to happen. Thus, the false positive rate is reduced. An algorithm sharing the same idea is ProSeed [Chen *et al.*, 2016]. It builds a probability network to capture the recurrent volatility of a data stream that transits among a few volatility patterns. Then it uses the network to predict the position of future drifts and only tries to find a drift in those instances where drifts are likely.

A data stream can experience different amounts of drift represented as drift severity. Kosina et al. [2010] presented a metric for severity measurements: when the stream with input space S changes from distribution C_1 to C_2, the percentage of S in C_2 with different class labels from it in C_1 is the drift severity. To estimate this percentage, they enable a warning period of the drift detector when it detects the stream is going to, but yet to, experience a drift, and it counts the number of misclassifications in this period. After a drift is detected, it computes the drift severity as

$$Severity = \frac{\#misclassified}{\#instances\ in\ warningperiod}.$$

Generally, this approach measures the amount of change in the one-dimensional indicator stream of the classifier. Additionally, to apply this metric measurement, one also needs to enable the existing detector to detect the warning period. Kosina et al. [2010] use two different thresholds: the drift warning can be detected with the lower threshold before detecting an actual drift. Recently Chen et al. [2015] proposed the MagSeed algorithm that is a warning level enabled version of the original Seed detector. Similarly, it applies a relaxed threshold to detect a warning level prior to detecting the actual drift with the drift threshold. Then it calculates the severity of a stream. However, to the best of our knowledge, all current approaches for tracking drift severity only capture the severity of previous drifts, and they do not proactively predict the severity trends.

3 Severity Detection Method Without Warning Periods

In this paper, we follow the general definition of concept drift severity proposed by Webb et al. [2016]. At any time t and u, the concept drift severity can be calculated by $Severity_{t,u} = D(t, u)$ where D is some distance function returning a non-negative value. For example, one can compute the joint probability distance between $P_t(X, Y)$ and $P_u(X, Y)$ to show the concept drift magnitude in a classification task.

Methods building upon the severity metric by Kosina et al. [2010] requires a warning period. A major limitation of the warning period is that one needs to set an appropriate warning threshold. A poorly set threshold may result in either the detector directly signalling a drift without encountering any warning periods, or it enters the warning period that is not followed by an actual drift (false warning alarm). In this section, we present a method that is suitable to be used with current detectors, such as ADWIN and Seed. Our method detects concept drift severity without requiring a warning period. Our technique compares the distance between two concept drifts in a predictor's indicator stream, for example, an error stream. We calculate the distance between stream distributions after two consecutive drifts d_1 and d_2 as the drift severity of drift d_2.

ADWIN and Seed both signal a drift when they detect a significant difference between two sub-window W_1 and W_2, where W_2 represents the most recent instances after a drift occurs. Whenever a drift happens, we store the mean of all instances in W_2 in the variable called "snapshot" S. When a new drift is detected, we then compute the severity (the distance) as $|\overline{W_2} - S|$, and then assign the mean of all instances in W_2 to S. We repeat this process when the detector performs its task.

4 Relationship between Severity and Detector Performances

Different levels of severity affect a drift detector's performance. We show the influences of these through experimentation. We use Seed and ADWIN detectors with different confidence levels as the base drift detector for this experiment. We run those two detectors on our synthetic data streams. The detail about generating these data streams are discussed in Section 6.

To show our findings, we plot the relationship between severity v.s. false negatives and detection delays for the Seed detector in Figure 1. We use different colours to represent changing confidence levels. The experiment concludes that, a concept drift with higher severity is "easier" to be detected when the severity is relatively high, because these experiments show that the detector experiences lower delay and false negative rate when processing a high severity drift. Thus, it is not necessary to set a relatively high detection sensitivity in the high severity case, given that a high detection sensitivity will result in an increasing false positive rates. A

similar phenomenon was also seen using the ADWIN detector.

In general, the result suggests that it is feasible to attempt to improve the performance of drift detection by reducing the sensitivity of the drift detector when the severity is high.

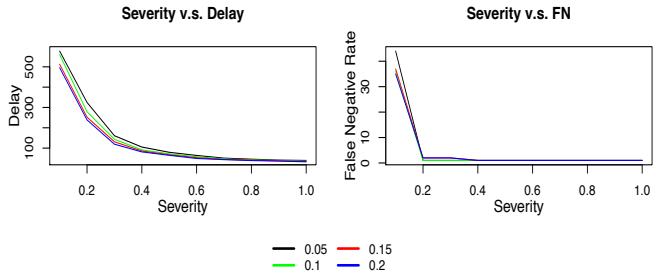

Figure 1: Severity and Drift Detector Behaviour

5 PRESS Algorithm

In this section we review the drift detector SEED, which we use in conjunction with our method to demonstrate the advantages of PRESS. We then detail our PRESS algorithm.

