PMTK: Probabilistic Modeling Toolkit

Matt Dunham
Kevin Murphy

Dept. Computer Science
Univ. British Columbia
Canada

Presented at UBC LCI Forum, 25 May 2009

Outline

- Why?
- What?
- How?

Why yet another toolkit?

- I need software for my forthcoming textbook (“Machine learning: a probabilistic approach”, MIT Press Fall 2010)
- Book describes *simple*, but widely used, probabilistic models and algorithms (linear and logistic regression, mixture models, HMMs, CRFs / Newton’s method, stochastic gradient, EM, Gibbs sampling, etc)
- Want *unified interface* to all models/ algorithms, to enable mix&match, and better conceptual understanding
- Want *readable*, but reasonably efficient, high-level source code implementations of these models/ algorithms
- Existing toolkits inadequate
  - ML toolkits often not probabilistic
  - GM toolkits often not discriminative
  - Bayesian toolkits often not efficient

Generic ML toolkits

Name	Functionality	Language
PMTK	Probabilistic supervised learning (including kernel preprocessing), unsupervised density modeling	Matlab
Weka	Various supervised methods (dtrees, boosting, NN)	Java
Spider	Kernel-based supervised methods	Matlab
Netlab	NNs, mixtures, GPs	Matlab
Torch	NNs, mixtures, SVMs, HMMs, etc	C++
MLtools (Lawrence)	Various, including LLE, GPLVM, etc.	Matlab
Shogun	SVMs	C++
R packages	Many: random forests, CART, mixtures, etc.	R/C/Fortran

See www.mloss.org

Generic Bayesian toolkits

Name	Functionality	Language
PMTK	Exact conjugate analysis, MCMC, Importance sampling, Variational Bayes	Matlab
(Open)BUGS	Gibbs sampling	Component Pascal
JAGS	Gibbs sampling	Java
HBC (Daume)	Collapsed Gibbs, emphasis on non-parametric Bayes	C?
Infer.net (Winn&Minka)	EP, VB, Gibbs	closed
Blaise (Bonawitz)	MH	Java
FBM (Neal)	MH for NNs, mixture models, Dirichlet difusion trees	C
VIBES (Winn)	VB	Java

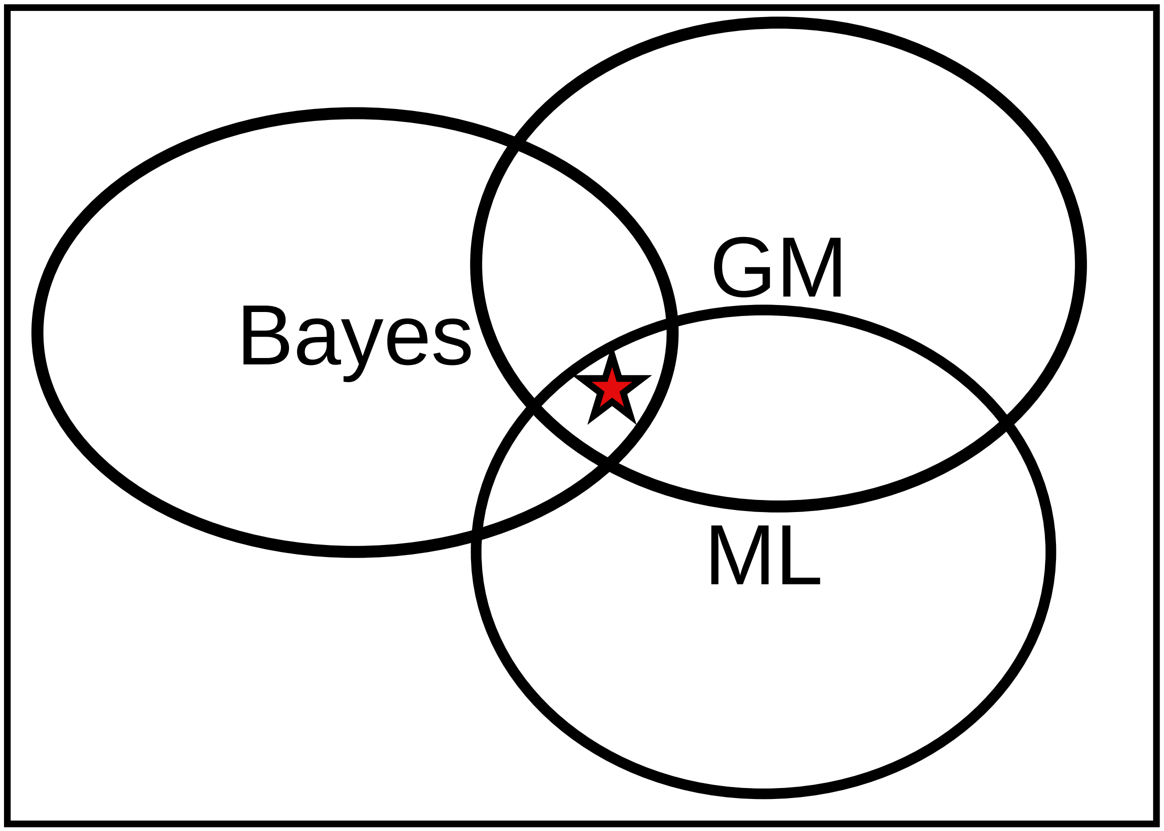
See "**Software for graphical models: a review**", Murphy, ISBA'07

Generic GM toolkits

Name	Functionality	Language
PMTK	DAGs, UGMs (Bayesian inference/ MAP estimation of states, parameters and structures)	Matlab
BNT	DAGs (parameter & state estimation, model selection)	Matlab
PNL (Intel)	DAGs, UGMs (parameter & state estimation)	C++
Hugin, Netica	DAGs, parameter & state estimation	$
VIBES, BUGS, infer.net	Hierarchical Bayes	-
GMTK (Bilmes)	DAGs, especially DBNs	(C++)
gR	Wrapper to various R packages	R
Smile/Genie	DAGs and influence diagrams	C++
Alchemy	Markov logic nets	C++?

See “**Software for graphical models: a review**”, Murphy, ISBA’07

The holy trinity



Why not BNT?

- BNT (Bayes Net Toolbox) is a very popular* Matlab package that I wrote in grad school
- But it does not support
  - Non-graph based probability models
  - Bayesian parameter estimation
  - Undirected graphical models
  - Non-parametric models (GPs, DPs, kNN, etc)
  - L1 priors
  - Kernels
  - Etc.
- Also
  - It is written in Matlab’s old object oriented system; the new version is much better (see later)
  - Various other design flaws

* About 120,000 visits between 1998-2002.

Why Matlab?

Pros

- Ideal for numerical computation
- Excellent plotting
- Easy rapid prototyping (interpreted, good IDE)
- Platform independent
- Succinct syntax
- Large code base
- Popular in ML comm.
- Functional / Object Oriented / Imperative

Cons

- Can be slow for anything other than matrix-vector computations
- Expensive for non-academics
- Not always backwards compatible

Natural alternatives: R, Python

Matlab 2008a's OO system

```
classdef ExampleClass < superclass1 & superclass2
% An example class definition

   properties
      prop1;        % field for storing data within objects of this clas
      prop2;
      prop3;
   end

   methods

       function obj = ExampleClass(varargin)
       % class constructor
            obj.prop1 = 1;
       end

       function obj = operation1(varargin)
       % public class method
       end

       function obj = operation2(varargin)

       end

       function val = get.prop1(obj)
       % this method is called, whenever a user tries to access the prop1
       % property of this class
       end

       function obj = set.prop1(obj)
       % this method is called, whenever the user tries to set the prop1 property
       end
   end

   methods(Access = 'private')
   % private helper methods not part of the class interface go here.
   end
end
```

New Syntax

- Single File Definition
- Abstract Classes
- Visibility Control
- Static Methods
- Handle Classes (can implement pointers, eg shared parameters)
- Events
- Event driven property access
- Operator Overloading
- Meta Classes

Outline

- Why?
- What?
- How?

Design Philosophy

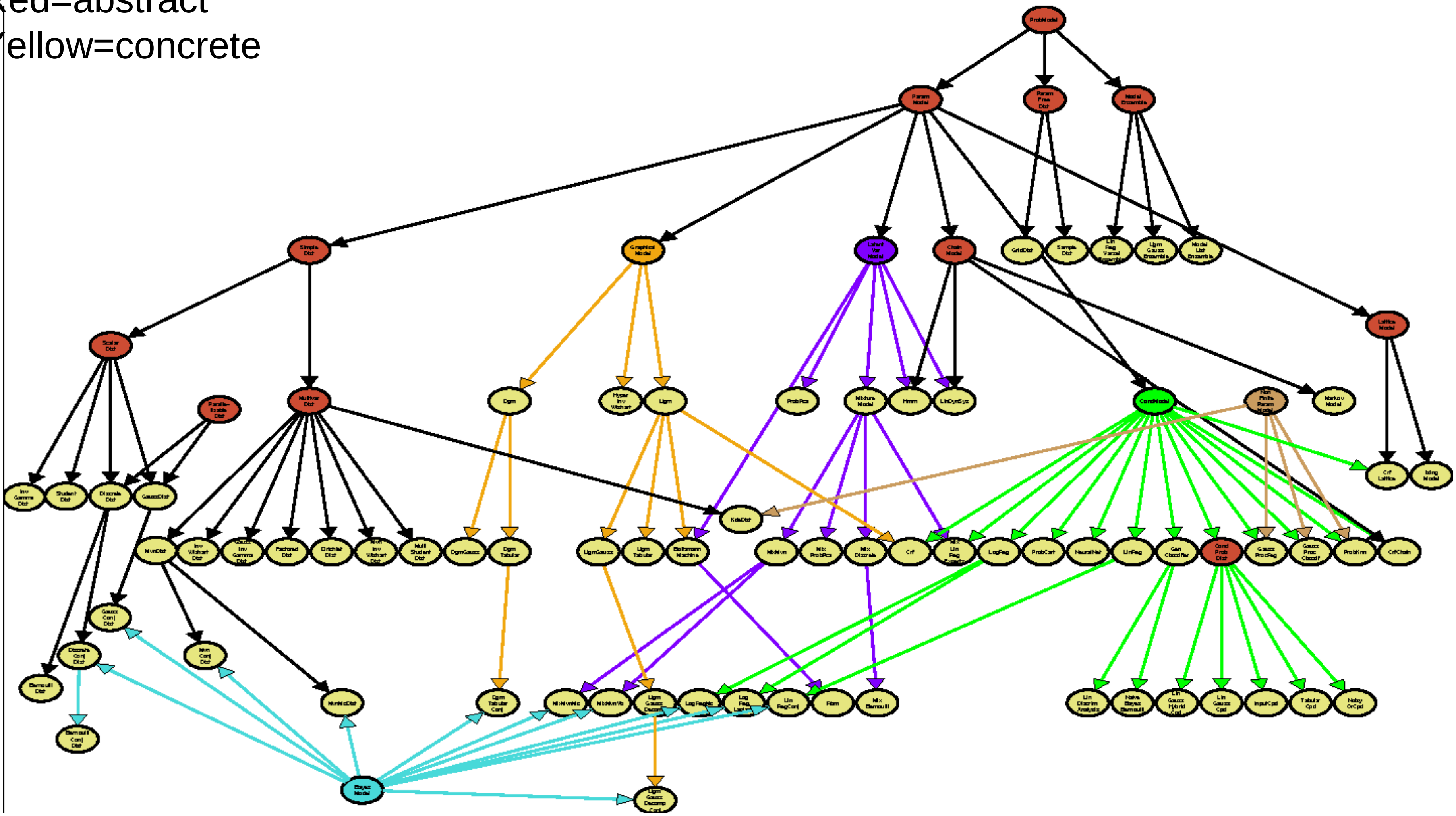
- Separate models from algorithms.
- Models: likelihood + prior + transformer (deterministic function of input, e.g., basis function expansion).
- Algorithms: methods to compute a posterior, or some function of it, such as its mode (MAP estimation), marginal, normalization constant, samples, etc.
- Point estimation (MAP/MLE) is treated as special case of Bayesian inference.
- Emphasis on multivariate models.

