Concept Drift Detection in Dynamic Probabilistic Relational Models

Nils Finke and **Tanya Braun** and **Marcel Gehrke** and **Ralf Möller**
Institute of Information Systems, University of Lübeck, 23562 Lübeck, Germany
{finke, braun, gehrke, moeller}@ifis.uni-luebeck.de

Abstract

Dynamic probabilistic relational models, which are factorized w.r.t. a full joint distribution, are used to cater for uncertainty and for relational and temporal aspects in real-world data. While these models assume the underlying temporal process to be stationary, real-world data often exhibits non-stationary behavior where the full joint distribution changes over time. We propose an approach to account for non-stationary processes w.r.t. to changing probability distributions over time, an effect known as concept drift. We use factorization and compact encoding of relations to efficiently detect drifts towards new probability distributions based on evidence.

1 Introduction

In order to cope with uncertainty and relational information of numerous objects over time, in many real-world applications, probabilistic temporal (also called dynamic) relational models (PDRMs) need often be employed (Finke et al. 2020). Lifted inference approaches use relations in PDRMs, allowing for tractable inference w.r.t. domain sizes (Niepert and Van den Broeck 2014). Typically, the temporal aspect is encoded under the first order Markov assumption, with probability distributions of future states necessarily being dependent on only the present state, i.e., it is stationary. Due to their efficient representation formalism and runtime behavior, PDRMs together with lifted inference approaches provide a powerful toolset for real-world challenges. Unfortunately, challenges in practice frequently clash with one of the assumptions, namely, an underlying stationary process. Non-stationarity can manifest itself in many aspects, such as probability distributions that change over time, also known as *concept drifts*. Research about over time changing probability distributions already provided a variety of different concept drift detection mechanisms. Robinson and Hartemink (2009) introduced non-stationary dynamic Baysian networks, which allow for an underlying conditional dependency structure to change over time. They explicitly encode drifts within the model by defining sets of edges to change, i.e., add or delete them to adjust conditional dependencies over time. Choi, Yeung, and Zhang (2001) define hidden-mode Markov decision processes (HM-MDP), in which changes are limited to a fixed and known number of modes. Each mode represents a stationary environment, formalized as an MDP. The modes are part of a hidden Markov model, encoding when the process switches from one mode to another. Hadoux, Beynier, and Weng (2014) also treat non-stationary environments within Markov models by a set of different contexts, i.e., modes, encoded as MDPs. They detect changes in the transition and reward function of an MDP and use hypothesis testing to test if a drift has occurred, assuming that the observations are independent and identically distributed (iid). If the test reveals that the observations no longer match the MDP, a new MDP is learned. To the best of our knowledge, we are the first to examine concept drifts for factorized (relational) models, without knowing the new process in advance, enabling us to efficiently detect concept drifts. Motivated by examples from a logistics application, we present an approach exploiting the factorization in PDRMs enabling for efficient concept drift detection. The approach is universal and applies to different kinds of PDRMs.

2 Reference Formalism

Note that our approach to concept drift detection does not depend on any specific formalism. However, we use probabilistic dynamic relational models (PDRMs) presented by Gehrke, Braun, and Möller (2018), based on parametric factor graphs introduced by Poole (2003) as a reference. Further, we here leave out formal definitions as much as possible and only give the intuition. We encourage to read on in (Gehrke, Braun, and Möller 2018) for a full specification.

Parameterized Dynamic Models PDRMs combine first-order logic with a factor graph, representing first-order constructs using logical variables (logvars) as parameters. Random variables (randvars) are parameterized with logvars (PRV) to compactly represent sets of randvars that are considered indistinguishable without further evidence. PRVs are linked through parametric factors (parfactors), which are functions that take those PRVs as arguments to represent their relation, and return a real number, called potential.

