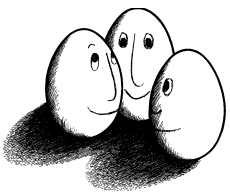
Bachelor-Arbeit

Behandlung von Concept Drift in zyklischen Prozessen

Stefan Rötner



Bachelor-Arbeit
Fakultät Informatik
Technische Universität Dortmund

Dortmund, 11. Juni 2014

Betreuer:

Prof. Dr. Katharina Morik
Dipl.-Inf. Hendrik Blom

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Hintergrund

In vielen Produktionsprozessen werden statistische Lernverfahren verwendet, z.B. um die Güte eines Produktes anhand von aktuellen Produktionsdaten vorherzusagen. Diese Produktionsdaten werden in der Regel von Sensoren in Echtzeit in einer derart hohen Frequenz erfasst, dass schon aufgrund von Speicherbeschränkungen starke Anforderung bezüglich der Verarbeitungszeit bestehen. In der Regel wird daher das zeitintensive Lernen eines neuen Modells auf einem relativ kleinen Trainingsdatensatz durchgeführt und dieses Modell anschließend zur *Online*-Vorhersage der gewünschten Zielgröße verwendet. *Online* bedeutet in diesem Kontext, dass jede Beobachtung sofort verarbeitet wird und nicht für eine spätere Verwendung zur Verfügung steht.

Ein *Concept Drift* bezeichnet eine Veränderung in der Verteilung der Produktionsdaten, die dazu führt, dass die Vorhersagen des gelernten Modells nicht mehr gültig sind, sodass ein neues Modell auf den aktuellen Daten gelernt werden muss. Am Beispiel der Stahlproduktion wären externe Einflussfaktoren, die zu einem *Concept Drift* führen können, ein Wechsel der verwendeten Schrottsorte oder der Verschleiß, der an einer Walze mit der Zeit auftritt.
Insbesondere zyklische Drifts, die sich durch einen Wartungszyklus an einer Produktionsanlage - also z.B. dem Erneuern der verschlissenen Walze - ergeben können, finden in der Forschung wenig Beachtung und sind daher von besonderem Interesse.

Die große Anforderung ist es den richtigen Zeitpunkt zum Neulernen des Modells zu finden. Auf der einen Seite sollen keine Ressourcen verschwendet werden, indem zu früh ein neues Modell gelernt wird. Auf der anderen Seite möchte man aber auch nicht zu lange ein schlechtes Modell verwenden. Wünschenswert ist also ein Verfahren, das die Gesamtfehlerzahl reduziert und die manuelle Analyse reduziert, indem automatisiert ein neues Modell gelernt wird, wenn das aktuelle keine befriedigenden Ergebnisse mehr liefert.

1.2 Aufbau der Arbeit

Zunächst wird Concept Drift formal definiert und ausgehend von dieser Definition werden die verschiedenen Arten von Concept Drift beschrieben. Anschließend werden die theoretischen Grundlagen der gängigen Methoden zur Erkennung und Behandlung von Concept Drift beschrieben und bezüglich der Anforderung eines realen Produktionspro-

zesses bewertet. Hierbei ergibt sich vor allem die Frage, welcher Ansatz für welche Art von Drift besonders geeignet ist und welcher Ansatz in zyklischen Prozessen die besten Ergebnisse liefert. Diese Fragestellung wird im weiteren Verlauf der Arbeit experimentell analysiert.

Zur Untersuchung der unterschiedlichen Methoden bezüglich der einzelnen Drifttypen werden verschiedene Szenarien definiert und ein Datengenerator-Framework entwickelt, mit dem sich diese Szenarien flexibel modellieren lassen. Mittels dieses Frameworks wird eine systematische Untersuchung der vorgestellten Methoden und Szenarien durchgeführt und die wichtigsten Erkenntnisse werden präsentiert.

Die Arbeit endet mit einem Kapitel zur Implementierung, damit einerseits die durchgeführten Experimente nachvollzogen werden können und wiederverwendbare Implementierungsarbeiten dokumentiert werden. So wurde das Datengenerator-Framework unter Verwendung des stream Framework [7] implementiert. Für die Behandlungs- und Erkennungsmethoden wurden soweit möglich bestehende MOA-Implementierungen [3] verwendet und ergänzt und daher eine entsprechende Schnittstelle zum stream Framework geschaffen.

2 Concept Drift

In dieser Arbeit wird maschinelles Lernen im Zusammenspiel mit Datenströmen und Concept Drift betrachtet. Es werden also Verfahren betrachtet, die auf gelabelten Daten ein Modell lernen und auf Grundlage dieses Modells für neu ankommende Daten deren Label vorhersagen. Ändert sich dabei die Verteilung der Daten, sprechen wir von einem *Concept Drift*. Um einen Concept Drift und Datenströme - als die Art von Umgebungen in denen die Behandlung von Concept Drifts untersucht werden soll - genauer und einheitlich zu beschreiben, müssen zunächst einige Begriffe festgelegt werden, für die in der Literatur viele Synonyme verwendet werden.

2.1 Definitionen

Begriff (Datenstrom)**.** Ein Datenstrom ist eine potenziell unendliche Folge von Data Items (x_t), die nur sequentiell verarbeitet werden kann. Dabei ist $t \in T$ und T eine (ggf. abzählbar unendliche) Indexmenge, auf der eine Totalordnung definiert ist. Im Folgenden wird t aber einfach als Zeitpunkt bezeichnet.

Begriff (Data Item)**.** Ein Data Item ist ein n-dimensionaler Vektor $\vec{x}_t \in X^n$. Die n Elemente von $\vec{x}_t$ nennen wir Attribute. Das Paar $(\vec{x}_t, y_t)$ eines Data Items zusammen mit einem Label $y_t \in Y$ nennen wir *Labeled Data Item* . Falls $Y \subseteq \mathbb{R}$ sprechen wir von Regression, falls Y eine diskrete Menge ist, von Klassifikation. Häufig verwendete Synonyme sind z.B. *instance, example, Beispiel* und *Beobachtung.*

In der Regel werden Concept Drift und Verfahren zu dessen Behandlung in Klassifikationsaufgaben betrachtet. Auch in dieser Arbeit werden zunächst die gängigen Behandlungsmethoden im Klassifikationskontext untersucht. Daher werden die Definitionen im weiteren Verlauf dieses Kapitels von Klassifikation ausgehen. Für Concept Drift in Regressionsaufgaben müssen kontinuierliche Verteilungsdichten anstelle von diskreten Wahrscheinlichkeiten betrachtet werden.
Zur Erläuterung des Begriffs *Concept Drift* bietet sich die Definition aus [20] an, da diese besonders dazu geeignet ist, verschiedene Driftursachen zu unterscheiden:

Begriff (Concept Drift)**.** Jedes Data Item $\vec{x}_t$ stammt aus einer zugehörigen Quelle (Verteilung) S_t. Stammen mehrere aufeinanderfolgende Data Items aus gleichen Quellen nennen wir dies Concept : $\mathbf{C} = S_1 = S_2 = \cdots = S_p$. Ein Concept Drift liegt vor, wenn für zwei Zeitpunkte i und j $S_i \neq S_j$ (vgl. Abb. 2.1).

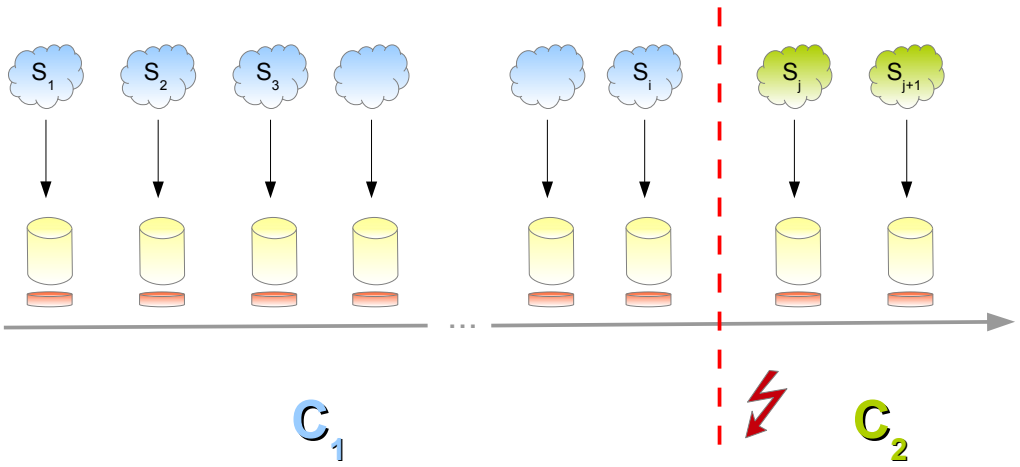


Abbildung 2.1: Concept Drift [20]

Begriff (Quelle)**.** Zunächst wurde davon ausgegangen, dass es sich bei der Quelle um eine Verteilung handelt. Betrachtet man eine Klassifikation mit $y_t \in \{c_1, c_2, \ldots, c_k\}$, dann ist der optimale Klassifikator zur Vorhersage von $\vec{x} \to c_i$ durch die apriori-Wahrscheinlichkeiten der Klassen $P(c_i)$ und die klassenbedingten Dichten $p(\vec{x} \mid c_i)$ eindeutig bestimmt ($i = 1, \ldots, k$). Damit können wir eine Quelle beschreiben als:

$$\mathbf{S} = \left\{ \Big(P(c_1), p(\vec{x} \mid c_1)\Big), \Big(P(c_2), p(\vec{x} \mid c_2)\Big), \ldots, \Big(P(c_k), p(\vec{x} \mid c_k)\Big) \right\}$$

2.2 Ursachen von Concept Drift

Aus der statistischen Entscheidungstheorie wissen wir, dass die Klassifikationsentscheidung bei gleichen Fehlklassifikationskosten von der maximalen aposteriori-Wahrscheinlichkeit bestimmt wird:

$$p(c_i \mid \vec{x}) = \frac{P(c_i)p(\vec{x} \mid c_i)}{p(\vec{x})}$$

Da $p(\vec{x})$ für alle Klassen gleich ist, kann ein Concept Drift durch eine Veränderung in einer der drei Wahrscheinlichkeiten (bzw. Verteilungsdichten) $p(c_i \mid \vec{x}), P(c_i)$ und $p(\vec{x} \mid c_i)$ verursacht werden [20].

Intuitiv stellt man sich einen Concept Drift als eine Änderung in $p(c_i \mid \vec{x})$ vor, weshalb dies oft (z.B. in [18, 17]) als *realer Drift* bezeichnet wird. Eine Änderung in $p(\vec{x} \mid c_i)$ wird, da sie nicht direkt beobachtbar ist, als *virtueller* Drift bezeichnet. Da die drei Wahrscheinlichkeiten jedoch gemäß dieser Gleichung voneinander abhängen, wird im Folgenden nicht zwischen virtuellem und realem Drift unterschieden.

2.3 Context Drift

Obwohl $p(\vec{x})$ also keine Auswirkung auf die einzelne Klassifikationsentscheidung hat, kann eine Änderung die generelle Vorhersagegüte entscheidend beeinflussen, wenn das zugrundeliegende Modell nicht perfekt ist.
Zur Anschauung (siehe auch Abb. 2.2) kann ein binäres Klassifikationsproblem im $\mathbb{R}^2$ dienen, bei dem die beiden Klassengebiete durch eine Gerade getrennt sind. Verwendet das Lernverfahren genau diese Gerade als Modell, so erhält auch nach einer Änderung in $p(\vec{x})$ jedes Data Item das korrekte Label. Wird zur Klassifikation jedoch ein nicht optimales Modell verwendet, gibt es Data Items, die nicht korrekt klassifiziert werden. Falls sich die Verteilung der Items nun so verändert, dass häufiger Items klassifiziert werden, für die das Modell keine korrekte Vorhersage trifft, wirkt sich das natürlich auch auf die Vorhersagegüte aus.

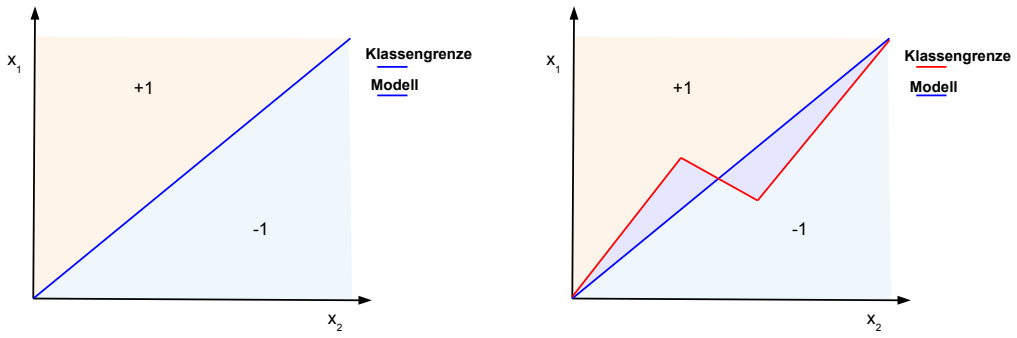


Abbildung 2.2: Context Drift

Auch in diesem Fall liegt ein Drift vor, auf den (zum Beispiel mit Neulernen des Modells) reagiert werden sollte. Wir sprechen dann von *Context Drift*.

Damit unsere Definition von Concept Drift auch den Context Drift abdeckt, müssen wir die Definition einer Quelle aus 2.1 noch um die Verteilungsdichte der Data Items $p(\vec{x})$ erweitern (da jedes Attribut natürlich aus einer anderen Verteilung stammen kann, handelt es sich dabei um eine multivariate Verteilung) :

Begriff (Quelle)**.**

$$\mathbf{S} = \left\{ \mathbf{p}(\vec{\mathbf{x}}), \Big(P(c_1), p(\vec{x} \mid c_1)\Big), \Big(P(c_2), p(\vec{x} \mid c_2)\Big), \ldots, \Big(P(c_k), p(\vec{x} \mid c_k)\Big) \right\}$$

2.4 Charakterisierung verschiedener Driftarten

Grundsätzlich werden die Drift-Typen *gradueller Drift* und *abrupter Drift* unterschieden. Beim abrupten Drift findet von einem Data Item zum nächsten ein Wechsel des Concepts statt, während beim graduellen Drift ein schleichender Übergang erfolgt.
[20] unterscheidet vom graduellen Drift, bei dem eine Übergangsphase von einem festen Concept zu einem anderen existiert, einen *inkrementellen Drift*. Bei einem inkrementellen Drift verändert ein Concept sich in jedem Schritt minimal, sodass Änderungen nur über einen längeren Zeitraum beobachtbar sind. In der Literatur werden inkrementelle Drifts gelegentlich ebenfalls als graduelle Drifts bezeichnet.