5.1 Review of Seed

Because PRESS uses the Seed [Huang *et al.*, 2014] detector as its component, we firstly review Seed. The Seed algorithm uses a sliding window storing the most recent instances seen in a stream. In addition, it accumulates instances in the window into fixed-size blocks. Seed assumes that instances of one block follow the same concept. To detect a drift, Seed compares the statistical difference between each of all the two possible sub-windows containing those blocks and signals a drift when the difference is greater than the threshold. For example, if Seed has three blocks B_1, B_2, B_3. It compares the difference between sub-window $\{B_1\}$ and sub-window $\{B_2, B_3\}$, and between $\{B_1, B_2\}$ and $\{B_3\}$.

In any partition of Seed's window, the Seed detector alarms a drift when $|\mu_{\hat{W}_0} - \mu_{\hat{W}_1}| > \epsilon$ where $\mu_{\hat{W}_0}$ and $\mu_{\hat{W}_1}$ are two sub windows' means and ϵ is a threshold value calculated by:

$$\epsilon = \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot \ln \frac{2}{\delta'}} + \frac{2}{3m} \cdot \ln \frac{2}{\delta'},\ \delta' = \frac{\delta}{n},$$
$$m = \frac{1}{1/n_0 + 1/n_1}$$

where σ_W^2 is the variance of the window; n_0 and n_1 are instance counts of two sub-windows of the partition; δ is a user set confidence level in range $(0, 1)$.

5.2 PRESS

In this section, we present the PRESS (PREdictive Severity Seed) algorithm.

We make the following assumption about the drift severity: in a stream fluctuating among a few fixed volatility patterns, if the volatility changes from a pattern p_1 to a pattern p_2 at time t_1, the severity of p_2 has the same distribution as p_2 when it changes from p_1 at another time t_2. In short, the most recent volatility transition determines the current pattern's concept drift severity at any time. In this case, severity changes also follow some fixed patterns. The assumption is reasonable if the same volatility transitions are caused by the same event. For example, the data generated by a manufacturing equipment monitoring sensor may experience recurrent volatility because the equipment may experience regular faults. A faulty event will cause a volatility transition such that the concept drift is becoming more frequent. In addition, upcoming concept drifts in the new volatility pattern are caused by the same factor - the faulty event, so they will share the same concept severity.

We use an example shown in Figure 2 to discuss our technique. In the two plots, the time element is signalled by the arrival of new instances. In the top plot (volatility v.s. instance), we observed that the volatility pattern changes among three patterns p_1, p_2, and p_3. Whenever we experience the same pattern transition we assume that the severity after the transition would be similar. In our example, there are two transitions from p_1 to p_2, so in the bottom plot (severity v.s. instance), the severity level denoted by "p_1 to p_2" in both of the cases are similar.

The intuition of PRESS is to learn the volatility transition using a probability network, and sample the concept drift severity for each volatility pattern by using reservoir sampling [Vitter, 1985]. Then, PRESS predicts the severity of future drifts and adjusts the Seed detector threshold correspondingly, so that it reduces the detector sensitivity by augmenting the threshold when the predicted severity is relatively low.

PRESS uses the volatility change detector presented by Huang et al. [2014]. It takes input of drift intervals. Whenever a drift is detected, it calculates the interval between the most recent drift and its previous drift. Then it inputs the interval value to the volatility change detector. The detector will alarm a change of volatility when it finds a significant difference between recent intervals and previous ones.

PRESS has a training phase and a testing phase. In the training phase, PRESS learns a probability network of stream volatility changes using the same approach as Chen et al. [2016]. In the network, each node is a certain volatility pattern containing samples from the distribution of one volatility state. One example of the pattern is $p = \{110, 108, 99\}$. Each number in p denotes the interval between two drifts. All numbers in p are assumed to be from one volatility distribution. Each edge of the network represents the probability of transiting from one volatility state to another state. Whenever a volatility change is detected, the detector can also return the most recent volatility pattern. Then, the most recently seen volatility pattern is compared with each pattern in the network using the Kolmogorov-Smirnov test. If the most recently seen pattern is not similar to any pattern in the network, we add it to the network along with a new transition edge. For example, the current network contains two patterns $p_1 = \{110, 108, 99\}$ and $p_2 = \{210, 190, 200\}$. The stream is currently in pattern p_2. When the volatility change detector detects a new pattern $p_3 = \{500, 490, 470\}$, it decides that p_3 is not statistically

