Probabilistic Programming – ein neuer Baustein für die KI

Motto: Wenig Daten? Dann schlau Raten!

Thomas Wengerek [1]

Abstract: Wenn statt riesiger Datenmengen ein generatives Modell (also ein probabilistischer Simulator) für die Erzeugung bzw. Entstehung der Daten der Anwendungsdomäne zur Verfügung steht, kann dieser Simulator durch bayesianische Inferenz „invertiert“ werden, d.h. es können diejenigen latenten Parameter des Modells bestimmt werden, die für jeweils vorgegebene Daten verantwortlich sind. Universelle Probabilistische Programmiersprachen (PPL) sind derart expressiv, dass sich mit ihnen das generative Modell, etwaige Bedingungen an die Daten und das Vorwissen des Anwenders bzgl. der latenten Parameter des Modells als ein integriertes Programm formulieren lassen. Die Ausführung dieses Programms erzeugt dann Stichproben der bayesianischen Posterior-Verteilung der gesuchten Modell-Parameter. Durch moderne Allzweck-Algorithmen zur Inferenz kann diese Verteilung exploriert und für den jeweiligen Anwendungszweck interpretiert werden. In komplexen Problemstellungen können somit in der Sprache der probabilistischen Logik formulierte Fragen durch Probabilistic Programming beantwortet werden. Dies eröffnet für viele Anwendungs-gebiete, auch in der Wirtschaftsinformatik, ungeahnte Möglichkeiten. Die vorliegende Arbeit soll die Ideen und Potentiale des Probabilistic Programming einem breiteren Kreis von Anwender*innen nahebringen. Dazu werden einerseits die grundlegenden Konzepte sorgfältig dargestellt und in ihrem Zusammenspiel motiviert und andererseits werden diese neuen Ideen anhand mehrerer Anwen-dungsbeispiele aus dem einschlägigen Forschungsumfeld vorgestellt. Dabei wurden einige dieser Beispiele, für die Softwaresysteme im Open Source Bereich verfügbar sind, auch praktisch erprobt.

Keywords: Bayesianische Inferenz, generatives Modell, probabilistische Logik, Inferenz-Algorithmen, Modell-Invertierung

1 Generative Modelle

Akteure in der Welt müssen mit Unsicherheit umgehen können. Das Wissen über die Welt ist ausschnitthaft und verrauscht, weil die Daten immer unvollständig und fehlerbehaftet sind. In dieser Lage muss ein solcher Akteur versuchen, das vorhandene Wissen so zu verwalten, dass es 1.) akkumuliert und flexiblen Schlussfolgerungsmechanismen zugänglich gemacht werden kann, und 2.) in geeigneter Weise probabilistisch interpretiert werden

1 Hochschule Stralsund/Fakultät Wirtschaft, Zur Schwedenschanze 15, 18435 Stralsund, thomas.wengerek@hochschule-stralsund.de

https://doi.org/10.30844/AKWI_2022_12

kann. Wie lassen sich diese beiden scheinbar unversöhnbaren Anforderungen produktiv in einem gemeinsamen Rahmen vereinen? In diesem Dilemma hilft die Beobachtung, dass es zwei verschiedene Wege [NJ01] zur Modellierung gibt: den Diskriminativen und den Generativen. Die Regressions- und Klassifikationsverfahren des maschinellen Lernens wählen den diskriminativen Weg. Dabei dienen inhärente Eigenschaften der konkret vorliegenden Daten als Unterscheidungsmerkmale. Beim generativen Weg hingegen wird versucht, einen Erzeugungsprozess (generatives Modell) für die zu beobachtenden Daten zu finden, um diese einerseits reproduzieren und andererseits vorhersagen zu können. Ein solcher Erzeugungsprozess wird in der Regel stochastischer Natur sein. Und wenn die Beschreibung eines solchen generativen Modells mittels eines geeigneten, strukturreichen Formalismus erfolgt, dann kann das Modell auch zur Arbeitsgrundlage für Schlussfolgerungsmechanismen werden, um mit ihrer Hilfe „Was-wäre-wenn-Fragen“ im Kontext des Modells zu beantworten (darauf wird im nächsten Abschnitt konkreter eingegangen). Darüber hinaus könnte es sogar möglich werden, eine über die aktuellen Daten hinausgehende, abstraktere Betrachtungsebene einzunehmen und hypothetische Überlegungen über das Modell selbst anzustellen.

2 Probabilistische Programme

Das Alles ist vielversprechend und ambitioniert – bleibt nur die Frage: wie lässt es sich formalisieren?

Hierzu bietet sich die Verwendung der probabilistischen Logik [Ja03] in der Form grafischer Modelle in Bayes-Netzen [Pe88, KF09] oder noch konsequenter im Model-based Machine Learning [Bi13, Wi19] an. Ein Bayes-Netz definiert eine Wahrschein-lichkeitsverteilung, die auf einer gerichteten, azyklischen Graphenstruktur beruht, die die wechselseitigen Unabhängigkeiten der bedingten Wahrscheinlichkeiten von Teilmengen der relevanten Größen widerspiegelt. Es sei angemerkt, dass in der probabilistischen Logik mit bedingten Wahrscheinlichkeiten sogar nicht-monotone Schlussfolgerungen in natürlicher Weise vorkommen können [Gro86] – eine Eigenschaft, die in der klassischen Logik nicht ohne gravierende Erweiterungen realisierbar ist. Dieser Graph ist somit das Gerüst des zugrundeliegenden Modells. Da die Expressivität graphischer Modelle für viele Anwendungen aber nicht ausreicht, wurde vorgeschlagen, Logik und Wahrscheinlichkeit in einer neuen, formalen Struktur, dem stochastischen Lambda-Kalkül [RP02], zu vereinen und als eine universelle Probabilistische Programmiersprache (PPL) zu realisieren [Gn08], wobei dieser Sprache zwei weitere Merkmale hinzugefügt werden: (i) Variablen können expliziten Bedingungen hinsichtlich ihres erlaubten Wertebereichs unterworfen werden (sog. „conditioning“). Und (ii) um die Ergebnisse von Berechnungen dieser Sprache auch in herkömmlichen Programmiersprachen weiterverwenden zu können, wird eine spezielle Schnittstelle benötigt (sog. „query“). Berechnungen, die in dieser Sprache formuliert sind, werden fortan als „Probabilistische Programme“ (PP) bezeichnet. PP beschreiben immer eine Wahrscheinlichkeitsverteilung – allerdings in Stichproben-Semantik („sample-based semantics“). Was bedeutet das?

