pomegranate

fast and flexible probabilistic modelling in python

Jacob Schreiber
Paul G. Allen School of Computer Science
University of Washington

jmschreiber91
@jmschrei
@jmschreiber91

PLATO

FIAT SLUG

UNIVERSITY OF CALIFORNIA SANTA CRUZ

UNIVERSITY *of* WASHINGTON

eScience Institute

ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS

Inria

INVENTORS FOR THE DIGITAL WORLD

PARIETAL

aspentech

Overview

pomegranate is more flexible than other packages, faster, is intuitive to use, and can do it all in parallel

Probability
Distributions

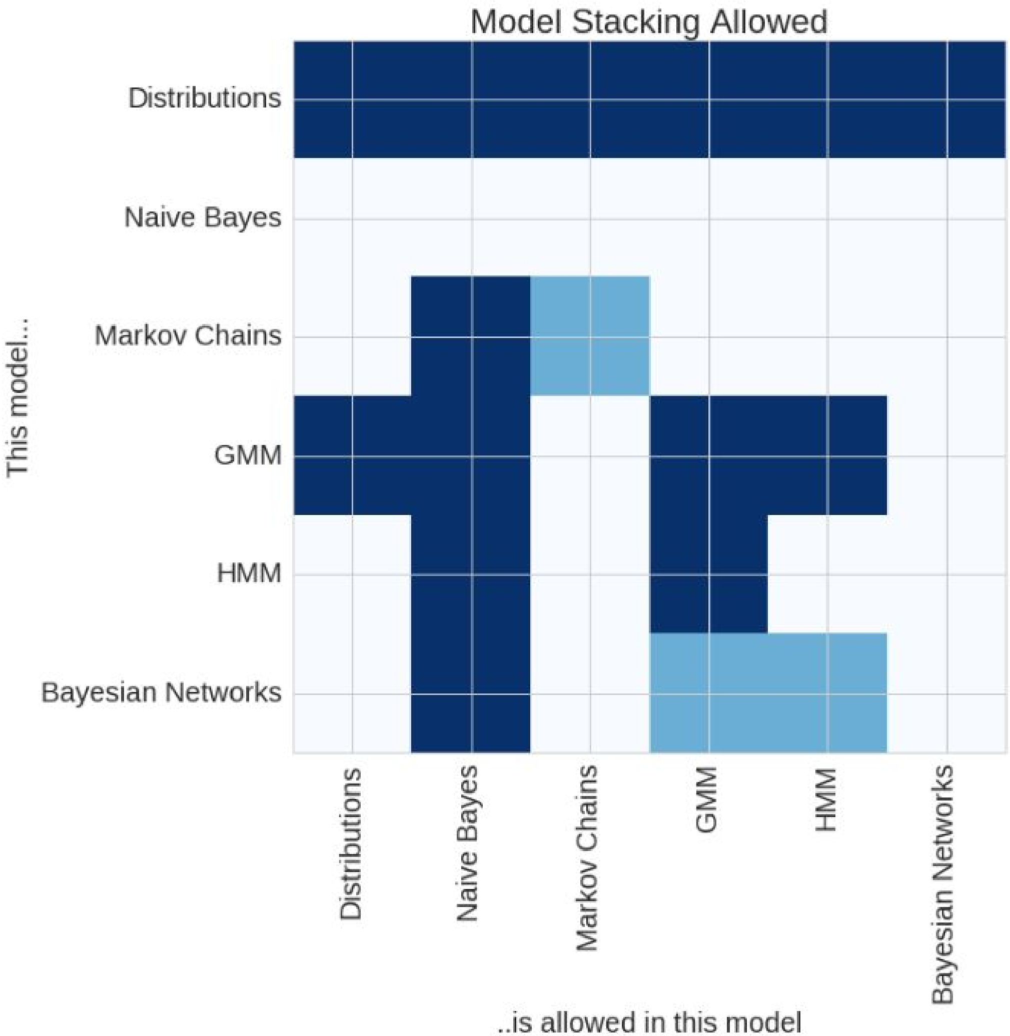
Bayes Classifiers

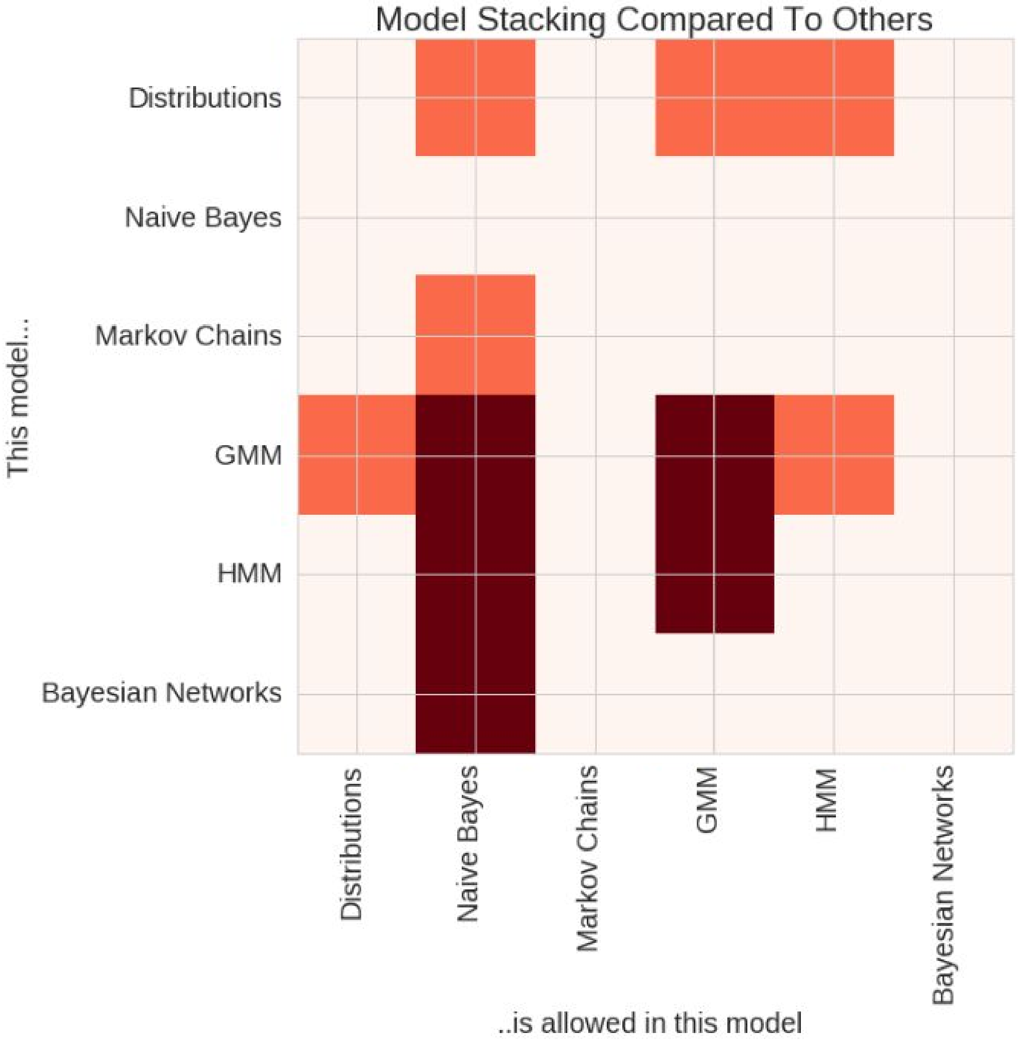
Bayesian
Networks

Markov Chains

Hidden Markov
Models

General Mixture
Models


Model Stacking Allowed
This model...
Distributions
Naive Bayes
Markov Chains
GMM
HMM
Bayesian Networks
..is allowed in this model
Distributions
Naive Bayes
Markov Chains
GMM
HMM
Bayesian Networks



Model Stacking Compared To Others

Overview

The API

Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

Finale: Train a mixture of hidden markov models in parallel

All models share most methods

model.log_probability(X) / model.probability(X)

model.sample()

model.fit(X, weights, inertia)

model.summarize(X, weights)

model.from_summaries(inertia)

All models have these methods!

model.predict(X)

model.predict_proba(X)

model.predict_log_proba(X)

All models composed of distributions (like GMM, HMM...) have these methods too!

Model.from_samples(X, weights)

All models except HMMs have this (coming soon!)

All models share most methods

model.log_probability(X) / model.probability(X)

model.sample()

model.fit(X, weights, inertia)

model.summarize(X, weights)

model.from_summaries(inertia)

All models have these methods!

model.predict(X)

model.predict_proba(X)

model.predict_log_proba(X)

All models composed of distributions (like GMM, HMM...) have these methods too!

Model.from_samples(X, weights)

All models except HMMs have this (coming soon!)

pomegranate supports many distributions

Univariate Distributions

1. UniformDistribution
2. BernoulliDistribution
3. NormalDistribution
4. LogNormalDistribution
5. ExponentialDistribution
6. BetaDistribution
7. GammaDistribution
8. DiscreteDistribution
9. PoissonDistribution

Kernel Densities

1. GaussianKernelDensity
2. UniformKernelDensity
3. TriangleKernelDensity

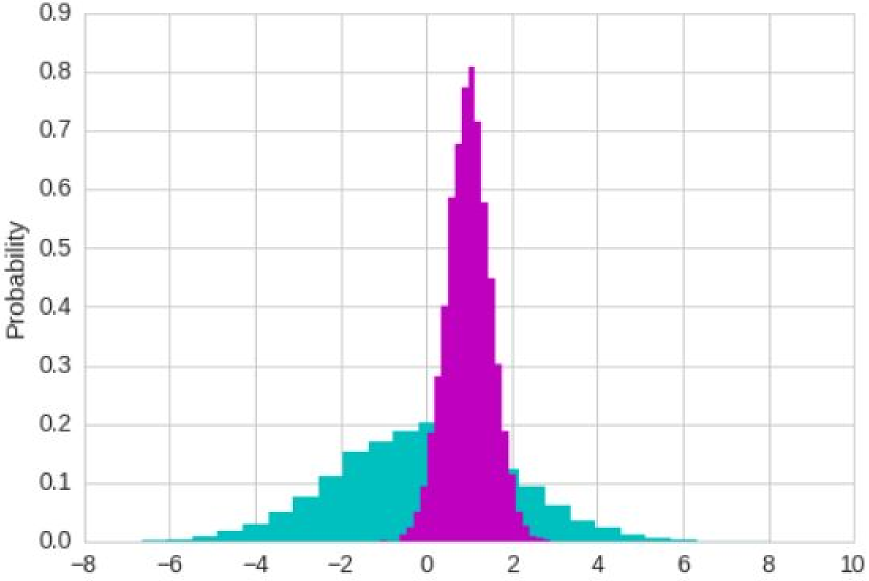
Multivariate Distributions

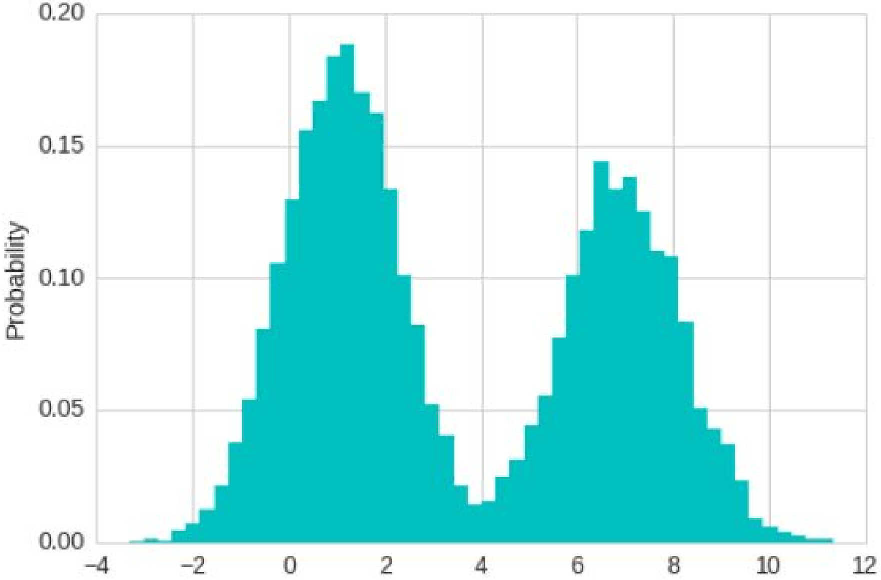
1. IndependentComponentsDistribution
2. MultivariateGaussianDistribution
3. DirichletDistribution
4. ConditionalProbabilityTable
5. JointProbabilityTable

Models can be created from known values

```
mu, sig = 0, 2
a = NormalDistribution(mu, sig)
```

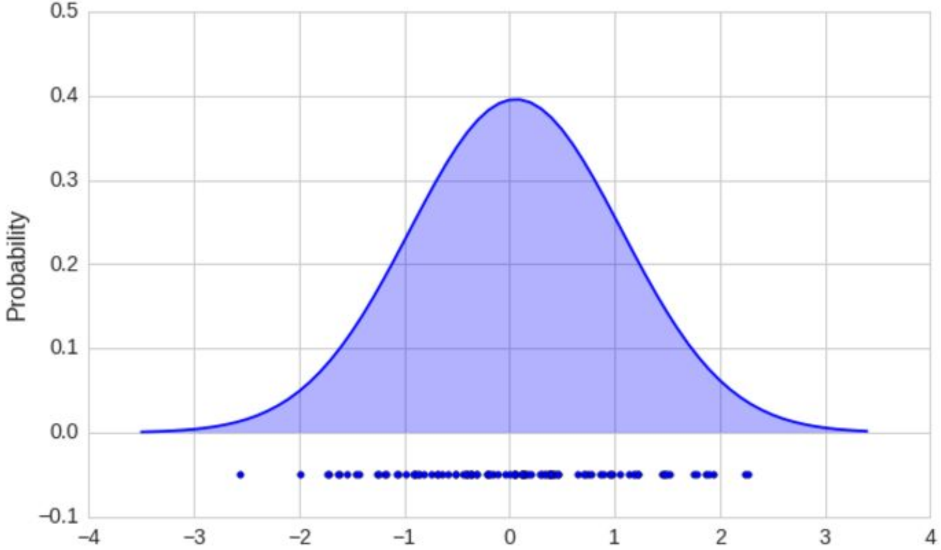
```
X = [0, 1, 1, 2, 1.5, 6, 7, 8, 7]
a = GaussianKernelDensity(X)
```