Main classes in PMTK

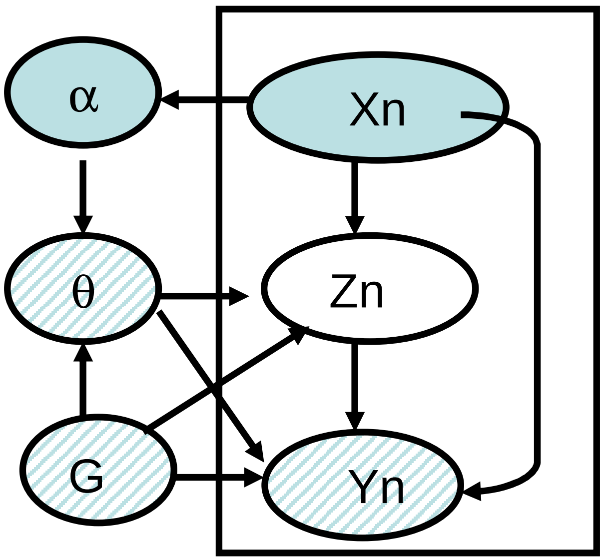
- DataStore
- ProbModel
- Transformer
- FitEng
- ModelSelEng
- InfEng
- Query
- Graph
- Graphlayout
- UnitTest