Jeder Lauf des PP entspricht einem Pfad („trace“) durch den Berechnungsbaum. An Knotenpunkten dieses Baums können probabilistische Funktionen aufgerufen werden, die den weiteren Weg durch den Baum nicht-deterministisch beeinflussen. Das Ergebnis eines solchen Pfades wird also durch die akkumulierten Wirkungen der Wahrscheinlichkeits-verteilungen, die an den Knoten des Berechnungsbaumes wirken, bestimmt. Insofern gehorcht das Ergebnis ebenfalls einer Gesamtwahrscheinlichkeitsverteilung – eben derjenigen des gesamten PP. Jeder Lauf des PP resultiert daher in einer Stichprobe („sample“) der Gesamtwahrscheinlichkeitsverteilung, die durch den Programmcode des PP definiert wird. Man erkauft sich also die universelle Expressivität einer Turing-vollständigen Probabilistischen Programmiersprache damit, dass die Ergebnisse der mit ihrer Hilfe geschriebenen PP, zwar als Wahrscheinlichkeitsverteilungen vorliegen – aber eben nicht in geschlossener Form, sondern nur als Stichproben-Prozess beim wiederholten Ausführen des PP.

Halten wir fest: mit einer universellen PPL kann - qua Programmcode - jede vorstellbare Wahrscheinlichkeitsverteilung (in Stichprobensemantik) beschrieben werden. Für die Implementierung generativer Modelle steht somit ein universelles Konstruktions-werkzeug zur Verfügung.

3 Bayesianische Inferenz und moderne Inferenz-Algorithmen

Daten sollen nicht nur durch probabilistische Simulationen (d.h. generative Modelle) reproduziert werden können, sondern in Bezug auf diese Simulationen sollen auch Fragen gestellt und Schlussfolgerungen [Me18] gezogen werden können. Wie sieht ein geeigneter, probabilistischer Schlussfolgerungsmechanismus in diesem Kontext nun aus?

Das Konzept der bayesianischen Inferenz [Mc20] in der Form der elementaren Bayes-Regel hilft hier weiter:

$$P(\theta \mid D) = (P(D \mid \theta) * P(\theta)) / (P(D)) \quad (1)$$

Das generative Modell wird intern durch Modellparameter θ (sog. latente Variablen) kontrolliert. Je nachdem welche Werte θ annimmt, wird das generative Modell bestimmte Daten D erzeugen. Dieser Zusammenhang wird durch die Likelihood P(D|θ) dargestellt. Die bedingte Wahrscheinlichkeit P(D|θ) lässt sich auch als kausale Perspektive interpretieren, in der θ die Ursache und D die Wirkung ist. Zusätzlich sei es möglich, explizite Bedingungen („conditioning“) an die zu beobachtenden Daten zu stellen. Diese Bedingungen stellen die spezifischen Fragen dar, die im Zuge der Inferenz beantwortet werden sollen. Der Kürze der Darstellung halber sei hier angenommen, dass diese Bedingungen in der Likelihood bereits subsumiert seien. Das im Vorfeld vorhandene Wissen (oder „Un-Wissen“) über die Modellparameter wird durch die Apriori-Verteilung P(θ) ausgedrückt. Dies ist derjenige Aspekt der bayesianischen Statistik, der häufig als sog. „subjektives Wissen“ kontrovers diskutiert wird [La18]. P(D) ist die Randverteilung. Sie stellt einen Normierungsfaktor dar. Die linke Seite von (1) ist die Posterior-Verteilung P(θ|D). Sie stellt das eigentliche Ziel der

Inferenz dar. Die Posterior-Verteilung beschreibt, welche Modellparameter wahrscheinlich dafür verantwortlich sind, wenn bestimmte Daten vorliegen. Man kann auch sagen: die Inferenz invertiert die kausale Richtung von Ursache und Wirkung, da von P(D|θ) auf P(θ|D) geschlossen wird:

$$P(\theta \mid D) \quad \leftharpoondown \quad P(D \mid \theta) \tag{2}$$

Diese Invertierung ist der entscheidende Aspekt der Inferenz. Zwar ist die Bayes-Regel formal gesehen symmetrisch, aber in der konkreten Anwendung gibt es nun einmal die Unterscheidung von Ursache (latente Variablen, Modellparameter) und Wirkung (Daten), und wir interessieren uns hier für die Modellparameter. Durch die Invertierung wird es somit möglich, die Ursache, also die Modellparameter, einer gegebenen Wirkung zu bestimmen. Man kann dies formal auch so ausdrücken [La18]:

$$P(\text{Ursache} \mid \text{Wirkung}) \quad \Leftarrow \quad P(\text{Wirkung} \mid \text{Ursache}) \tag{3}$$

Mit modernen Inferenzalgorithmen [WMM14] kann die in Stichprobensemantik vorliegende Posterior-Verteilung P(θ|D) näherungsweise bestimmt werden. D.h. die implizit als Programmcode des PP vorliegende Posterior-Verteilung

$$P(\theta \mid D) \quad \simeq \quad \text{Probabilistisches Programm} \tag{4}$$

wird durch einen Inferenzalgorithmus explizit ausgewertet. Bei komplexen, hoch-dimensionalen Problemen kann dies approximativ geschehen oder es wird zumindest eine Maximum-Aposteriori Schätzung (MAP) [Mc20] bestimmt. Eine wichtige Klasse derartiger Algorithmen basiert auf Markov-Chain Monte-Carlo Methoden (MCMC) in der speziellen Variante von Metropolis-Hastings [La18]. Das PP wird dazu eigens in Continuation-Passing-Style (CPS) transformiert, damit der Inferenzalgorithmus optimal mit dem Code arbeiten kann [GS22]. Aber daneben werden viele weitere Verfahren erprobt. Die Verbesserung der Inferenzalgorithmen bzgl. Problemgröße und Geschwin-digkeit ist Teil der aktuellen Forschung. Einen guten Überblick dazu bietet [Me18].

Die übergeordneten Forschungsbemühungen um die „Probabilistischen Programmierung“ setzen sich das ambitionierte Ziel [Me18], viele Gebiete des maschinellen Lernens auch für Nicht-Expert*innen leichter zugänglich zu machen, indem die Modellbildung auf ein Programmierproblem zurückgeführt wird, für das keine tiefgehenden Spezialkenntnisse erforderlich sind. Und die Inferenz erfolgt dann automatisch durch leistungsfähige, universell einsetzbare Verfahren. Das Potential dieser Forschungsagenda wird auch von der Anwendungsseite mit Interesse verfolgt und aktiv unterstützt [Pyr17].