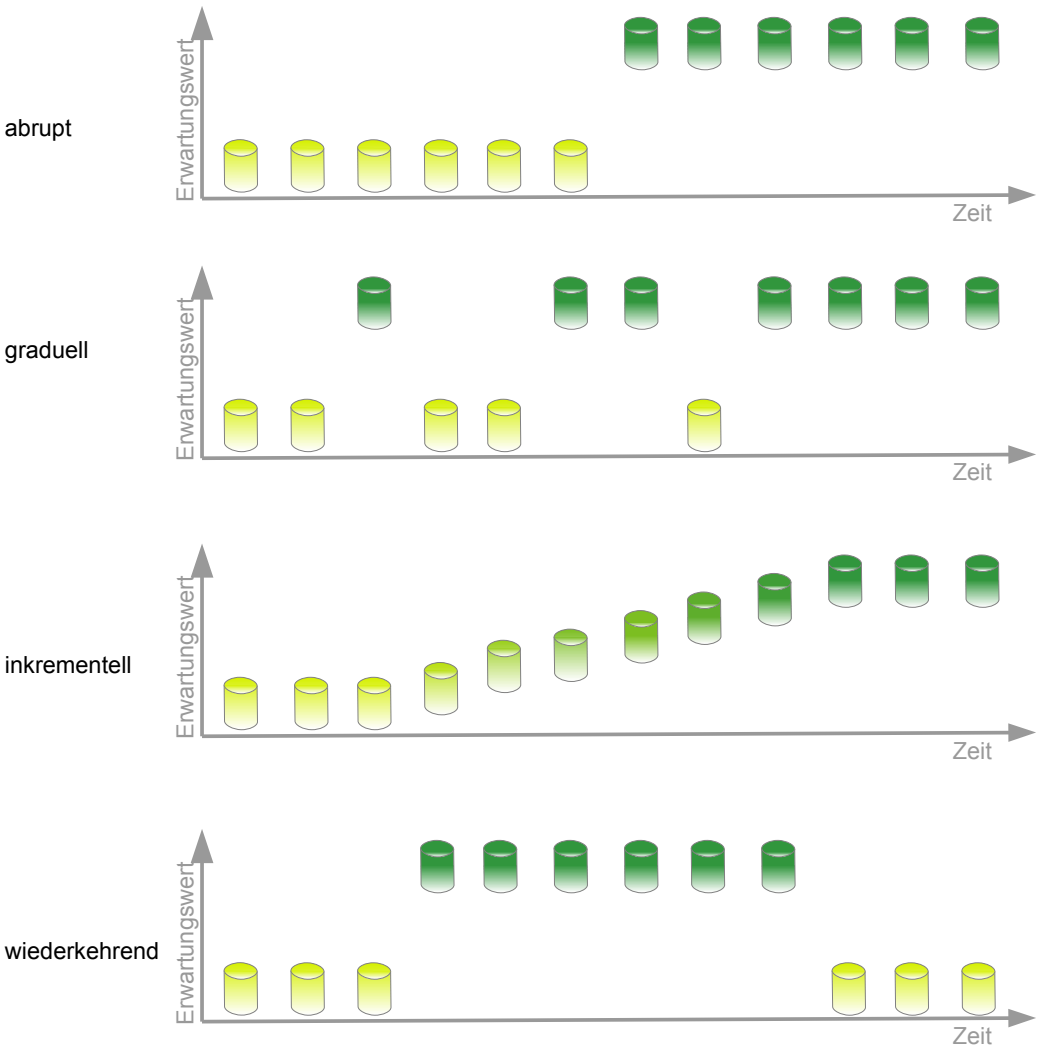


Abbildung 2.3: Driftarten [20]

Der hier definierte Concept Drift wird in [4] daher als Concept Change bezeichnet. Ein abrupter Drift wird dann Concept Shift genannt, und ein gradueller Drift dann Concept Drift. Trotz dieser eleganten Namensgebung wird in der Literatur meistens Concept Drift stellvertretend für alle Arten von Drift verwendet. Daher greift diese Arbeit auf die zuvor definierten Begriffe zurück.
Viele Datenströme kehren nach einem Concept Drift noch einmal (oder sogar mehrmals) zu einem alten Concept zurück. Diese Art von Drift wird *wiederkehrender Concept Drift* genannt. Falls die gleichen Concepts in etwa der gleichen Reihenfolge wiederkehren, spricht man von einem *zyklischen* oder *periodischen Concept Drift*. Die Periodendauer kann sich jedoch von Periode zu Periode unterscheiden.
Ein Beispiel für einen periodischen Concept Drift ist die Vorhersage der Absatzmenge von Speiseeis, die saisonal verschiedene Concepts hat. Dennoch unterscheidet sich der Beginn der Gutwetterphase und somit der Beginn des Driftes von Jahr zu Jahr [20].

Zusammenfassend existieren also folgende Kriterien zur Einordnung eines Drift-Typs:

- abrupter, gradueller oder inkrementeller Drift
- Context Drift oder durch Änderung in den aposteriori-Klassenwahrscheinlichkeiten bedingter Concept Drift
- wiederkehrender und periodischer Concept Drift

Betrachtet man also einen einzelnen Drift kann man diesen gemäß der Matrix 2.4 klassifizieren. Eine weitere Zeile wäre denkbar für Drifts, bei denen sich die aposteriori-Klassenwahrscheinlichkeiten und die Verteilung der Data Items ändern - also Concept und Context Drift gleichzeitig auftreten. Einen wiederkehrenden oder periodischen Drift kann man dann als eine Abfolge von atomaren Drifts betrachten, von denen jeder einzelne gemäß der Abbildung klassifiziert werden kann.

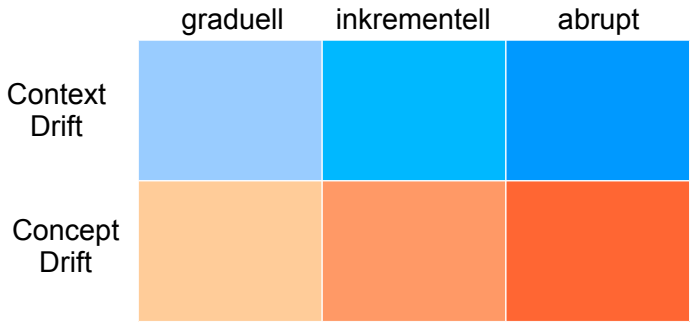


Abbildung 2.4: Klassifikation eines Concept Drifts

3 Behandlung von Concept Drift

Ausgehend von den Anforderungen an die Driftbehandlung auf Datenströmen, werden nun die wichtigsten Ansätze zur Behandlung von Concept Drift präsentiert und Gütemaße eingeführt, anhand derer diese Ansätze miteinander verglichen werden können.

3.1 Concept Drift in Datenströmen

Eine initiale Motivation dieser Arbeit war die statistische Vorhersage von Größen in einem Datenstrom eines realen Produktionsprozesses mit Concept Drift. Mit einer Datenstromanwendung gehen einige besondere Probleme einher, auf die auch bei der Entwicklung praxistauglicher Methoden zur Driftbehandlung Rücksicht genommen werden muss:

- Ein Datenstrom kann eine so große Datenmenge sein, dass sie nicht komplett im Speicher abgelegt werden kann.
- In der Realität treffen die einzelnen Data Items oft sehr schnell hintereinander ein, sodass unter Umständen die Verarbeitung noch nicht abgeschlossen ist, wenn das nächste Data Item (zum Lernen oder zur Vorhersage) eintrifft.
- Die sequentielle Verarbeitung schließt insbesondere den wahlfreien Zugriff auf historische Data Items aus.

Um diesen Problemen gerecht zu werden, wurden in [4] vier Anforderungen an Lernverfahren auf Datenströmen identifiziert:

1. Verarbeite zu jedem Zeitpunkt nur ein einzelnes Item und betrachte jedes Item nur einmal!
2. Verwende nur beschränkt viel Speicherplatz!
3. Arbeite mit einer beschränkten Verarbeitungszeit! (Echtzeitverarbeitung)
4. Sei jederzeit bereit eine Vorhersage für eine neues Data Item zu treffen! (*anytime prediction*)

Grundsätzlich gibt es zwei Ansätze, wie solche Lernverfahren vorgehen können, die im Folgenden als *Online*- und *Batch*-Verfahren unterschieden werden.

Für Batch-Verfahren wird im Speicher eine feste Anzahl von Data Items gepuffert, sodass auf den so entstandenen Datensatz ein maschinelles Lernverfahren angewendet werden kann. Aus der Anforderung der beschränkten Verarbeitungszeit folgt dann jedoch, dass nicht nach jedem Beispiel ein neues Modell gelernt werden kann. Dennoch kann für jedes

neue Item *online* eine Vorhersage des Klassenlabels durchgeführt werden. Die besondere Herausforderung eines solchen Verfahrens in Gegenwart von Concept Drift ist es also, die Zeitpunkte zum Neulernen des Modells zu bestimmen.

Online-Verfahren führen jeweils mit einem einzelnen neu eintreffenden Data Item ein Update des aktuellen Modells durch. Viele maschinelle Lernverfahren müssen die Items der Trainingsmenge zum Lernen mehrfach betrachten, sodass eine Übertragung auf ein entsprechendes Update mit jedem Item nicht möglich ist oder zu einem schlechteren Modell führt.
Da eine kontinuierliche Anpassung an die aktuellen Daten erfolgt, können Online-Verfahren außerdem ein Ansatz zur Behandlung von Concept Drift sein. Da auch für Online-Verfahren eine schnellere Anpassung erreicht werden kann, wenn nach einem Drift ein neues Modell gelernt wird, werden in dieser Arbeit Online-Verfahren nicht als Alternative zu den Ansätzen des nächsten Abschnitts, sondern als ein Basislernverfahren betrachtet, das möglicherweise eine bessere Anpassung an ein neues Concept ermöglicht als ein Batch-Verfahren.

3.2 Ansätze zur Behandlung von Concept Drift

Für Batch- und Online-Lernverfahren ist zu erwarten, dass nach einem Concept Drift das Modell bezüglich der neuen Verteilung schlechte Vorhersagen liefert. Das Ziel der Behandlung ist es, auch nach einem Concept Drift ein akkurates Modell zur Verfügung zu stellen. In der Regel lässt sich dies besser durch Lernen eines neuen Modells als durch eine Aktualisierung des bisherigen Modells realisieren.
Eine Driftbehandlungsmethode übernimmt entsprechend das automatisierte Neulernen des Modells nach einem Drift. Zum Neulernen kann dann ein Lernverfahren ohne spezielle Behandlung von Concept Drift (im Folgenden *Basislerner* genannt) verwendet werden. Im Wesentlichen unterscheiden die Methoden sich in der Auswahl der Zeitpunkte zum Neulernen und der Auswahl der dazu verwendeten Data Items.
Folgende Ansätze erlauben eine Behandlung unabhängig vom konkreten Basislerner:

Naiv Der einfachste Ansatz besteht darin, in regelmäßigen Abständen ein neues Modell zu lernen. Die Schwierigkeit ist hier die Wahl des Abstandes. Ein zu geringer Abstand bedeutet unnötigen Aufwand für das Modelllernen und aufgrund der kleineren Trainingsmenge ein schlechtes Modell. Ein zu großer Abstand führt dazu, dass erst spät auf einen Drift reagiert wird.

Fenster-Verfahren Diese Verfahren verwalten ein Fenster einer bestimmten Größe, in dem nur die letzten Beobachtungen gespeichert werden, während ältere Beobachtungen nicht mehr betrachtet werden. Auf den gepufferten Beobachtungen könnte nun ein statistischer Test durchgeführt werden, um zu entscheiden, ob sich die Verteilung verändert hat. Üblicherweise wird aber einfach das Modell aufgrund der Beobachtungen innerhalb des Fensters aktualisiert. Die Schwierigkeit bildet hierbei die Wahl der Fenstergröße. Besonders gute Ergebnisse lassen sich mit adaptiven Verfahren erreichen, welche die Fenstergröße in Abhängigkeit von der aktuellen

Vorhersagegüte anpassen [16].
Fensterverfahren entscheiden also insbesondere welche Data Items zum Training verwendet werden sollen.

Driftdetektor Idealerweise wird ein neues Modell gelernt, sobald ein Concept Drift stattgefunden hat. Außerdem muss nicht neu gelernt werden, wenn sich die Verteilung der Daten nicht verändert hat. Driftdetektoren versuchen daher mit statistischen Testverfahren einen Concept Drift zu erkennen. Sie entscheiden also, wann ein neues Modell gelernt werden soll.
Verschiedene Driftdetektoren unterscheiden sich in der konkreten Wahl der Teststatistik. Eine generelle Beurteilung welche Teststatistik die besten Ergebnisse liefert, ist nicht möglich, da zu jeder Teststatistik zwei unterschiedliche Verteilungen mit gleicher Verteilung der Teststatistik existieren.

Drifterkennung Drifterkennungsmethoden kombinieren einen Driftdetektor zur Bestimmung des Lernzeitpunktes mit einem Fensterverfahren zur Ermittlung der Trainingsmenge.
Besonders effektiv sind Erkennungsverfahren, welche mit einem Detektor wie der *Drift Detection Method (DDM)* [11] anhand der Teststatistik bereits vor Erkennung des Drifts eine Driftwarnung ableiten und das Trainingsfenster entsprechend wählen (Abb. 3.1).

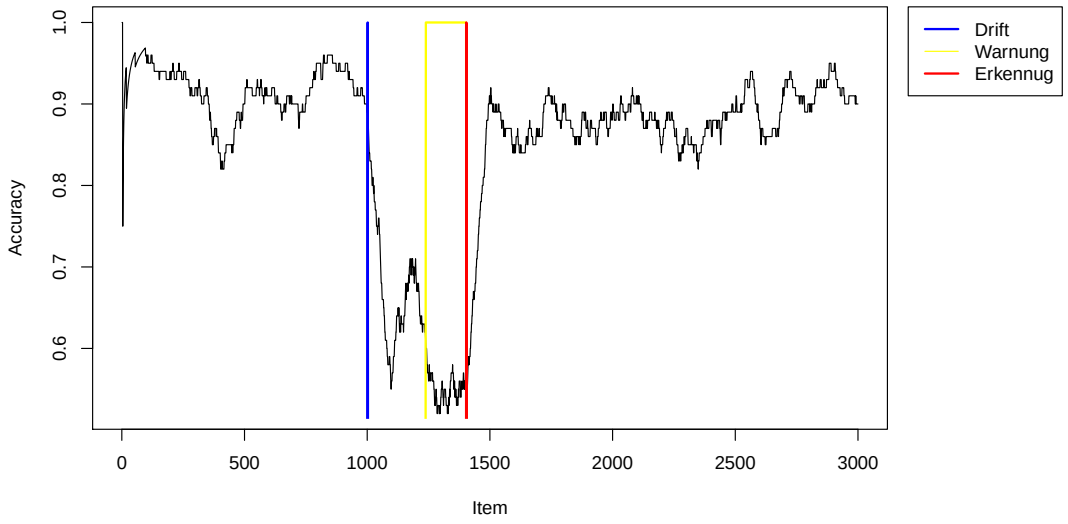


Abbildung 3.1: Drifterkennung am Beispiel der DDM

Mit Erreichen des sogenannten *warning level* wird vermutet, dass ein neues Concept vorliegt, und alle eintreffenden Data Items gepuffert. Mit Erreichen des *drift level* wird der Concept Drift als sicher angesehen und auf diesen Items ein neues Modell gelernt. Das *warning level* wird gelegentlich auch bedingt durch Rauschen und Varianz der Teststatistik erreicht, gepufferte Items werden dann bei einer

Rückkehr zum *normal level* verworfen (Abb. 3.2).

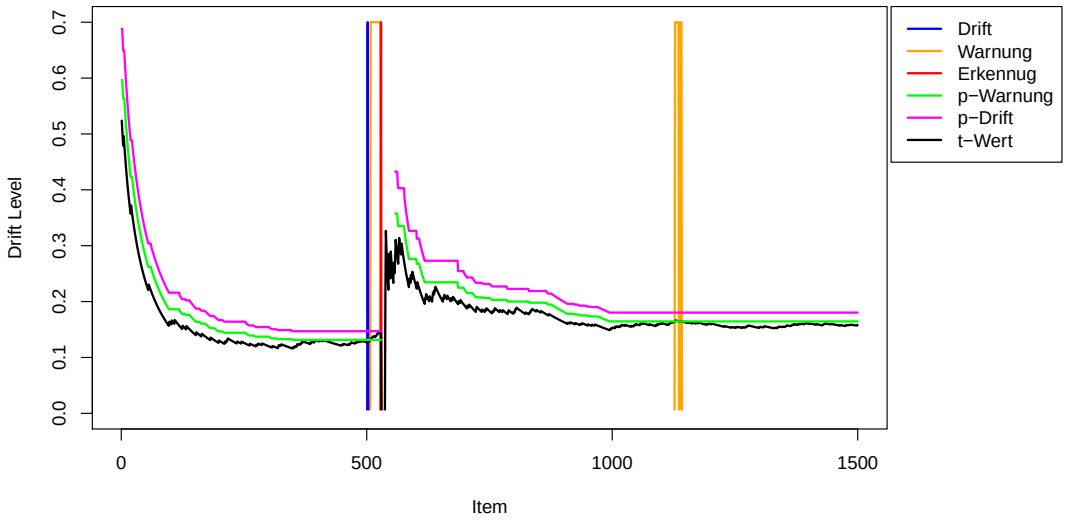


Abbildung 3.2: Drift Level am Beispiel der DDM

An der Abbildung ist außerdem schön zu Erkennung wie nicht nur die Teststatistik, sondern auch *warn* und *drift level* über die Zeit angepasst werden. Die DDM verwendet die Fehlerrate als Teststatistik und geht davon aus, dass diese sinkt, je länger ein Modell trainiert wird. Ein Anstieg der Fehlerrate um mehr als die dreifache Varianz dieser Teststatistik wird als Drift interpretiert und bei Anstieg um die doppelte Varianz wird eine Driftwarnung erzeugt.

Ensemble Ein Ensemble ist eine Menge von verschiedenen Lernern, die der Vorhersage der gleichen Zielgröße dienen. Die Grundannahme ist, dass die Kombination der Vorhersagen verschiedener Lerner besser ist, als die eines einzelnen Lerners. Ensemble sind generell eine bewährte Methode aus dem klassischen maschinellen Lernen.
Der Ensemble-Ansatz zur Behandlung von Concept Drift unterteilt den Datenstrom in Blöcke konstanter Länge (*Batch*) und lernt auf jedem solchen Block ein neues Modell. Das neue Modell wird in das Ensemble aufgenommen, sodass auch Trainingsdaten aus dem neuen Concept berücksichtigt werden. Aufgrund der Speicherbeschränkung muss dafür ein älteres Modell verworfen werden.
Der bekannteste Vertreter dieses Ansatzes ist das *Accuracy Weighted Ensemble* [19]. Hier wird die Kombination der Vorhersagen der einzelnen Modelle zur endgültigen Klassifikationsentscheidung dadurch realisiert, dass die Vorhersage jedes Lerners in Abhängigkeit von seiner aktuellen Klassifkationsgenauigkeit gewichtet wird. Zur Schätzung der aktuellen Vorhersagegenauigkeit wird jeweils auf dem letzten Batch für jeden Lerner eine Kreuzvalidierung durchgeführt.