Models

Red=abstract
Yellow=concrete



General Model Schema



Attributes

-CondProbModel: has Xn

-LatentVarModel: has Zn

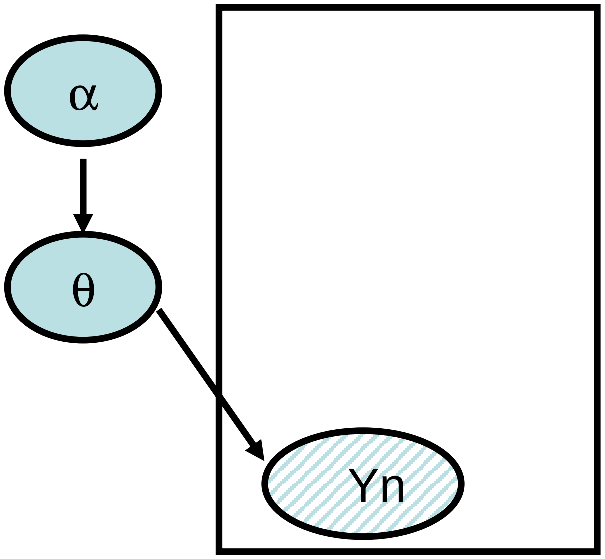
-BayesModel: θ is rv, otherwise const

-NonfiniteParamModel: θ grows with D

-GraphicalModel: G is rv, otherwise const

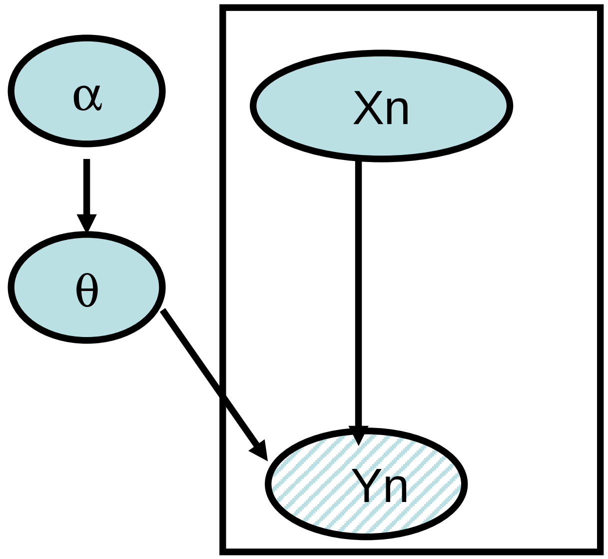
Some model classes

SimpleDist



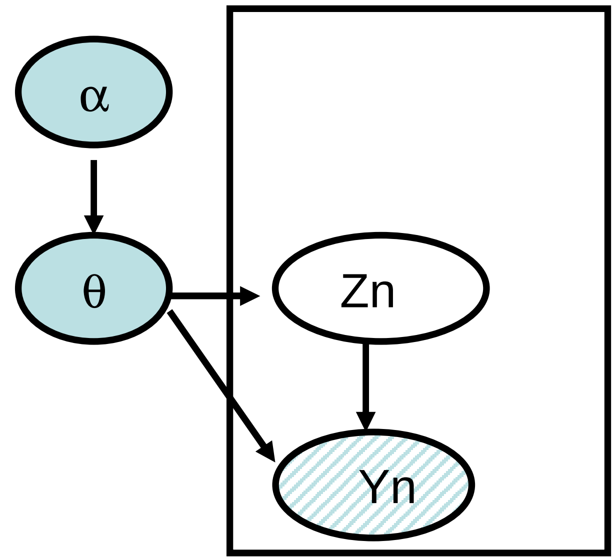
Mvn
Discrete
Dirichlet

CondModel



Linreg
Logreg
NeuralNet

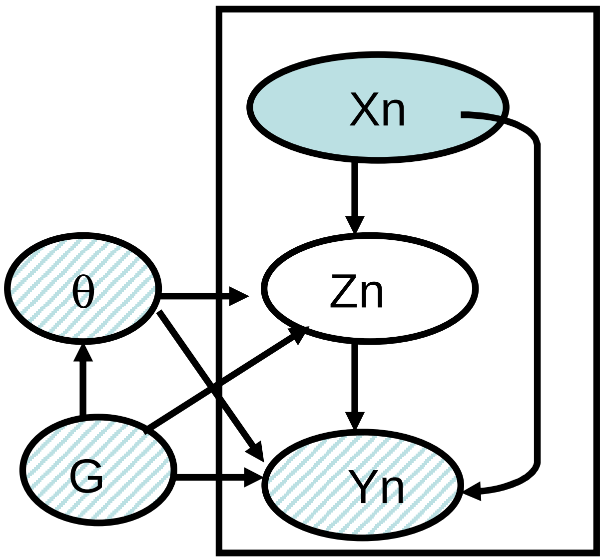
LatentVarModel



MixMvn
Hmm
BoltzmannMachine

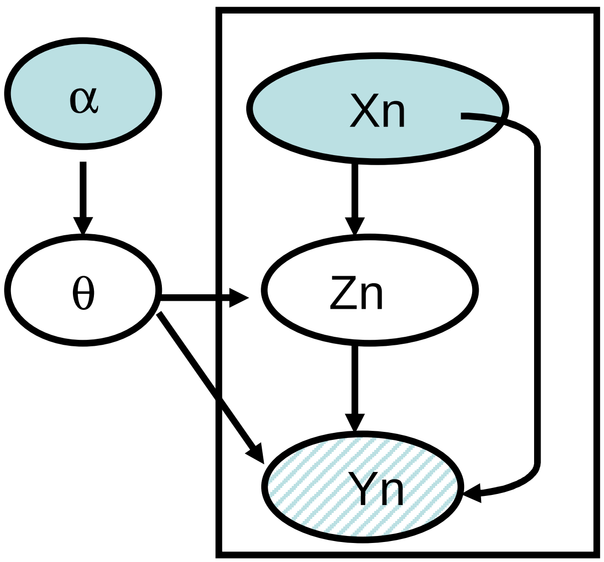
More exotic combinations

Cond+Latent+Graphical



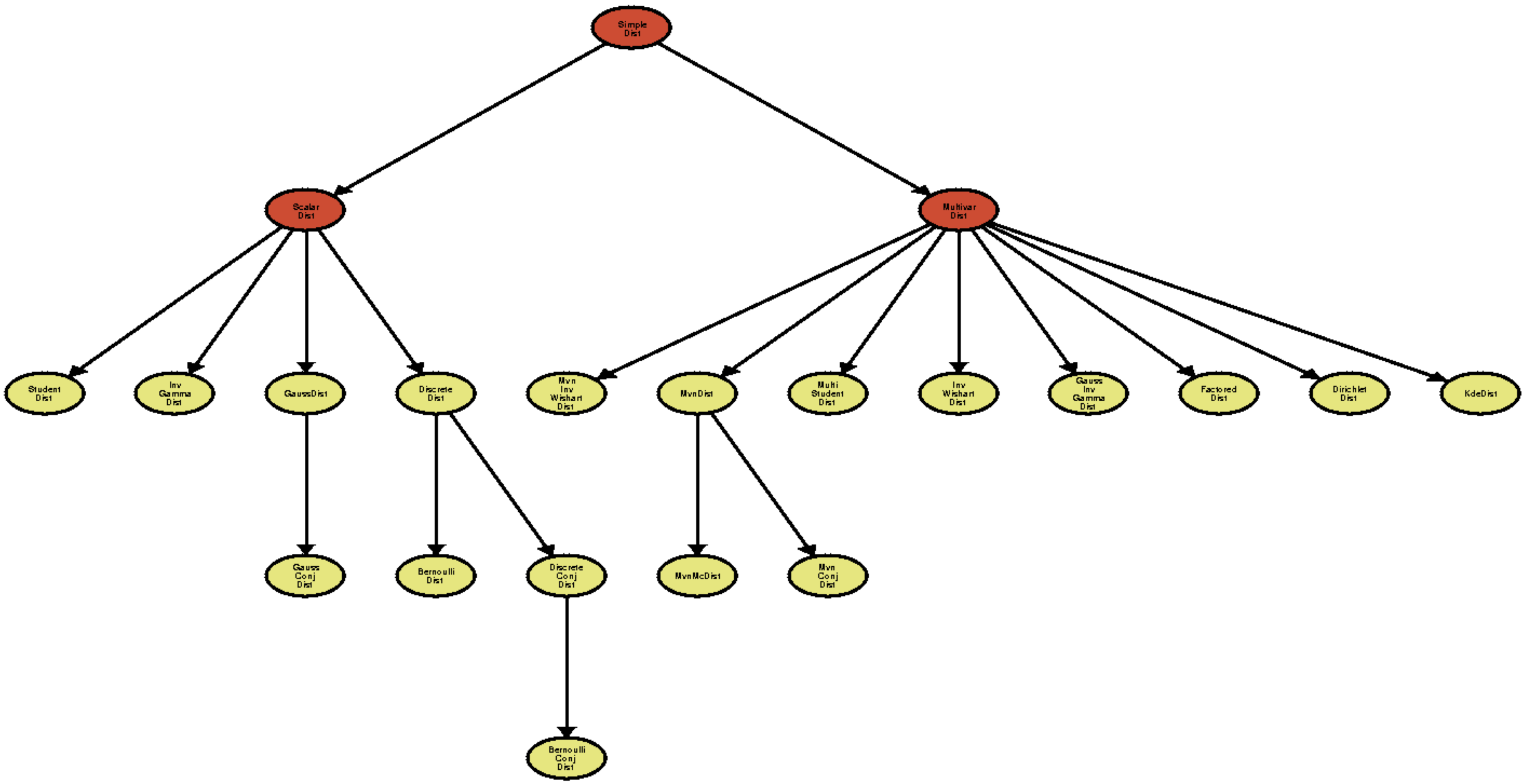
LatentCrf

Cond+Latent+Bayesian



MixLinregExperts

SimpleDist



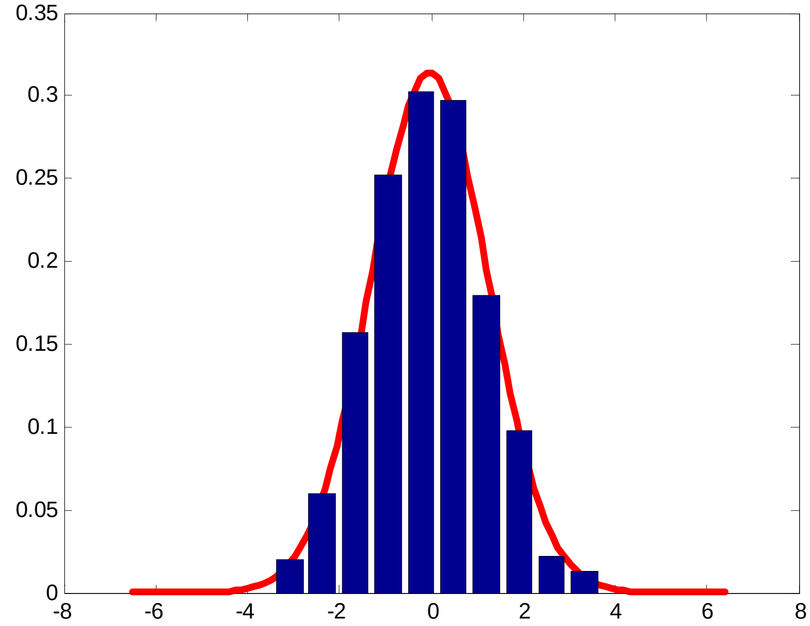
Simple Dist
Scalar Dist
Multivar Dist
Student Dist
Inv Gamma Dist
GaussDist
Discrete Dist
Mvn Inv Wishart Dist
MvnDist
Multi Student Dist
Inv Wishart Dist
Gauss Inv Gamma Dist
Factored Dist
Dirichlet Dist
KdeDist
Gauss Conj Dist
Bernoulli Dist
Discrete Conj Dist
MvnMcDist
Mvn Conj Dist
Bernoulli Conj Dist

Main methods for scalar distributions

- M = fit(M, D)
- out = mean/mode/var/entropy(M)
- [L,U] = credibleInterval(M, [p])
- L = logPdf(M, D),
- plotPdf(M)
- X = sample(M,n)
- d = M.ndimensions

- L = logPrior(M)
- SS = mkSuffStat(M, D)

Fitting a 1d Gaussian by MLE




```
Xtrain = randn(10,1);
Dtrain = DataTable(Xtrain);
M = MvnDist('-ndimensions', 1);
M = fit(M, Dtrain);
LLtrain = sum(logPdf(M, Dtrain))
figure;
h= plotPdf(M); set(h, 'linewidth', 3, 'color', 'r');
X = sample(M, 1000);
hold on
[freq,bins] = hist(X);
binWidth = bins(2)-bins(1);
bar(bins, normalize(freq)/binWidth);
```

Add'l methods for Bayesian models

- pTheta = getParamPost(M): returns p(θ|D,α)
- py = marginalizeOutParams(M): returns (prior/ posterior) predictive distribution

$$p(y|x,\alpha) = \int_\theta \sum_z p(y|x,z\theta)p(z|x,\theta)p(\theta|\alpha)$$

- All other methods (sample, logPdf, mean, mode, etc.) are defined wrt the predictive distribution.
- By contrast, non-Bayesian models use the plugin approximation

$$p(y|x,\theta) = \sum_z p(y|z,x,\theta)p(z|x,\theta)$$

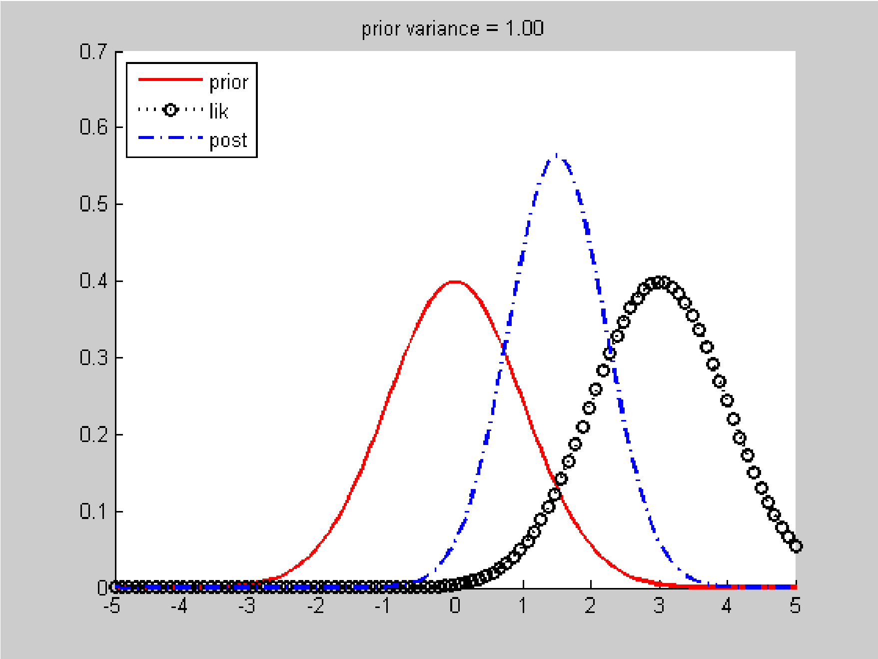
Flavors of Bayesian models

- We have to specify the representation of the posterior $p(\theta|D)$, eg Samples (MC), exact conjugate, approx. factored conjugate (VB).
- Hence we usually get 1 or 2 “flavors” for each base model, e.g., MvnMixMc, MvnMixVb, LogregMc, LogregLaplace. These models can use an inference engine to compute (functions of) the predictive distribution.
- This is orthogonal to *how* we compute $p(\theta|D)$ e.g., we could generate a bag of samples using Gibbs, MH, IS, etc. This is determined by the fitting engine.

Bayesian inference for 1d Gaussian *

Let us assume σ is known.
We use a conjugate prior

$$p(\mu|\alpha) = N(\mu|\mu_0, \sigma_0)$$

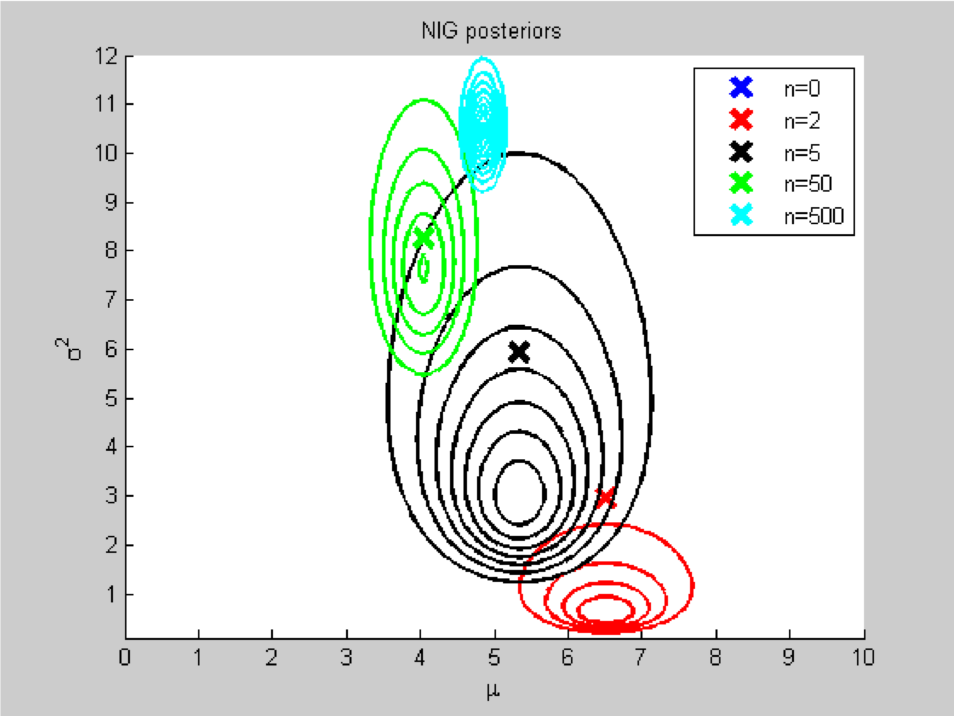



```
prior = MvnDist(0, 5); sigma2 = 1;
M = MvnConjDist('-muPrior', prior,  '-sigma', sigma2);
X = 3;
M = fit(m, DataTable(X));
post = getParamPost(M, 'mu');
figure; hold on
h1 =plotPdf(prior);  h2 = plotPdf(lik); h3 = plotPdf(post);
```

Bayesian inference for 1d Gaussian *

We use a conjugate prior

$$p(\mu, \sigma|\alpha) = N(\mu|\mu_0, k\sigma)IG(\sigma|a, b)$$




```
setSeed(1);
muTrue = 5; varTrue = 10;
X = sample(MvnDist(muTrue, varTrue), 500);
prior = NormInvGammaDist('-mu', 0, '-k', 0.001, '-a', 0.001, '-b', 0.
M = fit(MvnConjDist('-prior', prior), DataTable(X));
plotPdf(getParamPost(M));
```