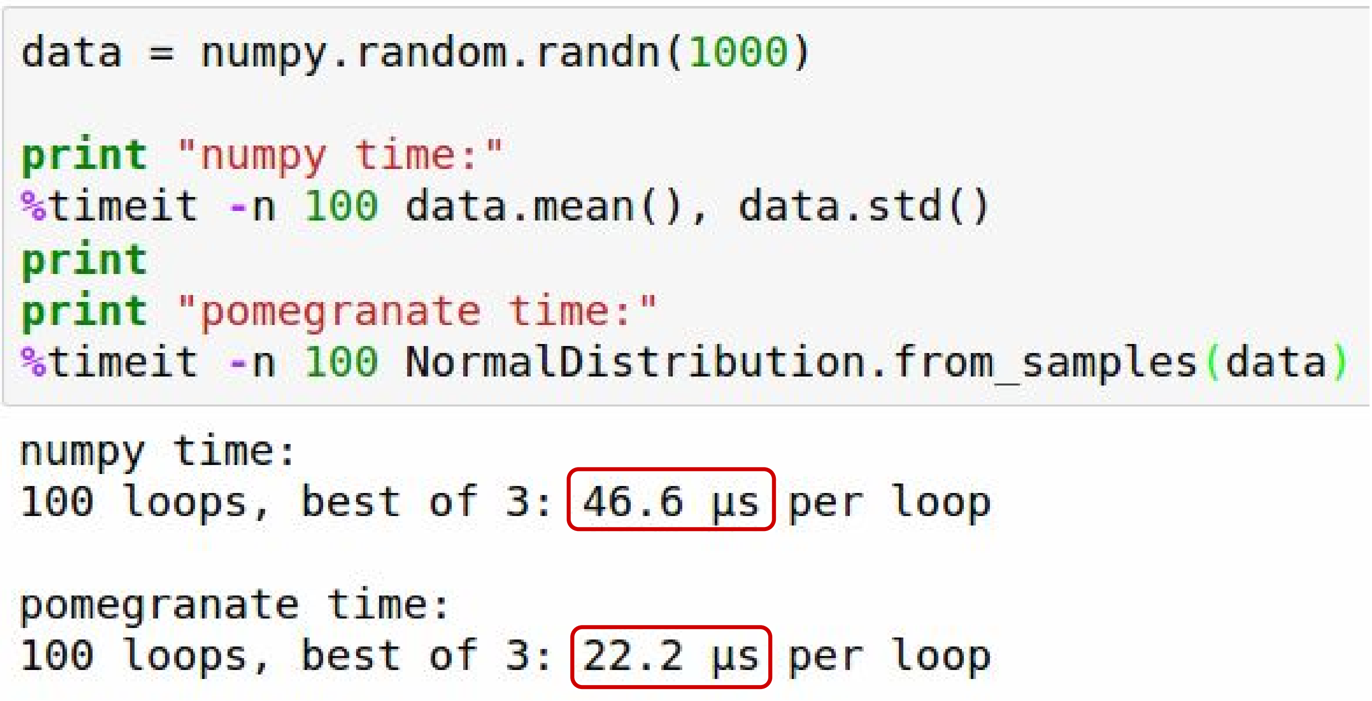
Models can also be learned directly from data

```
X = numpy.random.normal(0, 1, 100)
a = NormalDistribution.from_samples(X)
```



pomegranate can be faster than numpy

Fitting a Normal Distribution to 1,000 samples

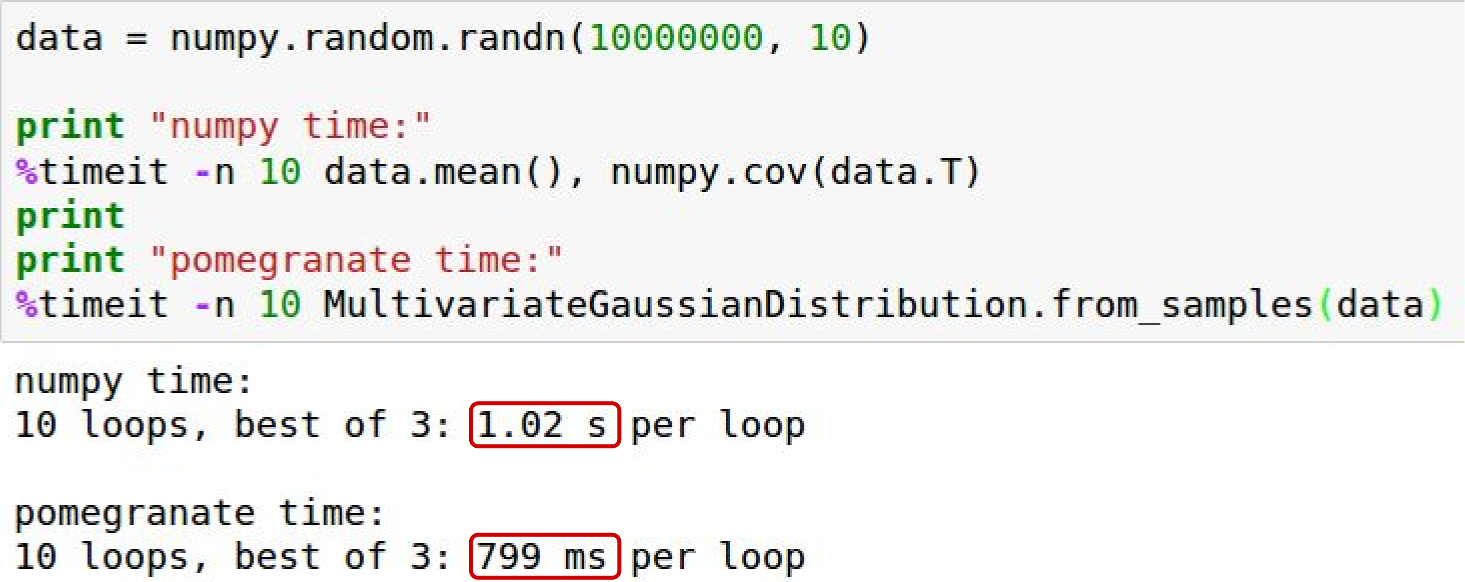


```
data = numpy.random.randn(1000)

print "numpy time:"
%timeit -n 100 data.mean(), data.std()
print
print "pomegranate time:"
%timeit -n 100 NormalDistribution.from_samples(data)
```

```
numpy time:
100 loops, best of 3: 46.6 µs per loop

pomegranate time:
100 loops, best of 3: 22.2 µs per loop
```

pomegranate can be faster than numpy

Fitting Multivariate Gaussian to 10,000,000 samples of 10 dimensions



```
data = numpy.random.randn(10000000, 10)

print "numpy time:"
%timeit -n 10 data.mean(), numpy.cov(data.T)
print
print "pomegranate time:"
%timeit -n 10 MultivariateGaussianDistribution.from_samples(data)
```

```
numpy time:
10 loops, best of 3: 1.02 s per loop

pomegranate time:
10 loops, best of 3: 799 ms per loop
```

pomegranate can be faster than numpy

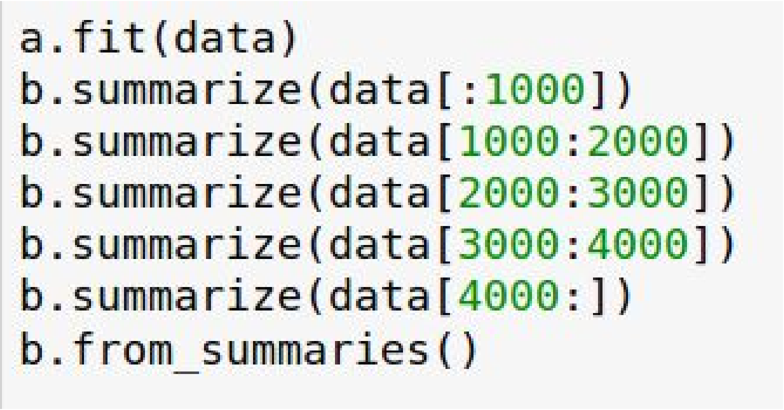
pomegranate reduces data to sufficient statistics for updates and so only has to go datasets once (for all models).

Here is an example of the Normal Distribution sufficient statistics

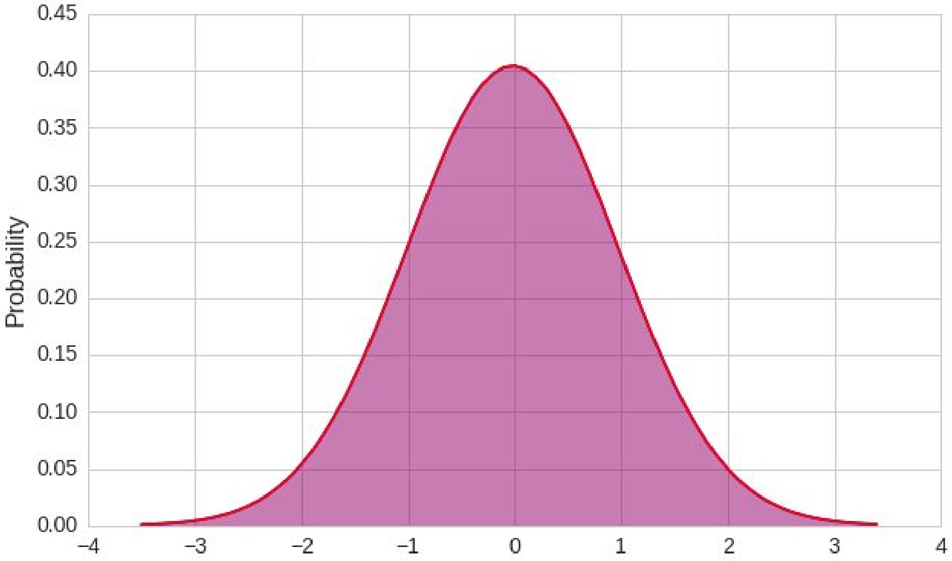
$$\sum_{i=1}^{n} w_i \qquad \sum_{i=1}^{n} w_i x_i \qquad \sum_{i=1}^{n} w_i x_i^2 \longrightarrow \begin{aligned} \mu &= \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \\ \sigma^2 &= \frac{\sum_{i=1}^{n} w_i x_i^2}{\sum_{i=1}^{n} w_i} - \frac{\left(\sum_{i=1}^{n} w_i x_i\right)^2}{\left(\sum_{i=1}^{n} w_i\right)^2} \end{aligned}$$

pomegranate supports out of core learning

Due to the use of sufficient statistics that are additive, pomegranate can natively support out-of-core/online learning, where you may not have the entire dataset in memory at a time

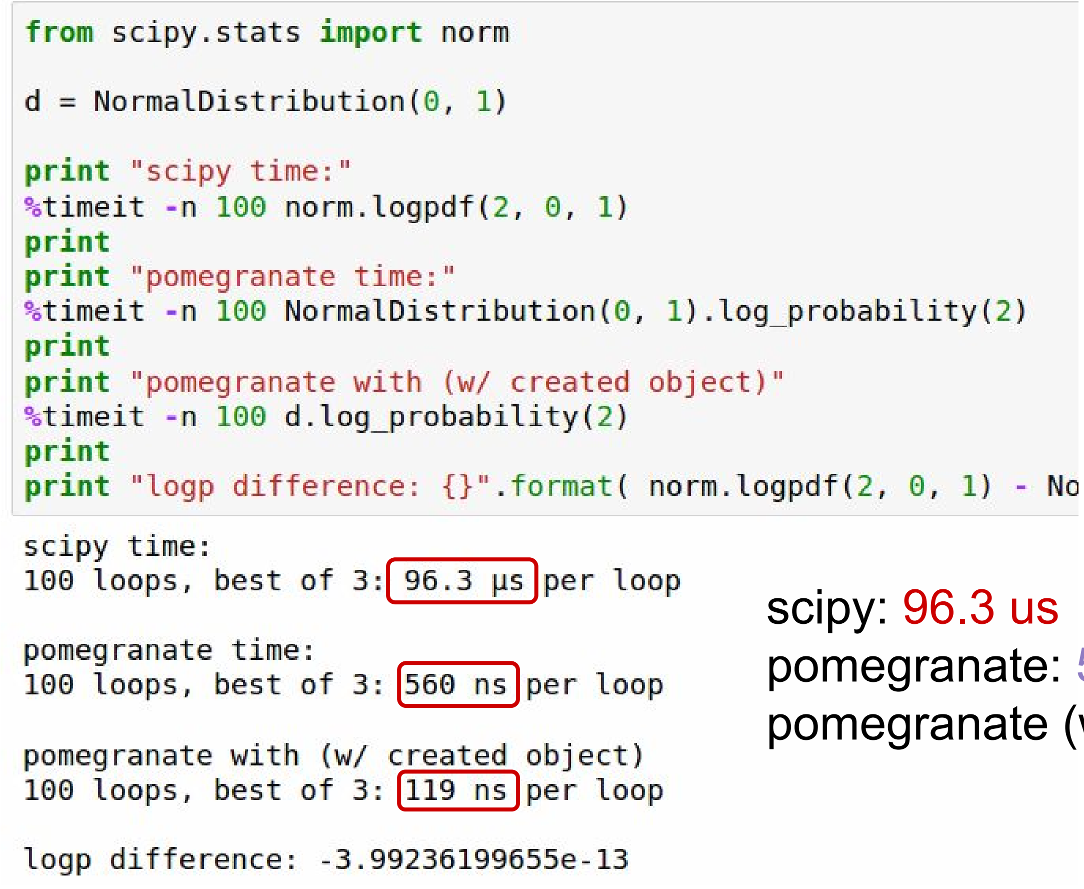


```
a.fit(data)
b.summarize(data[:1000])
b.summarize(data[1000:2000])
b.summarize(data[2000:3000])
b.summarize(data[3000:4000])
b.summarize(data[4000:])
b.from_summaries()
```



```
Fit Mean:       -0.0174820965846, Fit STD:       0.986767322871
Summarize Mean: -0.0174820965846, Summarize STD: 0.986767322871
```

pomegranate can be faster than scipy



```
from scipy.stats import norm

d = NormalDistribution(0, 1)

print "scipy time:"
%timeit -n 100 norm.logpdf(2, 0, 1)
print
print "pomegranate time:"
%timeit -n 100 NormalDistribution(0, 1).log_probability(2)
print
print "pomegranate with (w/ created object)"
%timeit -n 100 d.log_probability(2)
print
print "logp difference: {}".format( norm.logpdf(2, 0, 1) - No
```

```
scipy time:
100 loops, best of 3: 96.3 µs per loop

pomegranate time:
100 loops, best of 3: 560 ns per loop

pomegranate with (w/ created object)
100 loops, best of 3: 119 ns per loop

logp difference: -3.99236199655e-13
```

scipy: 96.3 us
pomegranate: 560 ns
pomegranate (w/ precreated object): 119 ns

pomegranate can be faster than scipy

pomegranate uses aggressive caching of values required for probability calculations to speed them up

$$P(X|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$logP(X|\mu,\sigma) = -\log(\sqrt{2\pi}\sigma) - \frac{(x-\mu)^2}{2\sigma^2}$$

$$logP(X|\mu,\sigma) = \alpha - \frac{(x-\mu)^2}{\beta}$$

GOSSIP GIRL

Example ‘blast’ from Gossip Girl

Spotted: Lonely Boy. Can't believe the love of his life has returned. If only she knew who he was. But everyone knows Serena. And everyone is talking. Wonder what Blair Waldorf thinks. Sure, they're BFF's, but we always thought Blair's boyfriend Nate had a thing for Serena.

Example ‘blast’ from Gossip Girl

Why'd she leave? Why'd she return? Send me all the deets. And who am I? That's the secret I'll never tell. The only one. —XOXO. Gossip Girl.

How do we encode these blasts?

Better lock it down with Nate, B. Clock's ticking.

+1 Nate
-1 Blair

How do we encode these blasts?