4 Anglican und ein didaktisches Beispiel

Es gibt eine Vielzahl von PPL, die auf den oben vorgestellten Konzepten beruhen. Darin ist die Gruppe der universellen PPL besonders interessant, weil es bei ihnen wenige Beschränkungen der Ausdrucksmittel bei der Code-Erstellung gibt und weil auch externe Bibliotheken für die Implementierung des generativen Modells herangezogen werden dürfen. Darüber hinaus stehen für diese universellen PPL eine ganze Reihe unter-schiedlicher Inferenzverfahren zur Verfügung. Eine dieser universellen, probabilistischen Programmiersprachen, die eine gewisse Vorreiterrolle einnimmt, ist Anglican [To16, Ang18]. Anglican basiert auf der modernen Lisp-Variante Clojure [Clo22]. Ein instruk-tives Anwendungsbeispiel - quasi das „HelloWorld“ der „Probabilistischen Program-mierung“ - ist das Äquivalent zur linearen Regression (siehe Abb. 1). Dabei sollen die sechs Datenpunkte mit folgenden x-y-Werten [0, 0.6], [1, 0.7], [2, 1.2], [3, 3.2], [4, 6.8], [5, 8.2], [6, 8.4] durch eine lineare Funktion `f` mit der Steigung `s` und dem Achsen-abschnitt `b` approximiert werden. Das PP ist ein Anglican-Ausdruck folgenden Inhalts: die lineare Funktion `f` ist das generative Modell. Ihre Funktionsparameter, `s` und `b`, sind die latenten Variablen des Modells. Als Apriori-Verteilungen dieser latenten Variablen werden Normalverteilungen (Mittelwerte jeweils 0, Standardabweichungen 2 bzw. 6) angenommen. Das „conditioning“ (im Code mit `observe` bezeichnet) beschreibt die Anforderungen, die durch die Daten an das Modell gestellt werden. D.h. konkret: die Funktionswerte `(f x)` sollen möglichst nah bei den y-Werten der Datenpunkte liegen. Damit der Inferenz-Algorithmus stabil arbeiten kann, wird auch hier eine Normalverteilung angenommen. Das PP `linear-regression-posterior` stellt also insgesamt die Frage: welche Werte müssen die latenten Variablen des generativen Modells annehmen, damit es zu den Datenpunkten möglichst gut passt? Ganz unten im Code wird durch den Vektor `[s b]` expliziert, welche Form die Antwort dann haben soll.

```
(defquery linear-regression-posterior []

  (let [s (sample (normal 0 2))

        b (sample (normal 0 6))

        f (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) 0.5) 0.6)

    (observe (normal (f 1) 0.5) 0.7)

    (observe (normal (f 2) 0.5) 1.2)

    (observe (normal (f 3) 0.5) 3.2)

    (observe (normal (f 4) 0.5) 6.8)
```

```
(observe (normal (f 5) 0.5) 8.2)

(observe (normal (f 6) 0.5) 8.4)

[s b]))
```

Als Nächstes wird das PP mit dem Inferenz-Algorithmus Metropolis-Hastings Markov Chain Monte Carlo [La18] ausgewertet, um die gesuchte Posterior-Verteilung darzu-stellen. Sie ist zweidimensional, da es sich um zwei latente Variablen (`s` und `b`), eben die Geradenparameter, handelt. Um die Wirkung des conditioning zu verdeutlichen, wird in Abb.1+2 diese Posterior-Verteilung (bzw. die aus ihr resultierenden Geraden in Abb.1, wenn man die latenten Variablen als Geradenparameter interpretiert) einmal ohne (hierbei wurden alle `observe`-Anweisungen aus dem PP entfernt) und einmal mit conditioning dargestellt. In beiden Fällen ist der Inferenzalgorithmus dann jeweils für sich genommen ausgeführt worden. In Abb.2 fällt dabei sofort auf, dass die Posterior-Verteilung mit conditioning (in rot) in beiden Dimensionen signifikant schmaler ist als die ohne conditioning (in blau). Die Expressivität einer PPL erlaubt es also, gezielte (aber immer probabilistisch notierte) Bedingungen an die latenten Variablen im PP zu formulieren. Die daraus durch den Inferenzalgorithmus realisierte Posterior-Verteilung wird dann mit hoher Wahrscheinlichkeit Stichproben der latenten Variablen generieren können, die die gewünschten Zielvorgaben des conditioning möglichst gut erfüllen. Die Posterior-Verteilung ohne conditioning (in blau) hingegen reproduziert letztlich nur die Apriori-Verteilung der latenten Variablen (im PP die Normalverteilungen für `s` und `b`). Dass die Glockenkurven hierbei nur schemenhaft zu erkennen sind, beruht auf der geringen Stichprobenanzahl von 100. Diese geringe Anzahl wurde aus didaktischen Gründen gewählt, damit in Abb.1 einzelne Linien visuell noch unterschieden werden können.

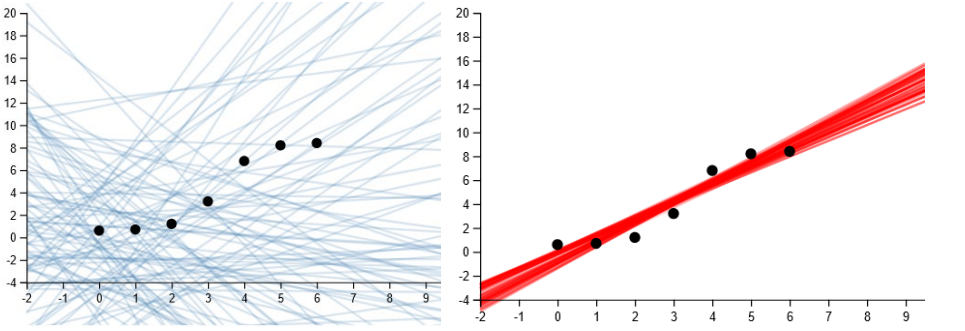

Abb.1: (links) Hier hat ein Probabilistisches Programm (ohne conditioning) 100 Paare `[s b]` der latenten Variablen erzeugt. Als Geradenparameter interpretiert ergeben sie eine regellose Anordnung von Geraden (in blau). (rechts) Das gleiche Programm, nun aber mit conditioning, erzeugt wieder Geradenparameter (in rot). Die zugehörigen Geraden approximieren die Datenpunkte nun aber in nahezu optimaler Weise. Es sei betont: hier wurde keines der üblichen Regressionsverfahren, z.B. die Kleinste-Quadrate-Methode o.ä., verwendet.