Add'l methods for multivariate distributions

- pQ = marginal(M, Q)

$$p(Y_Q) = \sum_{Y_H} p(Y_Q, Y_H | \theta)$$

- pQ = infer(M, Q, D) where domain=Q,V,H

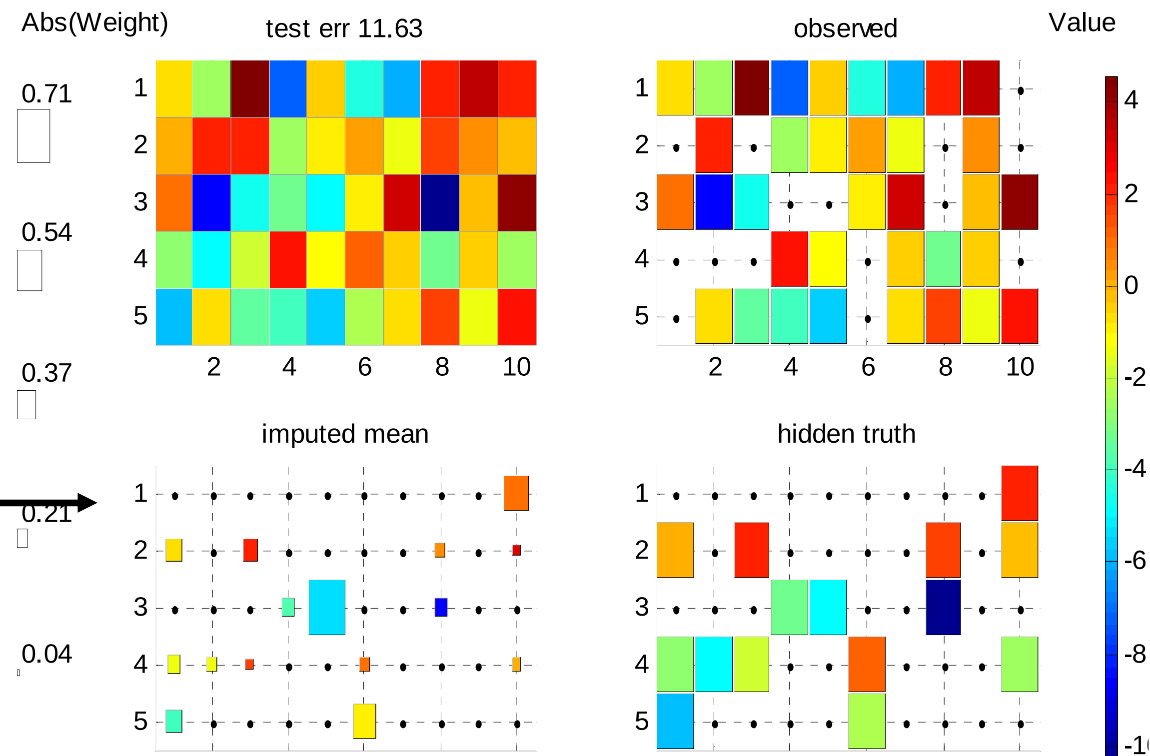
$$p(Y_Q | y_v) \propto \sum_{Y_H} p(Y_Q, Y_H, y_v | \theta)$$

- [X,Y,…] = computeFunPost(M, Q, D, funs)

  Funs can be 'mode', 'var', 'entropy', etc.
  Useful if cannot represent pQ conveniently.

- Query can be: 'joint', 'singles', 'pairs', 'missingJoint', 'missingSingles', int array or cell array of int arrays eg {[1], [1 2]}

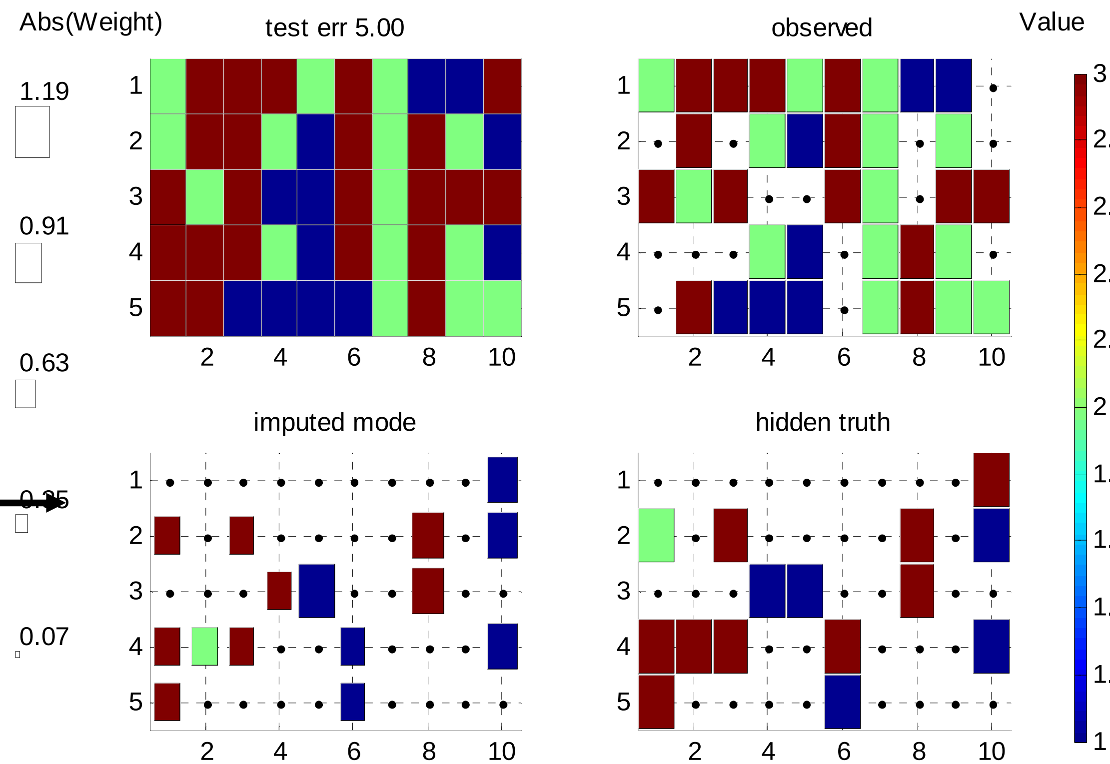
Imputation in an Mvn



Compute mean and var of p(Y10|y1:y9,θ)

```
M = MvnDist('-ndimensions', 10);
XtestMiss(missingTest) = NaN;
model = fit(M, DataTable(XtrainMiss));
Q = Query('missingSingles');
Dtrain = DataTable(XtrainMiss); Dtest  = DataTable(XtestMiss);
XimputeTrain = computeFunPost(model,Q,Dtrain,'mode');
[XimputeTest,Vtest] =  computeFunPost(model,Q,Dtest,{'mode','var'});
conf = (1./Vtest); conf(isinf(conf))=0; mm = max(conf(:));
hintonScale(XimputeTest, conf)
```

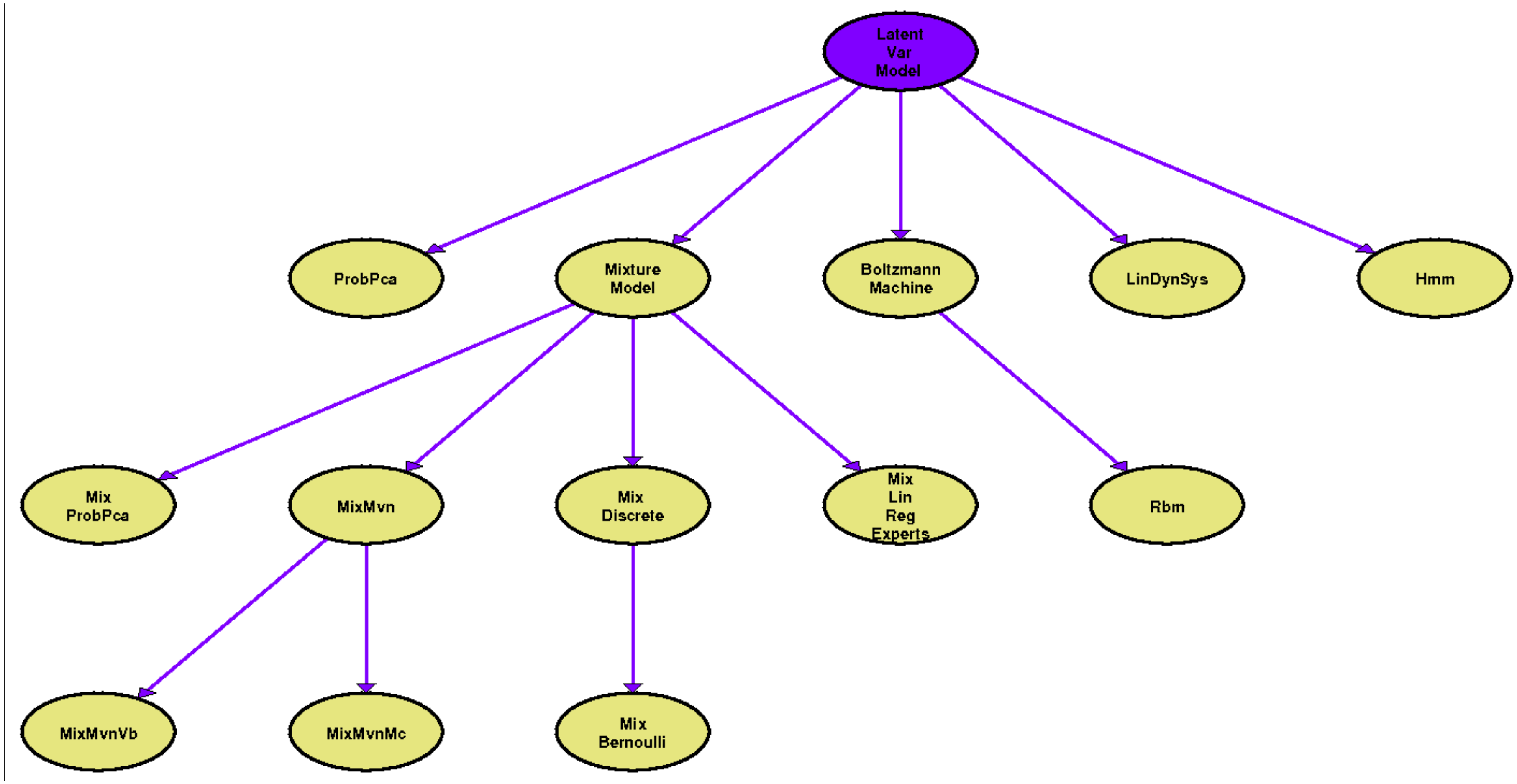
Imputation in a product of multinoullis *



Compute mode and entropy of p(Y10|y1:y9,θ)

```
M = ProdDist(DiscreteDist('-nstates',3), '-ndimensions', 10);
XtestMiss(missingTest) = NaN;
model = fit(M, DataTable(XtrainMiss));
Q = Query('missingSingles');
Dtrain = DataTable(XtrainMiss); Dtest  = DataTable(XtestMiss);
XimputeTrain = computeFunPost(model,Q,Dtrain,'mode');
[XimputeTest,Htest] =  computeFunPost(model,Q,Dtest,{'mode','entr
```

LatentVarModel



Add'l methods for LatentVarModel

- [Zhat, pZ] = inferLatent(M, D)
- This is syntactic sugar for

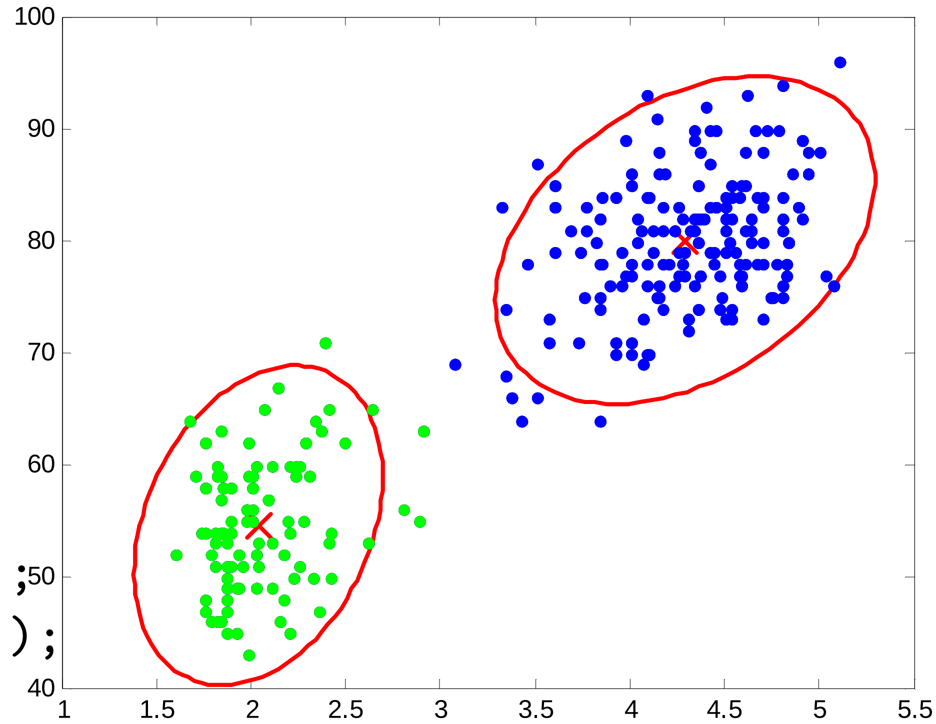
Zhat = computeFunPost(M, Query('latent'), D, 'mode') or 'mean'

pZ = infer(M, Query('latent'), D)