3.3 Gütemaße für Driftbehandlung

Das entscheidende Kriterium zur Beurteilung eines Lernverfahrens ist die Vorhersagegenauigkeit. Concept Drift muss gerade wegen der verursachten Verschlechterung dieser Vorhersagegenauigkeit behandelt werden, sodass wir den Erfolg einer Behandlung ebenfalls maßgeblich von der Vorhersagegenauigkeit abhängig machen.
Für die Klassifikation kann das vorhergesagte Label mit dem tatsächlichen übereinstimmen (korrekte Klassifkation) oder sich davon unterscheiden (Fehlklassifikation). Wir messen die Vorhersagegenauigkeit dann anhand der:

Korrektklassifikationrate (Accuracy).

$$\text{Accuracy} = \frac{\text{Anzahl korrekte Klassifikationen}}{\text{Anzahl korrekte Klassifikationen} + \text{Anzahl Fehlklassifikationen}}$$

Für die Regression ist der Fehler einer einzelnen Vorhersage als Abstand von vorhergesagtem und tatsächlichem Label bezüglich einer Metrik definiert. Die Vorhersagegenauigkeit wird dann gemessen, indem der Fehler über alle Beispiele gemittelt wird. Die zwei am häufigsten verwendeten Metriken liefern den *Mean Squared Error* und den *Mean Absolute Error*. Konkret aus einem Datensatz $(\vec{x}_1, \ldots, \vec{x}_n)$, wobei für das Beispiel $\vec{x}_i$ mit $(i = 1, \ldots, n)$ jeweils y_i das tatsächliche und $\hat{y}_i$ das vorhergesagte Label bezeichnet, schätzen wir die beiden Fehler wie folgt:

Mean Absolute Error (MAE).

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Mean Squared Error (MSE).

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Gerade im Zusammenspiel mit Datenstromalgorithmen sind der benötigte Speicherplatz und die zur Verarbeitung benötigte Zeit wichtige Kriterien für die Güte der Driftbehandlung [4]. Beide Kriterien können leicht explizit bestimmt werden. Außerdem wird eine minimale Parameterzahl gefordert. Da Driftbehandlungsverfahren in der Regel auf unbekannte Änderungen reagieren müssen, sollten sie so allgemein wie möglich sein und keine umfassende Optimierung zahlreicher Parameter erfordern.

3.3.1 Gütemaße für Driftdetektoren

Für Driftbehandlungsmethoden, die einen Driftdetektor benutzen, um den richtigen Zeitpunkt zum Neulernen zu bestimmen, können zusätzlich zur Vohersagegenauigkeit Kriterien betrachtet werden, die Aufschluss darüber geben, wie präzise die reine Erkennung

von Concept Drift ist. Die Erweiterung des Gütemaßes um solche Kriterien, erlaubt auch den direkten Vergleich reiner Detektoren ohne die Kopplung an den Erfolg eines konkreten Lernverfahrens. Besonders interessant ist eine solche Betrachtung vor dem Hintergrund, dass eine hohe Anzahl falscher Alarme (d.h. einer Erkennung, ohne dass tatsächlich ein Drift vorliegt) zu einer höheren Vorhersagegenauigkeit führen kann [5].

Konkret werden in [5] folgende Kriterien beschrieben:

Mean Time to Detection (MTD). Die Anzahl der Data Items von Beginn des Drifts bis zur Erkennung ist ein Maß für die Reaktivität des Detektors.

Mean Time between False Alarms (MTFA). Ein falscher Alarm impliziert das unnötige Neulernen des Modells. Die MTFA sollte also möglichst groß sein.

Missed Detection Rate (MDR). Neben falschem und korrektem Alarm bleibt noch der Fall eines nicht erkannten Alarms.

Mean Time Ratio (MTR). Als kombiniertes Maß zum Vergleich verschiedener Driftdetektoren anhand einer einzelnen Größe:

$$MTR = \frac{MTFA}{MTD} \times (1 - MDR)$$

Zusätzlich könnten folgende Kriterien aufschlussreich sein:

Detection Warning Ratio (DWR). Über die Anzahl der Driftwarnung im Verhältnis zur Anzahl der Drifterkennungen lässt sich die Wahl der Werte für *warning-* und *drift-level* beurteilen.

Average Number of Items to Learn (ANIL). Für die ursprüngliche Version der Drifterkennung werden zum Lernen des neuen Modells die Data Items zwischen Erreichen des *warning* und des *drift level* gespeichert. Ist die Anzahl der Items zu gering, ist die Qualität des neuen Modells möglicherweise unbefriedigend. Ist die Anzahl zu groß, kann das zu Konflikten mit der Speicherbeschränkung führen.

3.4 Fragestellung

In Anbetracht der verschiedenen Ansätze und der Vielzahl der konkreten Verfahren zur Driftbehandlung stellt sich die Frage, welches Verfahren in welchen Situationen besonders geeignet ist. Gibt es sogar ein Verfahren, dass in allen Szenarien besser geeignet ist als alle anderen? Wenig untersucht wurde bisher vor allem, welche Verfahren in zyklischen Prozessen besonders geeignet sind.

Um diese Fragen zu beantworten müssen verschiedene Ansätze zu Behandlung miteinander verglichen werden. Konkrete zu untersuchende Szenarien ergeben sich aus den in Kapitel 2 unterschiedenen Driftarten. Betrachtet werden müssen dabei verschiedene Lernverfahren als Basislerner, verschiedene Zeiten zum Lernen des Modells, verschiedene Driftgeschwindigkeiten und weitere ansatzspezifische Parameter, wie zum Beispiel der

verwendete Driftdetektor für die Drifterkennung oder Ensemble-Größe und Evaluationsabstände für den Ensemble-Ansatz.

Verglichen werden können die Verfahren nach den oben beschriebenen Kriterien.
Vor dem Hintergrund unseres konkreten Anwendungsfalls eines industriellen Produktionsprozesses, kann davon ausgegangen werden, dass die Verarbeitungszeit durch die Rate, in der neue Sensordaten eintreffen, bestimmt wird. Der verwendete Speicherplatz muss durch den konkreten Festplattenspeicher beschränkt sein. Aus den unter diesen Einschränkungen geeigneten Verfahren, muss dann das mit maximaler Vorhersagegenauigkeit ausgewählt werden, weshalb die Verfahren vor allem bezüglich der Vorhersagegenauigkeit verglichen werden.

4 Daten

In diesem Kapitel soll die Auswahl der Daten, die zur Durchführung der Experimente zum Vergleich verschiedener Methoden zur Driftbehandlung verwendet werden, erläutert werden. Betrachtet wurden synthetische Daten zur Abbildung des gesamten Spektrums verschiedener Driftarten und ein realer Standarddatensatz zur Evaluation von Driftbehandlung.
Da es für die Evaluation von Drifterkennungsmethoden, die auf der Analyse der Vorhersagefehler beruhen, ausreicht, die Fehlerwahrscheinlichkeit des aktuellen Lerners zu beurteilen, wird außerdem ein Fehlerraten-Generator verwendet.
Für die synthetischen Datensätze kann zur Evaluation auch auf Ground-Truth-Daten über den Driftzeitpunkt, bzw. darüber ob eine Beobachtung aus einem Driftbereich stammt oder nicht, zurückgegriffen werden.

4.1 Generator-Framework

Es ist zu erwarten, dass die in Kapitel 2 charakterisierten Arten von Concept Drift von verschiedenen Verfahren zur Driftbehandlung unterschiedlich gut erkannt werden. Außerdem sollen verschiedene Driftarten auch jeweils in zyklischen Prozessen betrachtet werden.
Da die ermittelten Charakteristika für Concept Drift nicht exklusiv sind, sondern zum Teil als orthogonale Kriterien aufgefasst werden können, ergeben sich aus beliebiger Kombination dieser Kriterien zahlreiche Szenarien verschiedener Drifttypen (zur Erinnerung Abb. 2.4).
Nicht für alle diese Szenarien stehen umfassende reale Datensätze zur Verfügung. Daher ist es zwingend notwendig auf synthetische Daten zurückzugreifen. Zur Generierung dieser Daten wurde eine modulares, leicht erweiterbares Generator-Framework entwickelt, das unserer Charakterisierung von Driftarten Rechnung trägt, und eine flexible Kombination der Driftarten - insbesondere auch in zyklischen Prozessen - ermöglicht.

4.1.1 Periodischer Drift-Generator

Für den kompletten Generator zur Erzeugung von Datensätzen mit Concept Drift wurde die Bezeichnung *periodischen Drift-Generator* gewählt. Ein periodischer Driftgenerator verwaltet mehrere potenziell verschiedene Concepts in einer festen Reihenfolge, die sequentiell nacheinander abgearbeitet werden (Abb. 4.1). Eine sequentielle Abarbeitung aller Concepts, zu denen jeweils die Anzahl der zugehörigen Items angegeben wird, bildet

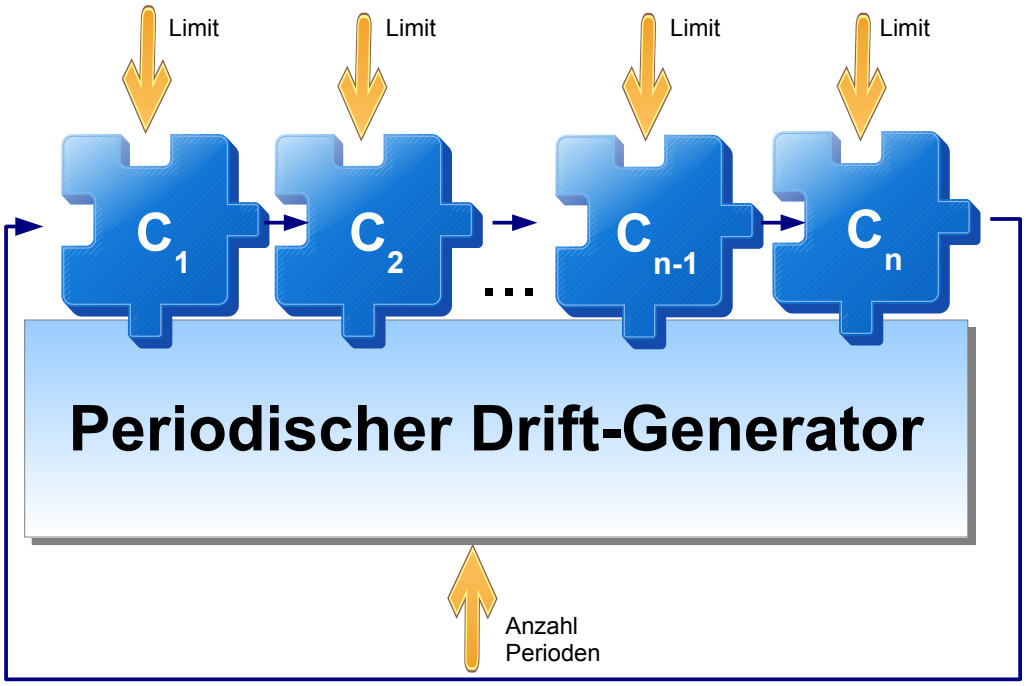


Abbildung 4.1: Baukastenprinzip des periodischen Drift-Generators

eine Periode. Eine neue Periode beginnt dann wieder beim ersten Concept. Neben den konkreten Concepts ist also der zentrale Parameter des periodischen Drift-Generators die Periodenanzahl.

Sowohl ein einzelner als auch ein wiederkehrender Concept Drift kann als periodischer Concept Drift mit nur einer Periode aufgefasst werden und dementsprechend ebenfalls mit diesem Generator erzeugt werden.
Da die Driftzeitpunkte und Periodengröße realer periodischer Drifts in der Regel nicht deterministisch und schon gar nicht konstant sind, kann für die einzelnen Concepts auch ein Intervall für die Anzahl der Elemente aus diesem Concept angegeben werden, aus dem dann in jeder Periode eine Anzahl zufällig gewählt wird.

Die einzelnen Concepts lassen sich wiederum durch verschiedene Datengeneratoren oder konkrete Datensätze umsetzen, sodass sich beliebige Szenarien in einen zyklischen Prozess einbetten lassen. Die einzelnen Drifttypen (abrupt, graduell, inkrementell), lassen sich dabei wie folgt modellieren:

Abrupter Drift kann ganz einfach durch Angabe zweier unterschiedlicher, aber konstanter (d.h. ohne Concept Drift), aufeinanderfolgender Concepts umgesetzt werden.

Gradueller Drift lässt sich ebenfalls aus Angabe zweier solcher Concepts generieren, indem eine entsprechende Übergangsphase zwischengeschaltet wird. Eine konstruktive Methode zur Realisierung der Übergangsphase für beliebige Concepts C_1, C_2 findet sich in [4]: Während der Übergangsphase der Länge W mit Mittelpunkt t_0 (im Datenstrom entspricht W einer festen Anzahl von Items) wird die Wahrscheinlichkeit modelliert, dass ein Item aus dem zweiten Concept generiert wird. Dann kann für jedes Item gemäß dieser Wahrscheinlichkeit entschieden werden, aus welchem Concept es entnommen werden soll. Zu Beginn der Übergangsphase sollte die Wahrscheinlichkeit nahe Null liegen und sich monoton steigend einem Wert nahe Eins am Ende der Übergangsphase annähern. Dieses Verhalten lässt sich beispielsweise durch eine Sigmoid-Funktion

$$f(t) = 1/(1 + e^{-s(t-t_0)})$$

beschreiben (Abb. 4.2). Über den Winkel, den die Tangente im Punkt t_0 mit der X-Achse einschließt, lässt sich die Wahl des Paramters s passend zur Länge der Übergangsphase als $s = 4/W$ ableiten.

Inkrementeller Drift In Ermangelung eines derart allgemeinen Verfahrens um einen Datensatz ohne Concept Drift nachträglich in einen inkrementellen Drift umzuwandeln, beschränken wir uns zur Modellierung von inkrementellem Drift auf Datengeneratoren. Diese werden mit optionalen Parametern zur Steuerung von Geschwindigkeit und Richtung des Drifts ausgestattet. Eine explizite Umsetzung des inkrementellen Drifts über derartige Parameter findet man in der Beschreibung des Rotating-Hyperplane-Generators.

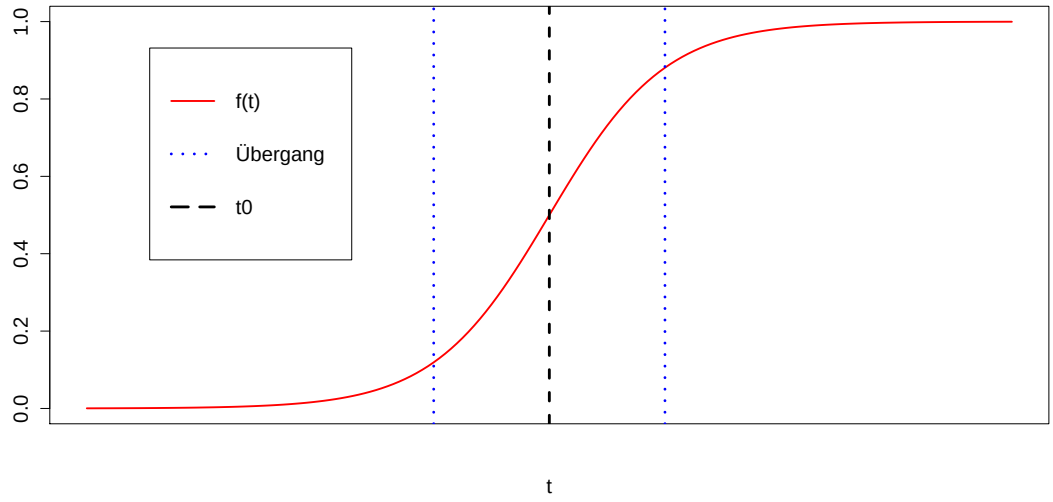


Abbildung 4.2: Sigmoid-Funktion $f(t) = 1/(1 + e^{-s(t-t_0)})$

Die beiden Realisierungen eines Datengenerators für ein konkretes Concept, Fehlerraten-Generator und Concept-Generator, werden im Folgenden genauer beschrieben.