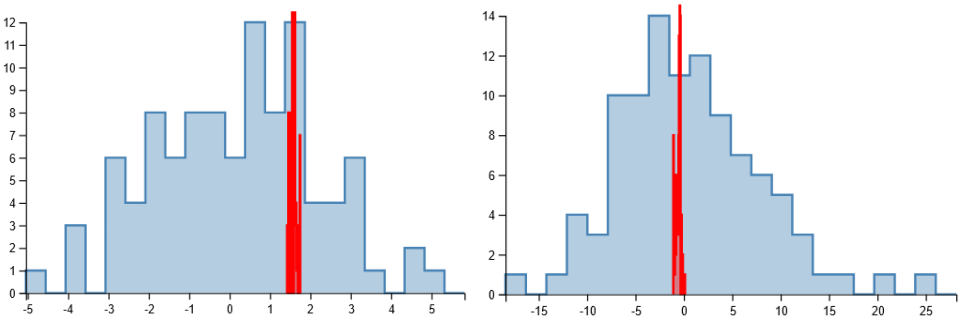

Abb.2: Diese Histogramme zeigen die durch die Inferenz bestimmten Posterior-Verteilungen der latenten Variablen: (links) für die Steigungen `s` und (rechts) für die Achsenabschnitte `b` der Geraden. Die breiten Verteilungen (in blau) gehören zum PP ohne conditioning und die extrem schmalen Verteilungen (in rot) gehören zum PP mit conditioning. Diese schmalen Verteilungen der Parameter `s` und `b` führen zur gebündelten Ausrichtung der Geraden in Abb.1 (rechts).

Dieses Beispiel verdeutlicht die Methodik und die Merkmale der „Probabilistischen Programmierung“. Folgende allgemeine Vorgehensweise lässt sich skizzieren:

- Wenn eine Situation vorliegt, in der ein Problem gelöst werden soll bzw. an die eine Frage gestellt werden soll, und nur wenige Daten vorhanden sind, es aber möglich ist, ein generatives Modell der Situation zu formulieren, welches hinreichend allgemeingültig ist und durch geeignete latente Variablen auch flexibel parametrisiert werden kann, dann besteht die Chance, „Probabilistische Programmierung“ einzusetzen.

- Ein Probabilistisches Programm (PP) beschreibt das Problem durch ein generatives Modell und eine in diesem Kontext gestellte, spezifische Frage, die durch geeignete Konditionierungen ausgedrückt wird.

- Das Inferenzverfahren arbeitet dann mit dem PP und realisiert die darin verborgene Inversion des konditionierten, generativen Modells, indem sie die zu dieser Inversion passende Verteilung der latenten Variablen des Modells bestimmt. Diese Verteilung ist die Antwort (z.B. in der Form einer MAP) auf die Fragestellung.

- Es gibt sehr viele unterschiedliche Inferenzverfahren mit jeweils eigenen Stärken und Schwächen. Manche dieser Inferenzverfahren sind „Allzweck-Werkzeuge“ mit breitgefächerten Einsatzbereichen. Der Metropolis-Hastings-Algorithmus in der obigen Regressionsaufgabe füllt z.B. die Rolle aus, die üblicherweise das Gauß-verfahren eingenommen hätte. Diese universelle Einsetzbarkeit von Inferenzver-fahren ist die Basis für die immer wieder artikulierte Ambition [Go14, Me18] des Forschungsfeldes der „Probabilistischen Programmierung“, nämlich viele Bereiche des maschinellen Lernens auch Nicht-Expertinnen und Nicht-Experten einfacher zugänglich zu machen bzw. sogar zu automatisieren.

- Anwenderinnen und Anwender können agnostisch hinsichtlich der zugrunde-liegenden Techniken und Konzepte der Inferenzverfahren vorgehen und sich ganz auf die konkrete Beschreibung ihres jeweiligen Anwendungsproblems und der zu beantwortenden Fragestellung im PP konzentrieren (Abb.3). Der Programmcode des PP ist daher rein deklarativ und somit kurz.

Abb.3: Fragen an generative Modelle werden als Konditionierungen in Probabilistischen Programmen formuliert. Inferenzverfahren beantworten diese Fragen durch die Berechnung passender Parameterwerte der Modelle in der Form von Wahrscheinlichkeitsverteilungen.

5 Komplexe Anwendungsbeispiele

Vier komplexe, sehr unterschiedliche Anwendungen werden im Folgenden vorgestellt, um die Breite der Anwendungsmöglichkeiten aufzuzeigen. (5.1) Zuerst wird eine Problemstellung aus der angewandten Mechanik untersucht. (5.2) Danach wird ein System vorgestellt, das große Datensätze, die Inkonsistenzen, Fehler und Lücken besitzen, automatisch bereinigt. Für diese ersten beiden Beispiele wurden eigene Experimente durchgeführt. Die eingesetzte Hardware bestand dabei aus einer Workstation mit einem AMD Ryzen 3955WX-Prozessor (4,30 GHz, 16 Kerne) und 256GB RAM. Danach werden zwei weitere repräsentative Beispiele (5.3 + 5.4) erwähnt, die aus der Forschungsliteratur stammen.

5.1 Lösen von Mechanik-Puzzles

Dieses Beispiel wurde zuerst auf der Machine Learning Summer School 2015 (MLSS2015) in Tübingen von F. Wood und Mitarbeitern vorgestellt [Phy15] und für die hier dargestellten Experimente verwendet und für die Auswertung (siehe Abb.6) um diverse Grafikfunktionen erweitert. Die zu lösende Aufgabe besteht darin, eine Anzahl von Bällen, die von einer erhöhten Rampe herunterrollen, in einen Zielbehälter zu befördern (siehe Abb.4). Dazu kann eine bestimmte Anzahl von kurzen, waagerechten Hinderniselementen („bumper“) völlig frei in der gesamten in Abb.4 gezeigten Szene so platziert werden, dass die Bälle auf diese bumper treffen, von diesen abprallen, auf weitere bumper treffen und sich auf diese Weise dem Zielbehälter nähern und mit etwas Glück letztlich in ihm landen.

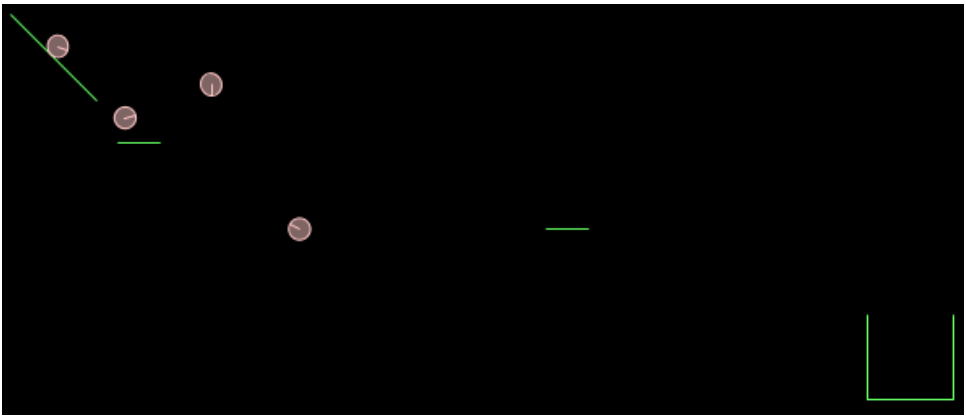

Abb.4: Das ist die Ausgangssituation. Die Bälle rollen von der Rampe oben links herunter, treffen auf den ersten zufällig platzierten bumper, prallen von diesem ab und fallen dann ins Nichts. Wie könnte man zusätzliche bumper so platzieren, dass die Bälle vom bumper zu bumper hüpfen und schließlich im Zielkorb unten rechts landen? Ein Lauf in dieser Simulation besteht immer aus 20 Bällen, danach wird gezählt wie viele von im Zielkorb landen. Dies führt allerdings zu zusätzlicher Komplexität, da die Bälle einer solchen Gruppe bei entsprechenden bumper-Platzierungen auch untereinander kollidieren können. In manchen Situationen kann das sogar hilfreich sein und für die Lösung ausgenutzt werden (siehe dazu Abb.7).