Like most dynamic model formalisms, PDRMs use two static parameterized models (PRMs) to describe how a model changes from one time step to the next. A PDRM encodes a sequential dimension by a pair of PRMs, one rep-

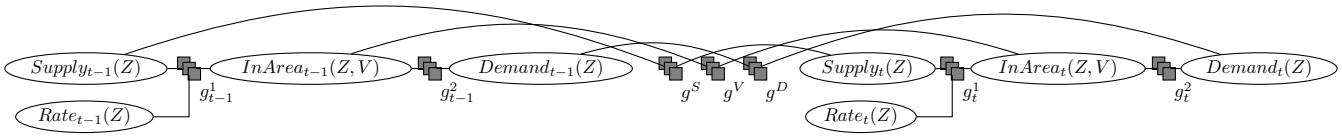

Figure 1: Two-slice parameterized probabilistic model $G_{\rightarrow}$ (and G_0)

resenting an initial time step and the other representing how the model transitions from one time step to the next. They follow the same idea as DBNs with an initial model and a copy pattern for further time steps. PDRMs are based on the first-order Markov assumption, i.e., randvars from each time slice t depend only on randvars from the preceding time slice $t-1$. PDRMs model a stationary process, i.e., changes from one time step to the next follow the same distribution. Semantics of a PDRM are given by instantiating a PDRM for a given number of time steps using G_0 for the initial time step and appending $G_{\rightarrow}$ for the other time steps, followed by grounding and building a full joint distribution.

Figure 1 shows a PDRM illustrating seaborne transportation. Variable nodes (ellipses) correspond to PRVs, factor nodes (boxes) to parfactors. Edges between factor and variable nodes denote relations between PRVs, encoded in parfactors. Parfactors g^S, g^V, and g^D are so called inter-slice parfactors. The submodel to the left and to the right of these inter-slice parfactors are duplicates of each other, with the left referencing time step $t-1$ and the right referencing time step t. The figure depicts $G_{\rightarrow}$, while the left part with $t = 0$ depicts G_0. Parfactors reference time-indexed PRVs, namely, a Boolean PRV $InArea(Z,V)$ and PRVs $Supply(Z)$, $Rate(Z)$, $Demand(Z)$ with range values $\{high, medium, low\}$, built from randvar names $\mathbf{R}$ = {Supply, Rate, InArea, Demand} and logvar names $\mathbf{L} = \{Z, V\}$. Thus, intuitively a PRV represents multiple entities of the same type represented through the logvar, e.g. here supply within zones $Z \in \{z_1, z_2, \ldots, z_n\}$. Seaborne transportation using vessels is mainly driven by supply ($Supply(Z)$) and demand ($Demand(Z)$) of commodities across various locations (zones Z). Vessels V move between these zones, captured by $InArea(Z,V)$, representing trade flows: Vessels are in zones with high supply (to load cargo), in zones with high demand (to discharge cargo), and in between while traveling. For transportation, a fee per ton, called freight rate ($Rate(Z)$), is charged.

Joining all parfactors within the model using a join over the arguments and multiplication for the potentials leads to a full joint probability distribution after normalising the join result (Sato 1995). Given a PDRM, one can ask queries for probability distributions or the probability of an event, possibly given a set of observations as evidence, like $P(Supply(z_1))$, $P(Demand(z_1) = high)$, or $P(Rate(z_1) \mid Supply(z_2) = high, Supply(z_3) = high)$. Here, $Supply(z_2) = high$ and $Supply(z_3) = high$, as given in the last query, denotes evidence.

Lifted Dynamic Junction Tree Algorithm For PDRMs, LDJT (Gehrke, Braun, and Möller 2018) provides an efficient way to answer a set of queries for probability distributions. LDJT constructs a so-called first-order junction tree (FO jtree) as an auxiliary structure. For details see the work of Gehrke, Braun, and Möller (2018). For query answering, LDJT uses lifted variable elimination as a subroutine that ensures lifted calculations (Poole 2003; Taghipour et al. 2013).

3 Concept Drift Detection

To ensure accuracy of an underlying model, it is important to detect and handle concept drifts. In the following, we consider an example from the field of dry-bulk shipping and characterize different types of drifts. We present an approach to detect such concept drifts in dynamic factorized models, using PDRMs as the reference formalism. Further, we show how to include concept drift detection in an online query answering algorithm, using LDJT as reference.