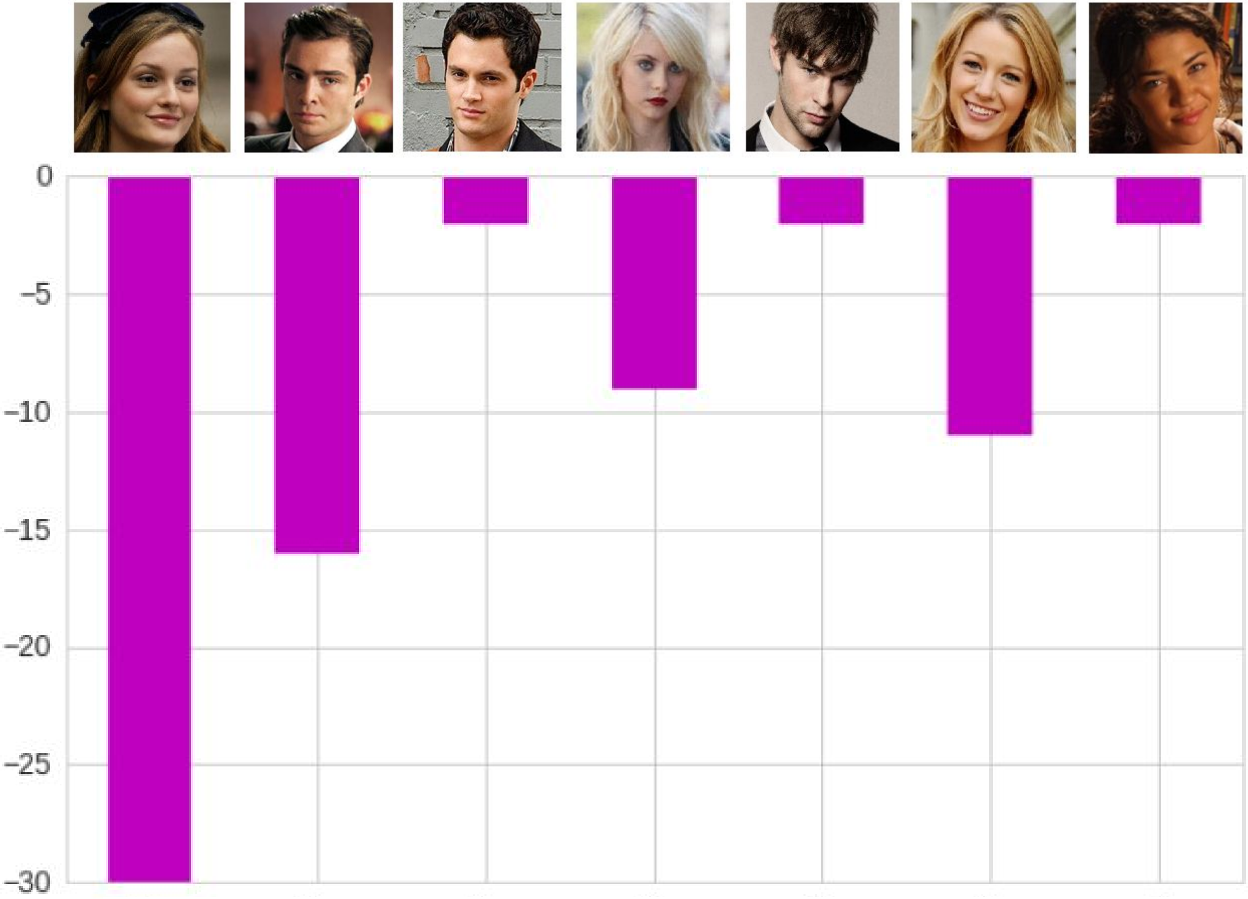
Better lock it down with Nate, B. Clock's ticking.

+1 Nate
-1 Blair

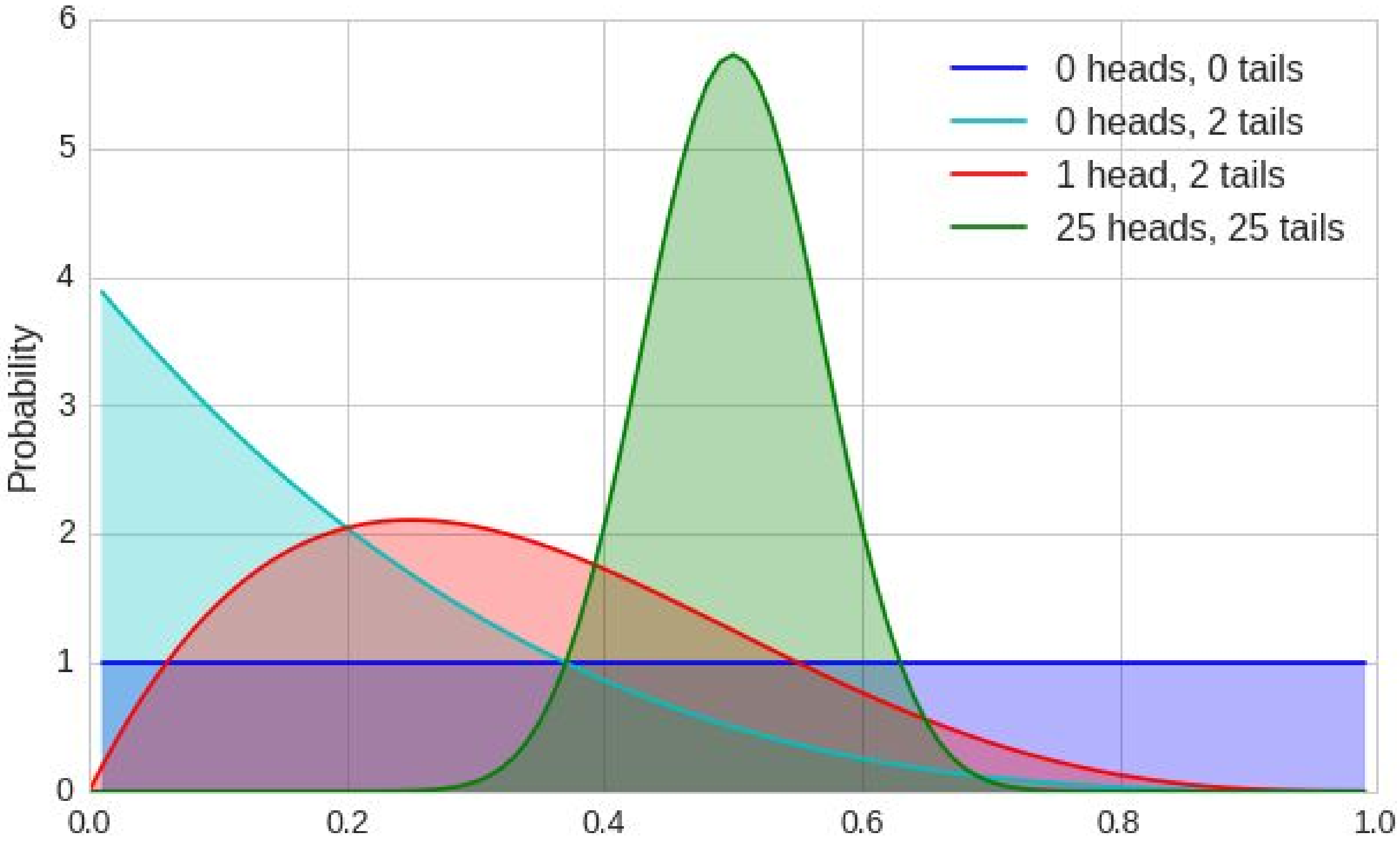
This just in: S and B committing a crime of fashion. Who doesn't love a five-finger discount. Especially if it's the middle one.

-1 Blair
-1 Serena

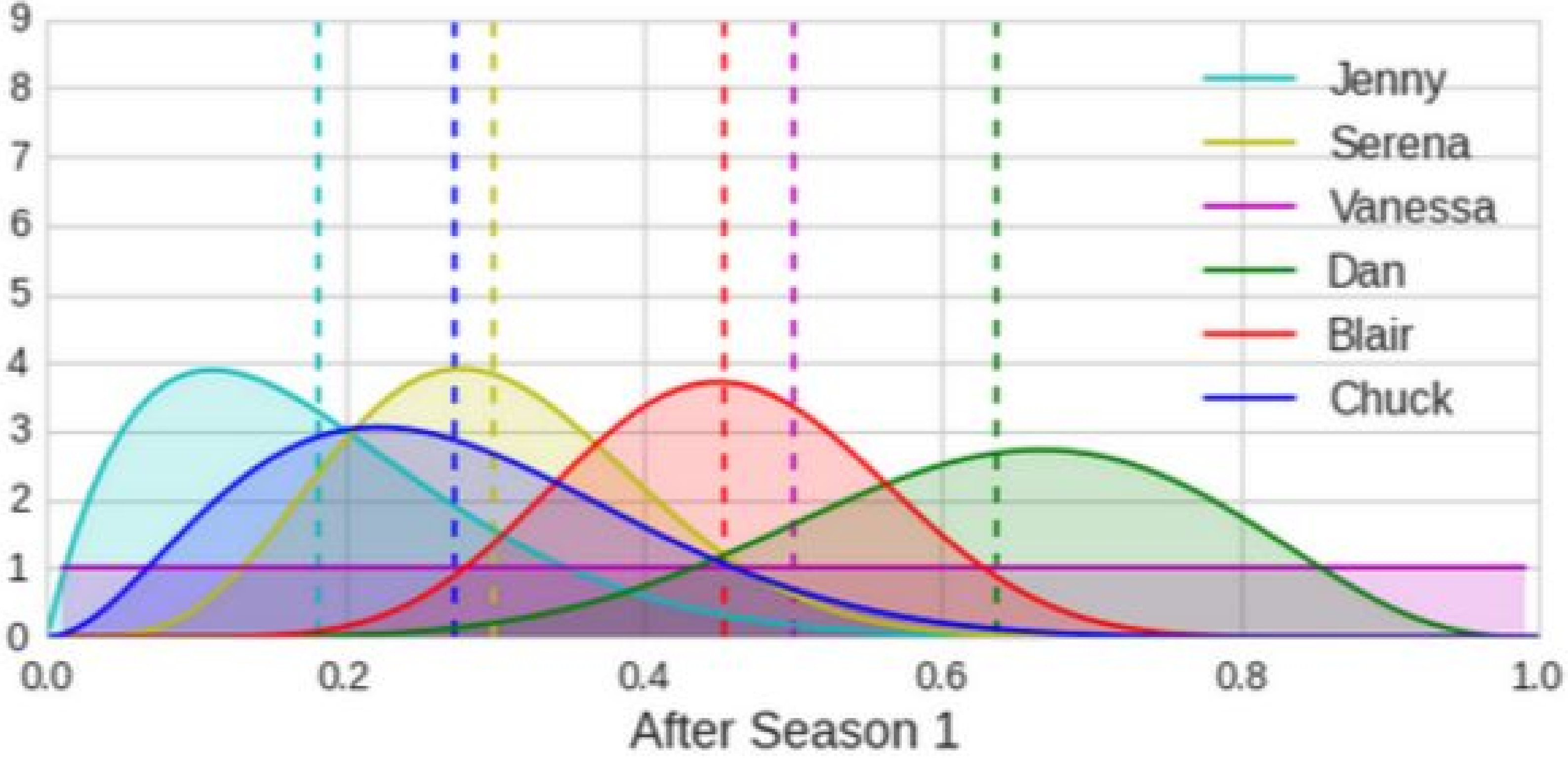
Simple summations don’t distinguish well



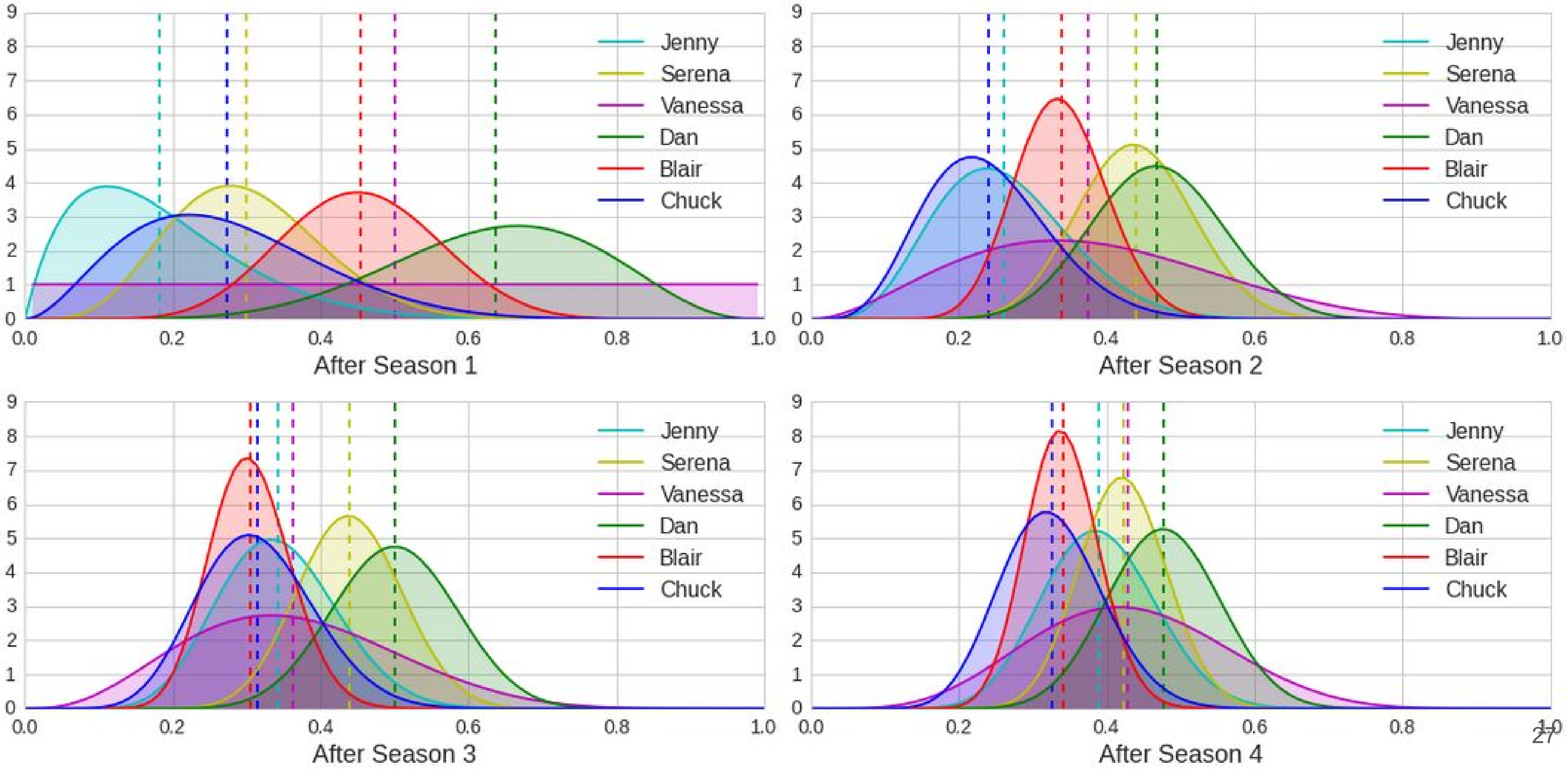
Beta distributions can model uncertainty well