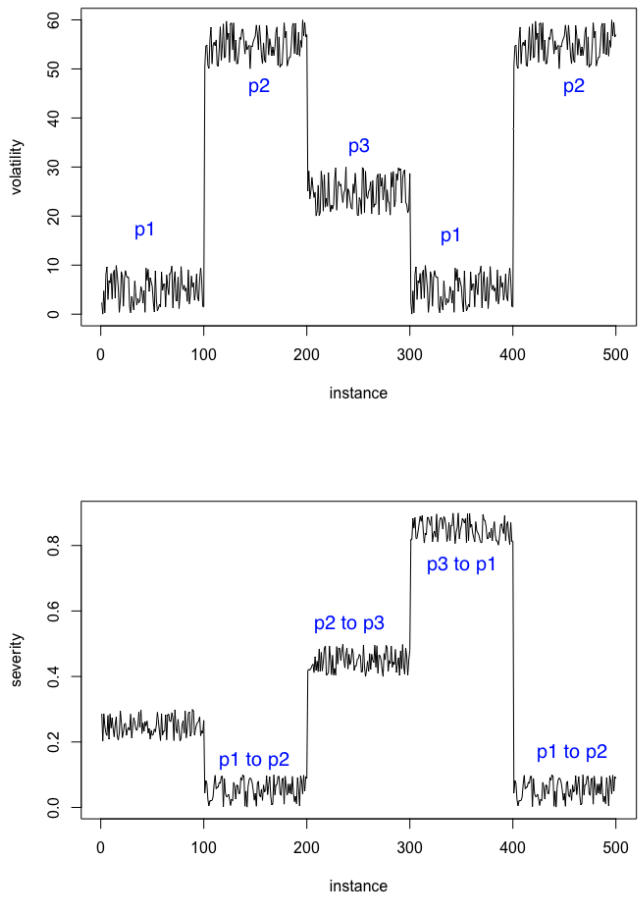

Figure 2: Relationship of Volatility and Severity

similar to any existing pattern, so p_3 will be added to the network and we add the transition edge from p_2 to p_3. In the other case, if we find a match of patterns, we simply insert samples of the most recently seen intervals to the matched patterns in the network and update the transition probability. By default, each pattern in the network can have up to 100 intervals. If the pattern is full, the oldest intervals are deleted to create space for new ones. The probability update is calculated based on the frequency of transitions. For example, we have seen that pattern p_2 has transited 3 times to p_3 and 2 times to p_1, so the transition probability from p_2 to p_3 and p_1 are 0.6 and 0.4 respectively.

When learning the network, PRESS also maintains a severity buffer of severity and a set of reservoirs denoted by $R(x, y)$ where x and y are any two volatility patterns. The severity buffer is a sliding window storing recently seen concept drift severity. It records a severity value whenever a drift occurs. When a volatility change is detected, PRESS inserts all samples in the buffer to $R(p_j, p_i)$ where p_i is the most recently seen pattern and p_j is the pattern seen before p_i. Then, it clears the buffer and repeats the process. For example, before detecting a volatility change, the stream is in pattern p_1, and the volatility change detector detects a volatility change to p_2, we add all severity values in the buffer to $R(p_1, p_2)$. PRESS uses the severity detection method in Section 3.

In the testing phase, PRESS is able to predict the severity of future concept drifts by using the network and severity reservoirs. It adjusts the threshold based on the predicted future severity. If the predicted severity in upcoming drifts is relatively high, it reduces the sensitivity by increasing the detection threshold. We calculate the threshold ϵ of PRESS using the following formula:

$$\epsilon = c \cdot \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot \ln \frac{2}{\delta'}} + \frac{2}{3m} \cdot \ln \frac{2}{\delta'}$$

This is a variation of the Hoeffding bound. Compared with the previous Seed detector's threshold formula, it has an additional c coefficient multiplying its first term. c is the threshold augmenting coefficient computed by $c = 1 + \beta \cdot \phi$. In this formula, β is a user set parameter controlling the magnitude of threshold augmentation. ϕ is the predicted severity value in range $(0, 1)$.

Algorithm 1: Learning Phase

input : S: Stream;
Let $Seed$ be the Seed Detector;
Let $VolDect$ be the Volatility Change Detector;
Let B be the severity buffer;
Let $R(x, y)$ denotes the severity reservoir of pattern x if x is transited from pattern y;
Let N be the probability network;
begin
  foreach $s \in S$ **do**
    Input s in $Seed$;
    if $Seed$ *detects a drift* **then**
      Let i be the interval between the most recent two drifts;
      Input i in $VolDect$;
      Compute the drift severity sv and input sv in B;
      if $VolDec$ *detects a volatility change* **then**
        Let p_i be the most recent seen pattern;
        Let p_j be the pattern seen before p_i;
        if *node* p_i *does not exist in* N **then**
          add node p_i to N;
          add edge $E(p_j, p_i)$ to N;
        increment probability of $E(p_j, p_i)$;
        if $R(p_j, p_i)$ *does not exist* **then**
          Create empty reservoir $R(p_j, p_i)$;
        Input all instances of B to $R(p_j, p_i)$;
        Clear B;
  end for
end

