MODEL-LEARNER PATTERN

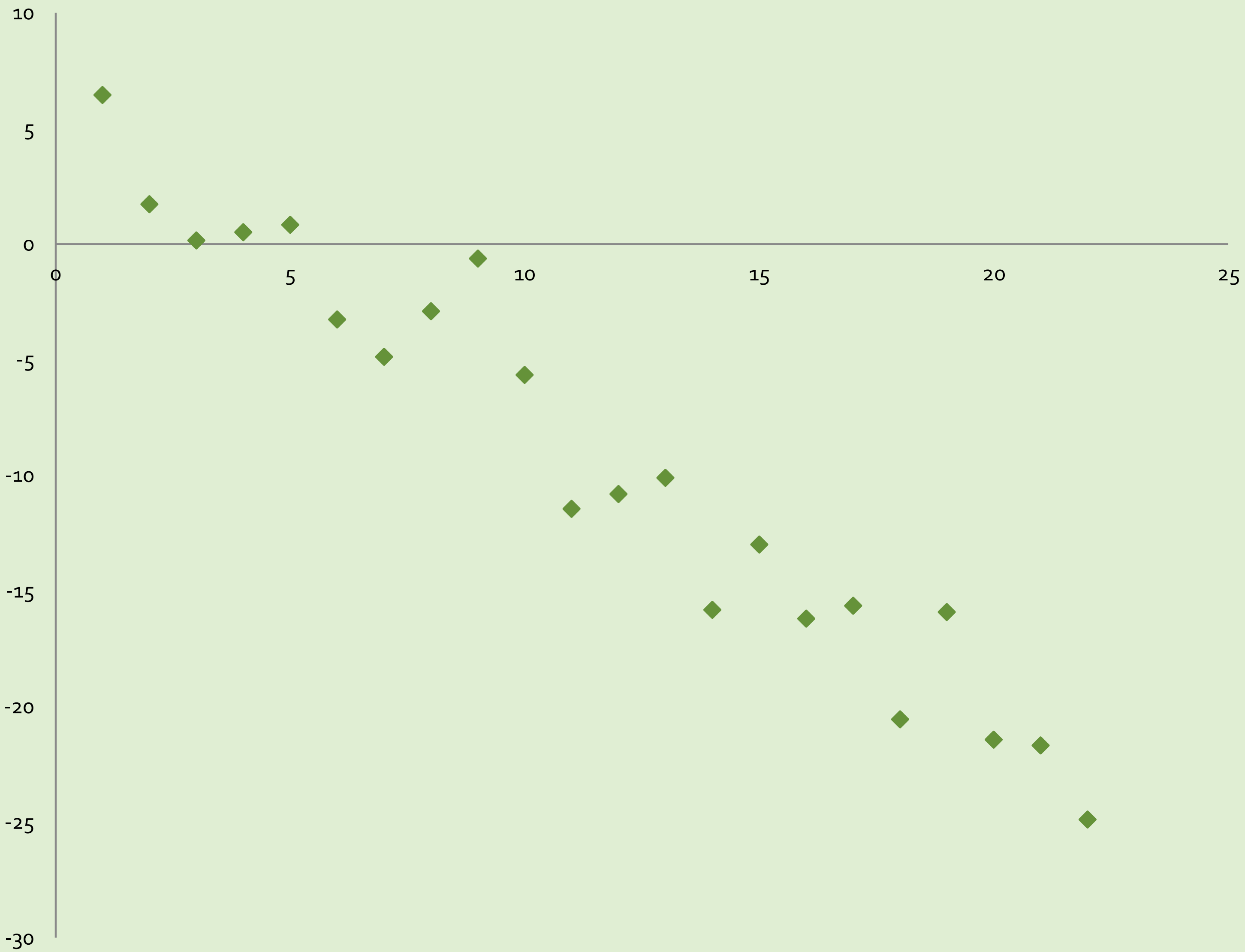
Andy Gordon (MSR and University of Edinburgh)

Joint work with Mihhail Aizatulin (OU), Johannes Borgström (Uppsala), Guillaume Claret (MSR), Thore Graepel (MSR), Aditya Nori (MSR), Sriram Rajamani (MSR), and Claudio Russo (MSR)

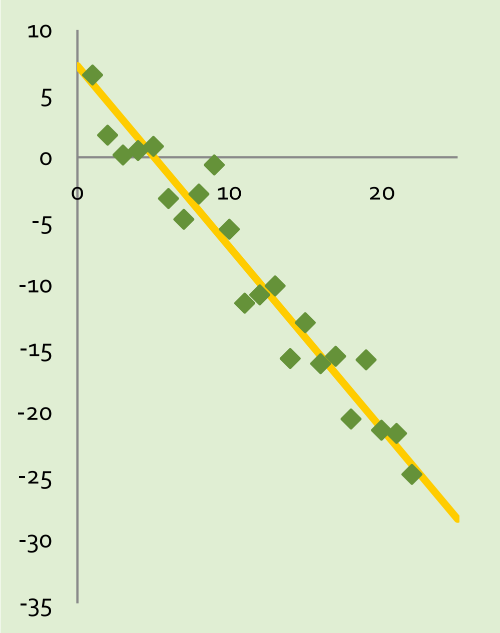
Microsoft Research

Microsoft

What next?