4.1.2 Fehlerraten-Generator

Da Drifterkennung in der Regel auf der Überwachung der Fehlerrate eines Lerners beruht, genügt es auch zur Beurteilung bezüglich Kriterien, die über die Vorhersagegenauigkeit hinaus gehen, die Fehlerrate eine Lerners zu betrachten. Um die konkreten Eigenarten eines Driftdetektors zu beurteilen, ist es oft wünschenswert, diese Fehlerrate gemäß theoretischer Überlegungen und unabhängig von konkreter Repräsentation der Daten oder Wahl und Genauigkeit eines spezifischen Basis-Lernverfahrens zu untersuchen. Ein entsprechender Generator wird zum Beispiel in [5] zum Vergleich verschiedener Drifterkennungsverfahren in Anwesenheit von inkrementellem Drift eingesetzt.
Allgemein kann davon ausgegangen werden, dass ein trainierter Lerner auf einem einzelnen Concept, eine in etwa konstante und verhältnismäßig geringe Fehlklassifikationsrate aufweist. Auf einem anderen, ebenfalls konstanten Concept wird der Lerner auch eine ungefähr konstante, jedoch geringere Klassifikationsgenauigkeit aufweisen. Entsprechend lassen sich mit mit einem Generator, der den Fehler gemäß einer gegebenen Fehlerrate erzeugt, mit den Mitteln des periodischen Drift-Generators - wie oben beschrieben- gradueller und abrupter Drift simulieren. Um inkrementellen Drift zu simulieren, muss die zugrundeliegende Fehlerrate entsprechend schrittweise erhöht werden.
Für die binäre Klassifikation müssen nur zwei Fehlergrößen erzeugt werden (0=kein Fehler, 1=Fehler). Dazu wird für jedes Data Item ein Wert β gleichverteilt aus $[0, 1]$ gezogen. Für $\beta \leq Fehlerrate$ ergibt sich entsprechend ein Fehler von 1, ansonsten ist der Fehler 0.
Für eine Übertragung auf reellwertige Fehler zur Simulation von Regressionsfehlern verwendet das Framework zunächst eine Normalverteilung um den erwarteten Fehler.

4.1.3 Concept-Generator

Für eine generelle Untersuchung von Methoden zur Behandlung von Concept Drift reicht die Fehlerrate jedoch nicht aus. Um auf den Daten ein Modell lernen zu können, muss auch ein Generator für *echte* Labeled Data Items mit konkreten Attributen existieren. Zur Abdeckung der von uns unterschiedenen Driftarten muss der Generator folgende Anforderungen abdecken:

1. **Simulation von Context Drift:**
   Austausch- und änderbare Verteilung Ω, aus der die Attributwerte $\vec{x}_t = (x_t^{(1)}, \ldots, x_t^{(n)}) \in X^n$ mit $x_t^{(i)} \sim \Omega, i = 1, \ldots, n$ generiert werden. Dabei ist Ω eine multivariate Verteilung, sodass die einzelnen Attribute gemäß unterschiedlicher Verteilungen generiert werden.

2. **Simulation von Concept Drift:**
   Austausch- und änderbare, bedingte Verteilung für das Label bei gegebenen Attributwerten.
   Realisierbar ist das über eine Label-Funktion $f : X^n \to Y$

3. Label für Klassifikation ($Y = \mathbb{Z}$) oder Regression ($Y = \mathbb{R}$)

Vor dem Hintergrund, dass die Verfahren später auf reale Produktionsprozesse angewendet werden sollen, muss berücksichtigt werden, dass unter Umständen auch die wahren Label der Data Items nicht immer korrekt sind (beispielsweise wenn die Label manuell von Fachleuten bestimmt werden). Ein solches *Rauschen* (Noise) kann für eine (binäre) Klassifikation realisiert werden, indem mit einer festen Rauschwahrscheinlichkeit das falsche Label vergeben wird. Für eine Regression lässt sich Rauschen beispielsweise durch Addition einer um Null normalverteilten Zufallsvariable zum Label realisieren.

Das Framework realisiert diese Modularität durch die Verknüpfung eines Attribut-Generators und einer Label-Funktion (und optional eines Noise-Generators) im Concept-Generator (Abb. 4.3).

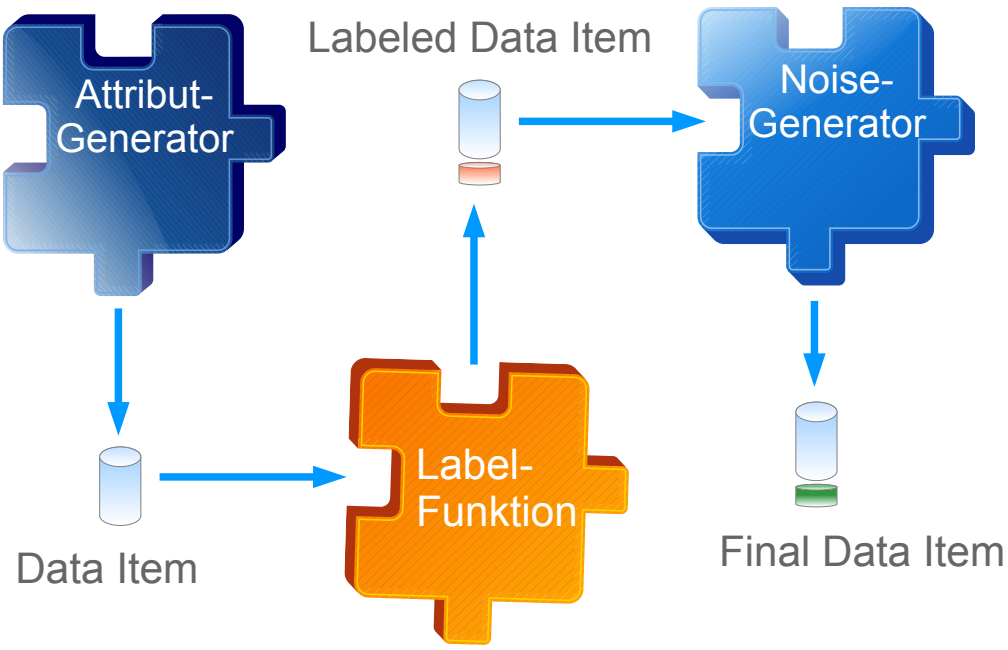


Abbildung 4.3: Baukastenprinzip des Concept-Generators

Attribut-Generatoren

Konkret bietet das Framework einen Attributgenerator für normalverteilte und einen für gleichverteilte Attributwerte. Parametrisierbar ist ein Attributgenerator durch die Verteilungsparameter und die Anzahl der Attribute n, wobei auch jedes Attribut gemäß einer eigenen Verteilung verteilt sein kann. Am Beispiel der Normalverteilung lässt sich ein inkrementeller Drift zum Beispiel durch eine konstante Änderungsrate des Mittelwertes erreichen, für die Gleichverteilung durch eine Änderungsrate der Intervallgrenzen.

Label-Generatoren

Als konkrete Label-Funktion steht für die Regression eine **Weighted Sum** $f(\vec{w}, \vec{x}) = \vec{w} \cdot \vec{x}$ mit Gewichten $\vec{w} \in \mathbb{R}^n$ zur Verfügung (beliebige reelle Funktionen sind analog umsetzbar). Auch hier lässt sich eine konstante Änderungsrate nach jedem berechneten Label auf die Gewichte addieren, um einen inkrementellen Drift zu simulieren.

Ein besonders anschaulicher Standarddatensatz (siehe [15, 18, 19, 10, 4]) für Concept-Drift im Kontext einer Klassifikationsaufgabe verwendet eine rotierende Hyperebene. Alle Beobachtungen auf der einen Seite der Ebene erhalten das Label *positiv*, die auf der anderen das Label *negativ*. Der Vorteil dieses Datensatzes ist neben der Anschaulichkeit, dass er sehr gut durch Parameter gesteuert werden kann. Durch eine langsame Rotation der Hyperebene lässt sich auch ein inkrementeller Concept Drift simulieren.
Für die Klassifikation steht daher die **Rotating Hyperplane** als Label-Funktion zur Verfügung. In der ursprünglichen Beschreibung des Generators werden gleichverteilte Attributwerte aus dem Intervall $[0, 1]$ angenommen, was sich in diesem Framework durch den gleichverteilten Attribut-Generator realisieren lässt.
Die konkrete Hyperebene mit Gewichten $w_i \in \mathbb{R}$ für $i = 0, ..n$ lässt sich durch folgende Gleichung beschreiben:

$$\sum_{i=1}^{n} w_i \; x_i = w_0$$

Entsprechend werden dann Data Items mit $\vec{w} \cdot \vec{x} \geq w_0$ als positiv und solche mit $\vec{w} \cdot \vec{x} < w_0$ als negativ klassifiziert. Wählt man $w_o = \frac{1}{2} \sum_{i=1}^{n} w_i$ erhält man zwei etwa gleichgroße Klassengebiete.
Eine Rotation der Ebene lässt sich durch eine Anpassung einiger/aller Gewichte nach jeder Vorhersage realiseren: $w_i^{t+1} = w_i^t + \delta\sigma$. Dabei beschreibt $\sigma \in \{-1, 1\}$ die Richtung der Rotation und $\delta \in \mathbb{R}$ die Rotationsgeschwindigkeit. Oft wird die Rotationsgeschwindigkeit in absoluter Änderung $\gamma \in \mathbb{R}$ pro N Beispiele angegeben (also $\delta = \frac{\gamma}{N}$).

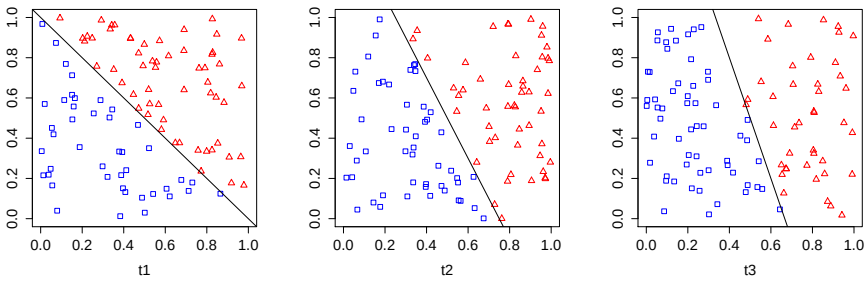


Abbildung 4.4: Rotating Hyperplane [18]

4.1.4 Ground-Truth-Annotation

Viele Kriterien zur Evaluation von Verfahren zur Behandlung von Concept Drift (z.B. die erwartete Zeit bis zur Erkennung) lassen sich nur bestimmen, wenn Informationen über den tatsächlichen Zeitpunkt des Drifts vorliegen. Daher stellen die synthetischen Datensätze diese Ground-Truth-Information über die Generierung der Annotation *@changeGT* mit den Ausprägungen *1* und *0* zur Verfügung.
Für einen abrupten Drift ist die erste Beobachtung nach einem Drift mit *1* annotiert und alle anderen mit *0*, sodass der Driftzeitpunkt bestimmt werden kann. Über einen Vergleich mit den Erkennungspunkten lassen sich Erkennungen auch in korrekte und fehlerhafte (*false positive*) Erkennungen unterteilen.
Ein gradueller Drift wird bis zum Beginn des Drifts mit *0* und ab Beginn des Drifts mit *1* annotiert. Alle Erkennungen, die vor dem Beginn des Drifts liegen, sind also naheliegenderweise fehlerhaft und alle danach korrekt.

4.2 Electricity Market

Einer der wichtigsten realen Datensätze (ELEC 2) zur Evaluation von Driftbehandlung beruht auf Daten die an australischen Strombörsen erhoben wurden und wurde erstmals in [14] vorgestellt. Der Strompreis wird alle 5 Minuten bestimmt. Neben Angebot und Nachfrage können auch unerwartete Ereignisse wie Umweltkatastrophen oder Unwetter den Preis kurzfristig beeinflussen. Langfristig ist darüber hinaus mit saisonalen Schwankungen zu rechnen.
Als Attribute wurden beispielsweise Nachfrage und Preis aus New South Wales und Victoria, der angesetzte Energietransfer zwischen den beiden Bundesstaaten und der konkrete Zeitstempel erhoben. Das Klassenlabel (mit den Ausprägungen UP und DOWN) bezeichnet die Richtung der Preisänderung gegenüber des *Moving Average* der letzten 24 Stunden. Jedes Beispiel bezieht sich auf einen Zeitraum von 30 Minuten, sodass sich bei einem Beobachtungszeitraum vom 7. Mai 1996 bis zum 5. Dezember 1998 eine Größe von 45312 Data Items ergibt [11].
Eine kritische Betrachtung, inwiefern der ELEC-Datensatz zur Beurteilung von Driftverfahren geeignet ist, findet sich in [5, 21]. Da jedoch ohnehin kaum alternative Benchmark-Datensätze verfügbar sind, werde ich für den Vergleich der verschiedenen Methoden zur Driftbehandlung auch diesen Datensatz verwenden.

5 Evaluation von Driftbehandlung

5.1 Szenarien

Für jede Driftart wurde ein Szenario definiert, das stellvertretend für alle Drifts dieser Art betrachtet werden soll. Für jedes Szenario wurden dann drei Ausprägungen des Drifts (schwach, mittel, stark) unterschieden.
Um die Anpassung nach einem Concept Drift zu beurteilen wird davon ausgegangen, dass für ein festes Concept vor dem Drift ein gutes Modell gelernt wurde. Daher beginnt jedes Szenario mit einer Vorlaufphase zum Lernen des Modells ohne Auswertung. Der konkrete Drift wird dann mit dem *Periodischen Drift-Generator* aus Kapitel 4 erzeugt. Alle Concepts werden dabei mit einer Rotating Hyperplane modelliert.

Abrupter Drift Für den abrupten Drift wurden zwei Concepts verwendet. Die Attribute werden jeweils gleichverteilt erzeugt. Das erste Concept stimmt mit dem zum Training verwendeten überein, das zweite unterscheidet sich in den Gewichten der Labelfunktion je nach Ausprägung.

Inkrementeller Drift Für den inkrementellen Drift wurde ein einzelnes Concept gewählt, das mit den gleichen Gewichten wie das zum Training verwendete beginnt und einige der Attribute gemäß einer Änderungsrate anpasst. Die Änderungsrate wird je nach Ausprägung des Drifts gewählt.

Gradueller Drift Der graduelle Drift wurde nicht explizit betrachtet, da der inkrementelle Drift als ein Spezialfall diese Art bereits abdeckt. Mit dem Generator-Frameworks lässt sich ein gradueller Drift aber leicht simulieren.

Context Drift Zur Simulation von Context Drift werden für alle Concepts die gleichen Gewichte zur Labelbestimmung verwendet. Die Attribute werden normalverteilt generiert und die Mittelwerte je nach Art und Ausprägung des Drifts geändert.

Zur Analyse der verschiedenen Driftarten wird zunächst ein einzelner Drift, also eine einzelne Periode des Generators betrachtet.

5.2 Vergleich der Driftbehandlungsmethoden

Die Ansätze Naiv, Ensemble und Drifterkennung wurden jeweils mit Online- und Batch-Basislerner auf den verschiedenen Szenerien systematisch getestet. Die wichtigsten Ergebnisse werden im Folgenden zunächst am Beispiel eines abrupten Drifts präsentiert. Die verschiedenen Plots wurden so ausgewählt, dass die Erkenntnisse möglichst anschau-

lich dargestellt werden, sodass die Plots von verschiedenen Ausprägungen stammen und sich in der Anzahl der Items im jeweiligen Concept unterscheiden können.

Als Basislerner wurde *Naive Bayes* verwendet, da mit einem Lerner mit hohem Bias schon auf kleinen Trainingsmengen gute Modelle gelernt werden können. Als Online-Verfahren wird das Modell mit jedem Item aktualisiert. Für das Batch-Verfahren werden die Items der Trainingsmenge zunächst zu einem Batch gepuffert und dann darauf das konkrete Modell gelernt, indem die Items nacheinander zum Lernen verwendet werden. Dass Online- und Batch-Verfahren zumindest auf einem festen Batch zum gleichen Modell führen, erleichtert die Vergleichbarkeit der beiden Verfahren.

Die Drifterkennung verwendet die *DDM* als Detektor und stellvertretend für den Ensemble-Ansatz wurde das *Accuracy Weighted Ensemble* betrachtet.

Vergleich der Behandlungsmethoden

Zunächst werden die Ansätze mit der Batch-Version von Naive Bayes als Basislerner betrachtet. Analysiert wurden die Verfahren anhand der Vorhersagegenauigkeit bezüglich der letzten 500 Data Items.