3.1 Systematic and Non-Systematic Drifts

Decomposition in time series analysis is a common approach to reveal reoccurring patterns in temporal data (Hyndman and Athanasopoulos 2018). Time series are mainly decomposed in trend, seasonality, and residual components. The former two can be summarized as systematic components. The latter is called non-systematic and carries what remains in the time series after systematic components are removed. While systematic components can be modeled directly, non-systematic components cannot. More specifically, when deriving probability distributions from historical data, a decomposition of the historical data to check for reoccurring patterns such as trend or seasonality can reveal potential concept drifts. In that case, it can be intimating to add variables for these components to encode all known influencing factors within the model. Although seasonality or trend can as well be understood as concept drifts (as the statistical properties of the variable change over time), we here focus on non-systematic components since they cannot be included within the model a priori.

3.2 Drift Detection

To detect concept drifts, we use evidence as we progress in time. Evidence should follow the full joint probability distribution encoded in the model. Using new evidence, we can compare the initial model with the current model afflicted with such new evidence, and learn a new model should the two differ significantly. We split drift detection in two steps: (i) Determine an ongoing *shift*-factor between the initial model and the current model afflicted with evidence. (ii) Detect a *drift* in the sequence of shift-factors. We assume that a concept drift manifests itself only across time steps, i.e., a

random event has occurred, changing the environment to the point that the model no longer accurately describes the environment. The event possibly occurs over multiple time steps.

Since a temporal model also considers the past and an altering event may still be ongoing, evidence will not instantly show that the initial model no longer accurately describes the environment. Instead, the shift-factor will increase until it plateaus. More specifically, we look for the point once the drift is complete to learn a new model. Consider Fig. 2 (a) for an example based on a sequence S of shift-factors s (blue dots). A shift-factor $s = 0$ indicates that both models are equal. We are interested in finding a new partial sequence of in average the same distance to the initial distribution, denoted as a *plateau*, such as the sequence starting of $t > 70$ (blue line). During $0 < t < 70$, the drift is still ongoing.

Shift We derive a shift-factor s that denotes the difference between the two models, i.e., a factor indicating how well evidence follows the initial model. To determine s, we use the Kullback–Leibler (KL) divergence. For discrete models, the KL divergence of two probability distributions P, Q over random variables $\mathbf{R}$ is defined as

$$D_{KL}(P,Q) = \sum_{\mathbf{r} \in \mathcal{R}(\mathbf{R})} P(\mathbf{R}=\mathbf{r}) \ \log\left(\frac{P(\mathbf{R}=\mathbf{r})}{Q(\mathbf{R}=\mathbf{r})}\right) \quad (1)$$

Typically, P denotes the real, observed probability distribution, while Q denotes the initial model probability distribution or the approximation of P. The naive way of using KL divergence for drift detection is to build a full joint distribution of the current model as well as the initial model. Unfortunately, working with full joint probability distributions yields a runtime that is exponential in the number of random variables. Therefore, we use factorization and compare the two models on the individual parfactors. Algorithm 1 shows an outline. To use KL divergence, we normalize each parfactor to enforce probability distributions for the comparison. The sum of all divergences is the shift-factor.

Drift Using the sequence S of shift factors s, we detect a concept drift by performing curve fitting to find plateaus in the sequence. We assume that the change in shift-factors follows a logistic function. Once curve fitting has returned the logistic function $\ell \in \mathbb{N}$ times as the best fit, where ℓ is a limit provided a priori, we determine the concept drift to be complete and thus, to be a signal that a new model needs to be learned. More specifically, we compare against a set of functions, namely, a linear, a sinusoid, and a logistic function. We perform curve fitting on the sequence S with all named functions and select the function, which has the best fit to observed data. Thus, based on regression modelling we determine the coefficients of the functions using the least square method. The coefficients are determined such that the mean squared error $\epsilon = \sum_{i=1}^{S}(F(x_i) - S_i)^2$ with $F(x)$ being either a linear, a sinusoid or a logistic function, is minimized. Algorithm 2 outlines drift detection given a sequence of shift-factors S and based on a limit ℓ. Note, that except the common definitions of the named functions,