Bayesian Models



A Model

$$y = Ax + B + e$$

where **noise** $e \sim N(0, P)$

x is an **input**, y is an **output**.
A, B, P are the model **parameters**.

- We **train** on the observed inputs and outputs to **learn** the parameters, and to **predict** new outputs on unseen inputs.
- **Bayesian** models capture uncertainty about model components as probability distributions.

Five Distributions

- **Prior** distribution: p(w)
  given by $w = (A, B, P), A \sim N(0,1), B \sim N(0,1)$ and $P \sim \Gamma(1,1)$
- **Sampling** distribution: $p(y|x, w)$
  given by $y \sim N(Ax + B, P)$ for $w = (A, B, P)$
- **(Prior) Predictive** distribution:
  $$p(y|x) = \int p(y|x, w)\, p(w)\, dw$$
- **Posterior** distribution, given training data $d = (x, y)$:
  $$p(w|d) = \frac{p(y|x, w)\, p(w)}{p(y|x)}$$
- **Posterior predictive** distribution, given $d = (x, y)$:
  $$p(y'|x', d) = \int p(y'|x', w)\, p(w|d)\, dw$$

Three Classes of Bayesian Inference

$$p(w|d) = \frac{p(y|x,w)\,p(w)}{p(y|x)} \text{ where } d = (x,y)$$

- **Exact inference** for discrete distributions:
  Representation: enumerations of probabilities
  Example: $[HH, \frac{1}{10}; HT, \frac{2}{10}; TH, \frac{7}{10}; TT, 0]$
- Approximate inference: sampling eg **Markov chain Monte Carlo**:
  Representation: finite ensemble of samples
  Example: $[A = 1.7, B = 1.6; A = 9.9, B = 9.8 ; \ldots]$
- Approximate inference: **belief propagation on factor graphs**:
  Representation: parameters for marginal of each variable
  Example: $[A = N(5.1{,}10), B = N(6.0{,}5)]$

Bayesian Models are Widely Applicable

- Many machine learning tasks may be cast as Bayesian models.
- We infer functions from inputs to outputs, governed by uncertain parameters.
- Examples include:
  - A **regression function** inputs a tuple of independent variables, and produces one (or more) dependent variables (typically continuous).
  - A **classifier** inputs a vector of **features** and outputs a single value, the **class** (typically discrete).
  - A **cluster analysis** groups items so that items in each cluster are more like each other than to items in other clusters.
  - A **recommender** predicts the rating or preference that a user would give to an item (such as music, books, or movies) based on previous ratings by a set of users.
  - A **rating system** assesses a player's strength in games of skill (such as chess or Go) based on observed game outcomes.

Promise of Probabilistic Programming

- Custom inference code is hard to write, depends on mechanism
- Instead, user writes a probabilistic model for a Bayesian inference problem as a short piece of code, while the compiler turns this code into an efficient inference routine.
- Systems include BUGS, IBAL, BLOG, Church, STAN, Infer.NET, Fun, Factorie, Passage, HBC, HANSEI, and more.
- Still, no linguistic abstractions for Bayesian models.
- **Our contribution:** a new typed **model** abstraction to represent a function from X to Y, governed by W:
  - may be composed to form richer models
  - via a **sampler**, may be run to draw from predictive distribution
  - via a **learner**, may be trained to make predictions

Distributions (1-3) as Probabilistic Code

- **Prior** distribution: $p(w|h)$ for **hyperparameter** h:

```
let prior (h:TH) =
  {A = random (Gaussian(h.MeanA, h.PrecA))
   B = random (Gaussian(h.MeanB, h.PrecB))
   P = random (Gamma(h.Shape, h.Scale))} : TW
```

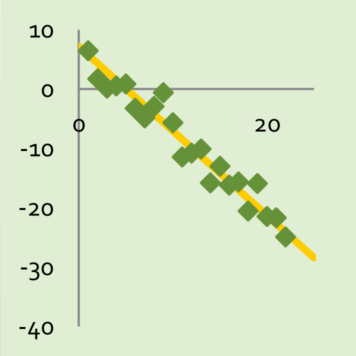
- **Sampling** distribution: $p(y|x, w)$

```
let gen(w,x) =
  [| for xi in x -> random(Gaussian(w.A * xi + w.B, w.P))|]
```