Predicted severity ϕ is the expected severity value of the

Algorithm 2: Predicting Phase

input : S: Stream;
Let $Seed$ be the Seed Detector;
Let $VolDect$ be the Volatility Change Detector;
Let B be the severity buffer;
Let $R(x, y)$ denotes the severity reservoir of pattern x if x is transited from pattern y;
Let $\overline{R}(x, y)$ denotes the mean of $R(x, y)$;
Let $Prob(y|x)$ denotes the probability of transiting to y given pattern x;
Let β be a parameter in range $(0, 1)$;
Let $Seed.c$ be the threshold coefficient of $Seed$ detector;
begin
    Normalise all severity samples in $R(x, y)$ in range $(0, 1)$ for any pattern x and y;
    foreach $s \in S$ **do**
        Input s in $Seed$;
        if $Seed$ *detects a drift* **then**
            Let i be the interval between the most recent two drifts;
            Input i in $VolDect$;
            if $VolDec$ *detects a volatility change* **then**
                Let p_i be the most recent seen pattern;
                Let H be the set of nodes where $Prob(H_j|p_i) > 0$;
                Set $\phi = \sum_{j=1}^{|H|}(Prob(H_j|p_i) \cdot \overline{R}(p_i, H_j))$;
                Set $Seed.c$ as $(1 + \beta \cdot \phi)$;
    end for
end

incoming concept drifts calculated by:

$$\phi = \sum_{j=1}^{k}(Prob(H_j|p_i) \cdot \overline{R}(p_i, H_j))$$

where p_i is the most recently seen volatility pattern and H is the set of all patterns transited from p_i obtained from the probability network. $Prob(y|x)$ denotes the probability of transiting to pattern y given the current pattern x. $\overline{R}$ is the mean of the severity reservoir. We show the pseudo-code in Algorithms 1 and 2.

The learning phase can be ran online. Every time a new instance arrives, the probability network structure is updated. The drawback of this is a slightly high computational cost. In reality batch processing, *i.e.* only updating the network after a number of instances, can be applied because it is likely that the same concept drift pattern will be constant for a certain period and it is unnecessary to update the network for every instance. In batch processing, we can drop also the old network after a period and then re-train a new one, so the algorithm will switch between the training phase and testing phase, to capture the most recent models.

Caveat: PRESS algorithm works well when there are some regularity of concept drift patterns. However it may not work well on some data streams, especially those without regular concept drift patterns. In these cases, the results will default to the base detector used. Thus performing no worst in terms of accuracy when compared current state-of-the-art techniques. Despite this many data streams concept drifts in the real-world follow behavioral patterns. In such real-world cases with regular concept drift, each concept drift is caused by some event (for example, concept drifts arise due to regular machine faults) and these events are also regular. Concept drift caused by the same event also share the same severity. So in the case of regular concept drifts, the severity and volatility both follow some distribution.

6 Experiments

Our experiments compare the performances of four drift detectors: PRESS, ProSeed, Seed, ADWIN. We test each detector with confidence level of 0.05, 0.1, 0.15, 0.2, 0.25. We carried out testing in both real world and synthetic datasets.

6.1 Synthetic Data

In this set of experiments, we tested our drift detectors with our synthetic data. Our synthetic generator outputs a series of values simulating the performance indicator of the predictor. The output of the stream is generated by formula $\mu+G$ where μ is an integer and G is a noise term, a Gaussian random variable with mean 0 and standard deviation 1.

To create drifts with different severity, we add different magnitudes of change to μ. We create datasets with both abrupt drift and gradual drift. Our synthetic datasets contain drifts with recurrent volatility and severity. For both abrupt and gradual concept drift streams, we generate datasets with different amounts of volatility patterns including 3 patterns, 5 patterns, and 10 patterns. The severity of the concept drift is determined by the transition of the previous pattern to the current pattern. (We follow the assumption in Section 5.2). Each dataset contains 1 million instances with 50000 concept drifts. We generate 50 copies for each dataset with different random seeds.

We measured False Positive Rate (FP), True Positive Rate (TP) and Delay. False positives is the count of false alarms in in a stream. The true positive is the count of actual drift that is successfully detected. Delay is the average count of instances between the actual drift place and the time of detecting it. For PRESS, we set $\beta = 0.4$. We set 32 as the block size for Seed and the volatility change detector.

The numbers in all results tables are average results across the 50 runs, and the standard deviation of the results are provided in the round brackets.

6.2 Evaluation of the True Positive Rate (TP) and Delay

We firstly evaluate the True Positive Rate (TP) and detection delay among four algorithms. We expect PRESS to have similar TP and Delay to other algorithms. TP in results tables is a rate of corrected detected concept drifts out of all real concept drifts in a stream.

For abrupt drift, we show the results for 10 patterns (Table 1). We note that the results for 3 and 5 patterns' results follow the same trend. We observe perfect TP (approximately