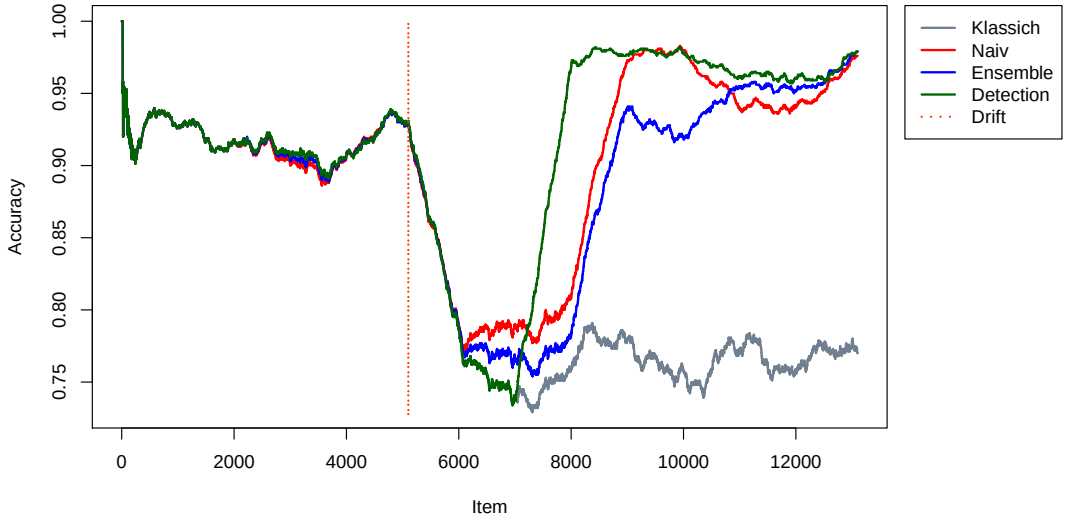


Abbildung 5.1: Vergleich von Behandlungsmethoden mit klassischem Naive Bayes.

Als Anhaltspunkt zur Beurteilung des Erfolgs der Behandlung und zum Vergleich der Behandlungsmethoden dient dabei ein Lerner ohne Driftbehandlung (klassisch). Der Lerner ohne Driftbehandlung erzielt auf dem neuen Concept eine in etwa konstante, aber deutlich geringere Vorhersagegüte als auf dem Concept, auf dem er trainiert wurde.

Für die verschiedenen Methoden zur Driftbehandlung lässt sich nun erkennen, wie weit nach dem Drift eine Verbesserung der Vorhersagegenauigkeit erreicht werden kann. Um

die Güte der Behandlung zu beurteilen ist jedoch vor allem interessant, welche Vorhersagegenauigkeit das neue Modell auf dem neuen Concept aufweist und wie schnell diese erreicht wird, da in der Praxis die Vorhersagen erst dann wieder verlässlich sind.

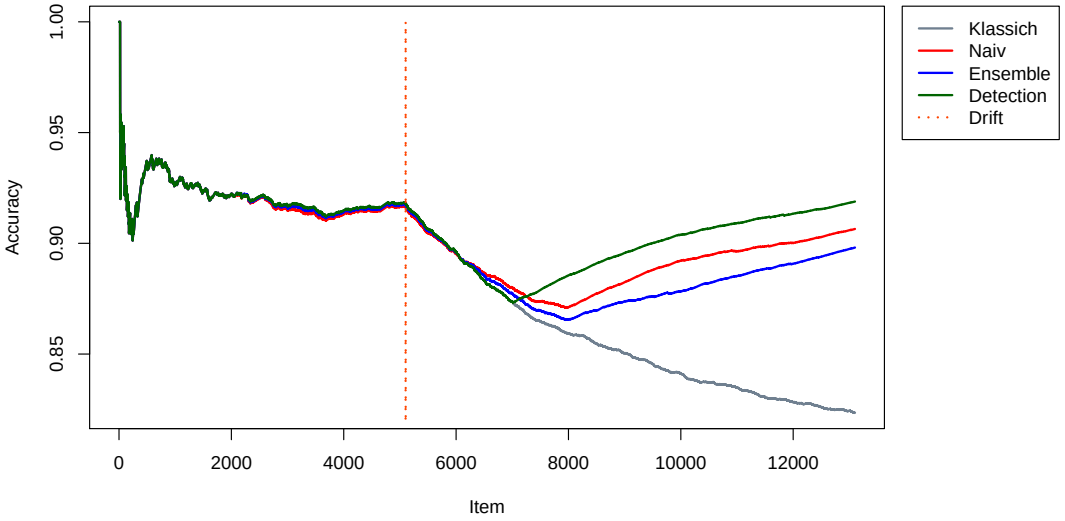


Abbildung 5.2: Vergleich von Behandlungsmethoden mit klassischem Naive-Bayes anhand der Gesamtvorhersagegüte.

Abbildung 5.2 vergleicht die Verfahren im gleichen Szenario anhand der Gesamtvorhersagegüte. Hier wird deutlich, dass diese tatsächlich ein gutes Maß ist, um die Güte von Driftbehandlungsverfahren anhand einer einzelnen Größe zu vergleichen. Da aber die Zeit bis zur Erkennung und zum Erreichen der neuen Vorhersagegenauigkeit nicht erkennbar ist, wird zum Vergleich der Verfahren jeweils die Vorhersagegenauigkeit bezüglich eines kleinen Fensters betrachtet.

Für den naiven Ansatz und das Ensemble wurde in dem konkreten Szenario aus Abb. 5.1 für jeden *Batch* eine Größe von 2000 gewählt. Entsprechend wird nach 6000 Items ein neues Modell gelernt, also zufällig sehr kurz nach dem Drift und insbesondere sogar bevor die DDM den Drift erkennt. Dennoch stammen die meisten Items in diesem Batch noch aus dem alten Concept. Dementsprechend kann eine leichte Verbesserung gegenüber der Methode ohne Behandlung beobachtet werden. Zuverlässige Vorhersagen stehen jedoch erst nach Lernen des nächsten Batch (nach 8000 Items) zur Verfügung. In realen Anwendungen, in denen es sehr lange dauert ein neues Modell zu Lernen, ist dieser Fall denkbar ungünstig.

Der Unterschied zwischen naivem Ansatz und Ensemble in der Abbildung lässt sich dadurch erklären, dass zwar in beiden Verfahren das gleiche neue Modell verwendet wird, für das Ensemble aber auch die Vorhersagen der älteren gelernten Modelle, die in diesem Fall auf dem alten Concept gelernt wurden, in die Klassifikationsentscheidung einfließen. Falls nicht alle alten Modelle mit 0 gewichtet werden, passt sich der naive Ansatz also zu Beginn besser an ein neues Concept an.

Unter der Grundannahme des Ensemble-Ansatzes, dass die kombinierte Vorhersage vieler Lerner besser ist als die eines einzelnen Lerners, werden mit einem Ensemble jedoch bessere Ergebnisse erzielt, wenn über mehr als einen Batch das gleiche Concept betrachtet wird. Durch eine generell höhere Klassifikationsgenauigkeit kann das Ensemble trotz schlechterer Anpassung auch direkt nach einem Drift eine absolut höhere Klassifikationsgenauigkeit erreichen.

Die Drifterkennungsmethode erkennt den Drift nach ca. 2000 Items - früher kann ein neues Modell des naiven Verfahrens, das ausschließlich auf dem neuen Concept gelernt wurde, nicht zur Verfügung stehen - und die gepufferte Trainingsmenge ist ausreichend groß um die endgültige Vorhersagegüte zu erreichen.
Das Verhältnis zwischen naivem Ansatz, bzw. Ensemble, und der Drifterkennung hängt maßgeblich davon ab, wie groß ein einzelner Batch ist. Wird im konkreten Fall eine kleinere Batchgröße verwendet, kann schneller ein gutes Modell zur Verfügung stehen als mit der Driftbehandlung. Jede Verkleinerung der Trainingsmenge kann sich jedoch negativ auf die Güte des neuen Modells auswirken.

Wenn ein Drift nicht oder erst spät erkannt wird, lassen sich mit dem naiven Verfahren oder einem Ensemble natürlich deutlich bessere Ergebnisse erzielen. Wenn die Driftdetektoren den Drift so zuverlässig und schnell erkennen wie in diesem Szenario, liefert die Drifterkennung deutlich die besten Ergebnisse und das bei deutlich geringerem Aufwand für das Neulernen von Modellen. Insbesondere in Anwendungsfällen, in denen das Modelllernen sehr lange dauert und ein großer Batch benötigt wird, um ein zuverlässiges Modell zu erhalten, ist die Drifterkennung aufgrund der expliziten Bestimmung des Zeitpunktes zum Neulernen überlegen.
Für einen konkreten Anwendungsfall hängt die Entscheidung, welcher Ansatz die besten Ergebnisse liefert, also vor allem davon ab, wie gut der Detektor den entsprechenden Drift erkennt.

5.2.1 Vergleich der Basislerner

Um auch das Online-Verfahren und die Drifterkennnung mit einem Online-Basislerner mit den bisher betrachten Ansätzen zu vergleichen, kann Abbildung 5.3 betrachtet werden:

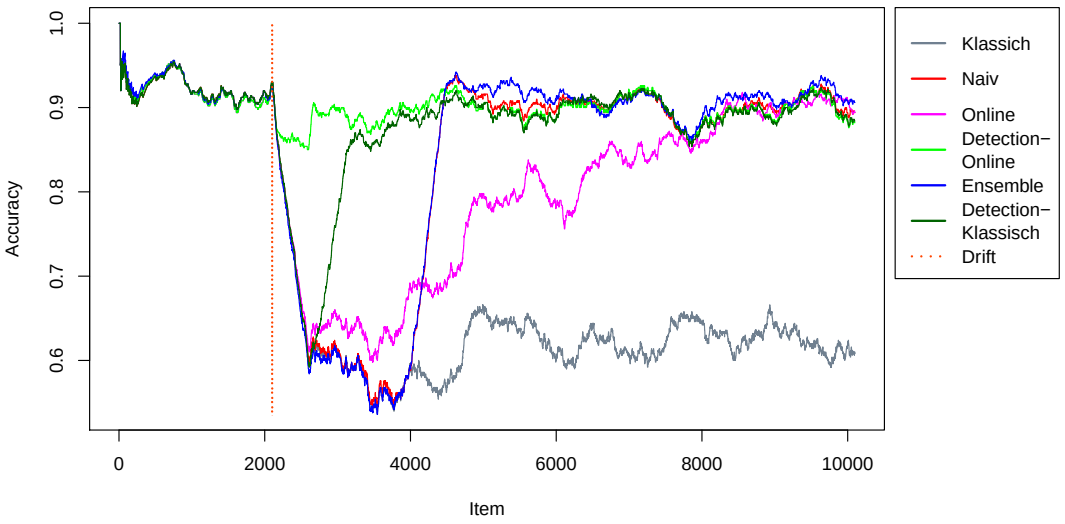


Abbildung 5.3: Vergleich von Behandlungsmethoden mit Online- und Batch-Basislerner.

Aufgrund der Aktualisierung des Modells mit jedem Item passt ein Online-Verfahren sich langsam dem neuen Concept an. *Klassisch* bezeichnet in dem Plot nach wie vor den Batch-Lerner ohne Driftbehandlung. Dennoch steht mit allen Ansätzen, die das Modell nach dem Drift neu lernen - egal ob online oder im Batch -, deutlich schneller ein gutes Modell zur Verfügung. Je leichter der Drift ist, desto besser passt sich das Online-Verfahren (auch im Verhältnis zu expliziten Driftbehandlungsverfahren) an.
Etwas überraschend ist für mich, dass in Verbindung mit Online-Verfahren der Drift meistens vom gleichen Drift-Detektor schneller erkannt wird, obwohl sich die Fehlerrate eigentlich weniger stark verändern sollte, als in Verbindung mit einem Batch-Verfahren.

5.2.2 Vergleich verschiedener Szenarien

Die experimentelle Auswertung zeigt, dass die strukturellen Unterschiede der Ansätze in allen Szenarien bestehen und sich die Szenarien hauptsächlich dadurch unterscheiden, wie zuverlässig und schnell die Driftdetektoren einen Drift erkennen (Abb. 5.4). Daher erfolgt eine Betrachtung der verschiedenen Szenarien in Kapitel 6 (Drifterkennung).

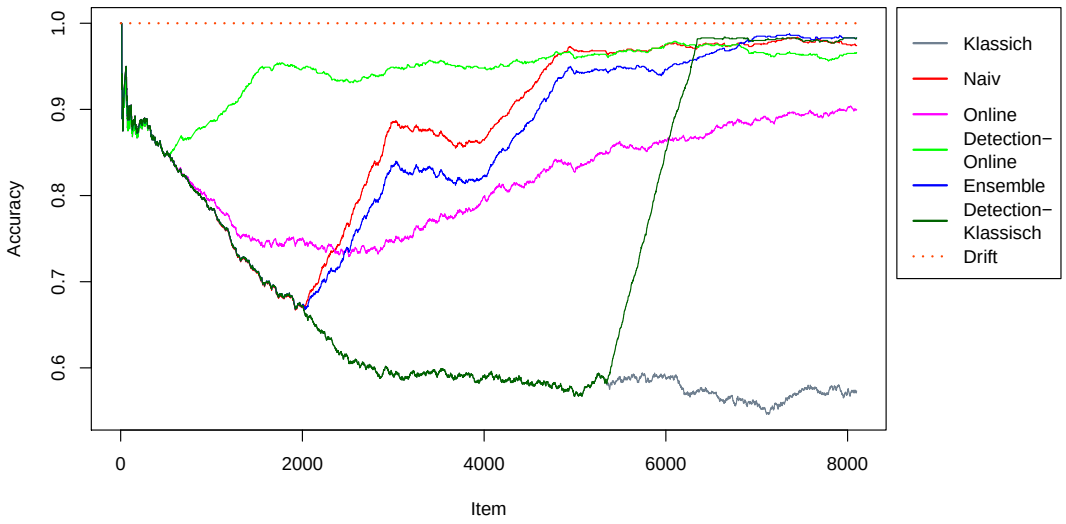


Abbildung 5.4: Vergleich der Behandlungsmethoden in einem Szenario für inkrementellen Drift

Abb. 5.4 zeigt zum Vergleich einen inkrementellen Drift. Für die Erkennung wird hier die EDDM [1] als Detektor verwendet, da sich damit auf langsamen graduellen Drifts gute Ergebnisse erzielen lassen [8]. Beim inkrementellen Drift ändert sich das Concept gewissermaßen mit jedem Data Item ein wenig. Entsprechend lassen sich für leichte inkrementelle Drifts mit Online-Verfahren gute Ergebnisse erzielen.

6 Evaluation von Drifterkennung

Wie im letzten Kapitel festgestellt wurde, hängt die Vorhersagegüte der Drifterkennung stark davon ab, wie gut der verwendete Driftdetektor funktioniert. Mit den in Kapitel 3 definierten Gütemaßen für Driftdetektoren lässt sich dies untersuchen.
Von Interesse ist vor allem die Zeit, die benötigt wird, um den Drift zu erkennen (MTD), da während dieser Zeit mit dem alten Modell schlechte Vorhersagen getroffen werden. Falls der Detektor keinen Drift erkennt, wird die MTD als Null gemessen. Dennoch werden, falls es tatsächlich einen Drift gab, dauerhaft schlechte Vorhersagen verwendet. Deshalb wird zusätzlich die Anzahl nicht erkannter Drifts (MDR) betrachtet.
Da mit jeder Erkennung ein neues Modell gelernt wird, kann eine hohe Zahl von falschen Alarmen die Vorhersagegenauigkeit erhöhen, weshalb ebenfalls die MTFA und das kombinierte Maß MTR betrachtet werden.

In [5] findet sich eine experimentelle Analyse verschiedener Driftdetektoren auf graduellen Drifts mit Hilfe eines Fehlerratengenerators. Besonders gute Ergebnisse erzielte dort der *ADWIN*-Driftdetektor [2].

Interessant ist nun vor allem, welche Resultate die verschiedenen Driftdetektoren auf abrupten Drifts erzielen. Daher wurden die Driftdetektoren ADWIN, DDM, EDDM jeweils mit dem Fehlergenerator für abrupten Drift untersucht.
Zunächst wurden 10.000 Items mit einer Fehlerrate von 0,2 erzeugt, und anschließend noch einmal 10.000 Items mit einer um β erhöhten Fehlerrate. Alle Werte wurden über 100 Experimente gemittelt. Um die unterschiedlichen Ausprägungen des Drifts zu simulieren wurden verschiedene Werte für β betrachtet:

DDM

Kriterium	$\beta = 0.05$	$\beta = 0.1$	$\beta = 0.2$
MTD	1110,1	1533,9	1083,8
MDR	0,42	0,18	0,02
MTFA	11.403,4	11.726,7	10.833,1
MTR	6,0	6,3	10,0

EDDM

Kriterium	$\beta = 0.05$	$\beta = 0.1$	$\beta = 0.2$
MTD	141,5	80,1	59,0
MDR	0	0	0
MTFA	563,0	612,9	831,75
MTR	4,0	7,7	14,1

ADWIN

Kriterium	$\beta = 0.05$	$\beta = 0.1$	$\beta = 0.2$
MTD	398,6	93,9	35,7
MDR	0	0	0
MTFA	841,5	223,8	125,1
MTR	2,1	2,4	3,5

Offensichtlich erkennen alle Detektoren einen stärken Drift auch schneller. Die DDM verursacht im Gegensatz zu beiden anderen Methoden kaum einen falschen Alarm, benötigt dafür aber auch deutlich länger um einen Drift zu erkennen. Die EDDM erreicht jeweils den höchsten MTR-Wert und wäre nach diesem Kriterium zu bevorzugen.