- **(Prior) Predictive** distribution:

$$p(y|x, h) = \int p(y|x, w)\, p(w|h)\, dw$$

```
let predictive(h,x) = let w = prior h in gen (w,x)
```



Distributions (4-5) as Probabilistic Code

- **Posterior** distribution, $p(w|d,h)$ where $d = (x,y)$:

$$p(w|d,h) = \frac{p(y|x,w)\,p(w,h)}{p(y|x,h)}$$

```
let posterior (h,x,y) =
  let w = prior h in observe (y = gen (w,x)); w
```

- **Posterior predictive** distribution:

$$p(y'|x',d,h) = \int p(y'|x',w)\,p(w|d,h)\,dw$$

```
let posterior_predictive (h,x,y,x') =
  let w = posterior (h,x,y) in gen (w,x')
```

Inference on Probabilistic Code

- F# quotations represent probabilistic code:

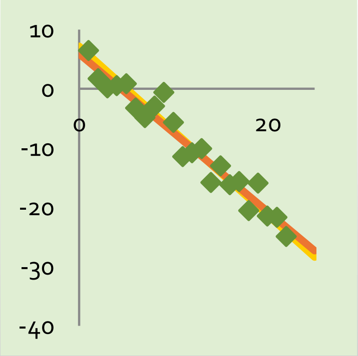
```
let d = <@ fun m -> (random(Gaussian(m,1.0), random(Bernoulli(0.5))) @>
: Expr<double -> double * bool>
```

- Infer.NET's inference invoked by a dynamically typed function, returning a marginalized representation `marginal('U)`

```
val infer : Expr<'T -> 'U> -> 'T -> marginal('U)
infer d 42.0 : Gaussian * Bernoulli
```

- Hence, we train our linear regression example:

```
let wD:{A=Gaussian;B=Gaussian;P=Gamma} =
      infer <@ posterior @> (x,y)
let yD:Gaussian[] =
      infer <@ posterior_predictive @> (x,y,x)
```



Abstraction 1: Model

- A model represents a probabilistic function from TX to TY, governed by an uncertain, learnable TW parameter, and a fixed TH hyperparameter.

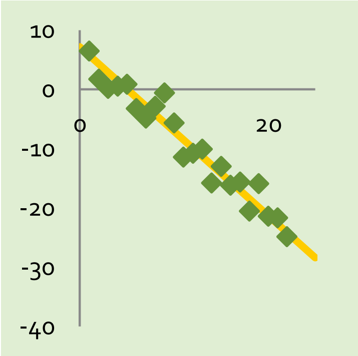
```
type Model<'TH,'TW,'TX,'TY> =
  { HyperParameter: 'TH
    Prior: Expr<'TH ->'TW>
    Gen: Expr<'TW *'TX ->'TY> }
```

```
{ HyperParameter = {MeanA=0.0; PrecA=1.0; … }
  Prior = <@ fun h ->
        { A = random(Gaussian(h.MeanA,h.PrecA))
          B = random(Gaussian(h.MeanB,h.PrecB))
          P = random(Gamma(h.ShapeN,h.ScaleN)) } @>
  Gen = <@ fun (w,x) -> [| for xi in x ->
               random(Gaussian(w.A * xi + w.B, w.P))|] @> }
```

Abstraction 2: Sampler

- A sampler is an object obtained from a model for sampling from the **prior** and **(prior) predictive** distributions, simply by running the code.

```
type ISampler<'TW,'TX,'TY> =
  interface
    abstract Parameters: 'TW
    abstract Sample: x:'TX -> 'TY
  end
```



Abstraction 3: Learner

- A learner is an object obtained from a model and an inference method, for computing the posterior and posterior predictive distributions, after training

```
type ILearner<'TDistW,'TX,'TY,'TDistY> =
  interface
    abstract Train: x:'TX * y:'TY -> unit
    abstract Posterior: unit -> 'TDistW
    abstract Predict: x:'TX -> 'TDistY
  end
```

Learner Semantics

```
type ReferenceLearner(m) =
  let mutable d = <@ (%m.Prior) (%m.HyperParameter) @>
  interface ILearner<Expr<'TW>,'TX,'TY,Expr<'TW> with
    member l.Train(x,y) =
      d <- <@ let w = %d in observe(y = (%m.Gen)(w,x)); w @>
    member l.Posterior() = d
    member l.Predict(x) = <@ let w = %d in (%m.Gen)(w,x) @>
```

- We have three efficient learners:
  - Exact (ADD/CUDD): algebraic decision diagrams
  - MCMC (Filzbach): ensembles of samples
  - Factor graphs (Infer.NET): marginal parametric distributions