Avoids creating unnecessary objects, or representing posterior joint

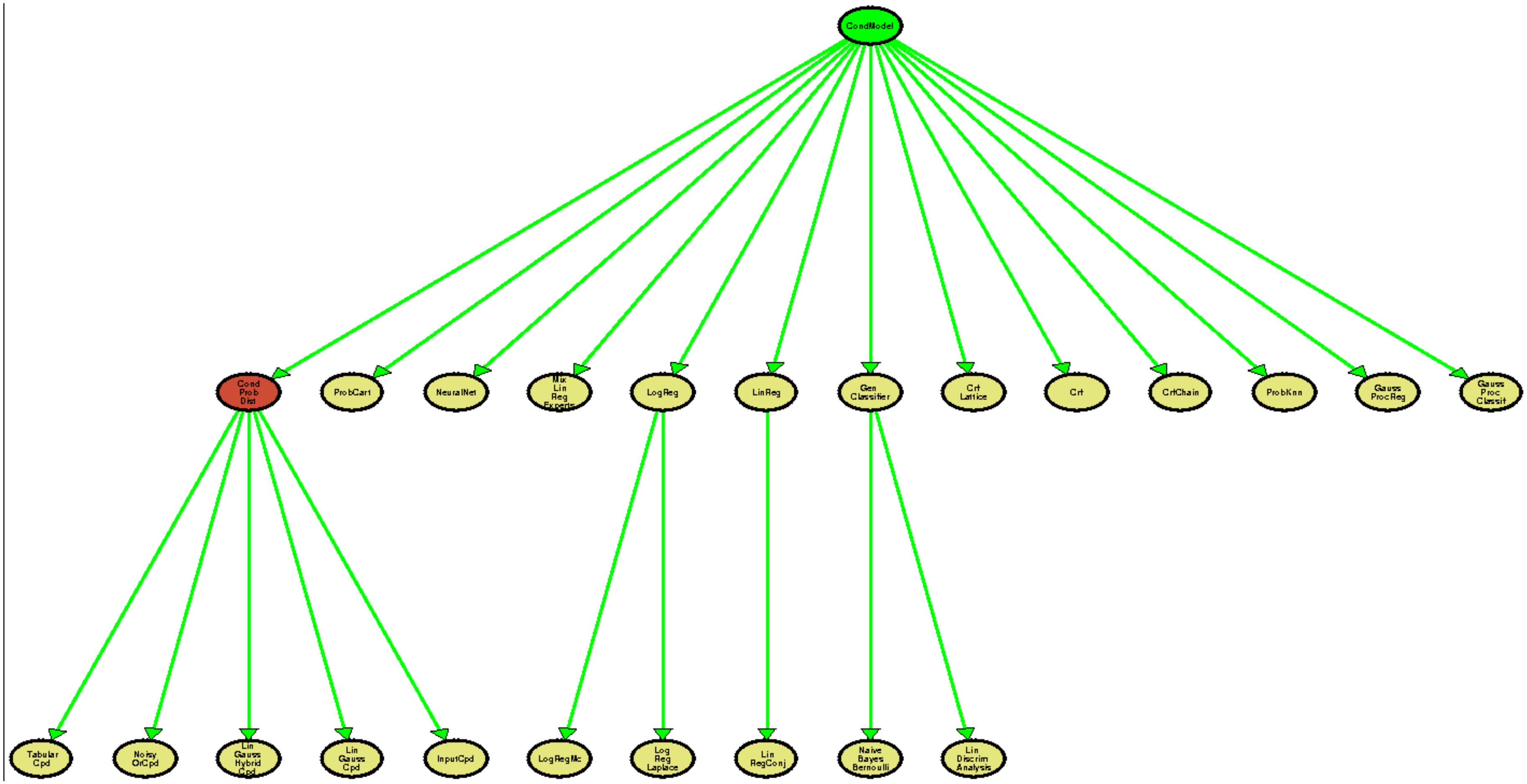
- Mixture models
  - Zhat(n) = most probable latent class (1..K)
  - pZ(n,k) = DiscreteDist(p(Z=k|yn,θ))
- PPCA
  - Zhat(:,n) = E[Z | yn, θ]
  - pZ = Gauss(Zhat, Cov[Z | yn, θ])
- Hmm
  - Zhat = Viterbi path (mode of joint)
  - pZ(k,t,n) = DiscreteDist(p(Zt=k|yn, θ)) % one-slice marginals

Mixture of 2d Gaussians




```
load oldFaith;
D = DataTable(X);
m  = MixMvn('-nmixtures',2);
%m.fitEng.verbose = true;
m = fit(m,D);
[Zhat, post] = inferLatent(m,D);
assertIsequal(Zhat,  mode(post));
hold on;
colors = {'g', 'b'};
for c=1:2
  plotPdf(m.mixtureComps{c});
  ndx = find((Zhat==c));
  plot(X(ndx,1),X(ndx,2), sprintf('%s.', colors{c}));
end
```

CondModel



CondModel
Cond Prob Dist
ProbCart
NeuralNet
Mix Lin Reg Experts
LogReg
LinReg
Gen Classifier
Crf Lattice
Crf
CrfChain
ProbKnn
Gauss ProcReg
Gauss Proc Classif
Tabular Cpd
Noisy OrCpd
Lin Gauss Hybrid Cpd
Lin Gauss Cpd
InputCpd
LogRegMc
Log Reg Laplace
Lin RegConj
Naive Bayes Bernoulli
Lin Discrim Analysis

Add'l methods for conditional models

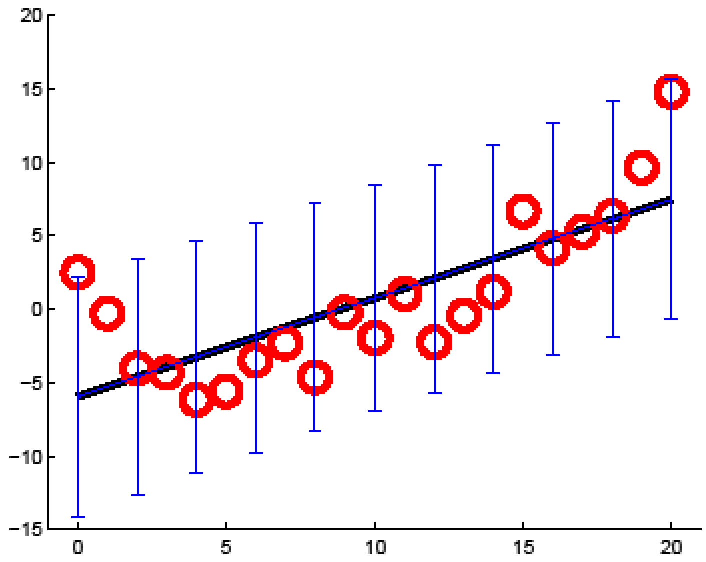
- [Yhat, pY] = inferOutput(M, D)
- This is syntactic sugar for

[Yhat, pY] = computeFunPost(M, Query('output'), D, 'mode') or 'mean'

pY = infer(M, Query('output'), D)

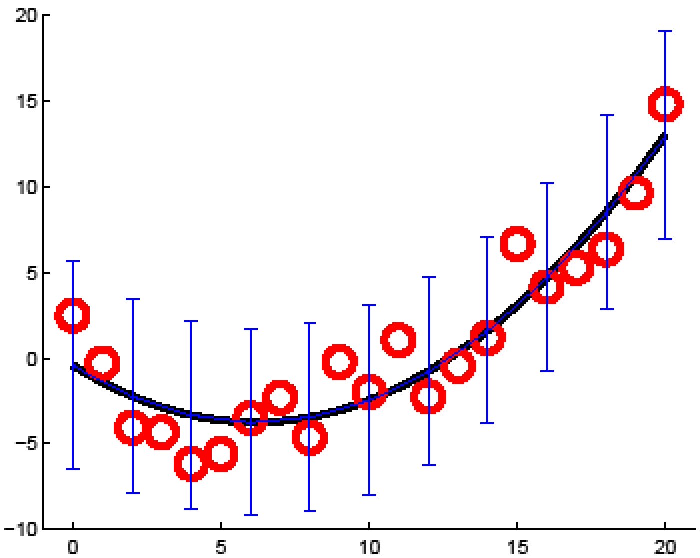
- Examples
- Linreg
  - yhat(n) = E[Y|xn, θ]
  - Py = GaussDist(yhat(n), Var[Y|xn, θ])
- Logreg
  - Yhat = mode[Y | xn, θ]
  - pY(n,k) = DiscreteDist(p(Y=k| xn, θ))

Vanilla linear regression *



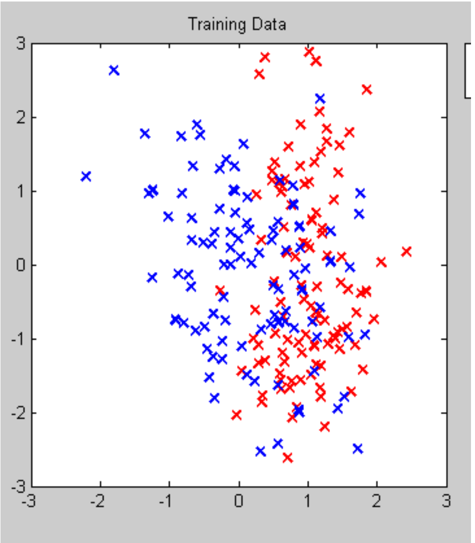

```
model = Linreg;
model = fit(model, DataTableXY(xtrain,ytrain));
[mu, py] = inferOutput(model, DataTable(xtest));
errorbar(xtest, mu, sqrt(var(py)));
```