Beta distributions capture our certainty about the identity of Gossip Girl



The distributions converge as the show progresses



Overview

The API

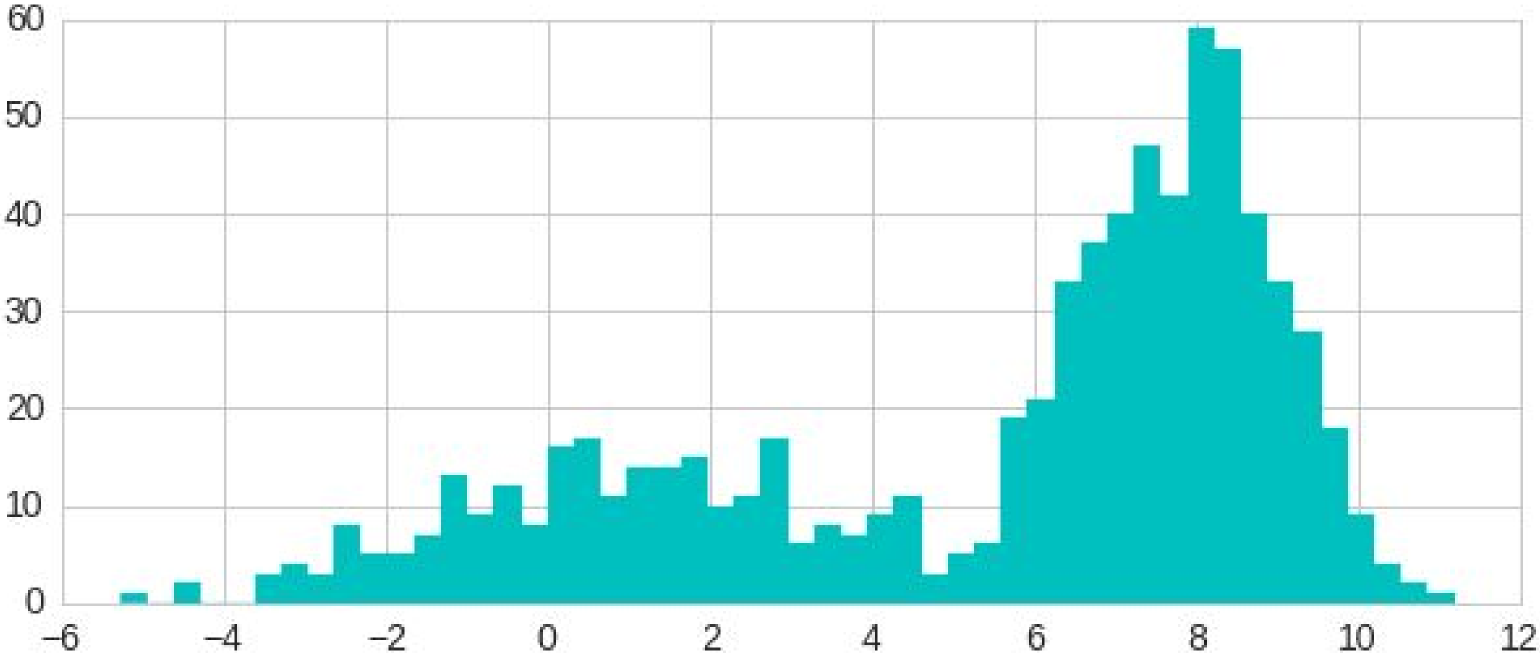
Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

Finale: Train a mixture of hidden markov models in parallel

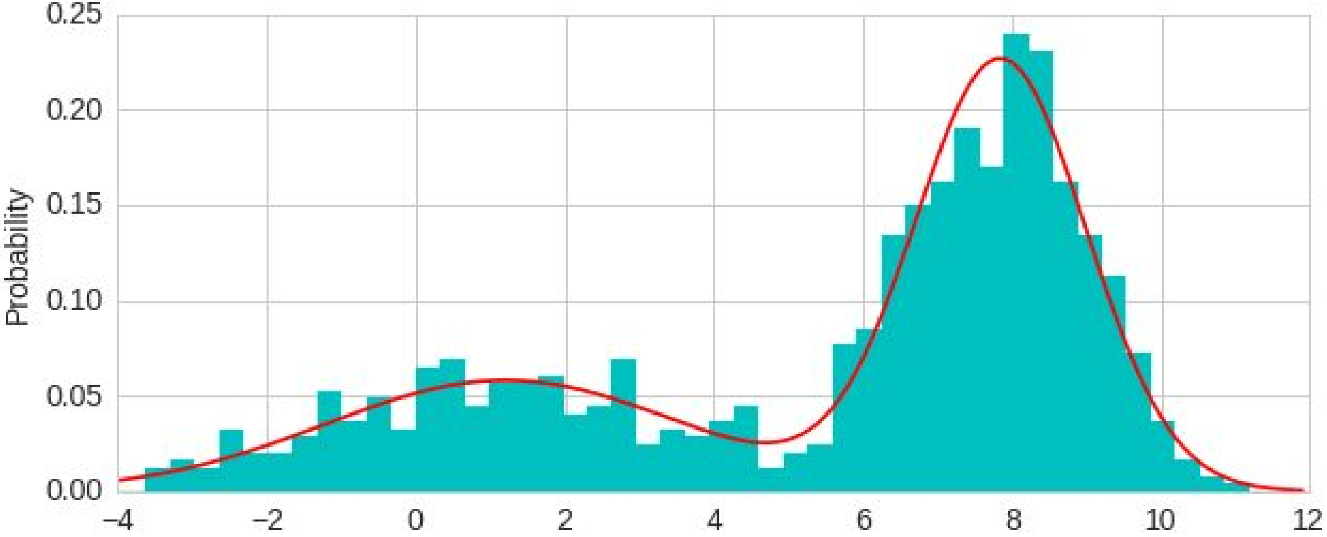
General Mixture Models (GMMs) can model multi-component distributions



GMMs use Expectation-Maximization (EM) to fit

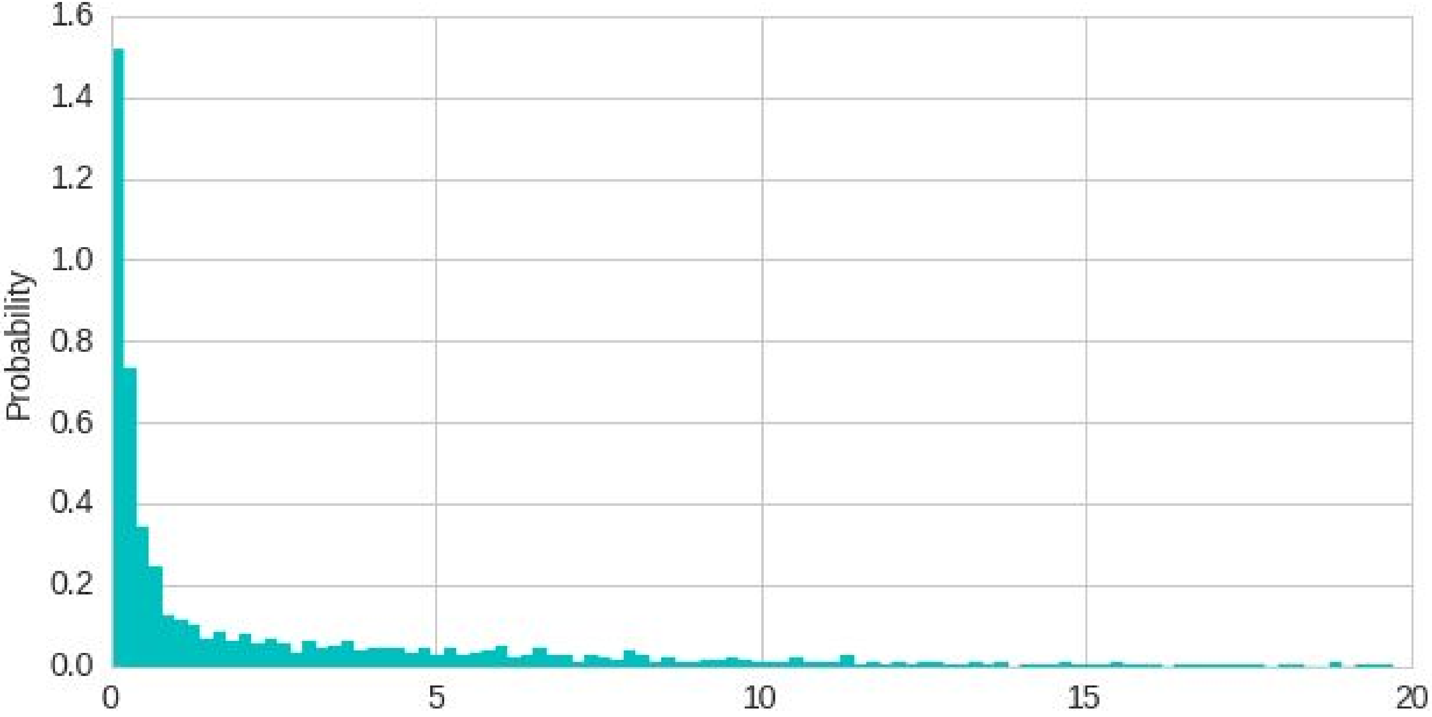
1. Initialize clusters using kmeans++ or kmeans||
2. Assign weights to all points equal to the posterior P(M|D) (E step)
3. Update distribution using weighted points (M step)
4. Repeat 2 and 3 ~~forever~~ until convergence

General Mixture Models (GMMs) can model multi-component distributions

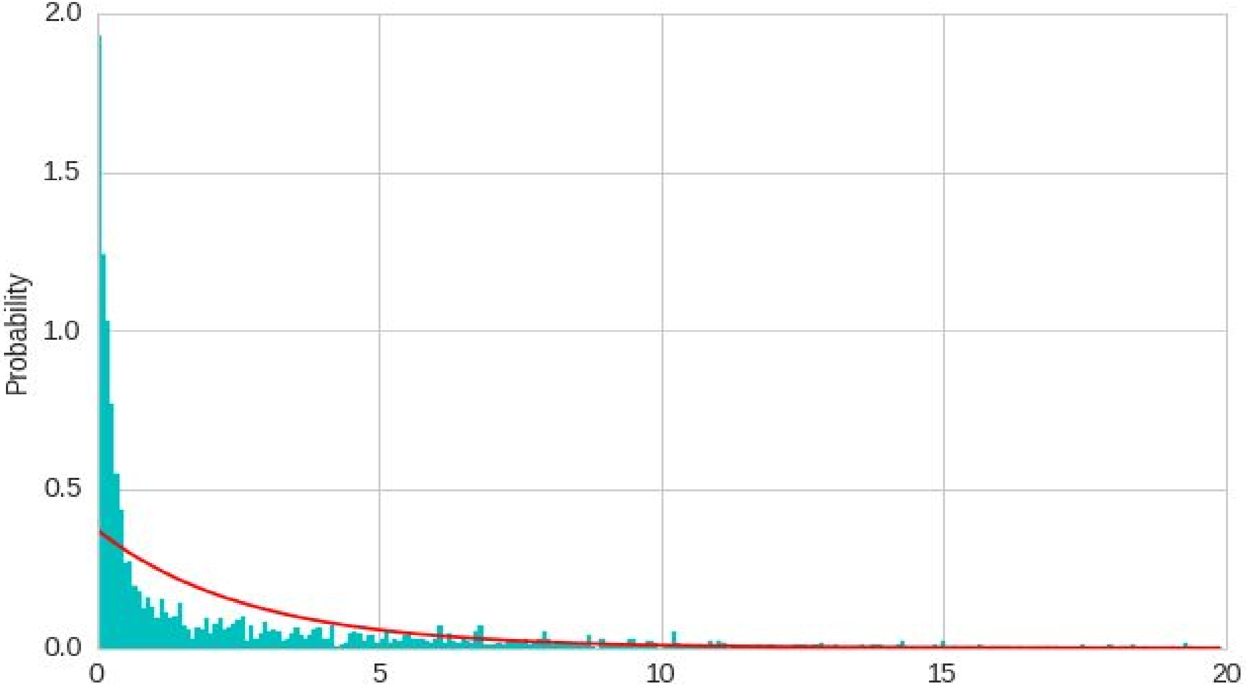


`model = GeneralMixtureModel.from_samples(NormalDistribution, 2, X)`

GMMs are not limited to Gaussian distributions

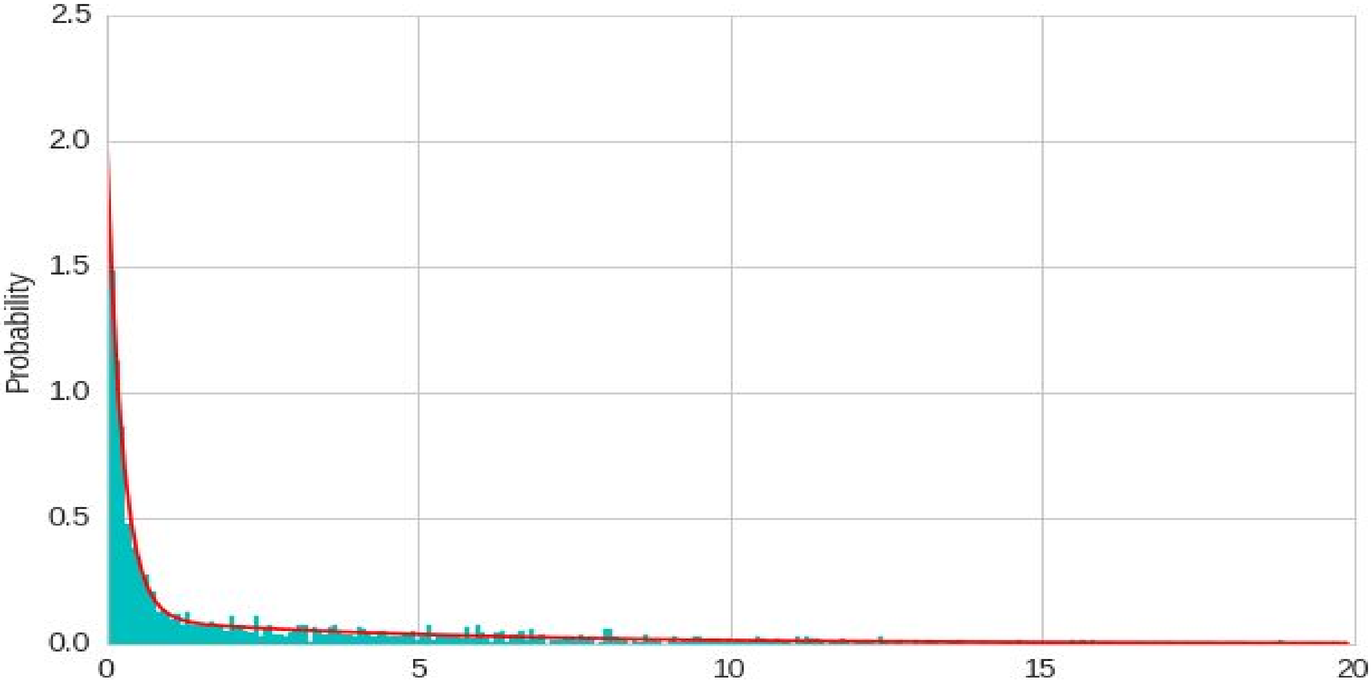


A single exponential distribution does not model this data well



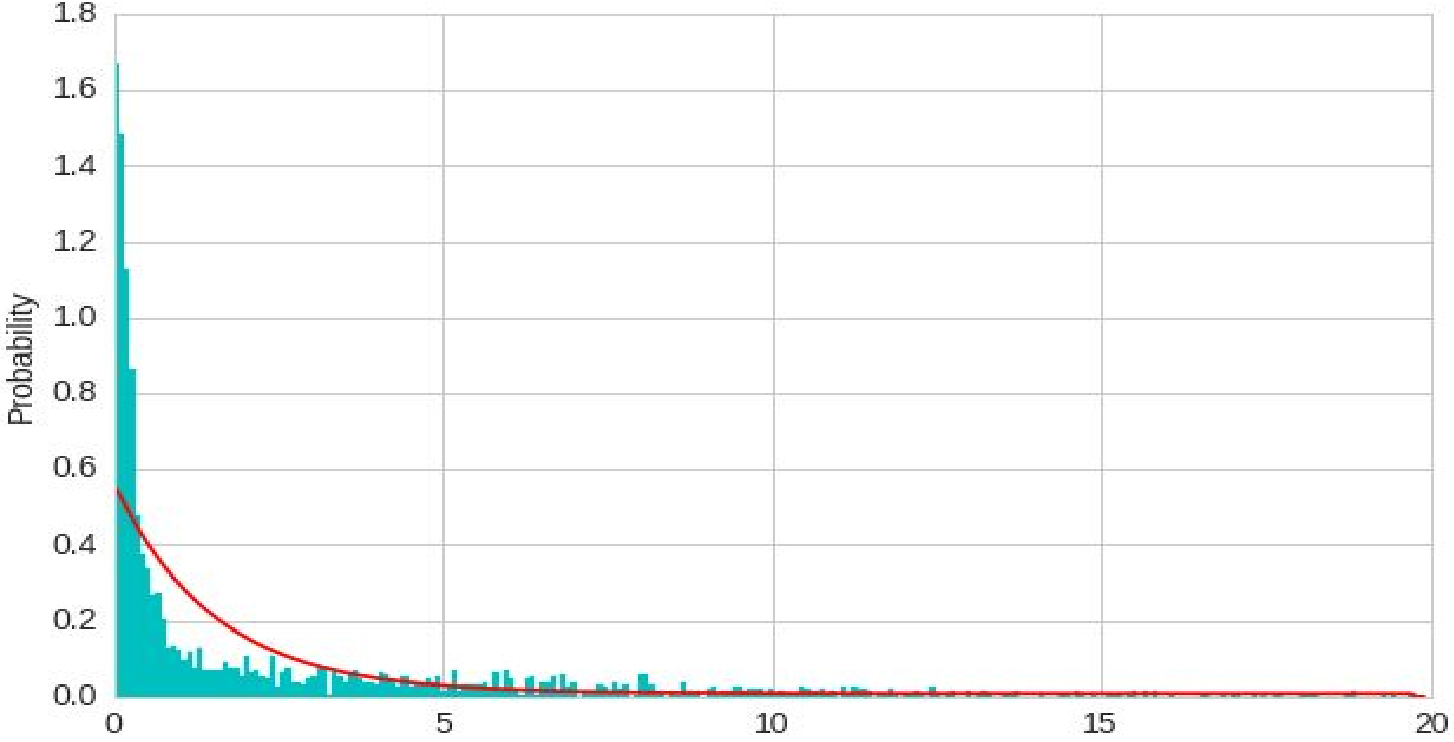
model = ExponentialDistribution.from_samples(X)