Three Examples

	Linear Regression	BPM Classifier	TrueSkill
TH	{MeanA: double; PrecA: double; … }	{Ncols:int}	{Players:int}
TW	{A:double; B:double; Noise:double}	{Noise: double; Weights: Vector}	{Skills: double[]; PerfPrec: double}
TX	double	Vector	{ P1:int; P2:int }
TY	double	bool	bool
Posterior	{A:Gaussian; B:Gaussian; Noise:Gamma}	{Noise:Gaussian, Weights:VectorGaussian>	{Skills: Gaussian[]}
Predict	double -> Gaussian	Vector -> Bernoulli	{ P1:int;P2:int } -> Bernoulli

Generic Loopback Function

- Given these abstractions, we can write generic machine learning code, such as **loopback testing**

```
let test (toLearner: Model<'TH,'TW,'TX,'TY> ->
                     ILearner<'DistW,'TX,'TY,'DistY>)
         (m:Model<'TH,'TW,'TX,'TY>)
         (x:'TX) : 'TW * 'DistW =
   let S = Sampler.FromModel(m)
   let y = S.Sample(x)
   let L = toLearner(m)
   do L.Train(x,y)
   (S.Parameters,L.Posterior())
```

Array Combinator

- Allows training and prediction on IID data

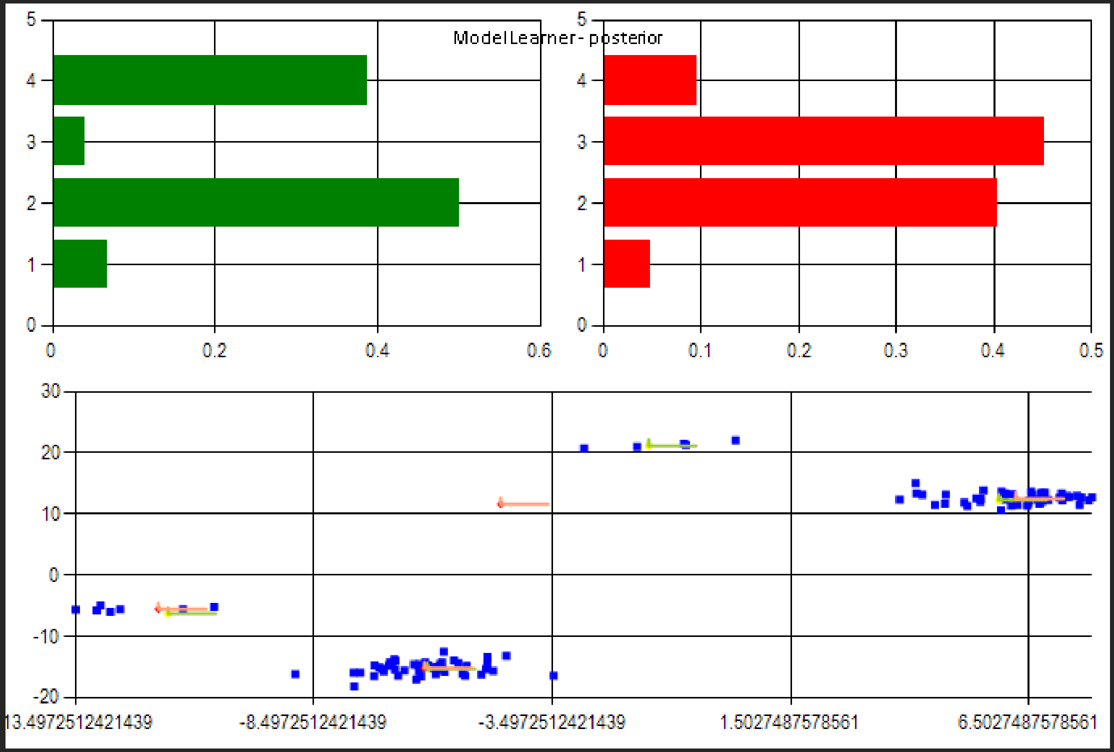
```
module IIDArray =
 let M(m:Model<'TH,'TW,'TX,'TY>)
      : Model<'TH,'TW,'TX[],'TY[]> =
     { Prior = m.Prior
       Gen = <@ fun (w,x) ->
         [| for xi in x -> (%m.Gen) (w,xi) |] @> }
```

Binary Mixture Combinator

- We code a variety of idioms as functions from models to models, eg, mixtures:

```
let Mixture(m1,m2) =
    {Prior =
      <@ fun h ->
         {Bias=random(Uniform(0.0,1.0))
          P1=(%m1.Prior) h
          P2=(%m2.Prior) h} @>
     Gen =
      <@ fun (w,x) ->
          if random(Bernoulli(w.Bias))
          then (%m1.Gen) (w.P1,x)
          else (%m2.Gen) (w.P2,x) @>}
```

