Probabilistische graphische Modelle mit Scala

Andreas Bille
rcs systems GmbH

- Ab 1500 G. Cardano Glücksspiel
- Ab 1600 Fermat, Pascal Diskrete Räume
- 1763 Bayes **Bayes Theorem**
- 1902 Gibbs Graphen als Distribution
- 1921 Wright Genetik
- 1930 Kolmogorov Moderne Formulierung
- …
- 1972 de Bombal et al. Naive Bayes model
- 1988 J. Pearl Probabilistic Reasoning
- 1992 Heckerman et.al. Pathfinder
- Today medical diagnosis, market data, natural language, genetic, communication, social science …

Resources

- http://probabilistic-programming.org/wiki/Home:

- **Existing probabilistic programming systems**
- Below we have compiled a list of probabilistic programming systems including languages, implementations/compilers, as well as software libraries for constructing probabilistic models and toolkits for building probabilistic inference algorithms.
- Anglican is a portable Turing-complete research probabilistic programming language that includes particle MCMC inference.
- BLOG, or Bayesian logic, is a probabilistic programming language with elements of first-order logic, as well as an MCMC-based inference algorithm. BLOG makes it relatively easy to represent uncertainty about the number of underlying objects explaining observed data.
- BUGS is a language for specifying finite graphical models and accompanying software for performing B(ayesian) I(nference) U(sing) G(ibbs) S(ampling), although modern implementations (such as WinBUGS, JAGS, and OpenBUGS) are based on Metropolis-Hastings. BiiPS is an implementation based on interacting particle systems methods like Sequential Monte Carlo.
- Church is a universal probabilistic programming language, extending Scheme with probabilistic semantics, and is well suited for describing infinite-dimensional stochastic processes and other recursively-defined generative processes (Goodman, Mansinghka, Roy, Bonawitz and Tenenbaum, 2008). Implementations of Church include MIT-Church, Cosh, Bher, and JSChurch.
- Dimple is a software tool that performs inference and learning on probabilistic graphical models via belief propagation algorithms or sampling based algorithms.
- FACTORIE is a **Scala** library for creating relational factor graphs, estimating parameters and performing inference.
- Figaro is a **Scala** library for constructing probabilistic models that also provides a number of built-in reasoning algorithms that can be applied automatically to any constructed models.
- HANSEI is a domain-specific language embedded in OCaml, which allows one to express discrete-distribution models with potentially infinite support, perform exact inference as well as importance sampling-based inference, and model inference over inference.
- Hierarchical Bayesian Compiler (HBC) is a language for expressing and compiler for implementing hierarchical Bayesian models, with a focus on large-dimension discrete models and support for a number of non-parametric process priors.
- PRISM is a general programming language intended for symbolic-statistical modeling, and the PRISM programming system is a tool that can be used to learn the parameters of a PRISM program from data, e.g., by expectation-maximization.
- Infer.NET is a software library developed by Microsoft for expressing graphical models and implementing Bayesian inference using a variety of algorithms.
- Probabilistic-C is a C-language probabilistic programming system that, using standard compilation tools, automatically produces a compiled parallel inference executable from C-language generative model code.
- ProbLog is a probabilistic extension of Prolog based on Sato's distribution semantics. While ProbLog1 focuses on calculating the success probability of a query, ProbLog2 can calculate both conditional probabilities and MPE states.
- PyMC is a python module that implements a suite of MCMC algorithms as python classes, and is extremely flexible and applicable to a large suite of problems. PyMC includes methods for summarizing output, plotting, goodness-of-fit and convergence diagnostics.
- R2 is a probabilistic programming system that employs powerful techniques from programming language design, program analysis and verification for scalable and efficient inference.
- Stan exposes a language for defining probability density functions for probabilistic models. Stan includes a compiler, which produces C++ code that performs Bayesian inference via a method similar to Hamiltonian Monte Carlo sampling.
- Venture is an interactive, Turing-complete, higher-order probabilistic programming platform that aims to be sufficiently expressive, extensible and efficient for general-purpose use. Its virtual machine supports multiple scalable, reprogrammable inference strategies, plus two front-end languages: VenChurch and VentureScript.

- Diverse commercial packages, packages für MatLab, Mathematica, carda

Gibt es „probabilistisches Programmieren“?