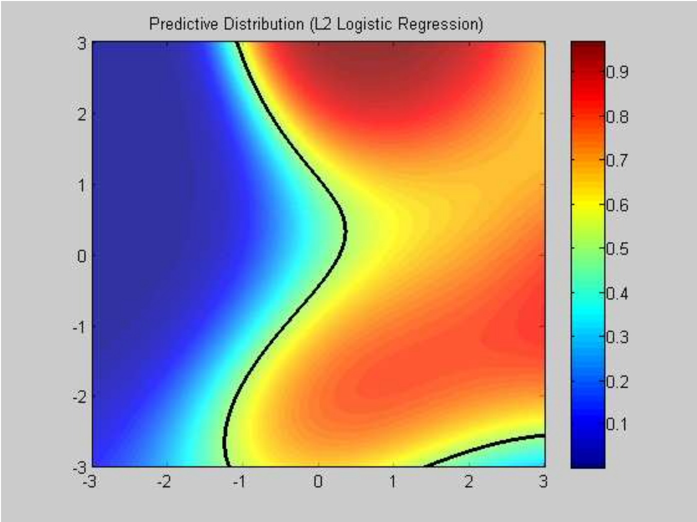
Polynomial linear regression *

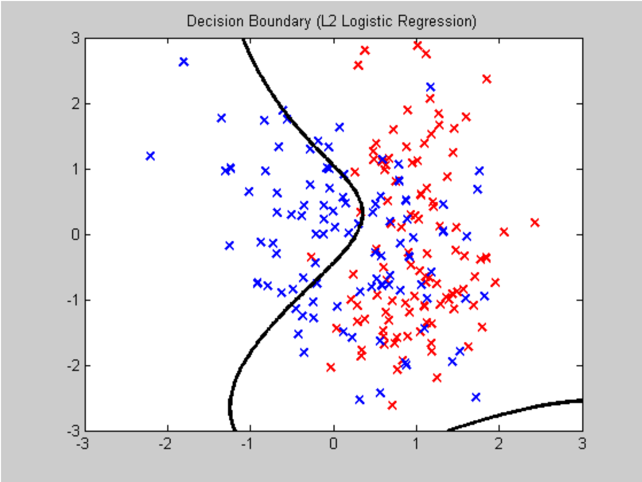



```
T =  ChainTransformer({RescaleTransformer, ..
   PolyBasisTransformer(2)});
model = Linreg('-transformer', T);
...
```

Kernel logistic regression

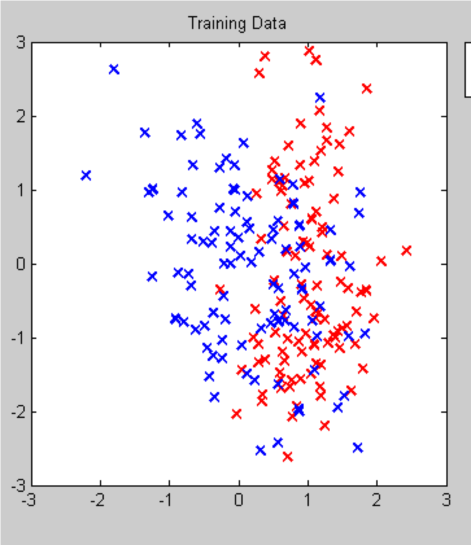


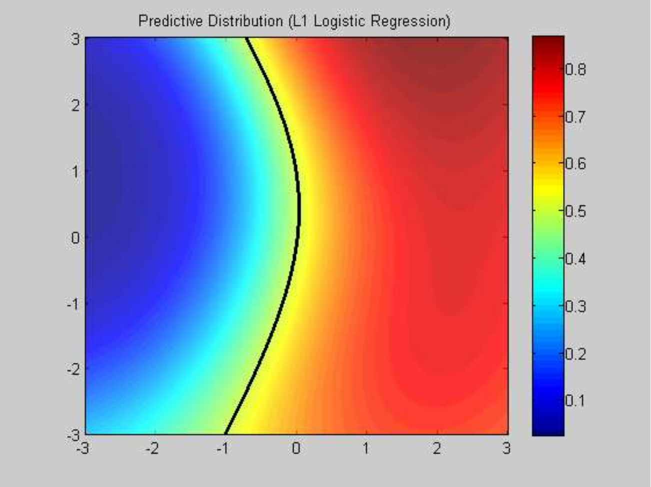


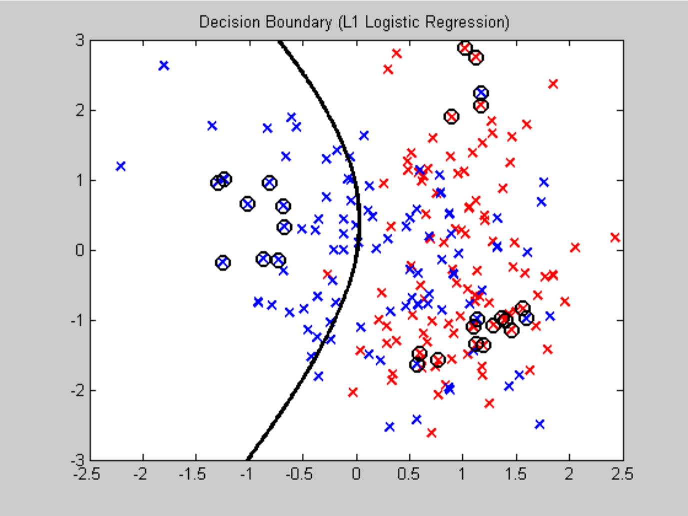



```
T = ChainTransformer({StandardizeTransformer,...
    KernelTransformer('-rbf',sigma)} );
model = Logreg('-nclasses',2, '-transformer', T,...
     '-prior', 'l2', '-lambda', lambda);
model.fitEng.optMethod =  'lbfgs';
model = fit(model, DataTable(xtrain, ytrain));
[X1grid, X2grid] = meshgrid(-3:0.02:3,-3:0.02:3);
[yhat, py] = inferOutput(model,DataTable([X1grid(:),X2gric
pgrid = reshape(py(:,1),nr,nc); surf(pgrid);
```

Sparse Kernel logreg ("RVM")



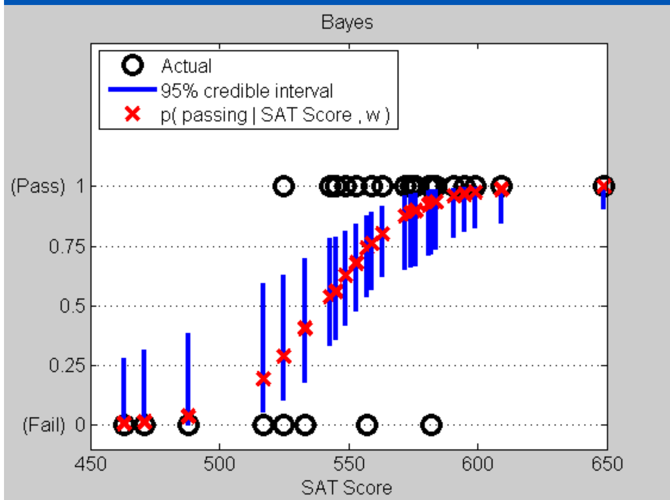


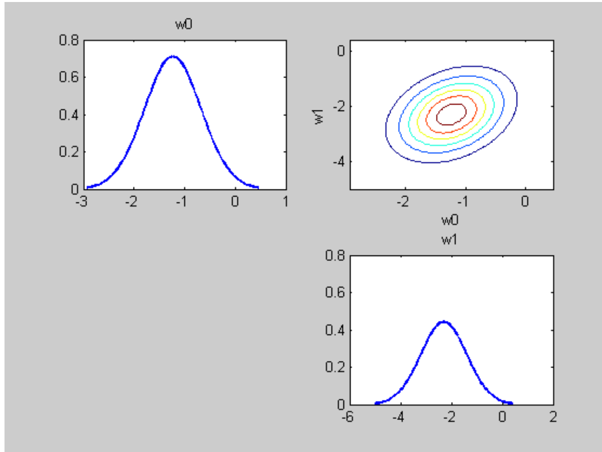


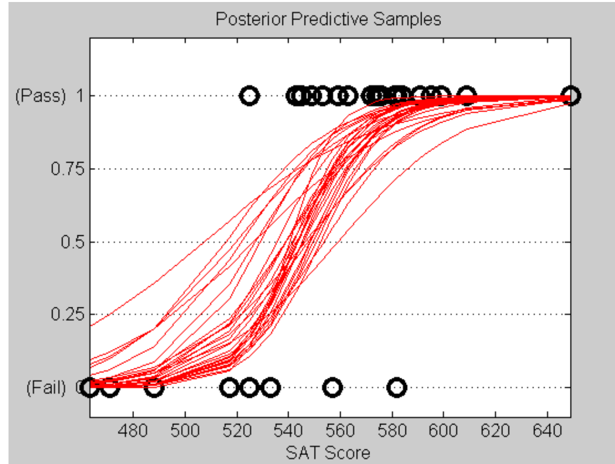

```
...
model = Logreg('-prior','l1','-lambda',lambda,
    '-transformer', T);
model.fitEng.optMethod = 'projection';
...
```

There is 1 feature for each training example (design matrix is NxN). Hence being sparse in the features/ weights is equivalent to selecting a subset of the examples ("relevance vectors").