Mixture Of Gaussians

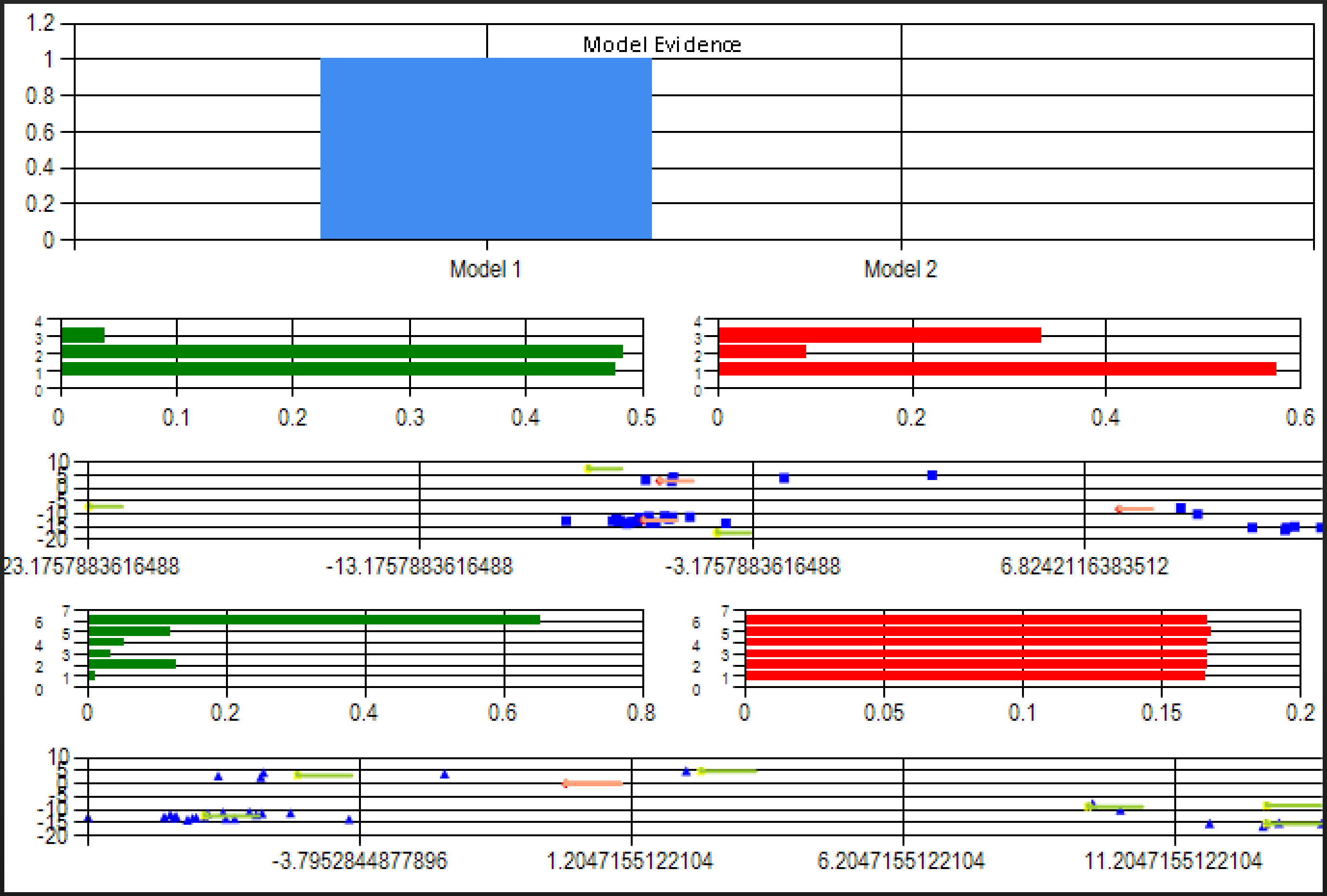



```
let k = 4 // number of clusters in the model
let M = IIDArray.M(KwayMixture.M(VectorGaussian.M,k))

let sampler1 = Sampler.FromModel(M);
let xs = [| for i in 1..100 -> () |]
let ys = sampler1.Sample(xs);

let learner1 = InferNetLearner.LearnerFromModel(M,mg0)
do  learner1.Train(xs,ys)
let (meansD2,precsD2,weightsD2) = learner1.Posterior()
```

Evidence Combinator

```
let M(m1,m2) =
  {Prior = <@ fun (bias,h1,h2) ->
              (breakSymmetry(random(Bernoulli(bias))),
               (%m1.Prior) h1, (%m2.Prior) h2) @>
   Gen = <@ fun ((switch,w1,w2),x) ->
            if switch then (%m1.Gen) (w1,x)
            else (%m2.Gen) (w2,x) @>}
```