Table 1: 10 patterns, abrupt change

PRESS			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	1.28×10^{-4} (8×10^{-6})	100 (0.0014)	19.54(0.2)
0.10	1.57×10^{-4} (1.3×10^{-5})	100 (0.0016)	19.04(0.15)
0.15	1.98×10^{-4} (1.4×10^{-5})	100 (0.0016)	18.86(0.13)
0.20	2.38×10^{-4} (2.3×10^{-5})	100 (0.0015)	18.63(0.17)
0.25	2.84×10^{-4} (1.7×10^{-5})	100 (0.0018)	18.58(0.08)
ProSeed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	1.25×10^{-4} (7×10^{-6})	100 (0.0016)	18.98(0.16)
0.10	1.89×10^{-4} (1.9×10^{-5})	100 (0.002)	18.53(0.16)
0.15	2.65×10^{-4} (1.3×10^{-5})	100 (0.0025)	18.43(0.09)
0.20	3.56×10^{-4} (2.1×10^{-5})	100 (0.0014)	18.31(0.12)
0.25	4.57×10^{-4} (2.5×10^{-5})	100 (0.003)	18.18(0.16)
Seed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	5.15×10^{-4} (4×10^{-4})	100 (0.0014)	18.87(0.2)
0.10	6.11×10^{-4} (4.33×10^{-4})	100 (0.0015)	18.48(0.16)
0.15	7.51×10^{-4} (5×10^{-4})	100 (0.002)	18.39(0.12)
0.20	9.08×10^{-4} (5.67×10^{-4})	100 (0.002)	18.22(0.16)
0.25	1.08×10^{-3} (6.41×10^{-4})	100 (0.0022)	18.1(0.21)
ADWIN			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	7.27×10^{-4} (8×10^{-6})	100 (0)	17.76(0.08)
0.10	1.02×10^{-3} (1.1×10^{-5})	100 (0.0008)	17.62(0.15)
0.15	1.38×10^{-3} (1.3×10^{-5})	100 (0.001)	17.34(0.16)
0.20	1.81×10^{-3} (1.1×10^{-5})	100 (0.0011)	17.29(0.13)
0.25	2.28×10^{-3} (1.8×10^{-5})	100 (0.001)	17.16(0.15)

Table 2: 3 patterns, gradual change

PRESS			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	3.73×10^{-3} (1.6×10^{-5})	84.69 (0.54)	58.56(0.38)
0.10	3.71×10^{-3} (5.2×10^{-5})	87.5 (0.85)	57.7(0.4)
0.15	3.72×10^{-3} (2.5×10^{-5})	88.76 (0.88)	57.1(0.41)
0.20	3.76×10^{-3} (5.4×10^{-5})	89.63 (0.67)	56.88(0.35)
0.25	3.76×10^{-3} (3.6×10^{-5})	90.62 (0.91)	56.26(0.5)
ProSeed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	3.67×10^{-3} (1.6×10^{-5})	90.77 (0.42)	56.63(0.43)
0.10	3.90×10^{-3} (1.6×10^{-5})	93.24 (0.49)	55.37(0.36)
0.15	4.22×10^{-3} (2.2×10^{-5})	94.35 (0.25)	54.5(0.32)
0.20	4.59×10^{-3} (2.9×10^{-5})	95.21 (0.28)	53.64(0.4)
0.25	5.02×10^{-3} (2.8×10^{-5})	95.64 (0.3)	53.08(0.53)
Seed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	4.17×10^{-3} (5.14×10^{-4})	91.08 (0.6)	56.43(0.41)
0.10	4.50×10^{-3} (6.2×10^{-4})	93.44 (0.44)	55.13(0.41)
0.15	4.91×10^{-3} (7.13×10^{-4})	94.56 (0.31)	54.33(0.34)
0.20	5.3×10^{-3} (7.95×10^{-4})	95.26 (0.29)	53.57(0.43)
0.25	5.87×10^{-3} (8.72×10^{-4})	95.81 (0.28)	52.85(0.46)
ADWIN			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	1.09×10^{-2} (1.08×10^{-4})	91.39 (0.39)	56(0.32)
0.10	1.07×10^{-2} (1.53×10^{-4})	93.52 (0.33)	54.79(0.62)
0.15	1.07×10^{-2} (7.1×10^{-5})	94.95 (0.20)	53.45(0.25)
0.20	1.10×10^{-2} (8.4×10^{-5})	95.67 (0.18)	52.86(0.38)
0.25	1.13×10^{-2} (1.2×10^{-4})	96.16 (0.22)	52.36(0.53)

Table 3: 5 patterns, gradual change

PRESS			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	2.613×10^{-3} (2.7×10^{-5})	97.76 (0.57)	50.08(0.62)
0.10	2.729×10^{-3} (4×10^{-5})	98.54 (0.28)	48.53(0.38)
0.15	2.862×10^{-3} (4.3×10^{-5})	98.64 (0.38)	47.84(0.46)
0.20	2.951×10^{-3} (2.3×10^{-5})	98.96 (0.27)	47.04(0.28)
0.25	3.07×10^{-3} (3.9×10^{-5})	98.88 (0.22)	46.69(0.38)
ProSeed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	2.957×10^{-3} (2.8×10^{-5})	98.69 (0.34)	46.94(0.55)
0.10	3.344×10^{-3} (1.5×10^{-5})	99.02 (0.18)	45.56(0.46)
0.15	3.739×10^{-3} (2.5×10^{-5})	99.1 (0.11)	44.86(0.39)
0.20	4.162×10^{-3} (1.5×10^{-5})	99.32 (0.09)	44.06(0.38)
0.25	4.641×10^{-3} (3.1×10^{-5})	99.44 (0.14)	43.47(0.51)
Seed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	3.383×10^{-3} (4.37×10^{-4})	98.64 (0.31)	46.82(0.59)
0.10	3.842×10^{-3} (5.12×10^{-4})	99.05 (0.19)	45.5(0.48)
0.15	4.334×10^{-3} (6.11×10^{-4})	99.16 (0.15)	44.81(0.35)
0.20	4.825×10^{-3} (6.81×10^{-4})	99.36 (0.12)	43.92(0.43)
0.25	5.354×10^{-3} (7.32×10^{-4})	99.43 (0.13)	43.34(0.49)
ADWIN			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	6.685×10^{-3} (7×10^{-5})	98.82 (0.16)	45.94(0.34)
0.10	6.881×10^{-3} (1.04×10^{-4})	99.01 (0.23)	44.75(0.46)
0.15	7.093×10^{-3} (1.02×10^{-4})	99.29 (0.20)	43.57(0.43)
0.20	7.453×10^{-3} (8.9×10^{-5})	99.32 (0.14)	42.96(0.4)
0.25	7.827×10^{-3} (1.04×10^{-4})	99.46 (0.15)	42.3(0.52)