Für das PP (s.u.) ist es notwendig, eine leistungsfähige Physik-Engine [Box18] als externe Bibliothek in das generative Modell `simulate-world` zu integrieren. Dass dies möglich ist, ist eine der besonderen Stärken von Anglican. Eine Variante dieser Engine wird auch in einschlägigen Spielen (z.B. Angry Birds o.ä.) eingesetzt. Mit `create-world` werden die Positionen der Rampe, des Zielkorbs und die Apriori-Verteilung von acht bumper definiert. Dazu werden die bumper in der gesamten in Abb.4 gezeigten Szene gleichförmig (`uniform-continuous`) verteilt und in der Liste (sog. lazy sequence) `bs` gespeichert. Zusätzlich können hier aber auch noch weitere Hindernisse eingebaut werden, die die Aufgabe drastisch erschweren. Ein Lauf besteht jeweils aus 20 Bällen. Mit `balls-in-box` wird beobachtet, wie viele von ihnen dann im Zielkorb landen. Die Konditionierung `observe` besagt, dass 20 Bälle im Korb sein sollen. Eine Stichprobe, die dieses PP erzeugt, besteht aus der aktuellen Trefferzahl `num-balls` und der Liste der aktuellen bumper-Positionen `bs` als latente Variablen des Modells, für die Werte durch den Inferenz-Algorithmus gesucht werden, die die Konditionierung erfüllen sollen.

```
(defquery physics1 []

  (let [n-bumpers 8

        f (fn [] (list (sample (uniform-continuous -5 14))

                       (sample (uniform-continuous 0 10))))
```

```
      bs (repeatedly n-bumpers f)
      w0 (create-world bs)
      w1 (simulate-world w0)
      num-balls (balls-in-box w1)]
  (observe (normal num-balls 1) 20)
  (list num-balls bs)))
```

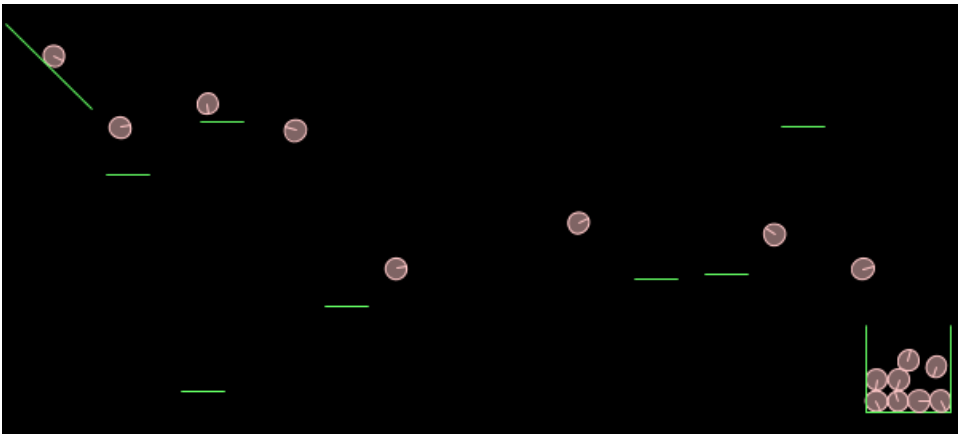

Abb.5: Nach der Inferenz: eine MAP-Stichprobe aus einer bumper-Verteilung, die zum Ziel führt.

Der hier verwendete Metropolis-Hastings Inferenzalgorithmus garantiert im Limes zwar die Konvergenz bis zur vollständigen, expliziten Darstellung der Posterior-Verteilung, aber in einem so komplexen, da hochdimensionalen Problem, kann wirkliche Konvergenz aufgrund des Rechenaufwands kaum je erreicht werden. Insofern dient die Inferenz hier als effizientes Explorationsverfahren (siehe dazu auch Abb.6), durch das relevante Bereiche der Verteilung zumindest approximiert werden können. Wenn dann derartige Bereiche explizit vorliegen, kann aus ihnen, z.B. in der Form einer MAP, eine mögliche Lösung der Aufgabenstellung leicht extrahiert werden (Abb.5 und 7).