7 Concept Drift in zyklischen Prozessen

In diesem Kapitel werden die vorgestellten Verfahren in einem zyklischen Szenario betrachtet, um insbesondere zu untersuchen, wie die Verfahren mit wiederkehrenden Concepts umgehen. Besonders effiziente Eigenschaften und Probleme der Verfahren werden herausgearbeitet, sodass eine Skizze für ein Verfahren speziell für die Behandlung von zyklischem Concept Drift entsteht.

7.1 Evaluation

Zum konkreten Vergleich der Ansätze wurden die Szenarien aus Kapitel 5 über mehrere Zyklen betrachtet. Das Verhalten der verschiedenen Ansätze lässt sich zum Beispiel an mehreren Zyklen des Szenarios für starken abrupten Concept Drift betrachten.
Die Größe für einen einzelnen Batch wurde (auch im Ensemble) erneut auf 2000 festgelegt und die Auswertung beginnt nach einem Vorlauf von 2000 Items, sodass zu Beginn der Auswertung für jeden Ansatz ein vergleichbares Modell zur Verfügung steht. Erneut wird Naive Bayes verwendet, um die Vergleichbarkeit von Online- und Batch-Basislerner zu verbessern. Während einer Periode werden aus jedem Concept jeweils 5000 Items betrachtet.

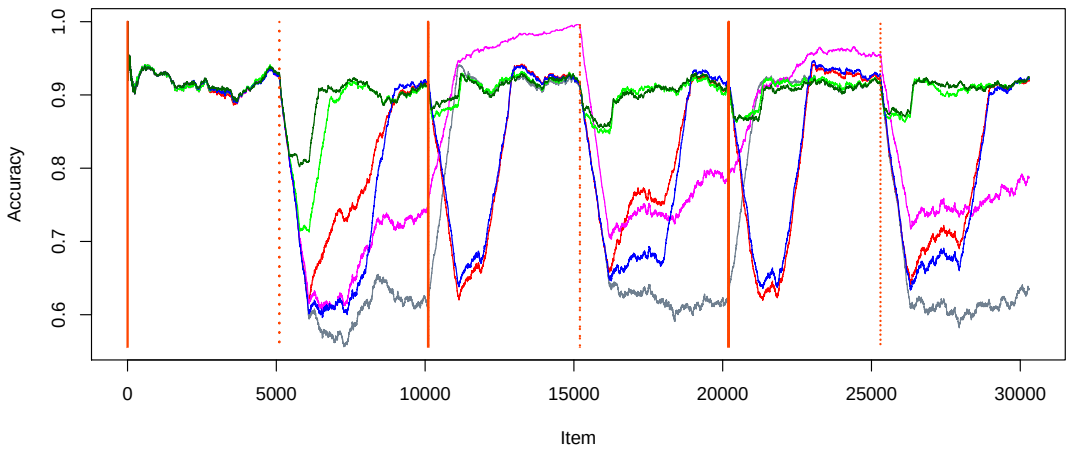


Abbildung 7.1: Vergleich der Behandlungsmethoden in einem Szenario mit zyklischem Drift.

Die Basislerner erreichen jeweils auf dem ersten Concept einer Periode, das mit dem der Trainingsmenge übereinstimmt, eine konstant hohe Vorhersagegenauigkeit und auf dem neuen Concept eine niedrige. Das online Verfahren passt sich auch über mehrere Perioden hinweg mit der Zeit an das neue Concept an.
Das Drifterkennungsverfahren erkennt jeden Drift und lernt jeweils entsprechend ein Modell. Auffällig ist, dass ab der zweiten Periode der Drift früher erkannt wird, sodass die Vorhersagegüte kürzer und weniger stark abfällt.
Für den naiven Ansatz und das Ensemble fällt die Vorhersagegüte nach einem Drift stark ab. Mit der gewählten Batchgröße steht jeweils 1000 Items nach dem Drift ein neues Modell zur Verfügung, das zur Hälfte auf dem alten und zur Hälfte auf dem neuen Concept gelernt wurde, und nach 3000 Items das erste Modell, das komplett auf dem neuen Drift gelernt wurde.
Insbesondere wird deutlich, dass das Ensemble nicht schneller auf den Drift reagiert, obwohl bereits bei der Auswertung nach 1000 Items Modelle zur Verfügung stehen, die komplett auf dem entsprechenden Concept gelernt wurden. Die Ursache liegt darin, dass der letzte Batch dann jeweils aus der Übergangsphase stammt, sodass die Modelle noch nicht gemäß dem neuen Concept gewichtet werden.

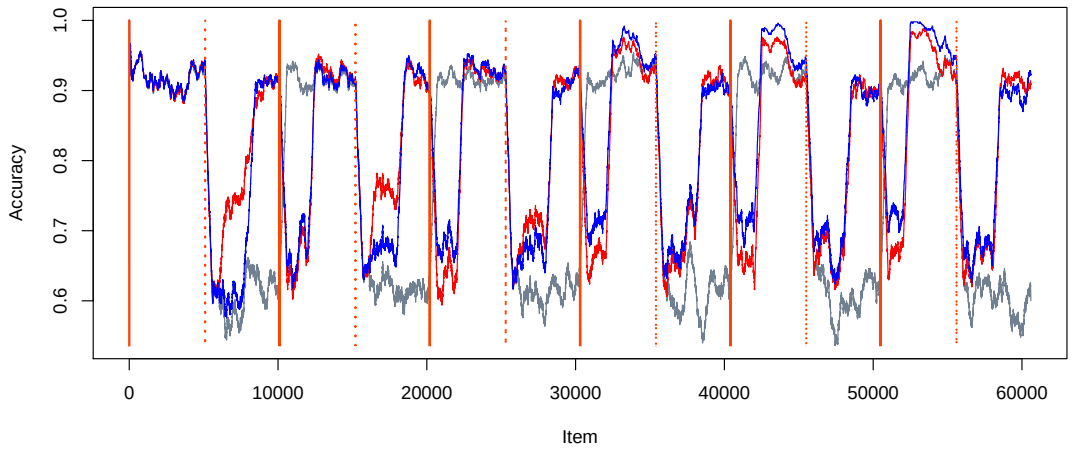


Abbildung 7.2: Vergleich von naivem Ansatz und Ensemble in einem zyklischen Drift.

Abbildung 7.2 betrachtet den naiven Ansatz und das Ensemble über einen längeren Zeitraum. Hier wird deutlich, dass die verzögerte Anpassung des Ensembles im Gegensatz zum naiven Verfahren besonders während der ersten Periode auftritt, wenn noch keine Modelle aus dem neuen Concept zur Verfügung stehen. Über die Zeit kann ein Ensemble, da mehr Modelle und somit mehr Trainingsdaten zur Verfügung stehen, bessere Ergebnisse erzielen als der naive Ansatz.
Zur Anschauung zeigt Abbildung 7.3 die totale Vorhersagegenauigkeit der beiden Ansätze, wenn beide Concepts übereinstimmen (das Experiment wurde so durchgeführt, dass

nicht nur die Parameter der Generatoren übereinstimmen, sondern exakt die gleichen Daten betrachtet werden, also insbesondere auf jedem Batch das gleiche neue Modell trainiert wird).

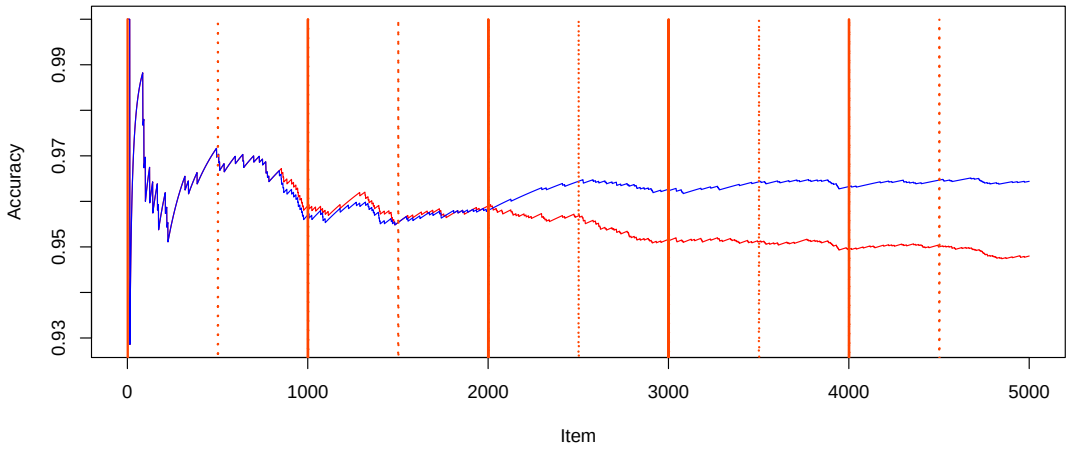


Abbildung 7.3: Naiver Ansatz und Ensemble auf einem konstanten Concept.

Im Vergleich der Methoden hat die Drifterkennung die besten Ergebnisse erzielt, wenn Szenarien betrachtet wurden, in denen die Detektoren die Drifts schnell und zuverlässig erkennen. Für den zyklischen Prozess ergeben sich für diesen Ansatz entsprechend gute Ergebnisse als Summe der guten Ergebnisse auf den einzelnen Drifts. Wiederkehrende Concepts werden jedoch nicht ausgenutzt, sondern jedes Mal erst ein neues Modell gelernt. Dem gegenüber kann ein Ensemble bei Eintreten eines Drifts auf Modelle zurückgreifen, die in einer früheren Periode auf dem neuen Concept gelernt wurden.

7.2 Behandlung von zyklischem Drift

Obwohl bei einem zyklischen Drift in einem Ensemble idealerweise zu Beginn jedes Drifts schon Modelle des neuen Concepts aus früheren Perioden zur Verfügung stehen, werden diese mit dem *Accuracy Weighted Ensemble (AWE)* nicht optimal ausgenutzt. Die bereits analysierten Probleme des Ansatzes sind:

Problem 1 Bedingt durch die feste Batchgröße reagiert das Ensemble (wie bereits in Kapitel 5 beschrieben) unter Umständen sehr spät auf einen Drift.

Problem 2 Die Verwendung eines Concepts aus der Übergangszone zur ersten Auswahl und Neugewichtung des Ensembles nach einem Drift bedeutet schlechte Vorhersagen für einen weiteren ganzen Batch.

Sinnvoll im Hinblick auf wiederkehrende Concepts ist, dass das AWE für ein Ensemble der Größe s insgesamt $m > s$ Modelle speichert, sodass jeweils nur die Vorhersagen der besten s verwalteten Modell kombiniert werden. So können Modelle des gerade nicht aktuellen Concepts verwahrt werden, ohne die aktuelle Vorhersagegüte zu beeinflussen. Aufgrund der Anforderung der Speicherbeschränkung muss mit Hinzunahme des neu gelernten Modells auch ein Modell entfernt werden. Im AWE wird jeweils das Modell entfernt, das auf dem letzten Batch die geringste Vorhersagegenauigkeit erreicht hat. Daraus ergeben sich jedoch weitere Probleme.

Problem 3 In Abhängigkeit von der festen Wahl von m und der Batchgröße b kann es passieren, dass bis zum nächsten Drift bereits alle Modelle des neuen Concepts ersetzt wurden. Wählt man beispielsweise in dem oben verwendeten konkreten Szenario mit Conceptgröße 5000 die Werte $m = 10$ und $b = 1000$ werden nach der ersten Periode fünf Modelle zu jedem Concept verwaltet. Während der ersten Hälfte der zweiten Periode werden nun die Modelle des zweiten Concepts durch die neugelernten Modelle ersetzt, sodass bei dem Drift nach dem 15.000sten Item nicht auf Modelle des bereits betrachteten zweiten Concepts zurückgegriffen werden kann.

Problem 4 Wenn immer nur das Modell mit der aktuell geringsten Vorhersagegenauigkeit entfernt wird, kann es passieren, dass ein Modell, welches auf einem anderen Concept hervorragende Ergebnisse liefert, entfernt wird und Modelle, die auf allen Concepts nur mittelmäßige Ergebnisse erreichen, dauerhaft gespeichert werden.
Zur Anschauung kann ein zyklischer Drift mit den Concepts C_1, C_2 und $m = s = 3$ betrachtet werden.
Angenommen, die verwalteten Concepts sind M_1, M_2 und M_3, wobei M_1 auf C_1 trainiert wurde, M_2 auf C_2, und M_3 aus der Übergangsphase stammt, also je zur Hälfte auf C_1 und C_2 trainiert wurde. Falls nun die Modelle auf einem neuen Batch B aus C_1 getestet werden, liefert M_2 vermutlich die schlechtesten Vorhersagen und wird ersetzt. Falls B aus C_2 stammt, würde analog M_1 entfernt. Stammt B ebenfalls aus der Übergangsphase wird vermutlich entweder M_1 oder M_2 ersetzt.

Das erste Problem kann für einen einzelnen Drift aufgrund der Abhängigkeit der Qualität des gelernten Modells von der verwendeten Trainingsgröße nicht durch eine beliebig kleine Batchgröße behoben werden. Stehen in einem zyklischen Prozess jedoch jederzeit Modelle des neuen Concepts zur Verfügung, kann bereits durch bloße Neugewichtung angemessen auf den Drift reagiert werden.
Entsprechend ist eine Entkopplung des zeitaufwendigen Neulernens, für das eine Mindestanzahl von Items benötigt wird, von der Auswahl des konkreten Ensembles aus den vorhandenen Lernern und der Aktualisierung der Gewichte dieser Lerner möglich. Falls das Intervall der Aktualisierungen der Gewichte deutlich kürzer als der zum Lernen verwendete Batch ist, kann schneller auf einen Drift reagiert werden.
Aufgrund des zweiten Problems sollte auch das Fenster zur Schätzung der aktuellen Vorhersagegenauigkeiten der einzelnen Lerner entsprechend verkleinert werden. Hier bleibt die Problematik der ungenauen Schätzung mit kleiner Testmenge bestehen.

Wenn die Anpassung an ein wiederkehrendes Concept bereits durch die Gewichtung erfolgt, dient das Lernen eines neuen Modells vor allem der Diversifizierung des Ensembles, um durch Kombination mit einem weiteren Modell des aktuellen Concepts bessere Vorhersagen zu erzielen. Dann reicht es theoretisch aus, zwischen zwei Driftzeitpunkten ein einziges Modell zu lernen. Dies entspricht genau der Idee der Drifterkennung, ließe sich also durch Bestimmung der Lernzeitpunkte und Trainingsmengen in einem gewichteten Ensemble von Lernern mittels eines Driftdetektors realisieren.
Dieses hybride Verfahren behandelt also ein wiederkehrendes Concept durch Neugewichtung und ein bisher unbekanntes Concept durch Drifterkennung.
Nachteil dieser Anpassung ist, dass nun wie bei der klassichen Drifterkennung eine Abhängigkeit von der Güte des Detektors entsteht.
Als Eingabe des Detektors ließen sich zur zuverlässigen Erkennung zum Beispiel die Fehler des Lerners mit dem höchsten Gewicht verwenden.

Das Auftreten des dritten Problems kann nicht immer verhindert werden, wie an dem Extremfall deutlich wird, dass ein Zyklus mehr verschiedene Concepts enthält, als Modelle gespeichert werden. Ein Ansatz zur Adressierung des generellen Problems könnte die Entwicklung eines Ähnlichkeitsmaßes für Concepts (zum Beispiel anhand der Definition der *Quelle* in Abschnitt 2.1) sein. Darüber könnten Modelle gemäß der zugehörigen Concepts gruppiert werden und die Abdeckung aller Concepts bei der Ersetzung beachtet werden.

Zur Vermeidung des vierten Problems kann zum Beispiel auf jedem Batch das Modell mit der besten Vorhersagegenauigkeit markiert werden und entsprechend jeweils das Modell, das am längsten nicht markiert wurde, ersetzt werden.