Table 4: 10 patterns, gradual change

PRESS			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	2.917×10^{-3} (4.5×10^{-5})	99.13 (0.2325)	41.56(0.52)
0.10	3.093×10^{-3} (5.8×10^{-5})	99.46 (0.2108)	40.26(0.47)
0.15	3.226×10^{-3} (4.7×10^{-5})	99.55 (0.1013)	39.68(0.5)
0.20	3.316×10^{-3} (4.5×10^{-5})	99.69 (0.0839)	39.01(0.5)
0.25	3.44×10^{-3} (6.8×10^{-5})	99.58 (0.1767)	38.83(0.27)
ProSeed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	3.331×10^{-3} (3.1×10^{-5})	99.43 (0.1563)	39.31(0.45)
0.10	3.695×10^{-3} (3.9×10^{-5})	99.67 (0.111)	37.87(0.5)
0.15	4.101×10^{-3} (3.7×10^{-5})	99.71 (0.1142)	37.49(0.29)
0.20	4.514×10^{-3} (3.4×10^{-5})	99.72 (0.0644)	36.92(0.41)
0.25	4.98×10^{-3} (5.5×10^{-5})	99.79 (0.0753)	36.56(0.46)
Seed			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	3.793×10^{-3} (4.76×10^{-4})	99.42 (0.1594)	39.18(0.54)
0.10	4.209×10^{-3} (5.29×10^{-4})	99.65 (0.0919)	37.94(0.49)
0.15	4.67×10^{-3} (5.85×10^{-4})	99.7 (0.0961)	37.5(0.34)
0.20	5.143×10^{-3} (6.47×10^{-4})	99.75 (0.0682)	36.85(0.43)
0.25	5.64×10^{-3} (6.79×10^{-4})	99.79 (0.0947)	36.44(0.57)
ADWIN			
δ	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.05	6.21×10^{-3} (6.4×10^{-5})	99.49 (0.1018)	37.66(0.25)
0.10	6.422×10^{-3} (1.06×10^{-4})	99.57 (0.1689)	37.02(0.42)
0.15	6.651×10^{-3} (1.21×10^{-4})	99.75 (0.1128)	36.03(0.48)
0.20	7.031×10^{-3} (1.08×10^{-4})	99.74 (0.0652)	35.68(0.41)
0.25	7.42×10^{-3} (1.2×10^{-4})	99.78 (0.1062)	35.16(0.49)

100%) for all algorithms, so we conclude that PRESS has a similar TP to others. In addition, all four algorithms have similar detection delays. For gradual drifts, we show results for 3, 5, and 10 patterns (Tables 2 to 4) , we observed that all detectors have a similar detection delay.

In these experiments, we ran 1 million training instances to learn the network to train the the PRESS network. Then we switch PRESS to the testing phase, we ran it on the subsequent incoming 1 million instances generated by using the same random seed. In the real-world application, deciding when to switch the phase depends on the data stream, we suggest to drop the old network and re-initiate the training phase when the detector's performances drop, for example, when the overall false positive rates starts to increase. Alertnatively, a weighted mechanisme for expiring older informa-