„Traditionelles Programmieren“	„Probabilistisches Programmieren“
Computer können erstmal nichts.	Computer sind autonom lernfähig.
Der Entwickler ist allwissend.	Der Entwickler wird zum Trainer.
Ein Programm ist in sich stark strukturiert – oder sollte es sein: Klassen, Module, Komponenten, Funktionen, Tiers, Bus, Service, Regeln, Workflow ….	Ein Graph setzt Wahrscheinlichkeitsvariablen zueinander in Beziehung. Systemmodellierung ist vor allem Bestimmung der relevanten WV.
Ein Programm ist eine eindeutige und deterministische Beschreibung des Systemverhaltens zu jeder Eingabe.	Der Computer lernt durch Training. Das Endergebnis wird nicht spezifiziert.
Logik, Eindeutigkeit, Determiniertheit	Datenqualität, Samplesize etc.
Spezifikation, Testen, Versionen ,Wartung, Application Lifecycle	Fehlerhafte Verhalten inhärent, Debugging, Lifecyle?
Ausgereift, erlernbar	Emerging, keine best practices etc.

Probabilistische Grundlagen

- Wahrscheinlichkeitsraum $(A, \sigma(A), P)$
- $A \in \sigma(A)$,
  $\emptyset \epsilon \sigma(A), abzählbare\ Vereinigung, Komplement$
- $P: \sigma(A) \to R, P(\emptyset) = 0, P(A) = 1$
- $P(\alpha \cup \beta) = P(\alpha) + P(\beta), falls\ \alpha \cap \beta = \emptyset$

Z.B. Würfel:

$$A = \{1,2,3,4,5,6),$$
$$\sigma(A) = \{Menge\ aller\ Teilmengen\ von\ A\},$$
$$P(\alpha) = |\alpha|/6$$

Bayes

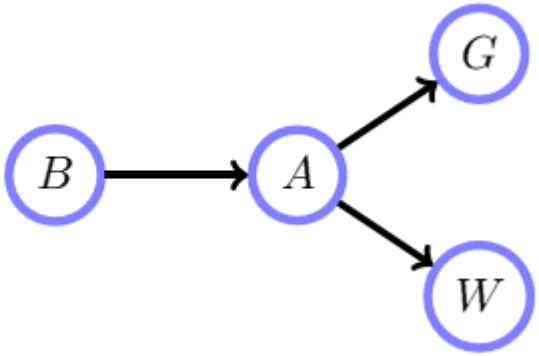
- Wahrscheinlichkeitsvariablen X,Y,Z…
- val X:$\sigma(A)$
- Wahrscheinlichkeitsverteilung über alle X,Y,Z,… P(X,Y,Z,…)
- Bayes bedingte Wahrscheinlichkeit:

$$\mathbf{P(X|Y)} = \frac{\boldsymbol{P(X \cap Y)}}{\boldsymbol{P(Y)}}, \mathbf{P(Y)} = \int \boldsymbol{dX}\ \boldsymbol{P(X, Y)}$$

Graphische Darstellung

$$P(G,W,A,B) = P(G|W,A,B)*P(W|A,B)*P(A|B)*P(B)$$

Aber:



$$P(G,W,A,B) = P(G|A)*P(W|A)*P(A|B)*P(B)$$

Figaro

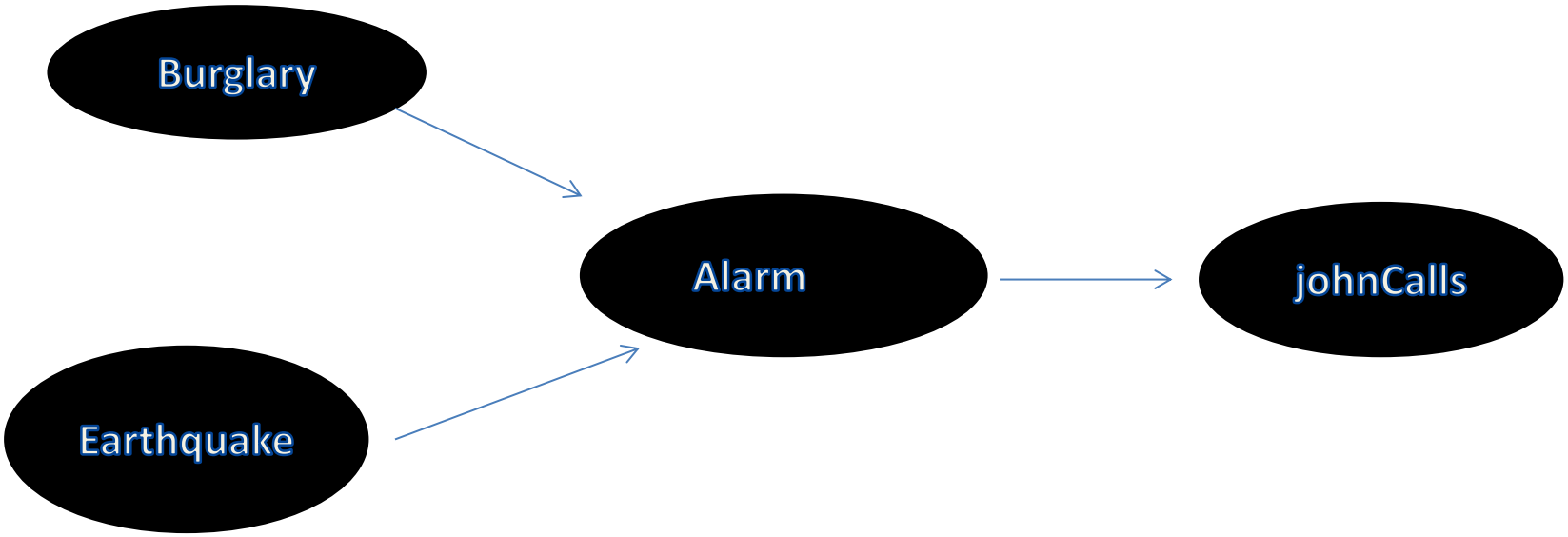
- Avi Pfeffer, Charles River Analytics
- Scala library
- Dokumentation gut
- Manning in Arbeit (2015)
- Umfangreich, erweiterbar

- Name: Der Autor komponiert klassische Musik

L1,L2

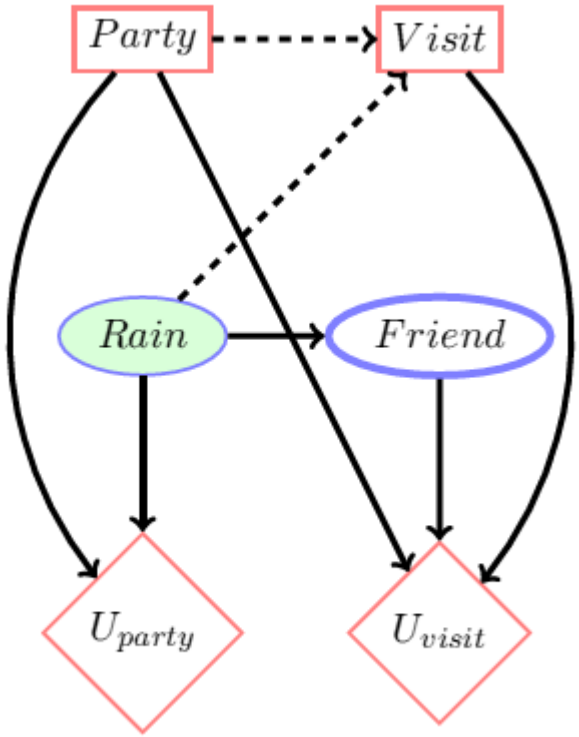
Abragen, Evidenz, Lernen

- $Q = \{Q1, Q2, \ldots.\}$,
- $E = \{E1, E2..\}$,
- $U = \{U1, U2, \ldots\}$
- P(Q,U,E) Wahrscheinlichkeitsverteilung
- $P(Q) = \frac{1}{N}\sum_U P(Q, U, E = \{e1, e2, \ldots\})$
- $N = \sum_{Q,U} P(Q, U, E = \{e1, e2, ..\})$
- Most probable explanation


Burglary
Alarm
johnCalls
Earthquake

L3

Informationen, Entscheidungen, Nutzen



Informationlink:
Informationen, die
in Entscheidungen einfliessen

Entscheidungen

Utility-Funktionen

L4

rcs

- Beratung, Entwicklung
- Schulung
- Eigenentwicklung carda (pre-alpha)
    - Fokus sehr große Systeme, Cloud, Web
    - **Parallelisierbarkeit** und Verteilbarkeit
    - Veröffentlichung von Modellen
    - Entscheiden lernen versus Model lernen

Literatur

- *Risk Assessment and Decision Analysis with Bayesian Networks*, N. Fenton, M. Neil, CRC Press
- *Bayesian Reasoning and Machine Learning*, D. Barber, Cambridge
- *Probabilistic Graphical Models*, D. Koller, N. Friedman, MIT Press
- *Modeling and Reasoning with Bayesian Networks*, A. Darwiche, Cambridge