7.3 ELEC

Abschließend wurden die Ansätze auch auf dem in Kapitel 4 beschriebenen, realen Datensatz des australischen Strommarktes miteinander verglichen. Auffallend im Verhältnis zu den Experimenten auf den synthetischen Datensätzen sind vor allem die schlechten Ergebnisse der Drifterkennung mit Batch-Basislerner. Die schnellere Erkennung von Drifts mit dem Online-Basislerner und die Abhängigkeit der Drifterkennung von der Zuverlässigkeit der Bestimmung der Driftzeitpunkte sind wieder zu erkennen. Das Online-Verfahren kann auch in diesem Fall durch die Verknüpfung mit einem Driftdetektor verbessert werden. Insgesamt lassen sich mit der Drifterkennung mit Online-Basislerner hier die besten Ergebnisse erzielen.

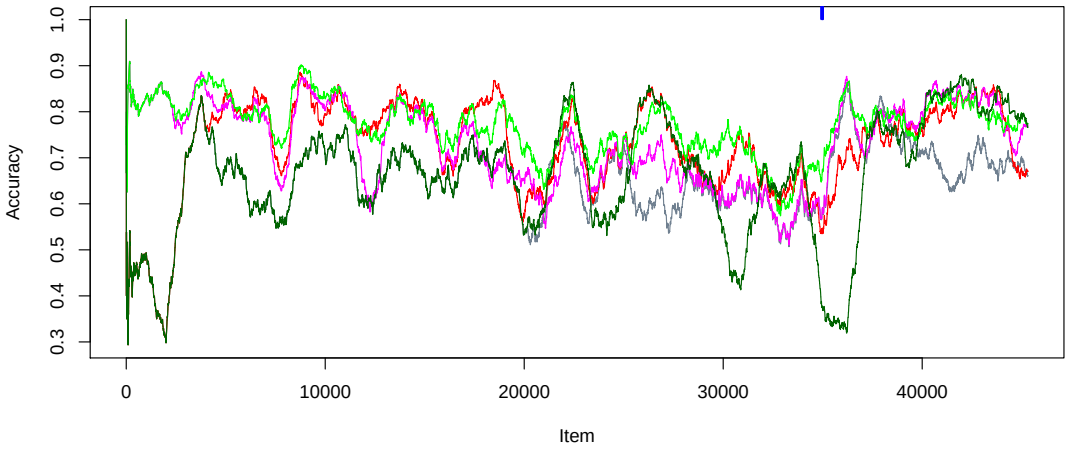


Abbildung 7.4: Vergleich der Behandlungsmethoden in einem Szenario mit zyklischem Drift.

8 Implementierung

Dieses Kapitel dient vor allem der besseren Nachvollziehbarkeit der durchgeführten Experimente und zur Dokumentation von wiederverwendbaren Implementierungen.
Zur Durchführung der Experimente wurde das streams Framework verwendet. Um größtmögliche Wiederverwendbarkeit der Lernverfahren und Driftbehandlungsmethoden des MOA-Frameworks zu gewährleisten, wurde eine Schnittstelle zwischen dem streams Framework und MOA geschaffen.
Das Projekt *Stream-ConceptDrift* liefert Funktionalität zur Evaluation von Methoden zur Driftbehandlung, eine Implementierung der in Kapitel 4 vorgestellten synthetischen Datengeneratoren, sowie eine Experimentierumgebung mit den Definitionen der zur Evaluation durchgeführten Experimente.
Für streams Framework und MOA wurde zunächst ein Fork erstellt. Der Code von Stream-ConceptDrift wird nach Auswahl wiederverwendbarer Elemente und ausführlicher Dokumentation – falls die erstellten Forks von MOA und streams Framework noch benötigt werden, gemeinsamen mit diesen – unter `https://bitbucket.org/StefanRot` veröffentlicht.

8.1 Stream-Framework

Das streams Framework [7, 6] dient - wie der Name nahe legt - der Verarbeitung von Datenströmen. Ein entscheidendes Feature ist, dass sich sowohl Datenströme als auch die Datenstromverarbeitung nach einem Baukastensystem in XML-Format fexibel definieren lassen. Die wichtigsten Elemente des Frameworks werden im Folgenden kurz vorgestellt:

Data Item Ein Data Item wird durch eine Map realisiert, in der die Attribute als (Name,Wert)-Paare verwaltet werden.

Stream Datenströme werden im Stream-Framework über das Interface *Stream* definiert. Ein Stream ist ein passives Element, das einem verarbeitenden *Process* auf Anforderung (über die Methode `public DataItem read()`) das nächste Data Item zur Verfügung stellt.

Processor Ein *Processor* ist ein atomarer Bestandteil eines *Process*, der eine Methode bereitstellt um das aktuelle Data Item zu verarbeiten (`public DataItem process(DataItem input)`). Als Ergebnis der Verarbeitung erhält man wieder ein Data Item, welches von nachfolgenden Prozessoren zur Verarbeitung genutzt wird. Damit können Prozessoren optional das Data Item verändern (beispielsweise At-

tribute umbenennen, entfernen oder solche, die von nachfolgenden Prozessoren benötigt werden, ergänzen).

Process Ein Verarbeitungsvorgang kann in einem Process zusammengefasst werden. Ein Process verwaltet eine Reihe von Prozessoren, die sequentiell abgearbeitet werden. Dazu wird dem ersten *Processor* das vom Stream erhaltene, aktuelle Data Item zur Verarbeitung übergeben. Das als Ergebnis erhaltene Data Item wird dann dem jeweils nächsten Prozessor übergeben. Nach Abarbeitung des letzten Prozessors wird vom zugehörigen Stream das nächste Data Item angefordert.

Service Anytime-Anforderungen werden im Stream-Framework über das Interface *Service* realisiert. Services sind über eine ID ansprechbar, sodass jederzeit auf sie zugegriffen werden kann. Konkrete Anytime-Anforderungen lassen sich dann durch Methoden in abgeleiteten Interfaces realisieren. Als Beispiel kann ein Online-Klassifikator im `streams` Framework dienen. Das inkrementelle Lernen wird über die Methode `process(DataItem input)` eines Prozessors realisiert. Die für einen Klassifikator typische Anytime-Anforderung der Vorhersage der Klasse eines neuen Data Items kann man dann dadurch realisieren, dass dieser Prozessor das Interface Prediction-Service mit der Methode `predict(DataItem item)` implementiert.

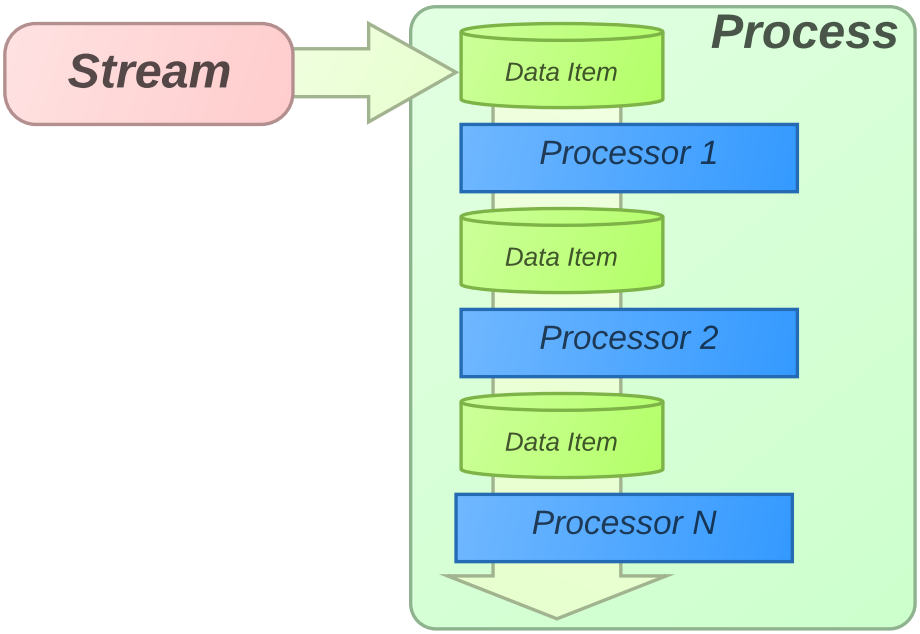


Abbildung 8.1: Datenstromverarbeitung im Stream-Framework [6]

```
<container>
        <stream  id="stream"
                class="stream.generator.ExampleStream"
                limit="1000"/>

        <process>
                <stream.learner.Prediction learner="lerner1" />
                <stream.learner.evaluation.PredictionError />
                <stream.classifier.MyClassifier id="lerner1" alpha="0.01" />
        </process>
</container>
```

Abbildung 8.2: XML-Definition einer Datenstromanwendung im streams Framework

Diese zentralen Komponenten stehen als abstrakte Java-Klassen oder Interfaces zur Verfügung und konkrete Instanzen lassen sich durch abgeleitete Klassen realisieren. Ein konkreter Prozessor muss beispielsweise die Methode `public DataItem process(DataItem input)` überschreiben um eigenes Verhalten umzusetzen. In einer XML-Datei werden diese Komponenten dann zu einer konkreten Anwendung zusammengesetzt. Dabei können Parameter der Java-Klassen, für die Getter und Setter implementiert sind, ebenfalls über die XML-Datei angegeben werden. Als einfaches Beispiel mit einem einzelnen Datenstrom, der von einem einzelnen Process verarbeitet wird, kann die XML-Datei zu obigem Beispiel eines typischen Klassifikators betrachtet werden (Abb. 8.2),

8.2 MOA

MOA [3] ist ein Framework für die Durchführung und Evaluation von Datenstrom-Experimenten und zur Implementierung entsprechender Lernverfahren. Ursprünglich lag der Fokus auf Klassifikation und Clustering. In jüngerer Zeit wurde das Framework um Funktionalität und Lernverfahren für Regressionsaufgaben und Concept Drift erweitert.

Als zentrales Konzept von MOA sei an dieser Stelle noch einmal auf die vier Anforderungen an Datenstrom-Lernverfahren verwiesen 3.1. Die Speicheranforderung wird durch die Auswahl aus drei festen Speichergrößen realisiert. Verarbeitungszeit und benötigter Speicher werden gemessen.

MOA ist eng mit dem WEKA-Projekt [13], einer bekannten Bibliothek für klassische Lernverfahren, verwandt und bietet Wrapper um auf diese Verfahren zurückzugreifen.

Neben Schnittstellen für die meisten Standardformate von Datensätzen finden sich in MOA auch eine ganze Reihe von Generatoren für synthetische Daten. Auch für Datenströme mit Concept Drift oder entsprechende Fehlerraten sind zumindest für Klassifikationsaufgaben Generatoren enthalten.

Zusätzlich zu Framework und Lernverfahren bietet MOA eine umfassende Sammlung an Evaluationsmethoden für verschiedene Experimentarten und ein GUI zur Konfiguration der Experimente.

Für diese Arbeit sind insbesondere die Implementierungen verschiedener Online-Lernverfahren (z.B. NaiveBayes, HoeffdingTree) und klassische Lernverfahren aus WEKA als Basislerner in Methoden zur Driftbehandlung interessant. Darüber hinaus wird die Implementierung der in Kapitel 3 beschriebenen Driftbehandlungsmethoden verwendet:

Ensemble Accuracy Weighted Ensemble

Driftdetektor Driftdetektoren implementieren in MOA das Interface *ChangeDetector*. Konkret stehen die Detektoren *Geometric Moving Average, Page-Hinkley-Test, EWMA-Chart, ADWIN, DDM* und *EDDM* zur Verfügung.
Eine kurze Beschreibung dieser Detektoren und der Concept Drift bezogenen Funktionalität von MOA findet sich in [5]

Drifterkennung Die Komibination eines Driftdetektors mit einem Basislerner wird durch den *DriftDetectionMethodClassifier* realisiert.

Naiv Der *WEKAClassifier* kann verwendet werden, um klassische WEKA-Lerner auf einen Stream anzuwenden - bietet also Funktionalität wie das Zusammenfassen von Items zu Batches in Verbindung mit regelmäßigem Neulernen des Modells.

8.2.1 Anpassungen

Im Zuge dieser Arbeit wurden Anpassungen an einigen MOA-Klassen vorgenommen. Dazu wurde zunächst ein Fork des MOA-Repository erstellt. Nach Abschluss der Arbeit werden die Änderungen in ein separates Maven Projekt [1] mit MOA als Dependency ausgelagert, sodass die Schnittstelle des streams Framework (auch ohne diese Änderungen) mit dem offiziellen MOA-Release verwendet werden kann.

Der **WEKAClassifier** dient dazu klassische Lernverfahren in MOA und insbesondere als Basislerner für den *DriftDetectionMethodClassifier* zur Verfügung zu stellen. Dabei kann als Parameter die Anzahl der Items, die zum Lernen des initialen Modells verwendet werden sollen, angegeben werden. Bei der Durchführung der Experimente zeigte sich in Verbindung mit klassischen Lernern unerwünschtes Verhalten, wenn ein Drift erkannt wurde, bevor seit Erreichen des *warn level* diese initiale Anzahl von Items gepuffert wurde (Abb. 8.3).

[1] `http://maven.apache.org/`

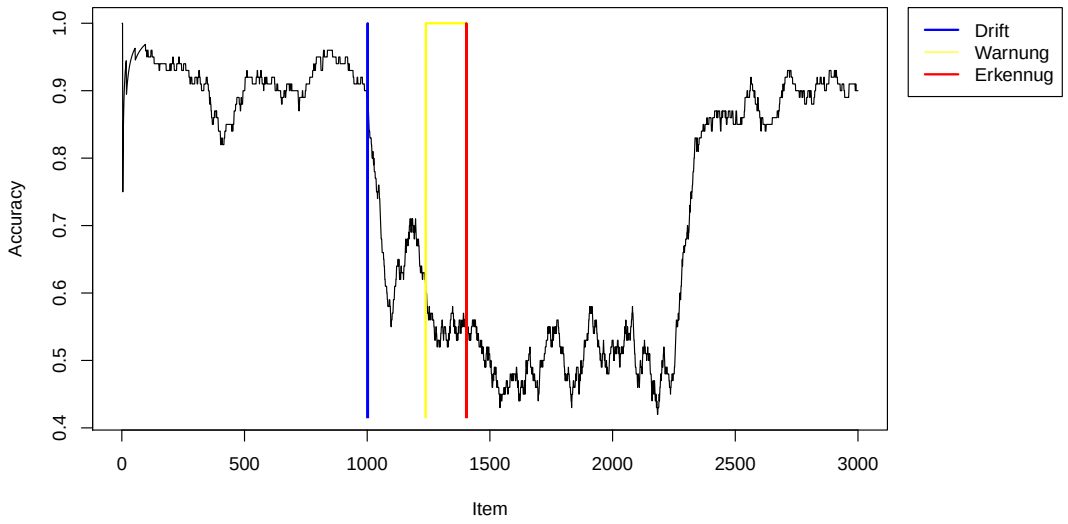


Abbildung 8.3: Drifterkennung mit WEKALearner vor der Anpassung

Effektiv wurde das neue Modell erst verwendet, nachdem diese Anzahl an Items wirklich vergangen war (in diesem Fall 1000). Dennoch wurden zum Lernen dann nur die Items, die zwischen Warnung und Erkennung gepuffert wurden, zum Lernen verwendet.

Dies wurde so korrigiert, dass nun mit Erreichen des *drift level* sofort ein neues Modell gelernt und zur Vorhersage verwendet wird. Um dennoch die Güte eines Modells mit der entsprechend angegebenen Trainingsgröße zu erreichen, werden die Data Items weiterhin gepuffert und dann erneut ein Modell gelernt (Abb. 8.4).

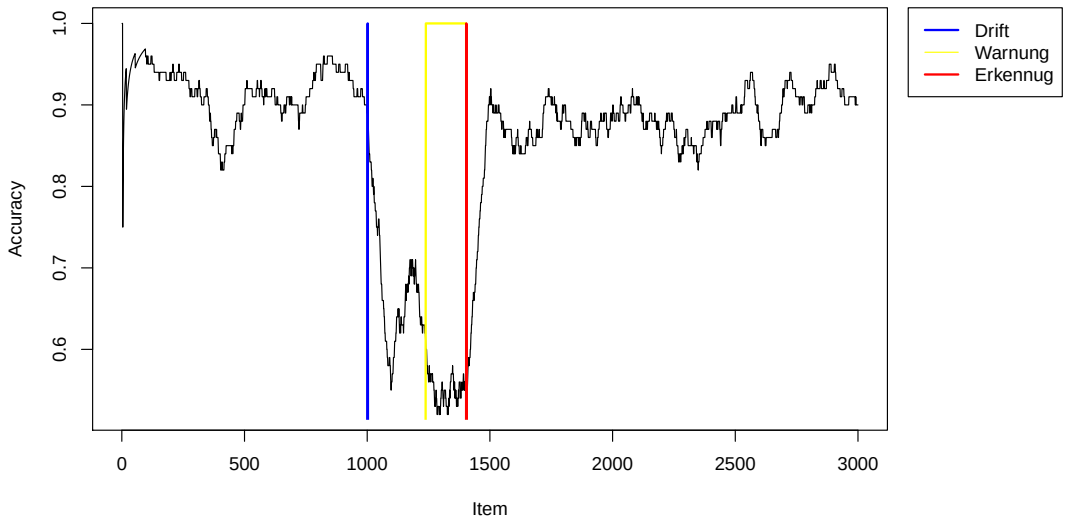


Abbildung 8.4: Drifterkennung mit WEKALearner nach der Anpassung

8.3 Schnittstelle: Streams-MOA

Der Hauptzweck der Schnittstelle zu MOA ist, dass MOA-Klassen als *Processor, Stream* oder *Service* im Stream-Framework verwendet werden können. Dazu wurden für diese Klassen nach dem in [12] beschriebenen Entwurfsmuster *Adapter* entsprechende Wrapper erstellt.