Bayesian logistic regression in 1D *

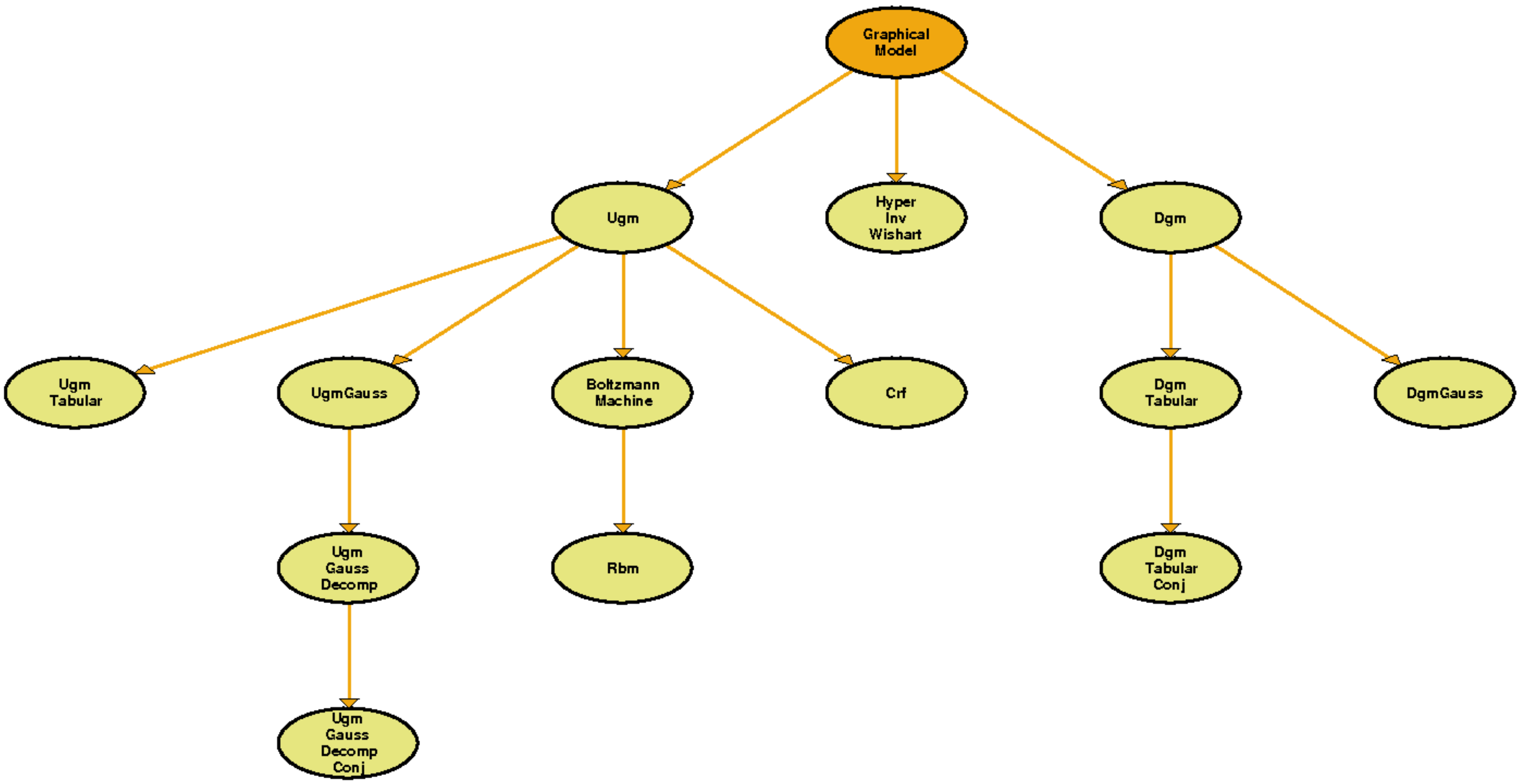






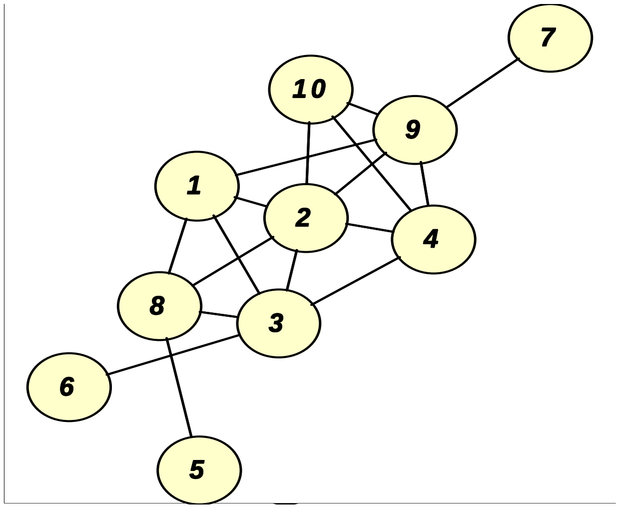

```
T = standardizeTransformer;
M = LogregLaplace('-nclasses',2,'-transformer',T,...
   '-prior','l2', 'lambda', 0.1);
M = fit(M, DataTableXY(X,y));
plotPdf(getParamPost(M, 'mu'));
[yhat, pred] = inferOutput(model,DataTable(X));
[Q5,Q95] = credibleInterval(pred);
med = median(pred); figure; hold on
plot(X, y, 'ko', 'linewidth', 3, 'markersize', 12);
for i=1:length(y)
    line([X(i) X(i)], [Q5(i) Q95(i)],   'linewidth', 3)
    plot(X(i), med(i), 'rx', 'linewidth', 3, 'markersiz
end
```

GraphicalModel



Add'l methods for Graphical Models

- M = fitStructure(M, D): returns MAP estimate of p(G|D)
- plotStructure(M): calls Graphlayout, which calls graphviz, but lets user subsequently interactively edit the layout in Matlab




```
M = UgmGauss;
M = fitStructure(M, DataTable(X)); plotStructure(M)
```

Model selection

- Examples: picking graph structure, number mixture components, L1/L2 regularizer
- Interface:
- M = UgmGauss('-G', UndirGraph.mkAllGraphs(5))
- M = UgmGauss('-ndimensions', 5)
- M = MixMvn('-ndimensions', 5, '-nmix', 1:10)
- M = Linreg('-prior', 'L2', '-lambda', Linreg.autoLambda('L2', D, 100))
- Internally, M=fit(M,D) calls a model selection engine, which implements a search method and a score method (eg BIC, CV)

Model ensemble

- This is a Bayesian version of model selection: it maintains a posterior distribution over a finite set of models
- Internally it uses a model selection engine to fit all the models.
- Subsequent calls to logPdf, sample, infer etc. are computed using Bayes Model Averaging

$$p(y|\mathcal{M}) = \sum_{m \in \mathcal{M}} p(y|m)p(m)$$

- Each sub-model may use a plug-in θ, or may integrate it out

Outline

- Why?
- What? (Models)
- How? (Algorithms)

Parameter estimation

- Every (parametric) model has a fit method. For non-Bayesian models, this solves the optimization problem

$$\max_\theta \left[\sum_{n=1}^N \log p(y_n|x_n,\theta)\right] + \log p(\theta)$$

$$p(y_n|x_n,\theta) = \sum_{z_n} p(y_n|z_n,x_n,\theta)p(z_n|x_n,\theta)$$

- This can either be implemented by the model (often by calling other people’s code), or by a fitEngine contained within the model.
- FitEngines are useful when we can factor out common code.

Fitting LatentVarModel

- If we have missing data, we use EmFitEng.
- The EM engine is abstract; model-specific subclasses implement the E and M steps, and initialization and optional plotting
- Subclasses:
  - MvnMissingEmFitEng
  - MixModelEmFitEng
  - HmmEmFitEng
- Subclasses of MixModelEmFitEng: MixMvnEmFitEng, MixDiscreteEmFitEng

Fitting CondModel

- Linreg+L2:
  - ‘QR’
  - ‘RLS’: Recursive least squares (Widrow-Hoff)
- Logreg+L2: several internal methods
  - ‘Minfunc’: we use Mark Schmidt’s minFunc, which supports many methods (LBFGS, CG).
  - ‘StochGrad’
  - ‘Perceptron’
- Linreg+L1:
  - ‘LassoShooting’ (Mark)
  - ‘L1LS’ (Boyd et al)
- Logreg+L1: Mark’s L1general code.
- CRFs: Mark’s code?

Fitting GraphicalModel

- UgmGauss:
  - Mark’s covsel
  - HastieTibshiraniFriedman algorithm
- UgmTabular
  - Mark’s pseudo-likelihood?
- Dgm: fit each CPD
- Missing data: use EM (needs inference)

Fitting Bayesian models

- Now fit must return a posterior
- Algorithm and model are conflated.
- MvnConjDist, DiscreteConjDist, LinRegConj: analytic results for conjugate priors
- LogRegMc: MH or IS
- LogRegLaplace: calls optimizer first
- MixMvnGibbs: Gibbs or collapsed Gibbs (Cody Severinski's code)
- MixMvnVb: Emtiyaz Khan's implementation of variational Bayes (as described in Bishop)

Model selection for LatentVarModel

- MixModel: grid search over K, plus selection based on BIC or CV
- May add support for VB later

Model selection for CondModel

- Linreg+L2: SVD
- Logreg+L2: grid search plus warm-start
- Linreg+L1: Lars (Karl Skoglund's implementation)
- Logreg+L1: grid search plus warm-start

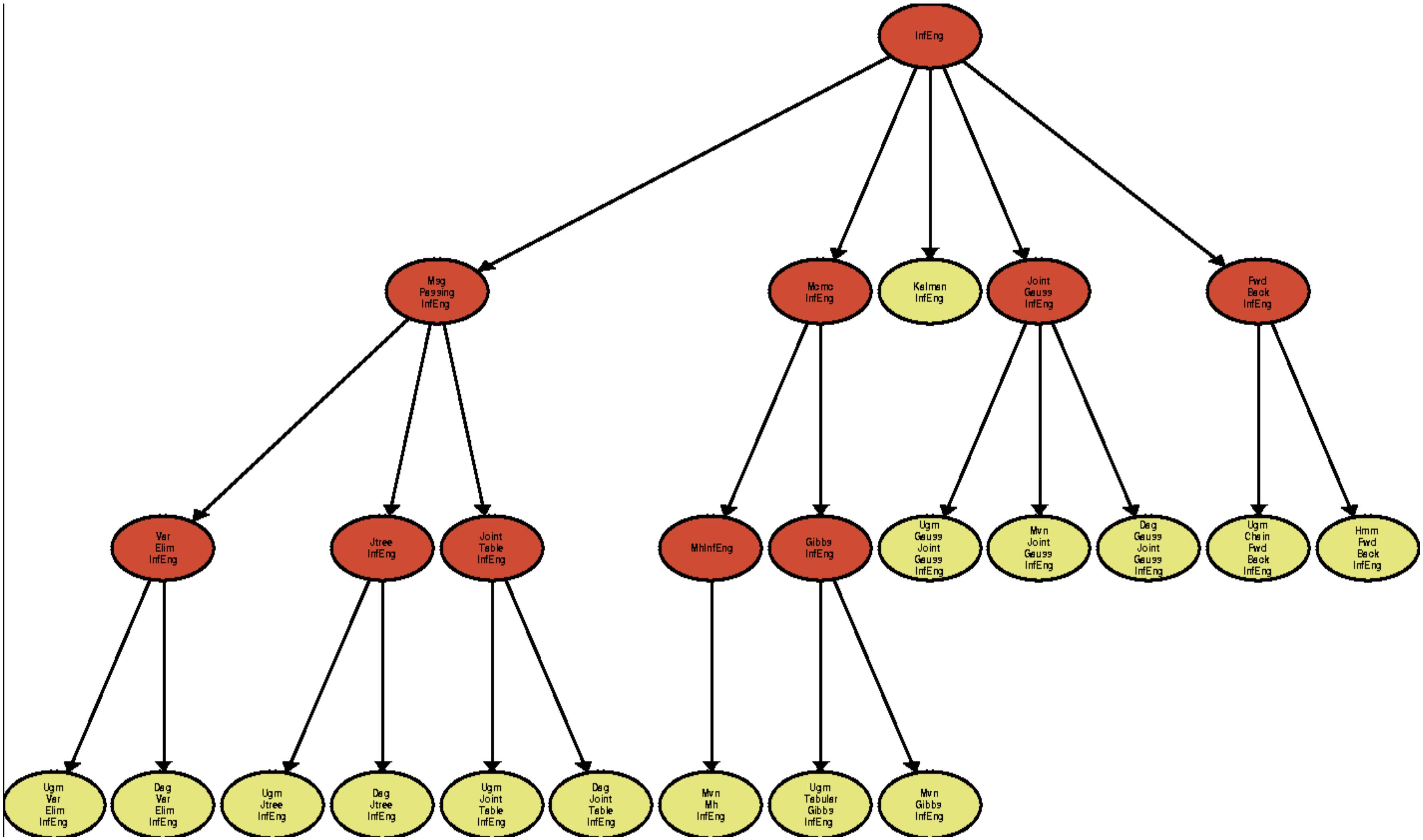
Model selection for GraphicalModel

- UgmGauss: many methods, work in progress
- DgmTabular:
  - ‘dp’: dynamic programming (Dan Eaton’s implementation of Silander06)
  - ‘sls’: hill climbing with random restarts (Mark’s code?)