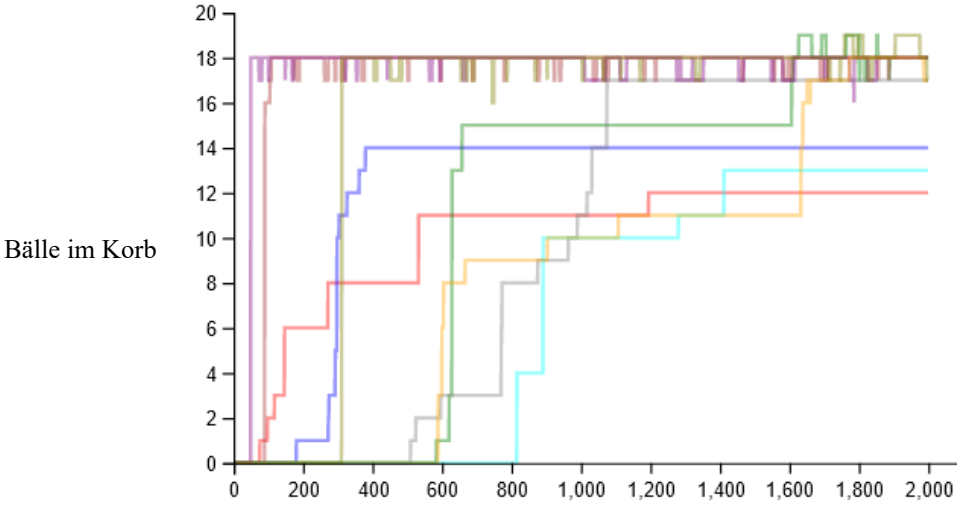

Abb.6: Der Inferenzalgorithmus bei der Arbeit: während der Burn-In-Phase wird die durch das „Probabilistische Programm“ definierte Verteilung stichprobenartig exploriert und dabei eine ihrer Modi gesucht. Ein Modus entspricht dem Teil der Posterior-Verteilung, der mit hoher Wahrscheinlichkeit die Konditionierung erfüllt. Die latenten Variablen des Modells und damit der gesuchten Posterior-Verteilung sind die x-y-Positionen der acht bumper. D.h. diese Verteilung ist eine zerklüftete Berg-und-Tal-Landschaft in einem 16-dimensionalen Raum. Wenn ein solcher Modus, d.h. ein Berg, gefunden wurde, wird er bestiegen. Dieser an einen Gradientenaufstieg erinnernde Vorgang ist typisch für Metropolis-Hastings und wurde hier zehnmal durchgeführt, da die anfängliche Stichprobe der Apriori-Verteilung der bumper-Positionen vorentscheidend dafür ist, welche Mode gefunden wird. Jedes Mal wird also eine neue Lösung gefunden. Im Diagramm wird dies jeweils mit einer bestimmten Farbe dargestellt. In jeder der zehn Durchführungen wurde die Simulation jeweils 2000-mal mit je 20 Bällen pro Simulationslauf ausgeführt. Die einzelnen Simulationsläufe entsprechen der Ordinate. Für jeden Simulationslauf wurde die Anzahl der Bälle im Zielkorb bestimmt und entsprechend auf der Abszisse eingetragen. Man erkennt anhand der stufenförmigen Linienverläufe deutlich, wie die Anzahl der Bälle im Zielkorb beim Fortschreiten der Inferenz jedes Mal sukzessive ansteigt.

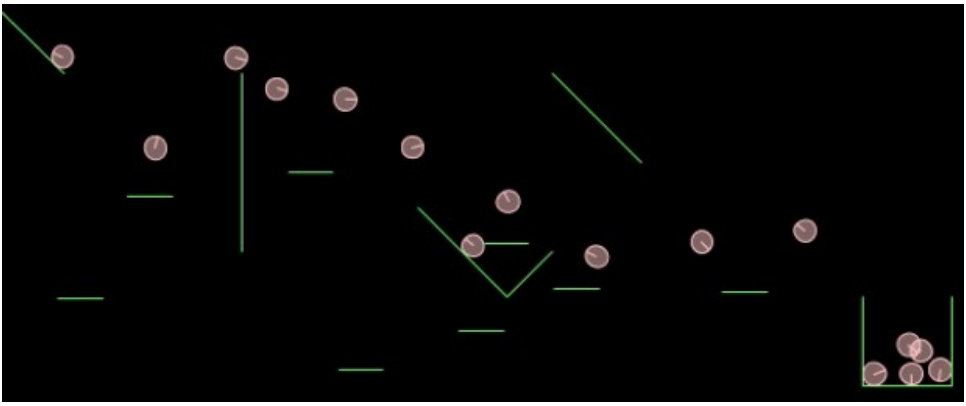

Abb.7: Diese Aufgabe wurde durch zusätzliche Hindernisse erschwert und die Komplexität der Wechselwirkung der Bälle untereinander muss hier ebenfalls berücksichtigt werden. Für die von der Inferenz gefundene Lösung wird eines der Hindernissse (senkrechte Linie im linken Bereich) von den Bällen als zusätzlicher bumper verwendet (blauer Pfeil links). Und der eingeklemmte Ball im Zentrum der Szene (blauer Pfeil in der Mitte) dient nun selbst als bumper für die anderen Bälle.

Von einer höheren Warte aus betrachtet, kann man sagen: die Inferenz invertiert das generative Modell in einem lokalen Gebiet seines Parameterraums gemäß den Vorgaben des „Probabilistischen Programms“ – d.h. sie findet qua expliziter Darstellung der Posterior-Verteilung Werte für diese Parameter, die zum conditioning des PP passen.

5.2 Datensäuberung mit PClean

Inkonsistente und fehlerhafte Daten sowie Datenlücken sind in allen Anwendungsbereichen von Wirtschaft, Verwaltung und Wissenschaft geradezu omnipräsent. Datensäube-rung ist sehr aufwendig. Daher besteht hier immenser Automatisierungsbedarf. Rein regel-basierte Verfahren haben sich als untauglich erwiesen. Kommerzielle Systeme kombi-nieren verschiedene Verfahren des maschinellen Lernens mit adaptierbaren Regel-systemen. PClean [Le21] ist ein neues, auf Prinzipien der „Probabilistischen Program-mierung“ basierendes System, das in einschlägigen Standard-Benchmarks mit gängigen, kommerziellen Systemen konkurrieren kann bzw. sogar leichte Vorteile aufweist und bei Datensätzen mit anspruchsvolleren Fehlerarten oder bei der Skalierung auf sehr große Datensätze (Tabellen mit 2 Mio. Zeilen) klar überlegen ist. PClean ist in Julia [Jul22] als Open Source Projekt implementiert [Pcl22]. Es definiert eine eigene DSL (domain specific language) zur Erstellung „Probabilistischer Programme“ zur Modellierung von Daten-tabellen und enthält spezielle Inferenzalgorithmen, die auf diese DSL abgestimmt sind, um die besonderen Skalierungsanforderungen zu erfüllen.

In einem mit PClean geschriebenen „Probabilistischen Programm“ kann mit Hilfe der DSL ein Vorschlag für die relevanten Klassen formuliert werden, aus denen sich der Gesamtdatensatz zusammensetzt. In diesen Klassen werden dann die Apriori-Vertei-lungen für die Daten angegeben und auch mögliche Fehlerquellen für diese Daten werden mit Hilfe der DSL spezifiziert, z.B. Vertauschungen von Buchstaben, falsche Einheiten bei numerischen Größen, Verwechslung von Groß- und Kleinschreibung, Falschschrei-bung von Namen, etc. Die Konditionierung auf die Daten geschieht durch den zu bereini-genden Gesamtdatensatz D selbst - also die große, fehlerbehaftete Tabelle, die gesäubert werden soll. Der spezielle Inferenzalgorithmus von PClean berechnet dann aus dem PP eine Posterior-Verteilung von relationalen Datenbanken, deren interne Daten nun aber gereinigt sind. Aus einer Stichprobe der Endphase dieser Inferenzberechnung kann dann wieder eine Gesamttabelle D‘ rekonstruiert werden, die gereinigt ist. Fassen wir zusam-men, was hier passiert: die Inferenz bestimmt für die Parameter des auf den verunreinigten Datensatz D konditionierten generativen Modells M eine geeignete Posterior-Verteilung über D‘. Das entspricht dem Schluss von der Wirkung auf die Ursache. Wenn man das in der anderen Richtung betrachtet, also von der Ursache auf die Wirkung, folgt daraus: das generative Modell mit genau diesen Parametern D‘ würde somit wieder den ursprüng-lichen, also verunreinigten Datensatz D hervorbringen – formal ausgedrückt: M(D‘) = D. Die durch die Inferenz gefundenen Parameter des generativen Modells repräsentieren aber das, was wir suchen, nämlich den gereinigten Datensatz D‘ bzw. eine relationale Datenbank, in der dieser Datensatz enthalten ist. In diesem Sinne invertiert die Inferenz also das generative Modell M.