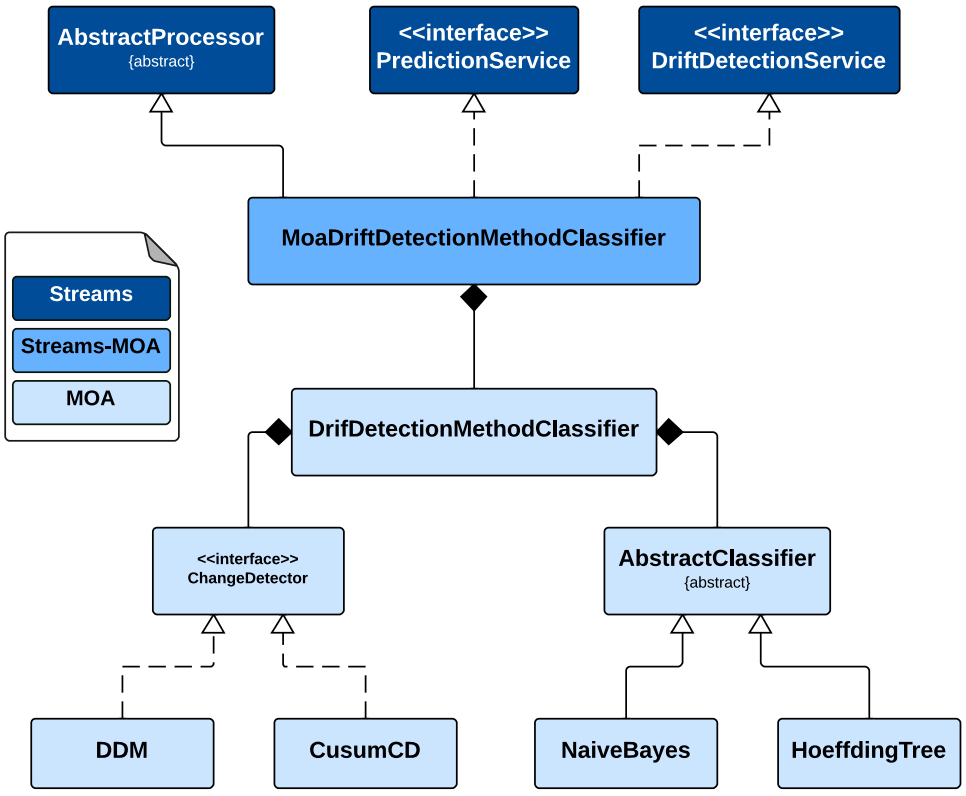


Abbildung 8.5: Schnittstelle MOA-Streams: Vereinfachtes Klassendiagramm am Beispiel des MoaDriftDetectionMethodClassifier

MoaClassifier und *MoaDriftDetectionMethodClassifier* erhalten die entsprechende MOA-Klasse als Attribut und können als Prozessor und PredictionService im Stream-Framework verwendet werden. *MoaChangeDetector* ist ein Wrapper für die Moa-Klasse *ChangeDetector* und kann als alleinstehender Prozessor (d.h. unabhängig von MoaDriftDetectionMethodClassifier) verwendet werden.
Sowohl *MoaDriftDetectionMethodClassifier* als auch *MoaChangeDetector* implementie-

ren den *DriftDetectionService* aus dem streams Framework, sodass sie in dem neuen Projekt Stream-ConceptDrift bezüglich ihrer Erkennungsgüte evaluiert werden können. Attribute der ursprünglichen MOA-Klassen können dabei in der XML-Datei als zusätzliche Attribute der streams Prozessoren angegeben werden oder gemäß des in MOA definierten Kommandozeileninterface zusammen mit dem Klassennamen angegeben werden.

```
<moa.classifiers.drift.DriftDetectionMethodClassifier id="ddm"
       driftDetectionMethod="DDM -n 100"
       baseLearner="meta.WEKAClassifier -l weka.classifiers.trees.J48"
       minNumInstances="1000"
/>
```

Abbildung 8.6: Beispiel für die Verwendung einer MOA-Klasse im streams Framework

Der andere zentrale Aspekt der Schnittstelle ist, dass die Lernverfahren aus MOA auf einer *Instance* (Moa-Interface) statt auf einem Data Item des streams Framework lernen. Daher musste mit der *DataInstance* ein spezielles Data Item angelegt werden, das von streams Prozessoren verarbeitet werden kann und alle Methoden einer Instance implementiert.
Eine Instance nutzt maßgeblich aus, dass in MOA Datensätze mit Hilfe des ARRF-Formats[2] definiert werden. Die Attribute sind daher mit Namen, Typ und Ausprägungen bereits bekannt und in den Lernern wird auf diese Informationen zugegriffen. Daher muss der entsprechende ARRF-Header aus den eintreffenden Data Items rekonstruiert werden und sowohl den DataInstances als auch dem MoaClassifier bekannt gemacht werden. Diese Aufgabe übernimmt die *DataInstanceFactory*.
Diese Idee stammt von Christian Bockermann und die konkrete Implementierung der Schnittstelle wurde daher dem entsprechenden Repository[3] als neuer Branch hinzugefügt.

Auch hier sei zur detaillierten Dokumentation auf Java-Docs und Code verwiesen.

8.4 Stream - ConceptDrift

Stream-ConceptDrift ist eine Erweiterung des Stream-Framework um Funktionalität zur Durchführung von Experimenten mit Concept Drift, die sich wie im Stream-Framework in XML-Dateien definieren lassen.
Das Projekt enthält eine Implementierung der in Kapitel 4 vorgestellten Datengeneratoren als *Stream*. Um einen Überblick über das Framework und die Benennung der entsprechenden Klassen zu erhalten, soll folgendes UML-Klassendiagramm (Abb. 8.7) dienen.

[2] Attribute-Relation File Format: `http://www.cs.waikato.ac.nz/ml/weka/arff.html`
[3] `https://bitbucket.org/cbockermann/streams-moa`

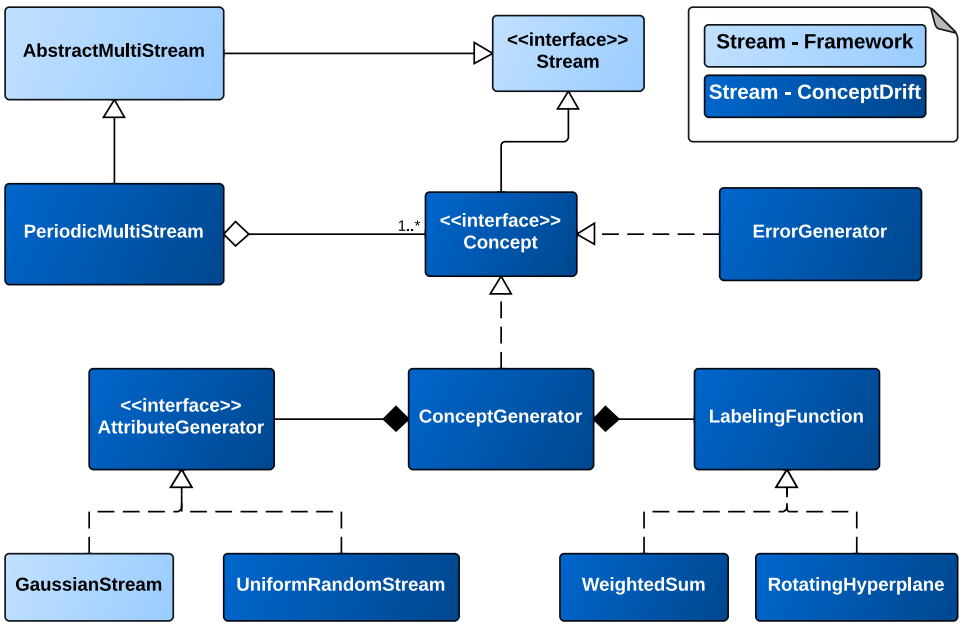


Abbildung 8.7: Vereinfachtes Klassendiagramm des Generator-Frameworks in Stream-ConceptDrift

Ein abrupter Concept Drift lässt sich dann beispielsweise in der XML-Datei des streams Framework wie in Abb. 8.8 definieren.

```
<stream id="drift" class="stream.io.multi.PeriodicMultiStream" numTotalPeriodes="5">
       <stream id="concept1" class="stream.generator.ConceptGenerator"
              limit="5000" weights="0.4,0.6,0.9" startWithChange="true">
              <stream id="attributes" keys="a,b,c"
                     class="stream.generator.UniformRandomStream"/>
       </stream>
       <stream id="concept2" class="stream.generator.ConceptGenerator"
              limit="3000" weights="0.2,0.5,0.9" startWithChange="true">
              <stream id="attributes" keys="a,b,c"
                     class="stream.generator.UniformRandomStream"/>
       </stream>
</stream>
```

Abbildung 8.8: Beispiel: Definition eines abrupten Drift mittels Generator-Framework.

Zur Bewertung von Driftbehandlung kann für die Accuracy auf den Prozessor PredictionError aus dem Stream-Framework zurückgegriffen werden. Entsprechende Prozessoren für die Bewertung von Regressionsverfahren (*MeanSquaredError, AbsoluteError*) wurden in StreamConceptDrift ergänzt.
Für die Beurteilung von Driftdetektoren steht der Prozessor *DriftDetectionEvaluator* zur Verfügung, welcher ein über das Interface *DriftDetectionService* zur Verfügung gestellten Driftdetektor bezüglich der im Abschnitt3.3.1 entwickelten Gütemaße bewertet. Zu beachten ist dabei, dass, falls die verarbeiteten Data Items nicht mit Ground-Truth-Informationen versehen sind, einige der Kriterien nicht berechenbar sind.

Sowohl im Generator-Framework als auch im DriftDetectionEvaluator lassen sich optional eine ganze Reihe von Annotationen für die Auswertung hinzufügen, die dann in eine Datei geschrieben oder geplottet werden können.

8.4.1 Definition von Experimenten

Zur systematischen Evaluation wurden die einzelnen Szenarien und die durchzuführenden Experimente in verschiedenen XML-Dateien definiert. Über *.properties*-Dateien lassen sich die konkreten Parameter der Szenarien und Lernverfahren festlegen und beliebige Szenarien und Experimente miteinander verknüpfen. Insbesondere können in einer *.properties*-Datei mehrere Experimente definiert werden, die parallel ausgeführt werden sollen.
Außerdem bietet der *ExperimentRunner* die Möglichkeit ein Experiment mehrmals hintereinander mit jeweils verschiedenen Parameterwerten auszuführen. Optional werden geeignete .R-Skripte aufgerufen, um die für das jeweilige Experiment relevanten Plots (zum Beispiel die im Evaluationkapitel 5 verwendeten) zu erstellen.

9 Fazit & Ausblick

Die Arbeit verschafft einen Überblick über verschiedene Arten von Concept Drift und Standardansätze zur Behandlung. Konkrete Behandlungsmethoden und Driftdetektoren wurden miteinander verglichen.
Vor dem Hintergrund einer realen Anwendung, in der das Lernen eines neuen Modells sehr aufwendig ist, wurden insbesondere verschiedene Ansätze zur Behandlung von Concept Drift in Verbindung mit Online- und Batch-Verfahren betrachtet. Es wurde experimentell bestätigt, dass sich die Vorhersagegüte unter Concept Drift mit den betrachteten Ansätzen auch für Batch-Verfahren steigern lässt. Insbesondere die Drifterkennung funktionierte jedoch in der Regel besser, wenn ein online Basislerner verwendet wurde.

In der Untersuchung der Ansätze zur Behandlung in zyklischen Prozessen konnte festgestellt werden, dass die Drifterkennung besonders dadurch gute Ergebnisse erzielt, dass schnell nach einem Drift ein neues Modell gelernt wird und nur Daten aus dem jeweiligen Concept zum Modelllernen verwendet wurden. Der große Vorteil eines Ensembles bezüglich zyklischem Concept Drift liegt darin, dass zu Beginn eines Drifts bereits ein Modell des neuen Concepts zur Verfügung steht.
Durch Kombination und Abwandlung dieser Verfahren konnte eine Skizze für einen neuen Behandlungsansatz entwickelt werden, der speziell auf zyklisch wiederkehrende Concepts ausgelegt ist. Zur Beurteilung der Qualität des vorgeschlagenen Ansatzes muss dieser implementiert und getestet werden.

Zur Erstellung dieser Arbeit wurde außerdem eine ganze Reihe von wiederverwendbarer Software erstellt, die in naher Zukunft ausführlich dokumentiert und öffentlich bereitgestellt werden soll.
Mit dem Generator-Framework können synthetische Daten für die verschiedenen Arten von Concept Drift für Klassifikations- und Regressionsaufgaben flexibel erzeugt werden. Anhand der dabei verfügbaren Ground Truth Daten, lassen sich auch Kriterien zur Bewertung von Driftdetektoren betrachten, die an echten Datensätzen nicht berechnet werden können.
Die gängigen Methoden zur Driftbehandlung und Driftdetektoren wurden mit diesem Framework miteinander verglichen. Da die konkreten Experimente in XML und Property-Dateien definiert sind, lassen sich mit dem *ExperimentRunner* neue Methoden zur Driftbehandlung leicht im Vergleich zu den bisherigen betrachten. Genauso lassen sich die Verfahren leicht auf anderen Daten vergleichen, sodass für ein reales Problem entschieden werden kann, welcher Ansatz zur Behandlung besonders geeignet ist.
Insbesondere die Schnittstelle des streams Framework zu MOA kann für viele Anwendungen von Interesse sein, da neben Concept Drift Behandlung auch zahlreiche Lernverfahren aus MOA und WEKA zur Verfügung stehen.

Literaturverzeichnis

[1] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldá, and Rafael Morales-Bueno. Early drift detection method. In *In Fourth International Workshop on Knowledge Discovery from Data Streams*, 2006.

[2] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*, 2007.

[3] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, August 2010.

[4] Albert Bifet and Richard Kirkby. Data stream mining: a practical approach. Technical report, The University of Waikato, August 2009.

[5] Albert Bifet, Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Indrė Žliobaitė. Cd-moa: Change detection framework for massive online analysis. In Allan Tucker, Frank Höppner, Arno Siebes, and Stephen Swift, editors, *Advances in Intelligent Data Analysis XII*, volume 8207 of *Lecture Notes in Computer Science*, pages 92–103. Springer Berlin Heidelberg, 2013.

[6] Christian Bockermann. The *streams* framework, 2012.

[7] Christian Bockermann and Hendrik Blom. The streams framework. Technical Report 5, TU Dortmund University, 12 2012.

[8] D. Brzezinski and J. Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(1):81–94, Jan 2014.

[9] Dariusz Brzezinski. Mining data streams with concept drift. Master's thesis, Poznan University of Technology, 2010.

[10] Wei Fan, Yi an Huang, Haixun Wang, and Philip S. Yu. Active mining of data streams. In *in Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 457–461.

[11] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In AnaL.C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer Berlin Heidelberg, 2004.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[13] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[14] Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing, 1999.

[15] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[16] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *In Proceedings of the Seventeenth International Conference on Machine Learning (ICML*, pages 487–494. Morgan Kaufmann, 2000.

[17] Alexey Tsymbal. The problem of concept drift: Definitions and related work. Technical report, 2004.

[18] Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. Dynamic integration of classifiers for handling concept drift. *Inf. Fusion*, 9(1):56–68, January 2008.

[19] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM.

[20] Indre Zliobaite. Learning under concept drift: an overview. *CoRR*, abs/1010.4784, 2010.

[21] Indre Zliobaite. How good is the electricity benchmark for evaluating concept drift adaptation. *CoRR*, abs/1301.3524, 2013.

Erklärung

Hiermit erkläre ich, Stefan Rötner, die vorliegende Bachelor-Arbeit mit dem Titel *Behandlung von Concept Drift in zyklischen Prozessen* selbständig verfasst und keine anderen als die hier angegebenen Hilfsmittel verwendet, sowie Zitate kenntlich gemacht zu haben.

Dortmund, 11. Juni 2014