A mixture of two exponentials models the data much better



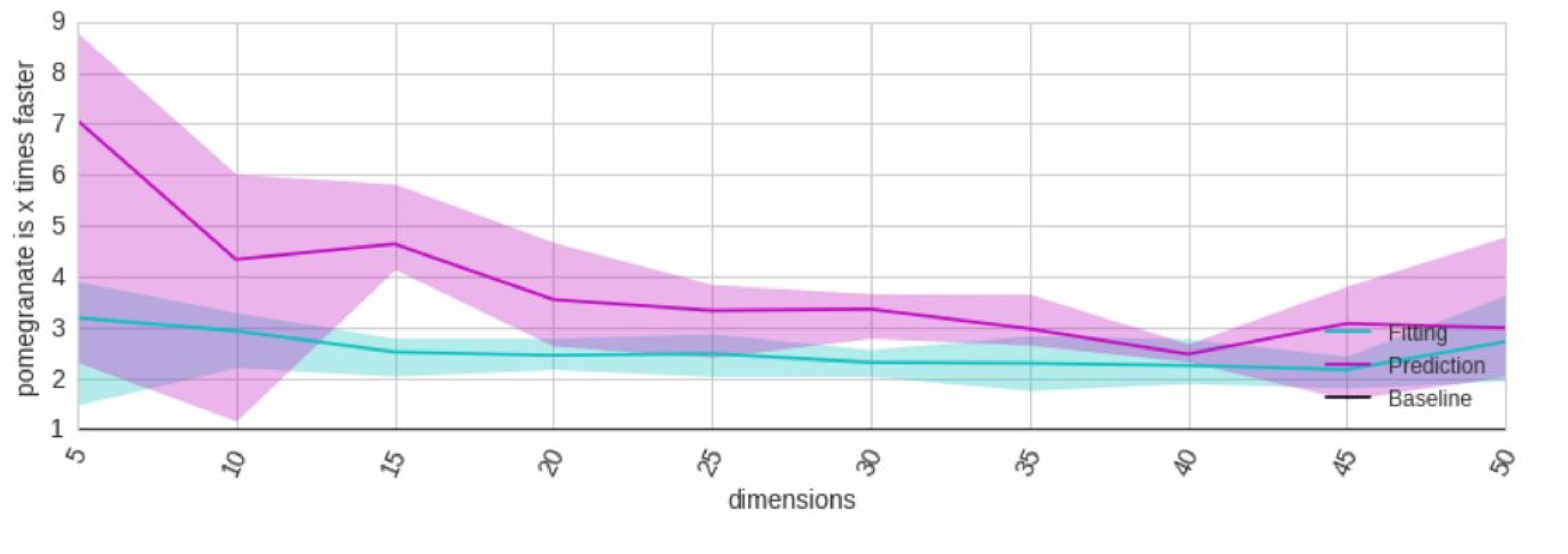
```
model = GeneralMixtureModel.from_samples(ExponentialDistribution, 2, X)
```

Heterogeneous mixtures are natively supported



model = GeneralMixtureModel.from_samples([ExponentialDistribution, UniformDistribution], 2, X)

general mixture models are faster than sklearn



Overview

The API

Major Models/Model Stacks

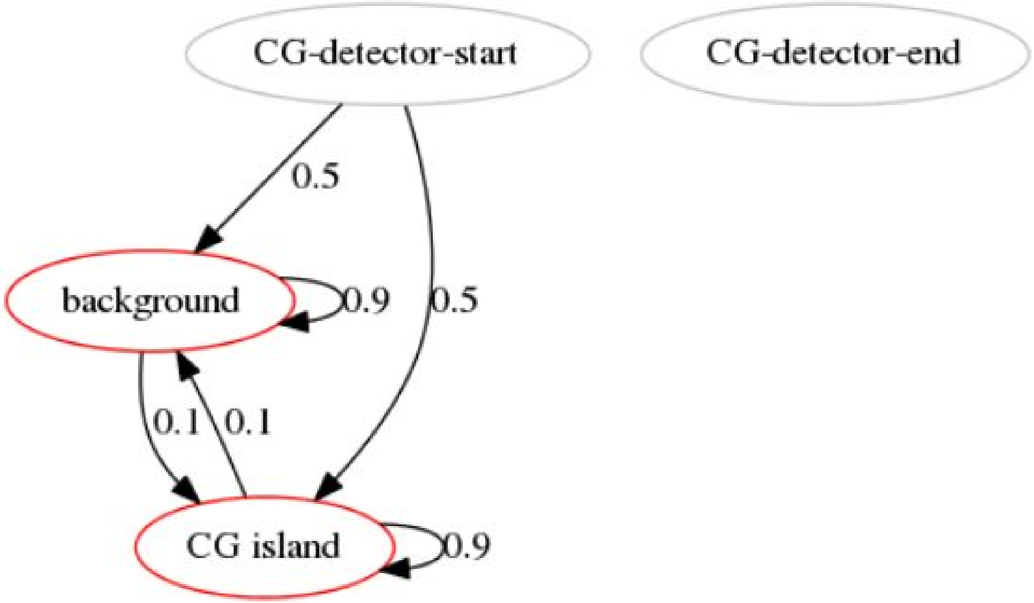
1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

Finale: Train a mixture of hidden markov models in parallel

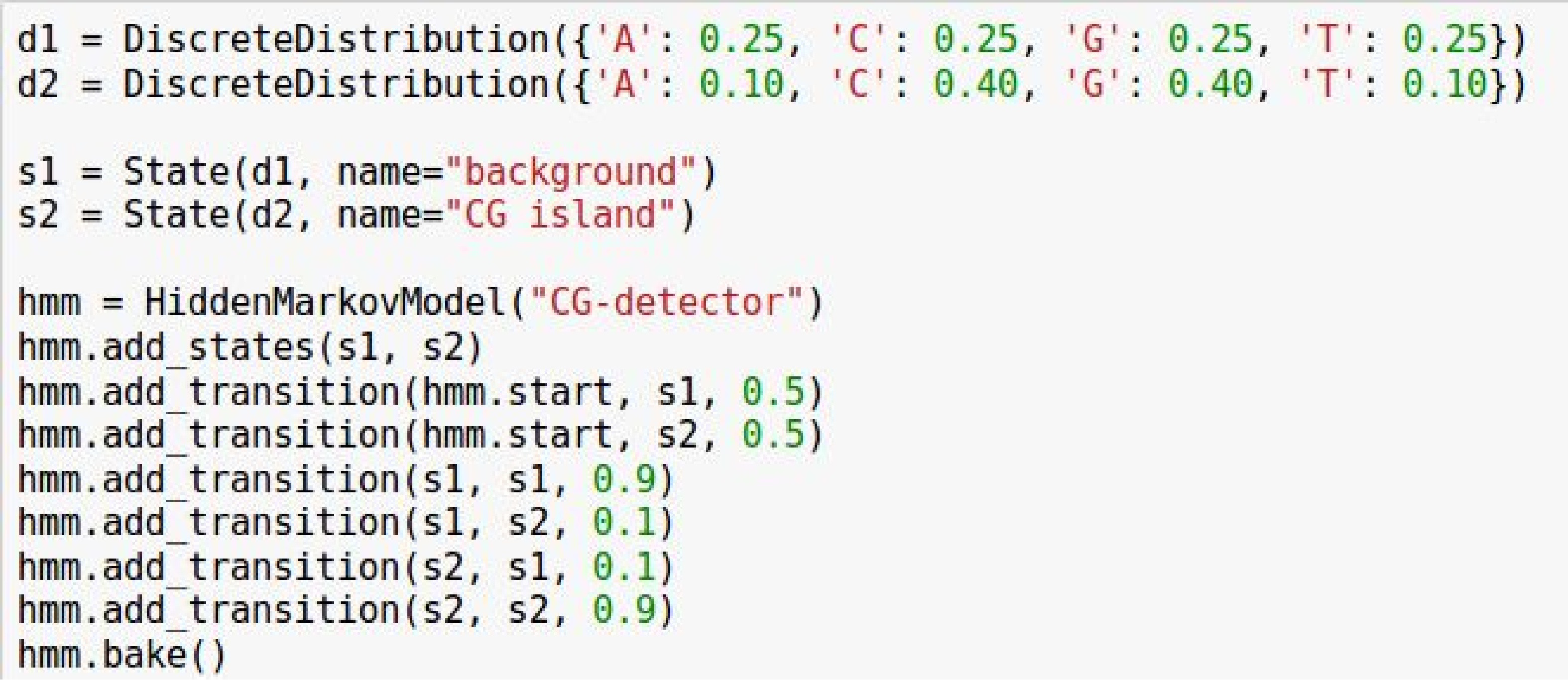
CG enrichment detection HMM

GACTACGACTCGCGCTCGCGCGACGCGCTCGACATCATCGACACGACACTC



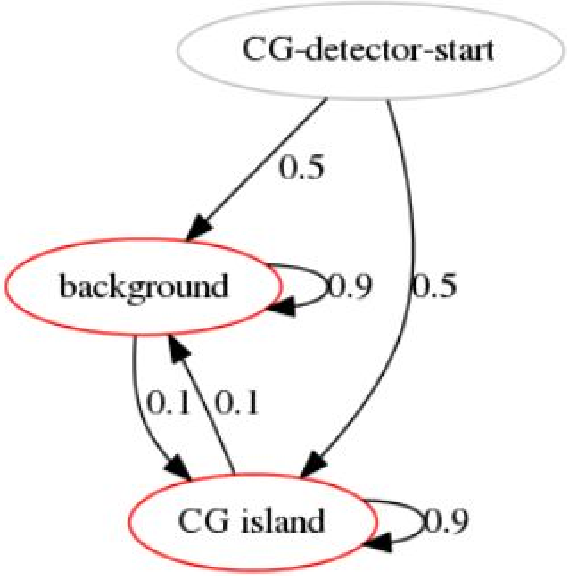
example: CG enrichment detector

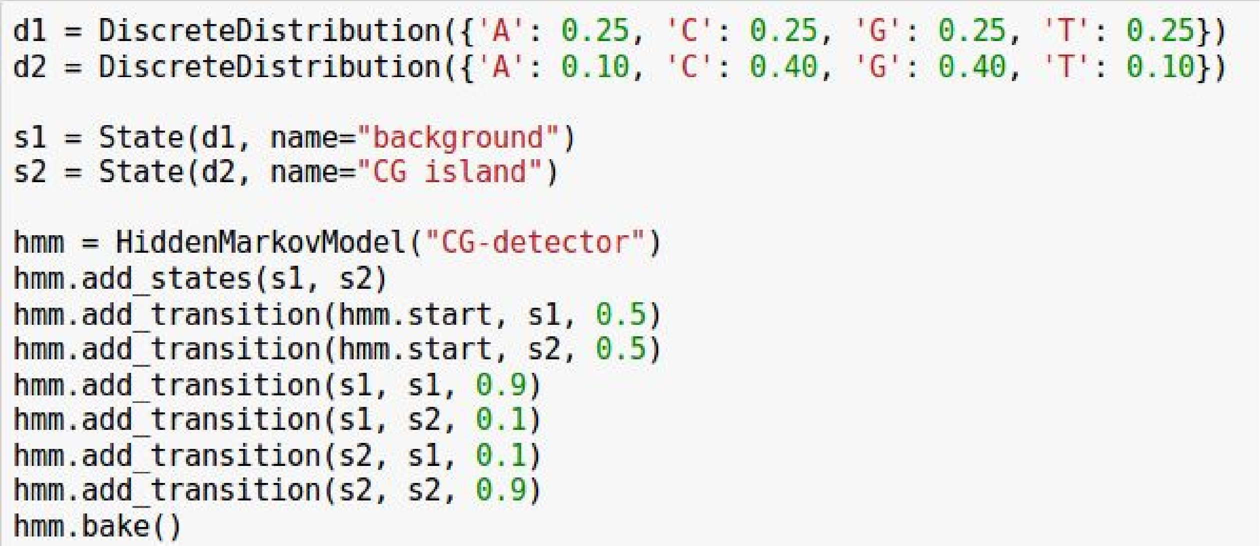
GACTACGACTCGCGCTCGCGCGACGCGCTCGACATCATCGACACGACACTC



```
d1 = DiscreteDistribution({'A': 0.25, 'C': 0.25, 'G': 0.25, 'T': 0.25})
d2 = DiscreteDistribution({'A': 0.10, 'C': 0.40, 'G': 0.40, 'T': 0.10})

s1 = State(d1, name="background")
s2 = State(d2, name="CG island")

hmm = HiddenMarkovModel("CG-detector")
hmm.add_states(s1, s2)
hmm.add_transition(hmm.start, s1, 0.5)
hmm.add_transition(hmm.start, s2, 0.5)
hmm.add_transition(s1, s1, 0.9)
hmm.add_transition(s1, s2, 0.1)
hmm.add_transition(s2, s1, 0.1)
hmm.add_transition(s2, s2, 0.9)
hmm.bake()
```