Table 5: False Postive Results on 3, 5, 10 Patterns

δ = 0.05, abrupt changes			
Patterns	3 patterns	5 patterns	10 patterns
PRESS	4.6×10^{-5} (1.9×10^{-5})	1.1×10^{-4} (1.6×10^{-5})	1.28×10^{-4} (8×10^{-6})
ProSeed	8.3×10^{-5} (1.5×10^{-5})	1.2×10^{-4} (5×10^{-6})	1.25×10^{-4} (7×10^{-6})
Seed	5.6×10^{-4} (4.6×10^{-4})	5.2×10^{-4} (4.1×10^{-4})	5.15×10^{-4} (4×10^{-4})
ADWIN	1.24×10^{-3} (2.6×10^{-5})	6.85×10^{-4} (8×10^{-6})	7.27×10^{-4} (8×10^{-6})
δ = 0.25, abrupt changes			
Patterns	3 patterns	5 patterns	10 patterns
PRESS	2.98×10^{-4} (5.5×10^{-5})	3.13×10^{-4} (2.7×10^{-5})	2.84×10^{-4} (1.7×10^{-5})
ProSeed	7.12×10^{-4} (2.1×10^{-5})	6.53×10^{-4} (2.3×10^{-5})	4.57×10^{-4} (2.5×10^{-5})
Seed	1.55×10^{-3} (8.57×10^{-4})	1.42×10^{-3} (7.86×10^{-4})	1.08×10^{-3} (6.41×10^{-4})
ADWIN	2.59×10^{-3} (2.4×10^{-5})	2.3×10^{-3} (2.3×10^{-5})	2.26×10^{-3} (1.8×10^{-5})
δ = 0.05, gradual changes			
Patterns	3 patterns	5 patterns	10 patterns
PRESS	3.73×10^{-3} (1.6×10^{-5})	2.61×10^{-3} (2.7×10^{-5})	2.92×10^{-3} (4.5×10^{-5})
ProSeed	3.67×10^{-3} (1.6×10^{-5})	2.95×10^{-3} (2.8×10^{-5})	3.33×10^{-3} (3.1×10^{-5})
Seed	4.17×10^{-3} (5.14×10^{-4})	3.38×10^{-3} (4.37×10^{-4})	3.79×10^{-3} (4.76×10^{-4})
ADWIN	1.09×10^{-2} (1.08×10^{-4})	6.69×10^{-3} (7×10^{-5})	6.21×10^{-3} (6.4×10^{-5})
δ = 0.25, gradual changes			
Patterns	3 patterns	5 patterns	10 patterns
PRESS	3.76×10^{-3} (3.6×10^{-5})	3.07×10^{-3} (3.9×10^{-5})	3.44×10^{-3} (6.8×10^{-5})
ProSeed	5.02×10^{-3} (2.8×10^{-5})	4.64×10^{-3} (3.1×10^{-5})	4.98×10^{-3} (5.5×10^{-5})
Seed	5.87×10^{-3} (8.72×10^{-4})	5.35×10^{-3} (7.32×10^{-4})	5.64×10^{-3} (6.79×10^{-4})
ADWIN	1.13×10^{-2} (1.2×10^{-4})	7.83×10^{-3} (1.04×10^{-4})	7.42×10^{-3} (1.2×10^{-4})

tion in the network can be used, this allows the information in the network to be up to date.

6.3 Evaluating False Positive Rate (FP)

In the experiments, we show that PRESS reduces the false positive. We compare FP for four detectors in both abrupt and gradual changing different streams with 3, 5, and 10 patterns. We use two confidence levels for each detectors as 0.05 and 0.25. FP is calculated as a percentage out of falsely detected concept drifts out of number of instances in the stream (in our case, the stream has 1,000,000 instances)

The results in Table 5 shows the false positive rate of the detector. The number in the () represents the standard deviation of the results. The performance of PRESS is promising as it has lower FP than Seed and ADWIN in both abrupt and gradual changing streams with 3, 5, and 10 patterns. In addition, PRESS has lower FP than ProSeed in any testing data stream when δ is greater than 0.10. These differences are statistically significant test by unpaired T-Test. We can conclude that PRESS achieves our research objectives: to reduce FP in data streams satisfying our assumptions.

6.4 Influences of β

In this section we show the effect of different values for parameter β. Table 6 shows FP, TP and Delay of PRESS on 10 patterns abrupt change streams. In abrupt changing streams, experiments show that an increasing FP will result in decreasing FP. However, it also comes with an increase in delay. Thus, β controls the trade-off between FP and Delay in PRESS. The TP does not vary significantly with changing β.

We observe similar trends in gradually changing streams, except that TP has an anti-correlation with β. It is because a larger β may result in overly augmented thresholds such that some true drifts cannot be detected. Choosing values for β depends on user scenario. If one considers that a low false positive rate is important, one should use a higher β value. However, a decreasing true positive rate may occur due to the trade-off. Whereas if the data stream is monitoring a critical task and TP is necessary then a lower β value should be used.

Table 6: Effects of different β values

Abrupt Changes			
β	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.10	4.63×10^{-4} (2.8×10^{-5})	99.98 (0.006)	24.14(0.13)
0.20	3.56×10^{-4} (4.2×10^{-5})	99.99 (0.0061)	24.61(0.17)
0.30	3.15×10^{-4} (5.6×10^{-5})	99.99 (0.0051)	24.92(0.2)
0.40	2.98×10^{-4} (5.5×10^{-5})	99.99 (0.0042)	25.26(0.26)
0.50	2.74×10^{-4} (6.1×10^{-5})	99.99 (0.0066)	25.89(0.2)
0.60	2.55×10^{-4} (4.5×10^{-5})	100.00 (0.0056)	26.22(0.16)
Gradual Changes			
β	Avg. FP (SD)	Avg. TP (SD)	Delay (SD)
0.10	4.273×10^{-3} (3.5×10^{-5})	99.77 (0.1)	37.09(0.41)
0.20	3.853×10^{-3} (5.4×10^{-5})	99.76 (0.06)	37.78(0.37)
0.30	3.607×10^{-3} (6.1×10^{-5})	99.75 (0.12)	38.22(0.45)
0.40	3.44×10^{-3} (6.8×10^{-5})	99.58 (0.18)	38.83(0.27)
0.50	3.314×10^{-3} (7.7×10^{-5})	99.42 (0.12)	38.25(0.35)
0.60	3.21×10^{-3} (5.3×10^{-5})	99.24 (0.1)	38.88(0.25)