Demo: Model Selection




```
let mx k = NwayMixture.M(VectorGaussian.M,k)
let M2 = Evidence.M(mx 3, mx 6)
```

A Dozen Models

Example / Learner	TH	TW	TDistW	TX	TY	TDistY
Sprinkler / A	SP.TH	SP.TW<bool>	ADD<SP.TW<bool>>	SP.TX	bool	ADD<bool>
TwoCoins / A	TC.TH	TC.TW<bool>	ADD<TC.TW<bool>>	TC.TX	bool	ADD<bool>
Two Coins / IN	TC.TH	TC.TW<bool>	TC.TW<Bernoulli>	TC.TX	bool	Bernoulli
Friends / A	bool[][]	bool list list	ADD<bool list list>	int * int * int	bool	ADD<bool>
Students / A	int * int	bool list list	ADD<bool list list>	int * int * int	bool	ADD<bool>
Gaussian / IN	GM.TH	GM.TW<$\mathscr{R}$,$\mathscr{R}$>	GM.TW<$\mathscr{N}$,Γ>	unit	real	$\mathscr{N}$
Gaussian Mix/ IN	MX1.TH	$\mathscr{R}$* GaussW*GaussW	β* GM.TW<$\mathscr{N}$,Γ>*GM.TW<$\mathscr{N}$,Γ>	unit	real	$\mathscr{N}$
Gaussian Mix / F	MX2.TH	(GaussW*GaussW)	(GaussW*GaussW)[]	unit	real	$\mathscr{R}$[]
PlantGrowth / F	unit	PG.TW	PG.TW[]	int	real	$\mathscr{R}$[]
TrueSkill / IN	TS.TH	TS.TW<$\mathscr{R}$>	TS.TW<$\mathscr{N}$>	TrueSkill.TX	bool	Bernoulli
Lin. Reg. / IN	LR.TH	LR.TW<$\mathscr{R}$,$\mathscr{R}$,$\mathscr{R}$>	LR.TW<$\mathscr{N}$,$\mathscr{N}$,Γ>	real	real	$\mathscr{N}$
MV Gaussian / IN	MVG.TH	MVG.TW<$\vec{\mathscr{R}}$,$\mathscr{M}$>	MVG.TW<$\vec{\mathscr{N}}$,$\mathscr{W}$>	unit	$\vec{\mathscr{R}}$	$\vec{\mathscr{N}}$

$\mathscr{R}$= real $\mathscr{N}$= Gaussian β= Beta Γ= Gamma
$\vec{\mathscr{R}}$= Vector $\mathscr{M}$= PositiveDefiniteMatrix
$\mathscr{W}$= Wishart // *generalizes* Γ *to multiple dimensions*
$\vec{\mathscr{N}}$= VectorGaussian // *multivariate Gaussian distribution*
GaussW= {Mean:$\mathscr{R}$; Precision:$\mathscr{R}$}
BetaW = {trueCount: $\mathscr{R}$; falseCount: $\mathscr{R}$}
SP.TH = {RainH: $\mathscr{R}$; SprinklerH: $\mathscr{R}$}
SP.TW<'TB> = {Rain: 'TB; Sprinkler: 'TB}
SP.TX = IsGrassWet // *a unit type*
TC.TH = {Bias1: $\mathscr{R}$; Bias2: $\mathscr{R}$}
TC.TW<'TB> = {Heads1: 'TB; Heads2: 'TB}
TC.TX = AreEitherHeads // *a unit type*
GM.TW<'TM,'TP> = {Mean:'TM; Precision:'TP}
GM.TH ={ Gaussian: GaussW, Gamma: GammaW }

MX1.TH = BetaW * GM.TH * GM.TH
MX2.TH = GM.TH * GM.TH
PG.TW = {alpha:$\mathscr{R}$;topt:$\mathscr{R}$;trho:$\mathscr{R}$;imass:$\mathscr{R}$;sigma:$\mathscr{R}$}
TS.TH = { Players: int; G: GaussW; P: GammaW }
TS.TW<'TA> = { Skills: 'TA[]}
TS.TX = { P1:int; P2: int }
LR.TH = {MeanA: $\mathscr{R}$; PrecA: $\mathscr{R}$; MeanB: $\mathscr{R}$; PrecB: $\mathscr{R}$; Shape: $\mathscr{R}$; Scale: $\mathscr{R}$}
LR.TW<'TA,'TB,'TN> = {A:'TA; B:'TB; Prec:'TN}
MVG.TH = {NCols:int; MeanVectorPrecisionCount:$\mathscr{R}$; WishartShapeConstant:$\mathscr{R}$; WishartScaleConstant:$\mathscr{R}$}
MVG.TW<'TM,'TC> = {Mean:'TM; Covariance:'TC}

Table 1. Rows show types for L : ILearner(TDistW,TX,TY,TDistY) for m : Model<TH,TW,TX,TY> (A=ADD, IN=Infer.NET, F=Filzbach)

Related and Future Work

- Roger Grosse's compositional theory of Bayesian image processing UAI 2012, plus greedy model selection algorithm – fits model-learner pattern.
- Extend our learner API to support partially observed output, eg, for Naïve Bayes or Hidden Markov Models.
- Completeness? Which Bayesian models don't fit?
- **Probabilistic metaprogramming** refers to automatic techniques for constructing probabilistic programs.
- Next, we aim to develop schema-directed probabilistic metaprogramming for inference on databases, an area in its infancy (cf Singh and Graepel's InfernoDB).