example: CG enrichment detector

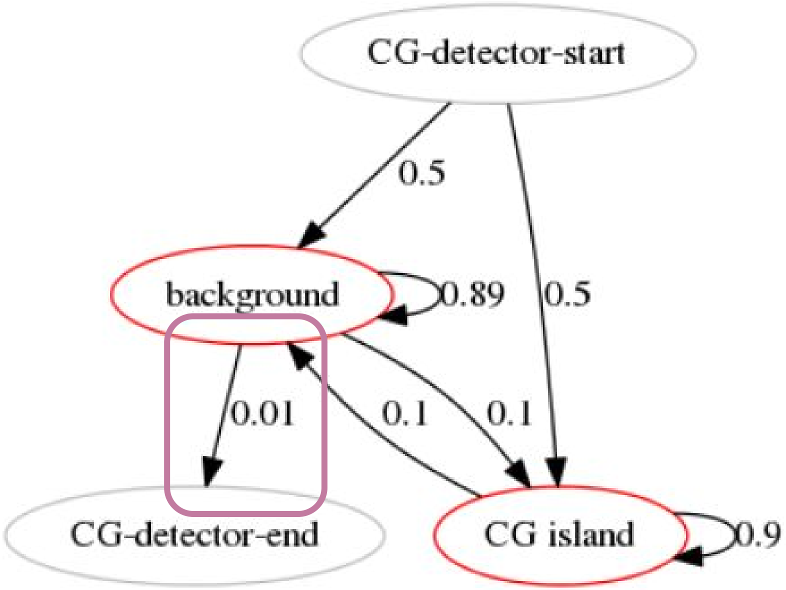


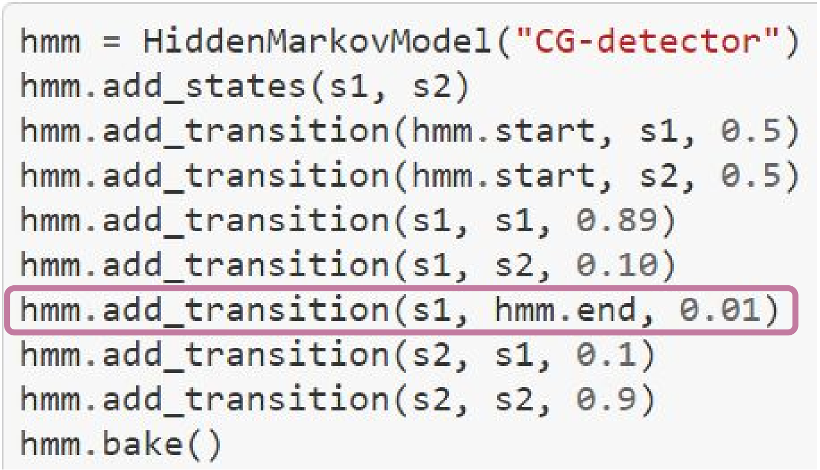


```
d1 = DiscreteDistribution({'A': 0.25, 'C': 0.25, 'G': 0.25, 'T': 0.25})
d2 = DiscreteDistribution({'A': 0.10, 'C': 0.40, 'G': 0.40, 'T': 0.10})

s1 = State(d1, name="background")
s2 = State(d2, name="CG island")

hmm = HiddenMarkovModel("CG-detector")
hmm.add_states(s1, s2)
hmm.add_transition(hmm.start, s1, 0.5)
hmm.add_transition(hmm.start, s2, 0.5)
hmm.add_transition(s1, s1, 0.9)
hmm.add_transition(s1, s2, 0.1)
hmm.add_transition(s2, s1, 0.1)
hmm.add_transition(s2, s2, 0.9)
hmm.bake()
```

```
sequence: CGACTACTGACTACTCGCGCACGCGGCACGCGTGCCGTCTATACTGCGCATACGGC
hmm pred: 11111111111111100000000000000000000001111111111111110000

hmm state 0: CG island
hmm state 1: background
```

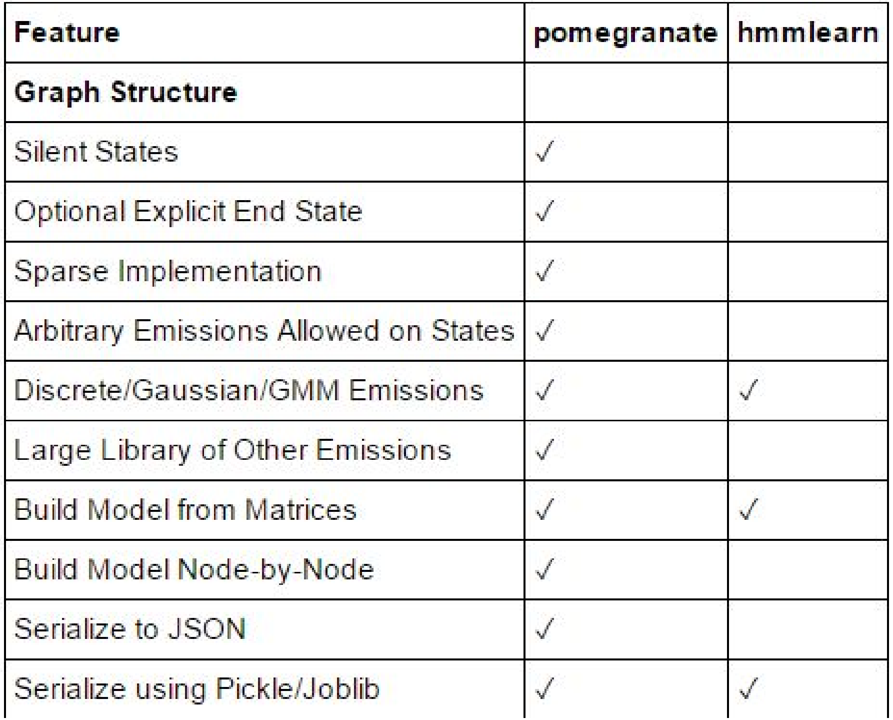
example: CG enrichment detector

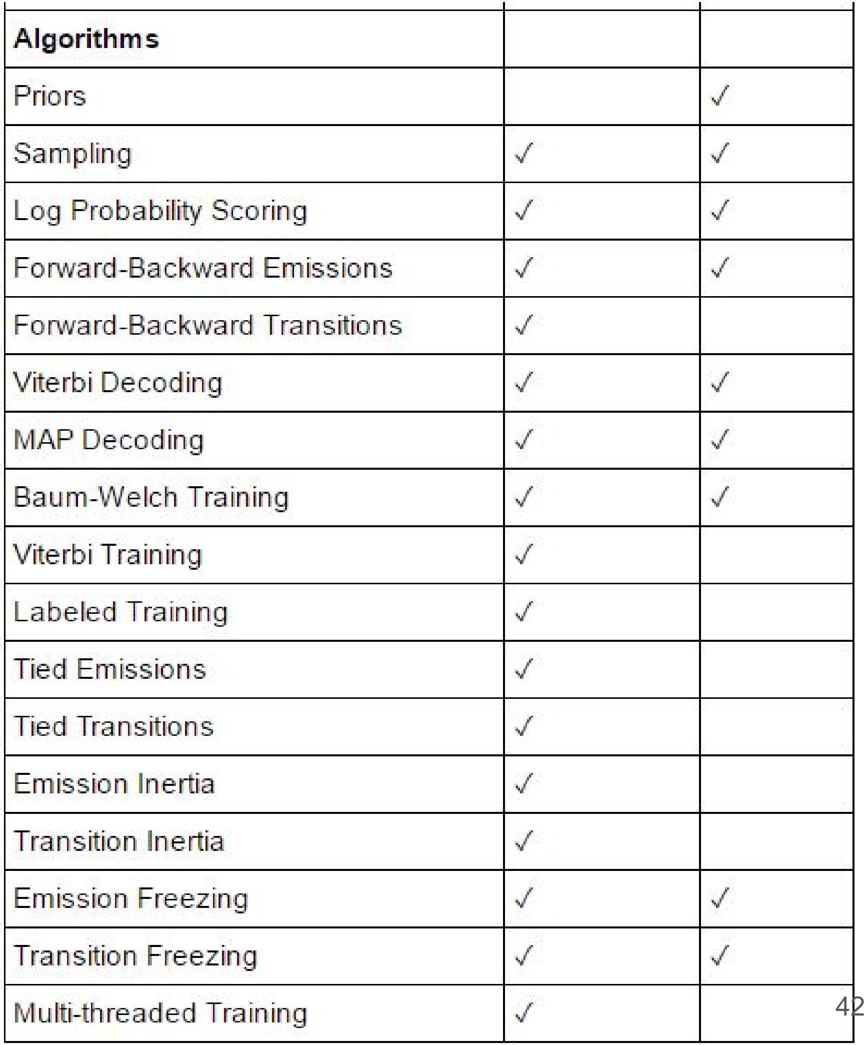




```
hmm = HiddenMarkovModel("CG-detector")
hmm.add_states(s1, s2)
hmm.add_transition(hmm.start, s1, 0.5)
hmm.add_transition(hmm.start, s2, 0.5)
hmm.add_transition(s1, s1, 0.89)
hmm.add_transition(s1, s2, 0.10)
hmm.add_transition(s1, hmm.end, 0.01)
hmm.add_transition(s2, s1, 0.1)
hmm.add_transition(s2, s2, 0.9)
hmm.bake()
```

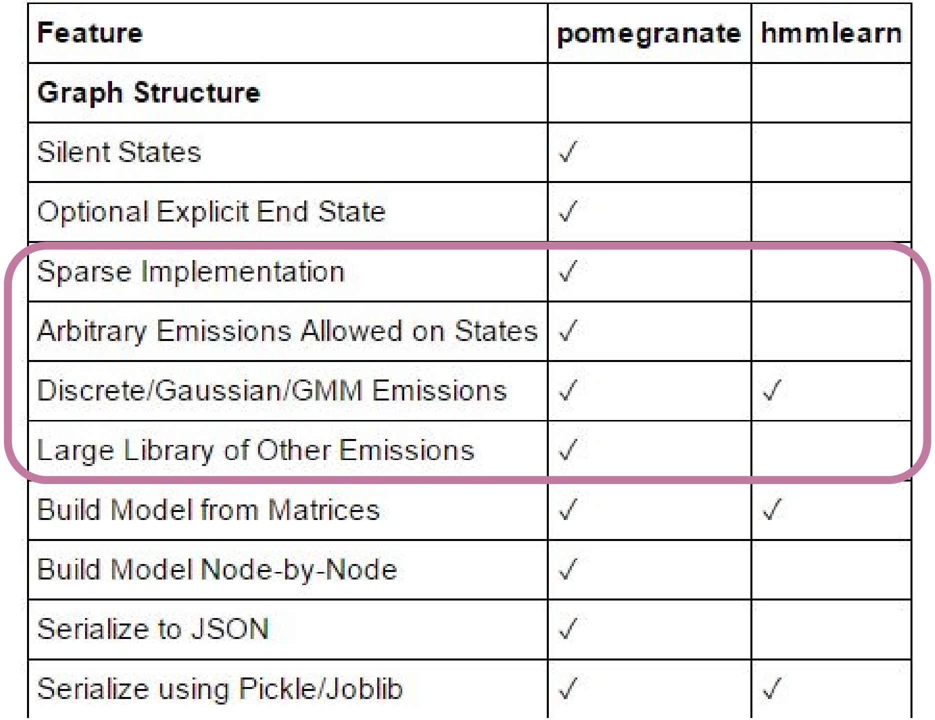
hidden markov models

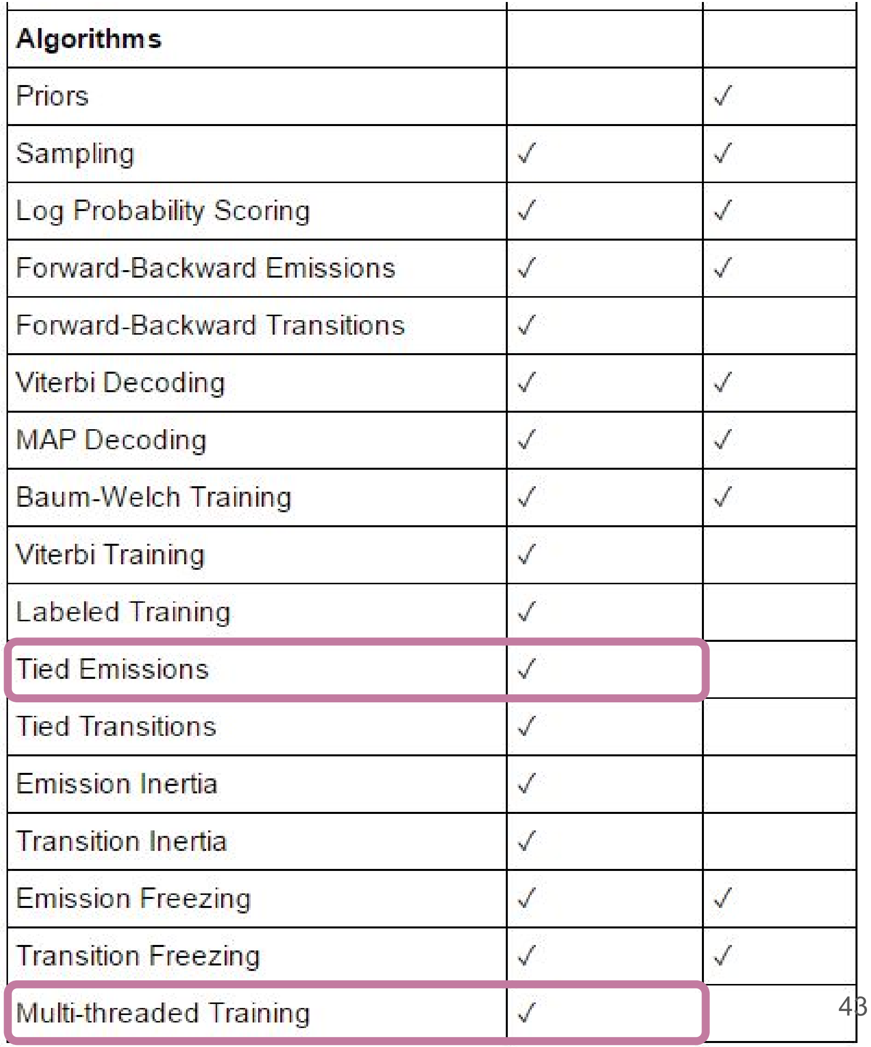




Feature	pomegranate	hmmlearn
Graph Structure		
Silent States	✓	
Optional Explicit End State	✓	
Sparse Implementation	✓	
Arbitrary Emissions Allowed on States	✓	
Discrete/Gaussian/GMM Emissions	✓	✓
Large Library of Other Emissions	✓	
Build Model from Matrices	✓	✓
Build Model Node-by-Node	✓	
Serialize to JSON	✓	
Serialize using Pickle/Joblib	✓	✓
Algorithms		
Priors		✓
Sampling	✓	✓
Log Probability Scoring	✓	✓
Forward-Backward Emissions	✓	✓
Forward-Backward Transitions	✓	
Viterbi Decoding	✓	✓
MAP Decoding	✓	✓
Baum-Welch Training	✓	✓
Viterbi Training	✓	
Labeled Training	✓	
Tied Emissions	✓	
Tied Transitions	✓	
Emission Inertia	✓	
Transition Inertia	✓	
Emission Freezing	✓	✓
Transition Freezing	✓	✓
Multi-threaded Training	✓	