Inference engines

- Many models support an infer* method.
- For certain graphical models (with fixed params, and discrete or Gaussian latent state spaces), we can implement generic algorithms, based on varelim or message passing (on graph or Jtree)
- An InfEng is an abstract class that defines
  - eng = enterEvidence(eng, M, D)
  - pQ = computeMarginal(eng, Q)
  - logZ = computeLogZ(eng)
- Subclasses: Jtree, Varelim, JointTable, JointGauss, FwdBack, Mcmc
- Model-specific subclasses specify how to create discrete or Gaussian factors
- Code mostly derived from BNT

Inference engines



Meta-tools

- viewClassTree: Graphlayout of class hierarchy
- methodReport: methods x models html table
- makeAuthorReport: searches for %#author tags, makes webpage listing external functions and their source
- runUnitTests: runs all tests and makes a summary table
- makeTestPMTK: call all examples with %#testPMTK tag
- makeRunDemos: script to call all examples
- publishExamples: call all examples and generate web page of their source code and output
- sendEmailToPmtkUsers (list from web download log)
- compilePMTKmex: searches for %#PMTKmex tag, then compiles C code using mex
- compileAndRun: uses embedded matlab compiler (emlmex) to generate fixed-sized code, then run it (requires %#eml)

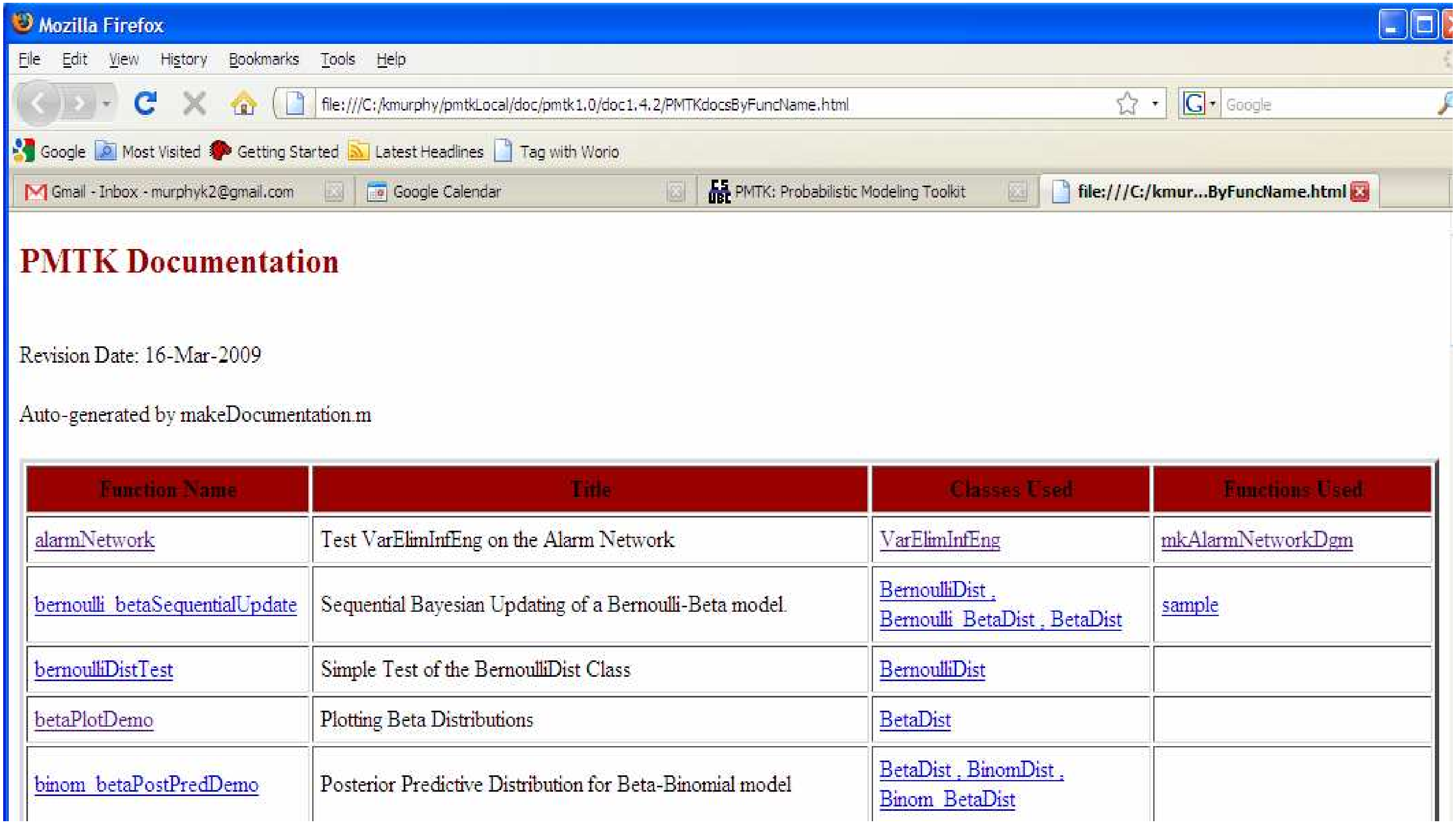
Methods x Models

	Simple Dist	Graphical Model	Param Free Dist	Factored Dist	Param Model	Prob Model	MvnDist	Cond Model	Latent Var Model	Bayes Model	Discrete Dist	Multivar Dist	Model Ensemble
copy	CI	CI	CI	CI	CI	CN	CI	CI	CI		CI	CI	CI
logPdf	AI	AI	AI	CO*	AI	AN	CO	AI	AI		CO	AI	AI
sample	AI	AI	AI	CO*	AI	AN	CO	AI	AI		CO	AI	AI
fit	AI	AI		CO*	AN		CO	AI	AI		CO	AI	AN
logPrior	CI	CI		CI	CN		CI	CI	CI		CO	CI	
mkSuffStat	CI	CI		CI	CN		CO	CI	CI		CO	CI	
mean	AN		AN	CO*			CO				CO	AI	
plotPdf	AN		AN	CO*			CO*				CO	AI	
var	AN		AN	CO*			CO				CO	AI	
entropy	AN			CO*			CO				CO	AI	
mode	AN			CO*			CO				CO	AI	
cov			AN	CO*			CO					AN	
infer		AN		CN*			CN						
computeMap		AN		CN*									
inferMissing		AN		CN*									
computeFunPost							CN						
computeMapLatent									AN				
computeMapOutput								AN					
getParamPost										AN			
inferLatent									AN				
inferOutput								AN					
logMargLik										AN			
plotTopology		AN											
pmf											CN		

Code	Description
A	Abstract
C	Concrete
N	Introduces
O	Overrides
I	Inherits
*	Unfinished

Created with methodReport

List of examples



PMTK Documentation

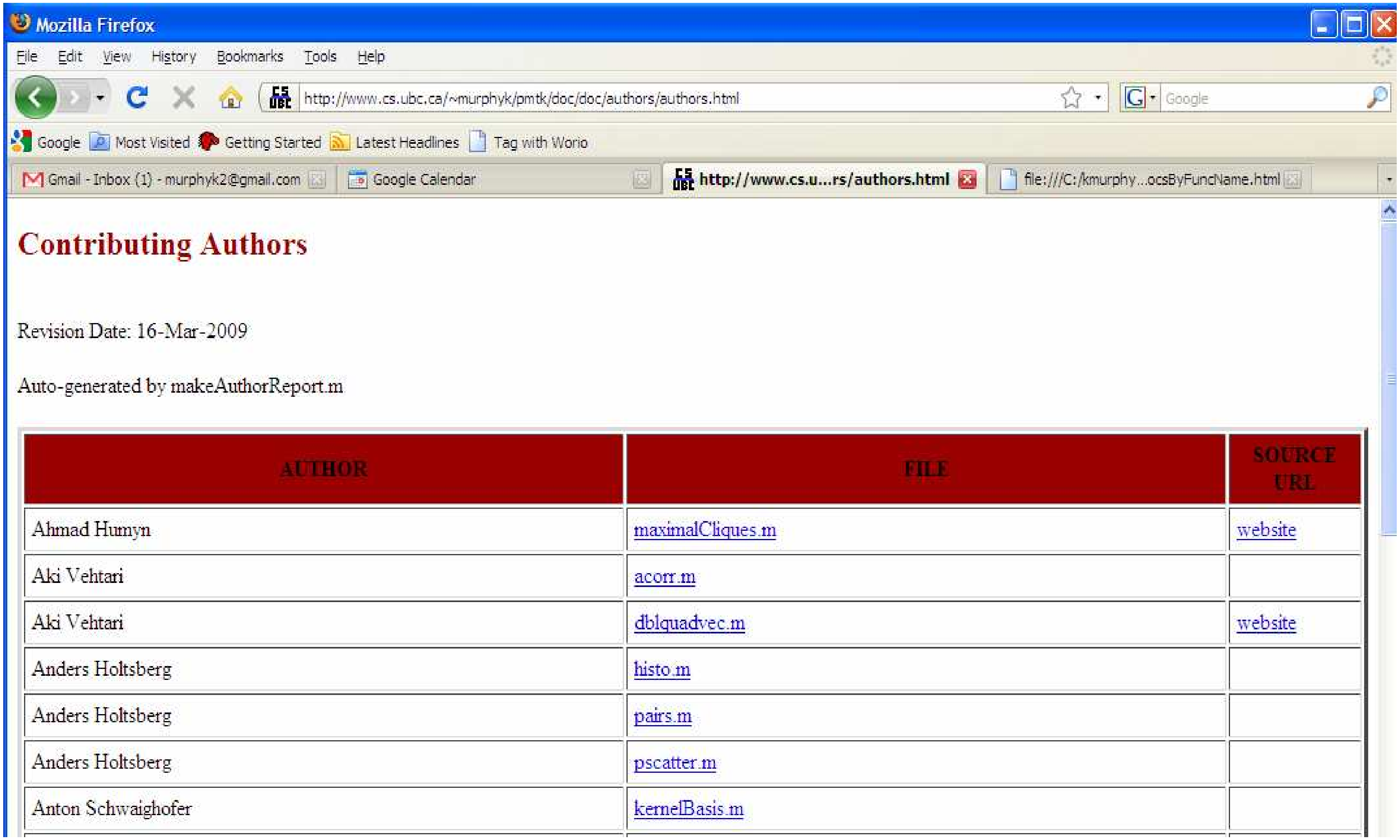
Revision Date: 16-Mar-2009

Auto-generated by makeDocumentation.m

Function Name	Title	Classes Used	Functions Used
alarmNetwork	Test VarElimInfEng on the Alarm Network	VarElimInfEng	mkAlarmNetworkDgm
bernoulli_betaSequentialUpdate	Sequential Bayesian Updating of a Bernoulli-Beta model.	BernoulliDist , Bernoulli_BetaDist , BetaDist	sample
bernoulliDistTest	Simple Test of the BernoulliDist Class	BernoulliDist	
betaPlotDemo	Plotting Beta Distributions	BetaDist	
binom_betaPostPredDemo	Posterior Predictive Distribution for Beta-Binomial model	BetaDist , BinomDist , Binom_BetaDist	

Created with publishExamples

Contributing authors



Contributing Authors

Revision Date: 16-Mar-2009

Auto-generated by makeAuthorReport.m

AUTHOR	FILE	SOURCE URL
Ahmad Humyn	maximalCliques.m	website
Aki Vehtari	acorr.m	
Aki Vehtari	dblquadvec.m	website
Anders Holtsberg	histo.m	
Anders Holtsberg	pairs.m	
Anders Holtsberg	pscatter.m	
Anton Schwaighofer	kernelBasis.m	

Made with makeAuthorReport

Conclusions

- PMTK strives to strike the right balance between simplicity, generality and efficiency.
- It combines elements from ML, GM and Bayesian communities.
- It provides a unified conceptual framework to data modeling, which is particularly useful for teaching.
- The source code is on pmtk.googlecode.com
- Email me if you want to use and/or develop it.