The **model-learner** pattern brings structure and types, as well as PL syntax, to probabilistic graphical models

```
module LinearRegression =
  type TH = {MeanA: double; PrecA: double; … }
  let h = {MeanA=0.0; PrecA=1.0; … }
  type TW<'a,'b,'c> = {A:'a; B:'b; Noise:'c}
  type TX = double
  type TY = double
  let M: Model<TH,TW<double,double,double>,TX,TY> =
    { Prior = <@ fun h ->
        { A = random(Gaussian(h.MeanA,h.PrecA))
          B = random(Gaussian(h.MeanB,h.PrecB))
          Noise = random(Gamma(h.ShapeN,h.ScaleN)) } @>
      Gen = <@ fun a -> let m = (a.W.A * a.X) + a.W.B
                 random(Gaussian(m, a.W.Noise)) @> }
```

```
module Classifier
  module Regression
   module TrueSkill
    module TopicModel
```

OrdNum	Delay	CusNum	SalID	Amt	Payed	State	Payment
100023	3	1030	A1	$21.80	TRUE	WA	Card
100024	2	1004	A2	$123.87	FALSE	WV	Cash
100025	5	1029	B1	[illegible]	TRUE (81%)	WV	Cash
100026	1	1030	B2	$89.69	TRUE	NY	Card
100027	2±1.2	1012	C1	$91.30	FALSE	WV (55%)	Cash
100028	5	1031	C2	$26.78	FALSE (71%)	WA	Cash
100029	1	1022	A1	$147.46	FALSE	WA	Cash
100030	4	1020	A2	$78.54	TRUE	NY	Card
100031	1	1037	B1	$139.84	FALSE	WA	Cash
100032	1	1018	B2	$134.21	FALSE	WV	Cash
100033	1	1016	C1	$21.69	FALSE (62%)	NY	Cash
100034	1	1034	C2	$131.29	FALSE	MA	Cash
100035	5	1012	A1	$12.40	FALSE (89%)	MA	Card
100036	2	1027	A2	$129.44	FALSE	NY	Cash
100037	3	1003	B1	$52.30	FALSE	WV	Cash
100038	1	1001	B2	$138.92	FALSE (66%)	NY	Cash
100039	2	1033	C1	$100.17	TRUE	MA (90%)	Card
100040	3±1.5	1008	C2	$8.90	FALSE	WA	Cash
100041	1	1034	B1	$101.91	FALSE (68%)	NY	Cash
100042	3	1032	B2	$131.82	FALSE	WA	Cash

Write your model in F# or C#

Or choose from library

Or automatically generate

```
type Model
```

Assemble multiple models

Choose algorithm (eg, EP, VMP, Gibbs, ADD, Filzbach)

Synthetic data to test learner

```
type ISampler
```

```
type ILearner
```

Train, predict, repeat

The Paper

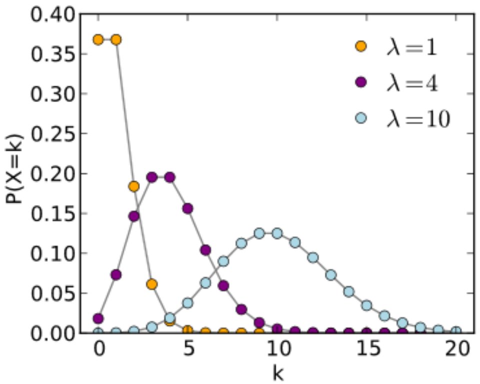
- The new conceptual insight is that code-based machine learning can be structured around **typed Bayesian models**, which are pairs of expressions representing prior and sampling distributions.
  - Definition of a type of Bayesian models, with combinators for compositionally constructing models, and operations to derive samplers and learners from an arbitrary model.
  - Many Bayesian examples expressed as models.
  - A formal semantics for models, learning, and prediction in Fun, and its semantics using measure transformers and probability monad.
  - Learners based on Algebraic Decision Diagrams, message-passing on factor graphs, and Markov chain Monte Carlo.