Algorithm 1: Shift-factor calculation at time t

```
Input: G_0, G_t models
Function getShift(G_0, G_t):
    s ← 0
    for each parfactor g ∈ G_t do
        Find corresponding parfactor g' in G_0
        Normalize distributions of g, g'
        s = s + D_KL(g, g')        // see Eq. (1)
    return s
```

Algorithm 2: Drift detection using curve fitting

```
Input: S sequence of shifts of length t
static Queue F ← () of length ℓ
Function detectDrift(S, ℓ):
    E ← ()
    for each f in {linear, sinusoid, logistic} do
        Perform curve fitting for f on S
        Get ε error between f and S
        E ← E ∘ ε
    Get f according to min(E)
    if F full then
        Dequeue first element of F
    Enqueue f in F
    if ∀f ∈ F[0, ℓ − 1] : f = logistic then
        return true
    return false
```

we describe the sinusoid function in combination with a linear function to encode upward or downward behaviour as $f(x) = A \cdot sin(\omega \cdot x + \phi) + (b_0 + b_1 \cdot x)$ with A as the mean level, ω the frequency, ϕ the phase, b_0 as the y-intercept and b_1 the overall slope.

Figure 2 (a) depicts all named fitted functions on S. Within the algorithm linear and sinusoid function are used as exclusion criteria. Based on the least square error ϵ, we select the function describing the data the best. Figure 2 (b-e) shows how ϵ evolves with getting more data (here $T = \{50, 60, ..., 80\}$). With $T > 70$ and $\epsilon = 0.123$ slowly the growth of a logistic functions settles, and a concept drift can be identified. Still, with $T < 70$ a sinusoid function fits better to the data. Thus, we use the sinusoid and the linear function to exclude data which does not follow a logistic function at all. A linear function with slope $a > 0$ fits better to data points in case no plateau at the end of the sequence S can be identified. Still, the KL divergences increases over time. Thus, we assume, that we are still within a drift, which has not leveled off yet. On the other hand, a sinusoid function fits better to data points, in case the KL divergence periodically increases and decreases. A general upward trend with sinusoidal abnormalities in Fig. 2 (a) is apparent. Until $T < 70$, a linear and a sinusoid function describes the data points better than a logistic function.

3.3 Online Query Answering under Concept Drift

As follows, we present an online version of LDJT that includes drift detection. Before turning to LDJT as a specific

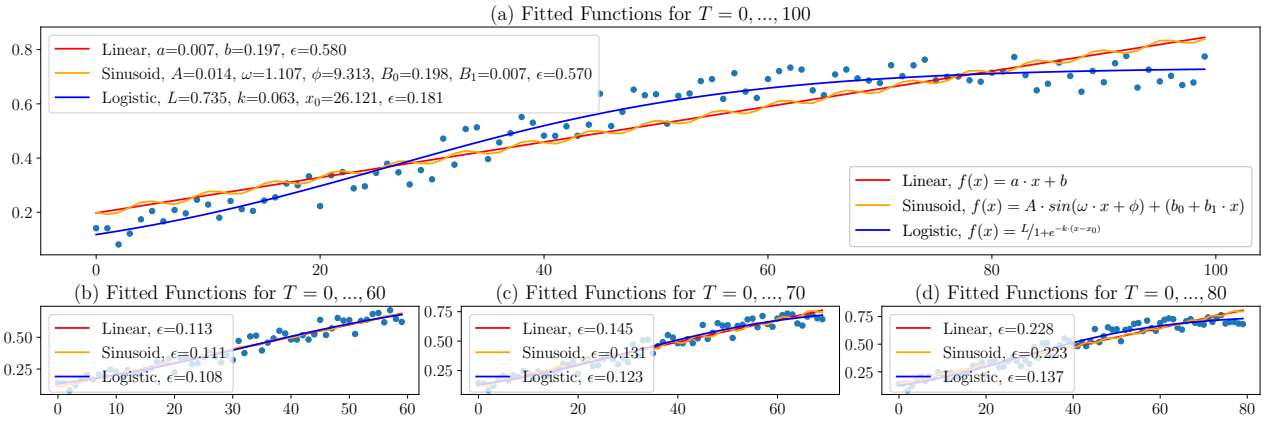

Figure 2: Curve fitting on D_{KL} (blue dots) with linear (red line), sinusoid (yellow line) and logistic (blue line) function.

algorithm, we look at the general case.