Um PClean auszuprobieren, wird nun ein synthetischer Datensatz einer Länge von 50.000 Zeilen mit Informationen zu Mietwohnungen verwendet, der viele verunreinigte Daten enthält. Dieser Datensatz wird via [Pcl22] zur Verfügung gestellt und in [Le21] detailliert erläutert. Er besteht aus vier Spalten: Zimmertyp, Mietpreis, Stadt bzw. Region, Bundesstaat. Folgende Fehler kommen vor: 1.) Namen von Städten/Regionen sind zu 2% falsch geschrieben. Viele unterschiedliche Korrekturvarianten sind i.d.R. möglich. 2.) Es fehlen 10% der Angaben zu Bundesstaaten. In den Bundesstaaten kommen häufig auch gleichlautende Städte bzw. Regionen vor, so dass die Zuordnung nicht eindeutig ist. 3.) Preisangaben sind zu 1% in einem falschen Zahlenformat bzw. in einer falschen Einheit angegeben. 4.) Der Wohnungstyp mit der Zimmeranzahl fehlt in 10% aller Fälle.

Die Datenreinigung dauerte mit der oben spezifizierten Hardware ca. 33 Sekunden, wobei ca. 27% dieser Zeit auf den Garbagekollektor und die Kompilation entfallen sind und während der Berechnung ca. 12,6 GB Speicherplatz benötigt wurde. Es steht ein korrekter Datensatz („ground truth“) zur Verfügung [Pcl22], der mit dem bereinigten Datensatz verglichen wurde, um die Leistungsfähigkeit von PClean zu messen: die Erfolgsquote (Verhältnis der Anzahl korrekter Reparaturen zur gesamten Fehlerzahl) betrug 0,69 und die Güte (Verhältnis der Anzahl korrekter Reparaturen zur Anzahl aller Reparaturversuche) betrug ebenfalls 0,69. Genau diese Werte wurden auch in [Le21] genannt. Die Autoren behaupten außerdem, dass PClean in weitergehenden Vergleichstests bzgl. dieses Datensatzes kommerziellen Systemen klar überlegen sei und belegen dies durch entsprechende Ergebnisse. Sie berichten darüber hinaus von der Bereinigung eines über 2 Mio. Zeilen

umfassenden Datensatzes aus dem medizinischen Bereich. Sie führen aus, dass ohne einen entsprechenden „ground truth“-Datensatz die Erfolgsmessung der Bereinigung aber natürlich nur stichprobenartig erfolgen könne. Hierbei hätte sich eine Erfolgsquote von 90% ergeben. Die Autoren betonen ausdrücklich, dass die Behandlung eines derart großer Datensatzes völlig außerhalb der Möglichkeiten kommerzieller Systeme läge.

5.3 Captcha Breaking

Es gibt mindestens drei Wege, eine Captcha-Hürde zu überwinden: (1) Man hat sehr viele, korrekt gekennzeichnete Captcha-Daten und trainiert damit ein Neuronales Netz. Dafür gibt es im Internet bereits eine Vielzahl von Do-it-yourself-Bastelanleitungen und Apps. Oder (2) man setzt ausgeklügelte Methoden aus der Bildverarbeitung und Muster-erkennung ein [MM03]. Oder (3) man verwendet „Probabilistische Programmierung“ wie in [Ma13]: darin wird mit Hilfe der universellen PPL Venture [Ven22] ein PP geschrie-ben, das einen Captcha-Generator als generatives Modell enthält. Durch ihn werden die Elemente einer Zeichenfolge zufällig angeordnet, skaliert und verrauscht. Die zugrunde-liegenden Zeichen sind die latenten Variablen des generativen Modells. Nun wird ein reales Captcha-Bild, dessen zugrundeliegenden Zeichen aber nicht gegeben sind, als Kon-ditionierung des PP verwendet. Durch Inferenz mittels Metropolis-Hastings MCMC wird das generative Modell (also der Captcha-Generator) invertiert, so dass Abschätzungen für die Werte der latenten Variablen möglich werden, d.h. die zugrundeliegenden Zeichen werden erkannt. Die Autoren [Ma13] berichten hier von einer Erkennungsrate von 70%.

5.4 Strategieplanung in dynamischen Modellen

Zur Entwicklung geeigneter Strategien zur Steuerung des dynamischen Ablaufs des Pandemie-Geschehens (COVID-19) wurden existierende epidemiologische Simulatoren in ein PP integriert [Wo22]. Als PPL wurde PyProb [Pyp22] verwendet, das auch bereits im CERN (Elementarteilchenphysik) eingesetzt wird. Dadurch dass sich Planungs-aufgaben als Inferenz darstellen lassen, konnten Maßnahmen zur Pandemiebekämpfung so aufeinander abgestimmt werden, dass sich die sozio-ökonomischen Kosten, die Belastung des Gesundheitssystems, Todesraten, etc. im Rahmen des Simulationsmodells minimieren ließen und auch Ausgangssperren konnten so ganz verhindert werden.

Zusammenfassung und Ausblick

Die zentrale Idee der „Probabilistischen Programmierung“ besteht in der auf ausgewählte Daten konditionierten Inversion eines generativen Modells des Anwendungsproblems, um geeignete Werte für die latenten Variablen dieses Modells zu bestimmen. Dafür reichen i.d.R. vergleichsweise geringe Datenmengen aus. Kurz gesagt: die Konditionierung ist eine Frage an das Modell, die dann von der Inferenz beantwortet wird. Wenn nur wenige Daten zur Verfügung stehen, aber ein generatives Modell für die Aufgabenstellung verfügbar ist, kann „Probabilistische Programmierung“ ein Weg sein, um Probleme anzu-gehen, die mit etablierten Methoden des Machine Learning nur schwer zu behandeln sind.

Das ambitionierte Fernziel des Forschungsfeldes der „Probabilistischen Programmierung“ ist die „automatisierte“ Lösung von Aufgabenstellungen des Machine Learning durch universelle Inferenzverfahren. Nicht-Experten sollen das jeweilige Problem dann nur noch deklarativ als „Probabilistisches Programm“ formulieren müssen. Den Rest übernimmt die Inferenz. Für dieses Fernziel wird die Verfügbarkeit leistungsfähigerer Inferenz-verfahren entscheidend sein. Als ein möglicher Weg wird dazu u.a. „Deep Probabilistic Programming“ [Pyr17, Pyp22, SWF16] propagiert. Besonders interessant ist auch die Möglichkeit der Integration einer externen Simulationsanwendung in ein Probabilistisches Programm (siehe z.B. 5.1). Die Simulationsanwendung übernimmt darin dann die Rolle des generativen Modells [Ang18, Pyp22].