6.5 Real-World Data

We evaluated the four drift detectors on MOA real-world data streams: Electricity, Forest Covertype and Poker Hand datasets. We use Naïve Bayes Classifier to generate error

streams for three datasets. Error streams are preprocessed. A classification error normally produces a value of "1" and a correct classification produces a value of '0". The error streams were pre-processed, and transformed in to an aggregated error stream based on a specific window size. The window size was set to 100. This meant that every 100 instances of the error stream was aggregated and transformed into an aggregated value. Finally, we obtained three integer streams to test four detectors. These datasets do not contain the actual positions of concept drifts so we only show the number of detected concept drifts in Table 7. PRESS produces the least number of drifts in most confidence levels. The reason we choose Naïve Bayes classifier is that it has been widely used in testing concept drift detectors. It has a reasonable prediction accuracy to test the effectiveness of most concept drift detectors. The real world data used the online training mode.

Table 7: Number of drifts detected in Real-World datasets

Forest Covertype					
δ	0.05	0.1	0.15	0.2	0.25
PRESS	864	864	864	864	864
ProSeed	843	889	922	957	969
Seed	824	872	932	961	989
ADWIN	1428	1472	1516	1561	1573
Electricity					
δ	0.05	0.1	0.15	0.2	0.25
PRESS	18	18	18	18	18
ProSeed	18	20	26	27	29
Seed	68	70	72	77	79
ADWIN	113	116	122	122	122
Poker-Hand					
δ	0.05	0.1	0.15	0.2	0.25
PRESS	1290	1290	1290	1290	1290
ProSeed	1382	1458	1502	1540	1575
Seed	1402	1478	1546	1573	1607
ADWIN	2227	2291	2322	2356	2386

7 Conclusion and Future Work

We investigated how concept drift severity will affect the performance of drift detectors. Based on the findings, we proposed the PRESS detector. PRESS uses a probability network and reservoir sampling to capture volatility and severity trends of a stream. Then, it can predict the severity of future drifts to adjust its detection sensitivity to reduce false positive rates. We compare PRESS with three other drift detectors in both synthetic and real-world datasets. Most experiments showed that PRESS generates the lowest false positive rate while maintaining similar true positive rates and delays to other drift detectors. We proposed a new method for severity detection for both Seed and ADWIN drift detectors and we applied this method in PRESS.

Our current limitation is that PRESS experiences slightly lower true positive rates with gradual concept drifts. In our future work, we want to make improvements on our severity detection method so that it can also work well in the case of gradual concept drift. In addition, we hope to further explore potential ways that drift severity may improve drift detection techniques besides adapting the drift detection threshold.

References

[Baena-Garcıa *et al.*, 2006] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86, 2006.

[Bifet and Gavalda, 2007] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, volume 7, pages 443–448. SIAM, 2007.

[Chen *et al.*, 2015] Kylie Chen, Yun Sing Koh, and Patricia Riddle. Tracking drift severity in data streams. In *Australasian Joint Conference on Artificial Intelligence*, pages 96–108. Springer, 2015.

[Chen *et al.*, 2016] Kylie Chen, Yun Sing Koh, and Patricia Riddle. Proactive drift detection: Predicting concept drifts in data streams using probabilistic networks. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 780–787. IEEE, 2016.

[Gama *et al.*, 2004] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer, 2004.

[Gama, 2010] Joao Gama. *Knowledge discovery from data streams*. CRC Press, 2010.

[Huang *et al.*, 2014] David Tse Jung Huang, Yun Sing Koh, Gillian Dobbie, and Russel Pears. Detecting volatility shift in data streams. In *2014 IEEE International Conference on Data Mining*, pages 863–868. IEEE, 2014.

[Huang *et al.*, 2015] David Tse Jung Huang, Yun Sing Koh, Gillian Dobbie, and Albert Bifet. Drift detection using stream volatility. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 417–432. Springer, 2015.

[Klinkenberg and Renz, 1998] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. *AAAI Workshop on Learning for Text Categorization(Madison, USA)*, pages 33–40, 1998.

[Kosina *et al.*, 2010] Petr Kosina, Joao Gama, and Raquel Sebastião. Drift severity metric. In *ECAI*, pages 1119–1120, 2010.

[Page, 1954] ES Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[Vitter, 1985] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[Webb *et al.*, 2016] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.