After having collected the newest evidence, one performs drift detection to check if a new model is required to be learned (see Alg. 2). If so, the parameters of the initial model need to be updated. Since we only consider non-systematic drifts in the distribution, we do not need to learn the structure anew, only update the potentials in the parfactors. The whole procedure looks as follows: (1) Collect current evidence. (2) Perform drift detection using Alg. 1, 2. (3) (a) If a drift is detected: Learn new parameters. (b) Otherwise: Continue. (4) Answer current queries and go back to (1).

For an online version of LDJT with concept drift detection, which we call LDJT$^{\text{drift}}$, LDJT needs to be extended with two streams as inputs. To make LDJT an online algorithm, sets of evidence and query terms, originally provided before its execution, are replaced with one stream for evidence and one for query terms. Before instantiating the FO jtree for the current time step, LDJT$^{\text{drift}}$ performs concept drift detection using Alg. 1,2. If a drift is detected, LDJT$^{\text{drift}}$ calls a learning algorithm to learn new parameters, updates the FO jtrees, and resets the current time t to 0.

4 Conclusion and Outlook

This paper presents an approach to efficiently detect concept drifts in dynamic factorized models, making use of evidence and the compact encoding of the models in terms of factorization and relations. As dynamic models are often modelled as stationary processes, they frequently clash in practice having non-stationary environments. This paper contributes an efficient detection algorithm to identify the point when a drift has settled to relearn the model. We present LDJT$^{\text{drift}}$, an online version of LDJT, incorporating the concept drift detection approach, while answering queries under evidence. Moving on, we run an empirical evaluation testing runtime performance and detection accuracy.

References

Choi, S. P. M.; Yeung, D.-Y.; and Zhang, N. L. 2001. *Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making*. Berlin, Heidelberg: Springer Berlin Heidelberg. 264–287.

Finke, N.; Gehrke, M.; Braun, T.; Potten, T.; and Möller, R. 2020. Investigating matureness of probabilistic graphical models for dry-bulk shipping. In Jaeger, M., and Nielsen, T. D., eds., *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, 197–208. PMLR.

Gehrke, M.; Braun, T.; and Möller, R. 2018. Lifted Dynamic Junction Tree Algorithm. In *Proceedings of the International Conference on Conceptual Structures*. Springer.

Hadoux, E.; Beynier, A.; and Weng, P. 2014. Sequential decision-making under non-stationary environments via sequential change-point detection. In *Learning over Multiple Contexts (LMCE)*.

Hyndman, R. J., and Athanasopoulos, G. 2018. *Forecasting: principles and practice*. OTexts.

Niepert, M., and Van den Broeck, G. 2014. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In *AAAI-14 Proceedings of the 28th AAAI Conference on Artificial Intelligence*. AAAI Press.

Poole, D. 2003. First-order Probabilistic Inference. In *Proc. of the 18th International Joint Conference on Artificial Intelligence*. IJCAI Organization.

Robinson, J. W., and Hartemink, A. J. 2009. Non-stationary dynamic bayesian networks. In Koller, D.; Schuurmans, D.; Bengio, Y.; and Bottou, L., eds., *Advances in Neural Information Processing Systems 21*. Curran Associates, Inc.

Sato, T. 1995. A Statistical Learning Method for Logic Programs with Distribution Semantics. In *Proceedings of the 12th International Conference on Logic Programming*, 715–729. MIT Press.

Taghipour, N.; Fierens, D.; Davis, J.; and Blockeel, H. 2013. Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. *Journal of Artificial Intelligence Research* 47(1).