In der Kognitionsforschung wird „Probabilistische Programmierung“ zur Analyse und Modellierung von Lern- und Entscheidungsprozessen sowie von Sozialverhalten verwendet [Te11, Go16, Ev22, Go17, St22]. Und auch die Einsatzmöglichkeiten für die Wirtschaftsinformatik i.A. sind breit gestreut: z.B. im Business Process Reengineering, in der modellbasierten Planung und Steuerung komplexer Prozesse und in der Risiko-Modellierung, in Business Analytics, in IT-Sicherheit und in Quantitative Finance.

Literaturverzeichnis

[Ang18] Anglican, probprog.github.io/anglican, Stand: 24.3.2022

[Bi13] Bishop C.M.: Model-based machine Learning, Phil Trans R Soc A 371:20120222, Royal Society Publ., 2013

[Box18] Clojure wrapper für die Java Physik-Engine jbox2d, github.com/floybix/cljbox2d, 2018, Stand: 24.3.2022

[Clo22] The Clojure Programming Language, clojure.org, Stand: 24.3.2022

[Ev22] Evans, O.; Stuhlmüller, A.; Salvatier, J.; Filan, D.: *Modeling Agents with Probabilistic Programs, agentmodels.org, Stand: 24.3.2022*

[Gn08] Goodman, Noah; Mansinghka, Vikash; Roy, D.M.; Bonawitz, K.; Tenenbaum, J.B.: Church: A Language for Generative Models. In: Proc. of the 24th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2008), AUAI Press, 2008

[Go16] Goodman, N.D.; Tenenbaum, J.B., Buchsbaum,D.; Hartshorne, J.; Hawkins, R.; O'Donnel, T.J.; Tessler, M.H.: Probabilistic Models of Cognition, probmods.org, 2016, Stand: 24.3.2022

[Go17] Gopnik, A.: Making AI more Human. In: Scientific American, Juni 2017

[Gro86] Grosof, B.N.: Non-Monotonicity in Probabilistic Reasoning, In: Proceedings of the Second Conference on Uncertainty in Artificial Intelligence (UAI1986), 1986

[GS22] Goodman, N.D.; Stuhlmüller, A.: The Design and Implementation of Probabilistic Programming Languages, dippl.org, Stand: 24.3.2022

[Ja03] Jaynes, E.T.: Probability Theory: The Logic of Science, Cambridge Univ. Press, 2003

[Jul22] The Julia Programming Language, julialang.org, Stand: 24.3.2022

[KF09] Koller, D.; Friedman, N.: Probabilistic Graphical Models, MIT Press, 2009

[La18] Lambert, B: A Student's Guide to Bayesian Statistics, SAGE Publications Ltd., 2018

[Le21] Lew, A.K.; Agrawal, M.; Sontag, D.; Mansinghka, V.K.: PClean: Bayesian Data Cleaning with Domain-Specific Probabilistic Programming. In: Proc. of the 24th Int. Conf. on AI and Statistics (AISTATS 2021), PMLR, 2021

[Ma13] Mansinghka, V.K.; Kulkarni, T.D.; Perov, Y.N.; Tenenbaum, J.B.: Approximate Bayesian Image Interpretation using Generative Probabilistic Graphics Programs. In: Adv. in Neural Information Processing Systems 26 (NIPS 2013), Curran Ass., Inc., 2013

[Mc20] McElreath, R.: Statistical Rethinking (2nd ed.), CRC Press – Taylor&Francis Group – Chapman & Hall Book, 2020

[Me18] van de Meent, J.-W.; Paige, B; Yang, H; Wood, F.: An Introduction to Probabilistic Programming, arxiv.org/abs/1809.10756, Stand: 24.3.2022

[MM03] Mori, G; Malik, Jitendra. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In: Proc. of the Int. Conf. on Comp. Vis. and Pattern Recog., IEEE, 2003

[NJ01] Ng, A.; Jordan, M.I.: On Discriminative vs. Generative classifiers: A comparison of logistic regression and naïve Bayes. In: Adv. in Neural Inf. Proc. Systems 14 (NIPS 01), MIT Press, 2002

[Pcl22] PClean: A Domain-Specific Probabilistic Programming Language for Bayesian Data Cleaning, github.com/probcomp/PClean, Stand: 24.3.2022

[Pe88] Pearl, J.: Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, 1988

[Phy15] Probabilistic Programming Repositories - MLSS2015 (practical materials for MLSS Tübingen) - physics, bitbucket.org/probprog/, Stand: 24.3.2022

[Pyp22] PyProb, github.com/pyprob/pyprob, Stand: 24.3.2022

[Pyr17] Uber AI Labs Open Sources Pyro, eng.uber.com/pyro, Stand: 24.3.2022

[RP02] Ramsey N.; Pfeffer, A.: Stochastic lambda calculus and monads of probability distributions. In: Proc. of the 29th ACM-SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002), ACM, 2002

[St22] Stuhlmüller, A.: Forest - repo. for generative models, forestdb.org, Stand: 24.3.2022

[SWF16] Salvatier, J.; Wiecki, T.V.; Fonnesbeck, C.: Probabilistic programming in Python using PyMC3. PeerJ Computer Science 2:e55, 2016

[Te11] Tenenbaum, J.B.; Kemp, C.; Griffiths, T.L.; Goodman, N.D.: How to Grow a Mind: Statistics, Structure, and Abstraction. In: Science, Vol. 331, 2011

[To16] Tolpin, D.; van de Meent, J.-W.; Yang, H.; Wood, F.: Design and Impl. of Probabilistic Programming Language Anglican, In: Proc. of the 28th Symp. on the Impl. and Appl. of Functional Progr. Lang. (IFL 2016), ACM, 2016

[Ven22] Venture, probcomp.csail.mit.edu/software/venture, Stand: 24.3.2022

[Wi19] Winn, J. (et al.): Model-based Machine Learning, mbmlbook.com, Stand: 24.3.2022

[WMM14] Wood, F; van de Meent, J.-W.; Mansinghka, Vikash: A New Approach to Probabilistic Programming Inference. In: Proc. of the 17th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2014), Vol. 33 of Proc. of PMLR, 2014

[Wo22] Wood, F.; Warrington, A.; Naderiparizi, S.; Weilbach, C.; Masrani, V.; Harvey, W.; Scibior, B.; Grefenstette, J.; Campbell, D.; Nasseri, A.: Planning as Inference in Epidemiological Dynamic Models. In: Frontiers in Artificial Intelligence (31. 3.2022), www.frontiersin.org