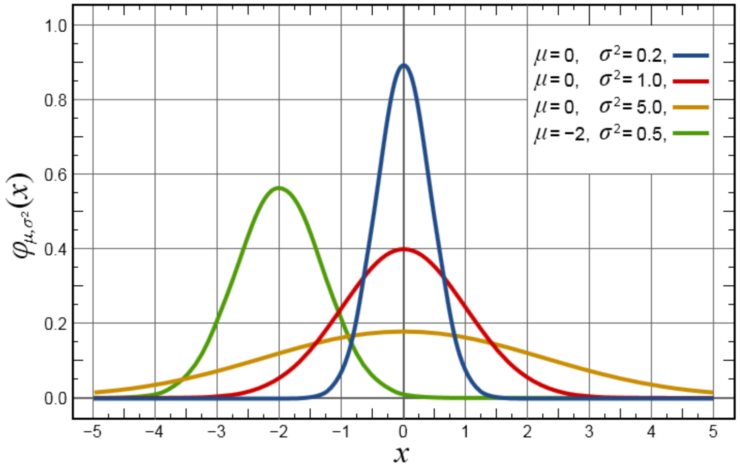
Questions?

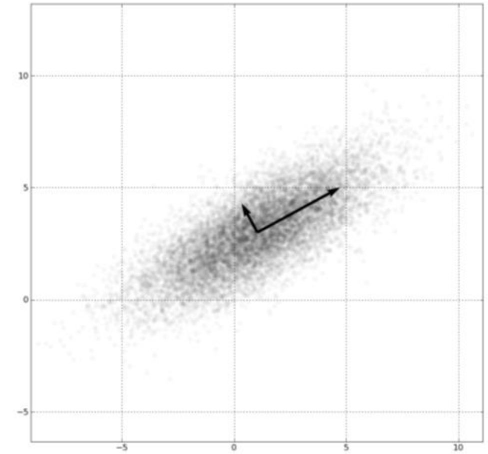
Infer.NET (since 2006)

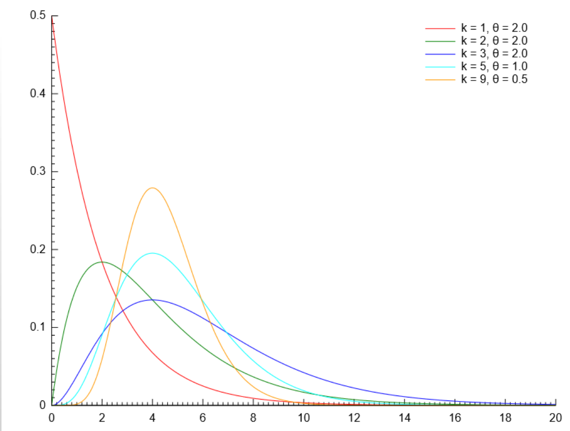
- Tom Minka, John Winn, John Guiver, and others
- A .NET library for probabilistic inference
  - Multiple inference algorithms on graphs
  - Far fewer LOC than coding inference directly
  - Designed for large scale inference
  - User extensible
- Supports rapid prototyping and deployment of Bayesian learning algorithms
  - Graphs represented by object model for pseudo code, but not as runnable code

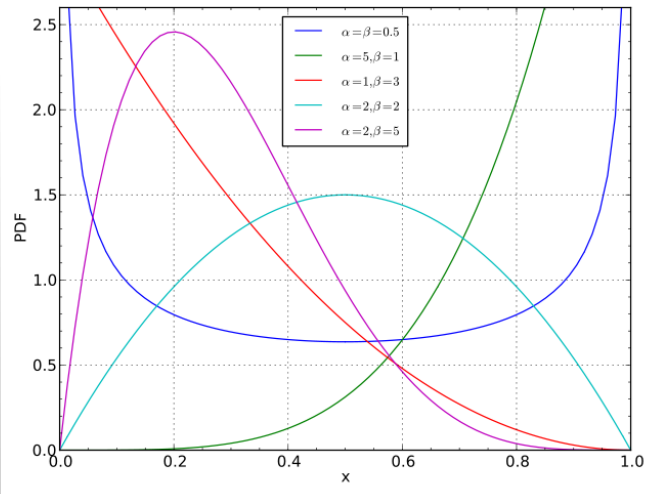
Some Probability Distributions in Fun

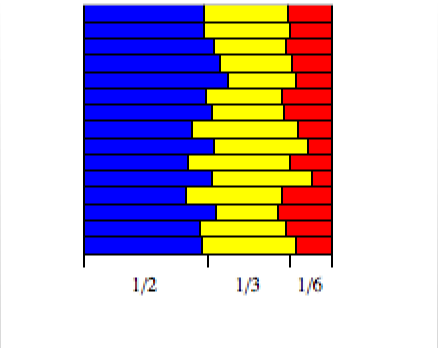












Source: Wikipedia