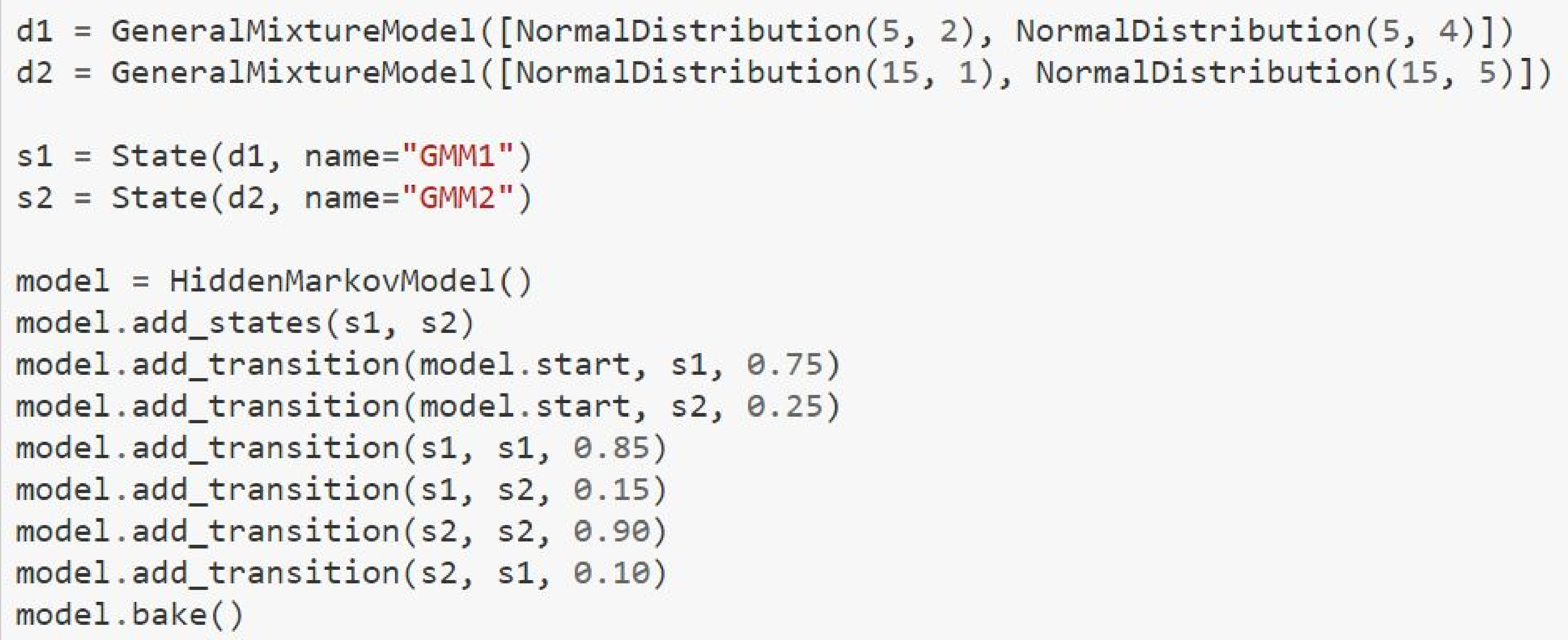
hidden markov models





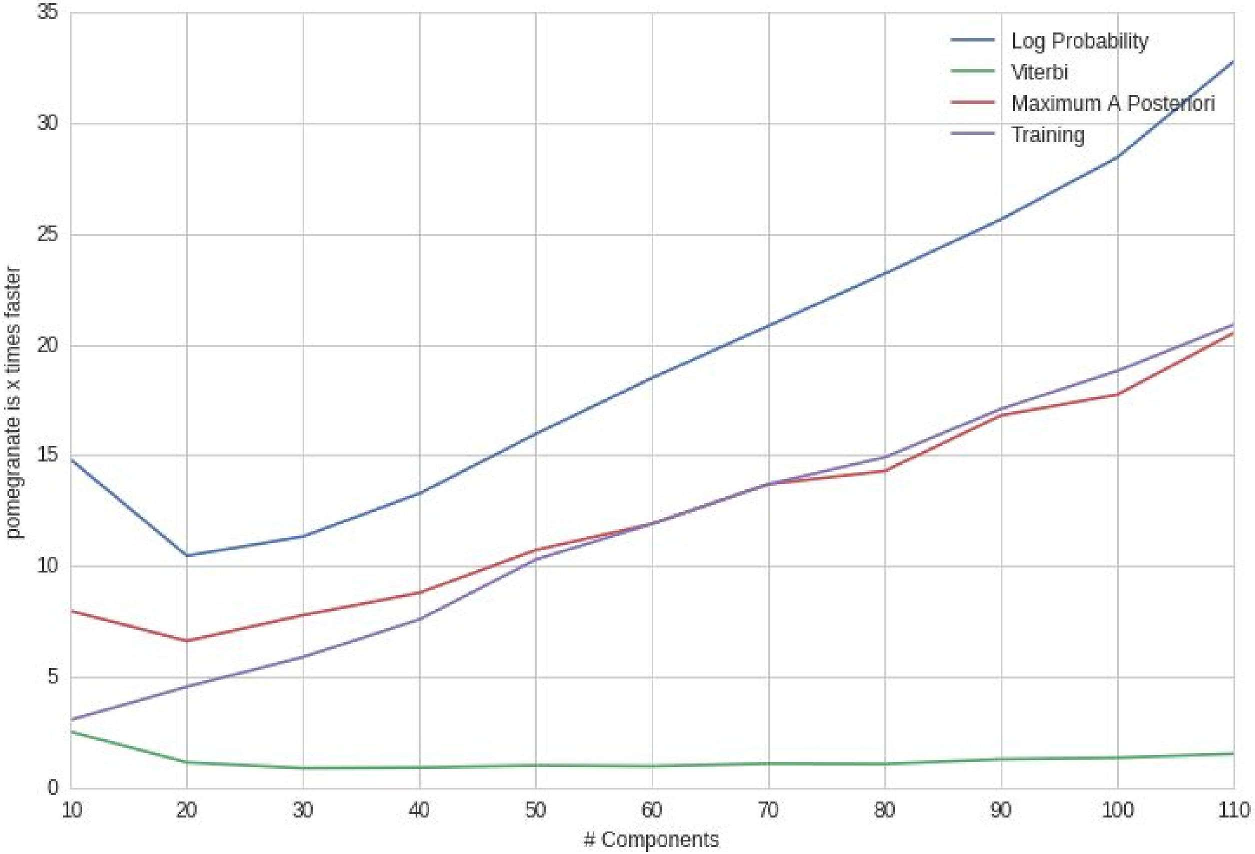
Feature	**pomegranate**	**hmmlearn**
Graph Structure		
Silent States	✓	
Optional Explicit End State	✓	
Sparse Implementation	✓	
Arbitrary Emissions Allowed on States	✓	
Discrete/Gaussian/GMM Emissions	✓	✓
Large Library of Other Emissions	✓	
Build Model from Matrices	✓	✓
Build Model Node-by-Node	✓	
Serialize to JSON	✓	
Serialize using Pickle/Joblib	✓	✓
Algorithms		
Priors		✓
Sampling	✓	✓
Log Probability Scoring	✓	✓
Forward-Backward Emissions	✓	✓
Forward-Backward Transitions	✓	
Viterbi Decoding	✓	✓
MAP Decoding	✓	✓
Baum-Welch Training	✓	✓
Viterbi Training	✓	
Labeled Training	✓	
Tied Emissions	✓	
Tied Transitions	✓	
Emission Inertia	✓	
Transition Inertia	✓	
Emission Freezing	✓	✓
Transition Freezing	✓	✓
Multi-threaded Training	✓	

example: GMM-HMM



```
d1 = GeneralMixtureModel([NormalDistribution(5, 2), NormalDistribution(5, 4)])
d2 = GeneralMixtureModel([NormalDistribution(15, 1), NormalDistribution(15, 5)])

s1 = State(d1, name="GMM1")
s2 = State(d2, name="GMM2")

model = HiddenMarkovModel()
model.add_states(s1, s2)
model.add_transition(model.start, s1, 0.75)
model.add_transition(model.start, s2, 0.25)
model.add_transition(s1, s1, 0.85)
model.add_transition(s1, s2, 0.15)
model.add_transition(s2, s2, 0.90)
model.add_transition(s2, s1, 0.10)
model.bake()
```

hidden markov models are faster than hmmlearn



Overview

The API

Major Models/Model Stacks

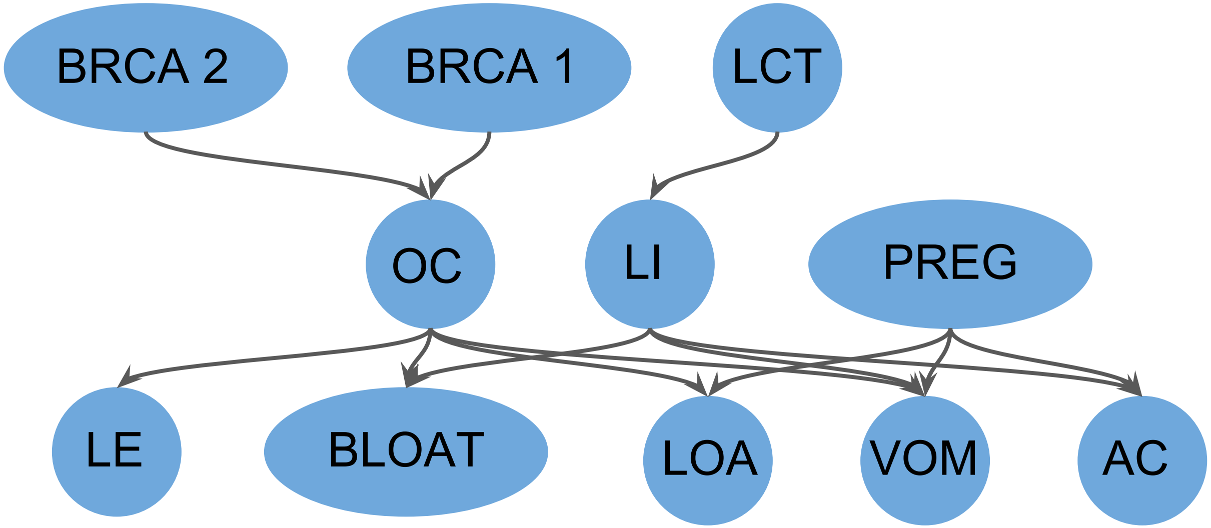
1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

Finale: Train a mixture of hidden markov models in parallel

Bayesian networks

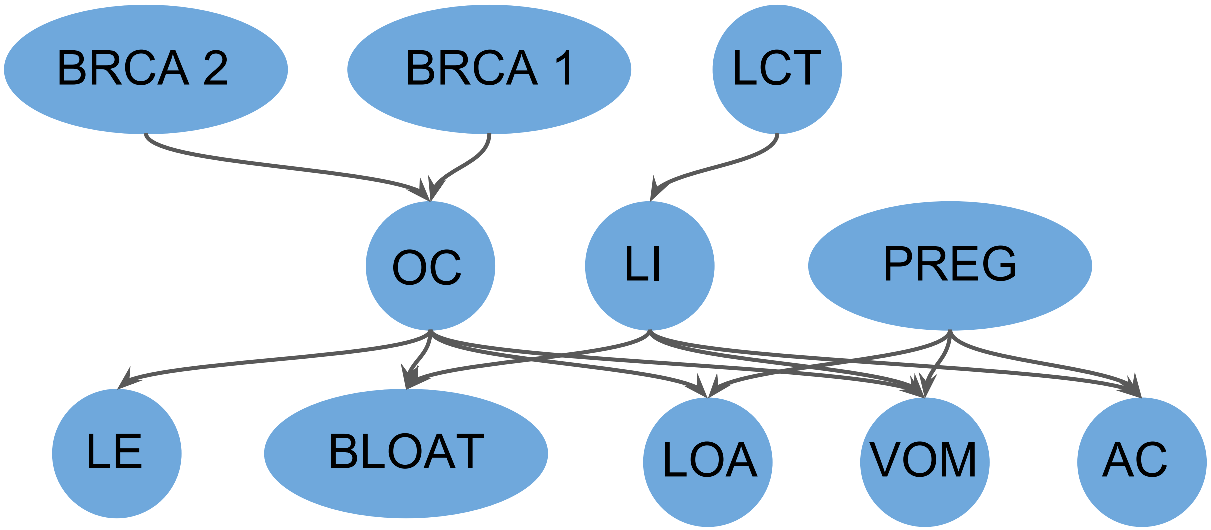
Bayesian networks are powerful inference tools which define a dependency structure between variables.



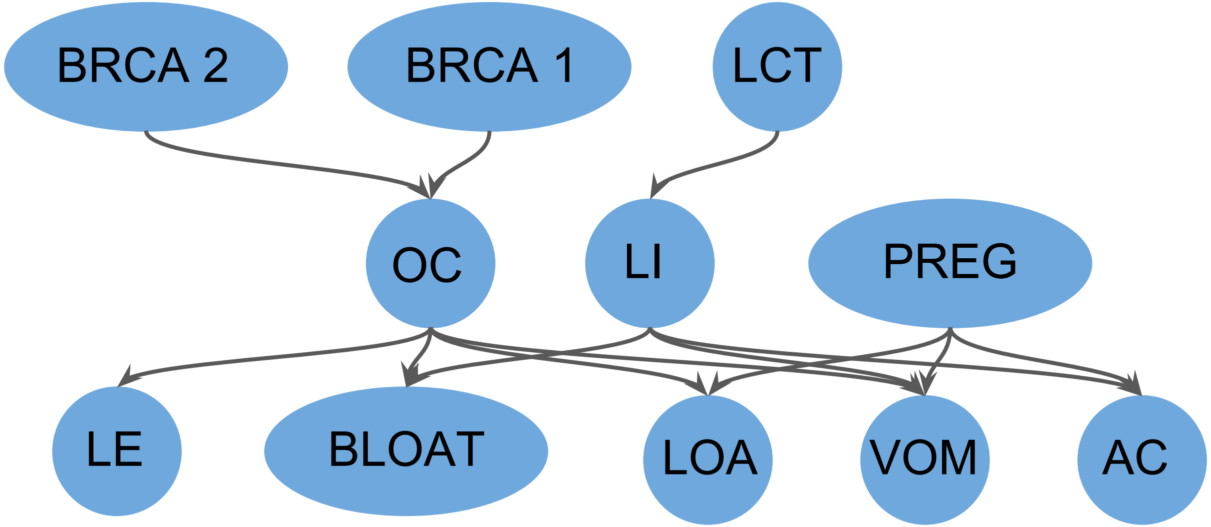
Bayesian networks

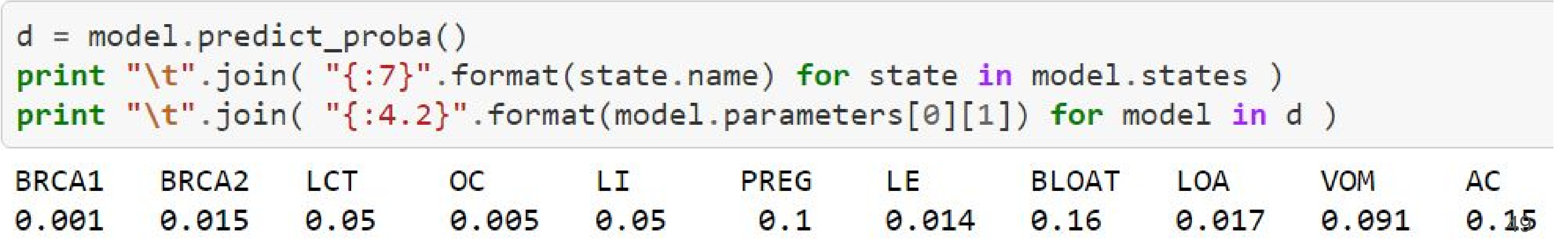
Two main non-trivial tasks:

1. Inference given incomplete information
2. Learning the dependency structure from data



Inference given incomplete information

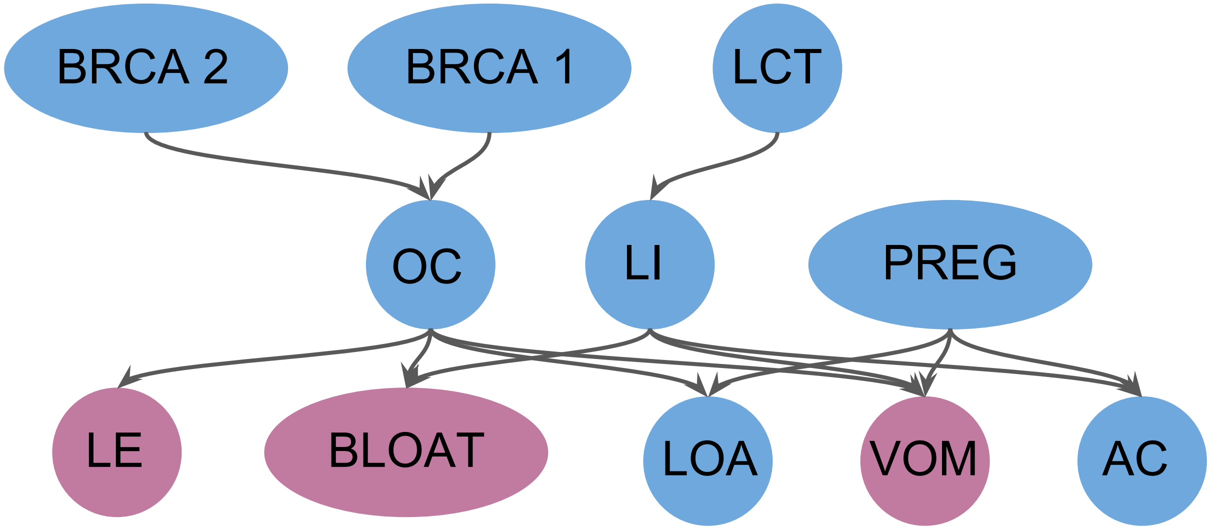


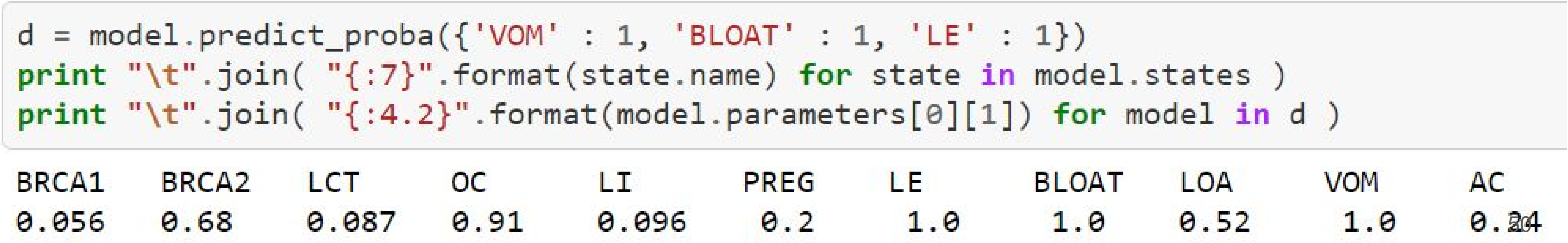


```
d = model.predict_proba()
print "\t".join( "{:7}".format(state.name) for state in model.states )
print "\t".join( "{:4.2}".format(model.parameters[0][1]) for model in d )
```

```
BRCA1    BRCA2    LCT      OC       LI       PREG     LE       BLOAT    LOA      VOM      AC
0.001    0.015    0.05     0.005    0.05      0.1     0.014    0.16     0.017    0.091    0.15
```

Inference given incomplete information

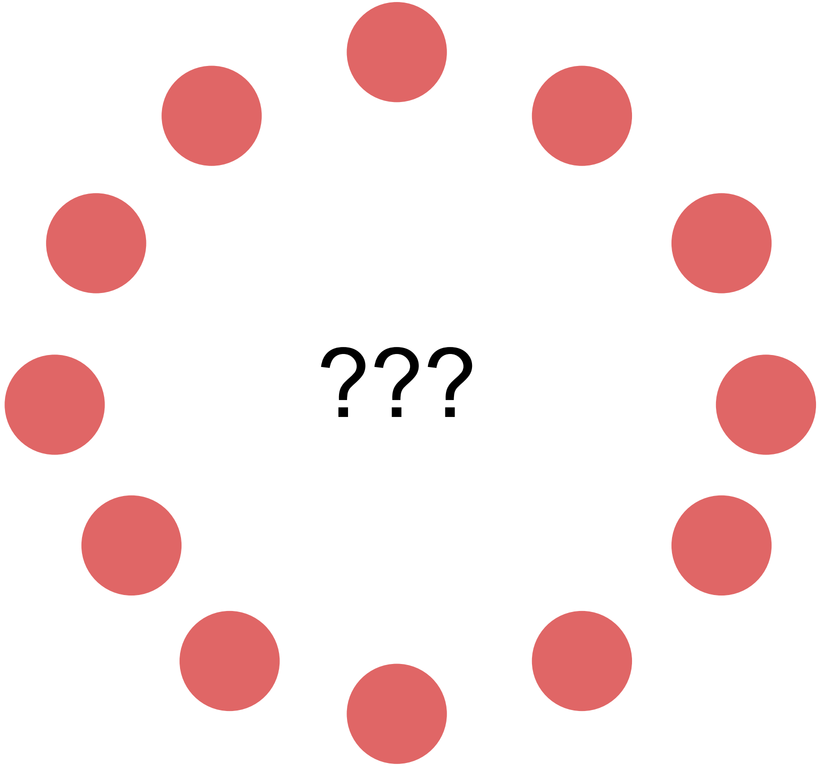




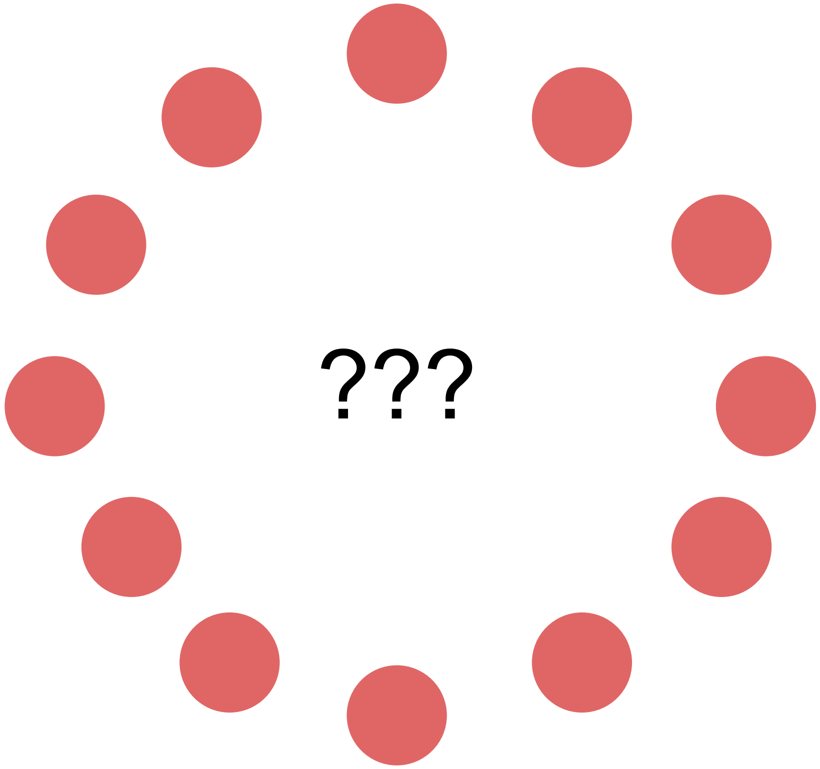
```
d = model.predict_proba({'VOM' : 1, 'BLOAT' : 1, 'LE' : 1})
print "\t".join( "{:7}".format(state.name) for state in model.states )
print "\t".join( "{:4.2}".format(model.parameters[0][1]) for model in d )
```

```
BRCA1   BRCA2   LCT     OC      LI      PREG    LE      BLOAT   LOA     VOM     AC
0.056   0.68    0.087   0.91    0.096    0.2     1.0     1.0    0.52     1.0    0.24
```

Sometimes we want to learn structure from data



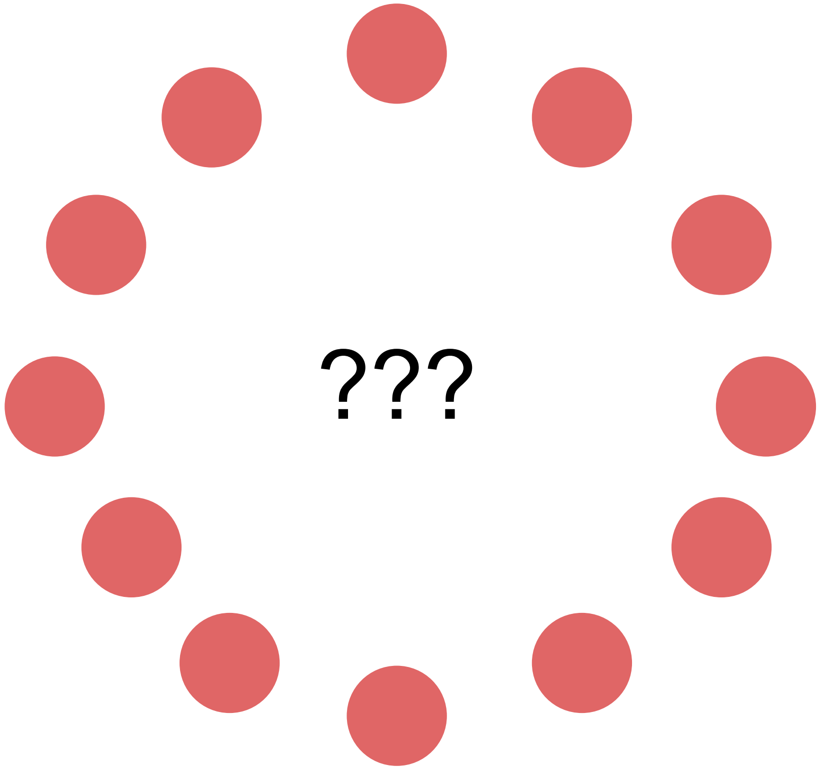
Sometimes we want to learn structure from data



Three primary ways:

- “Search and score” / Exact
- “Constraint Learning” / PC
- Heuristics

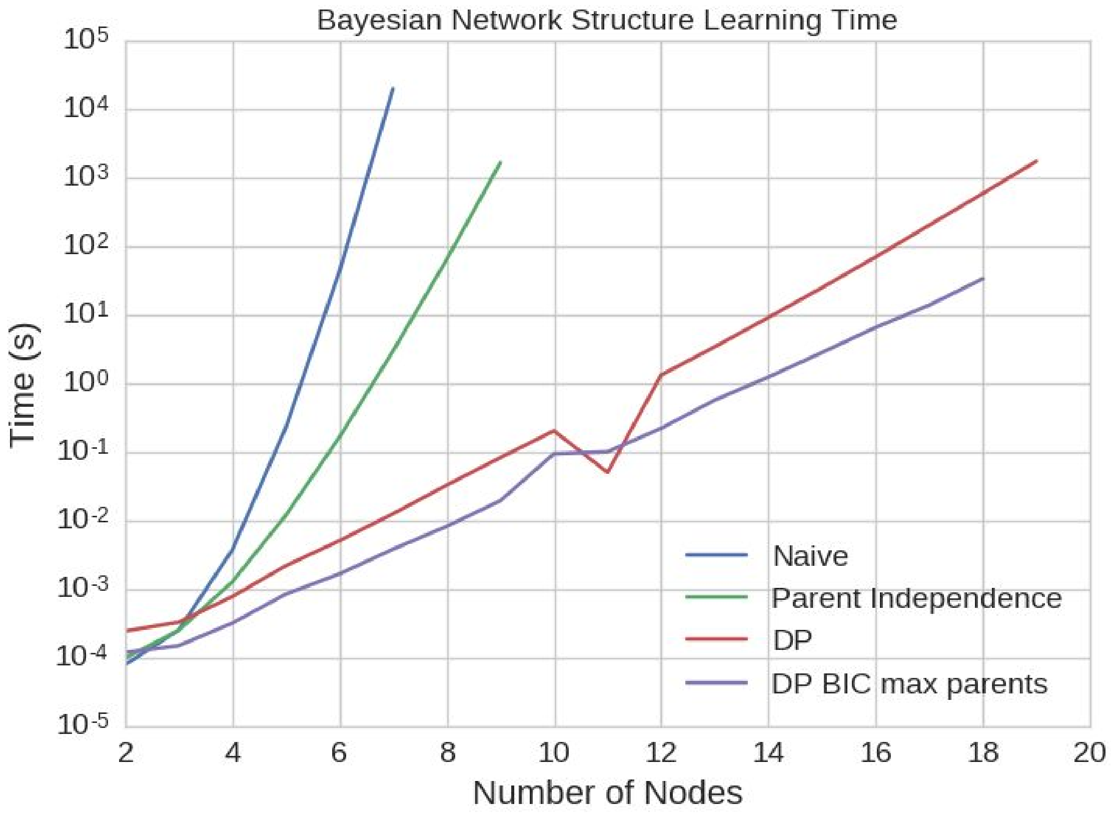
Sometimes we want to learn structure from data



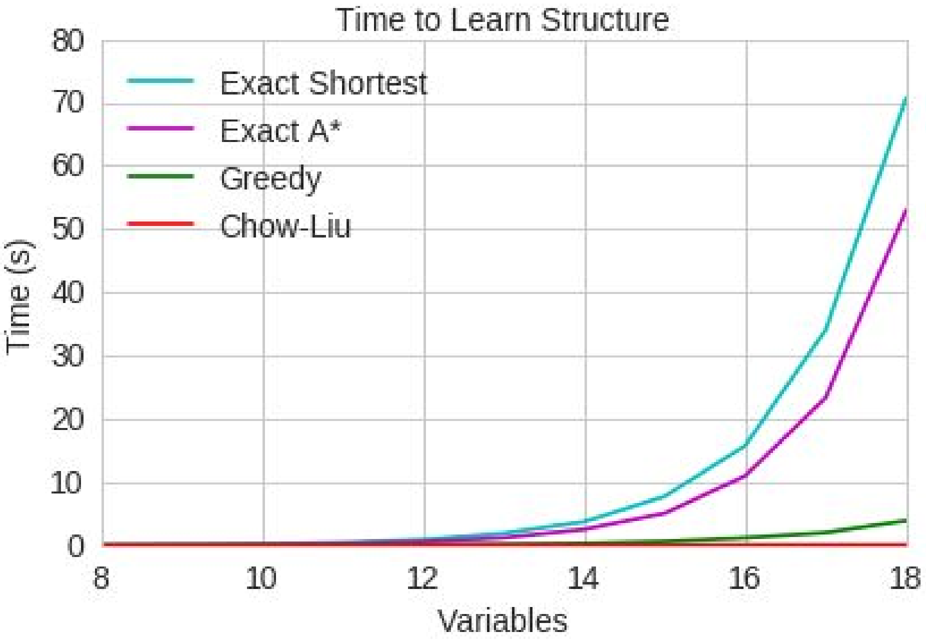
pomegranate supports

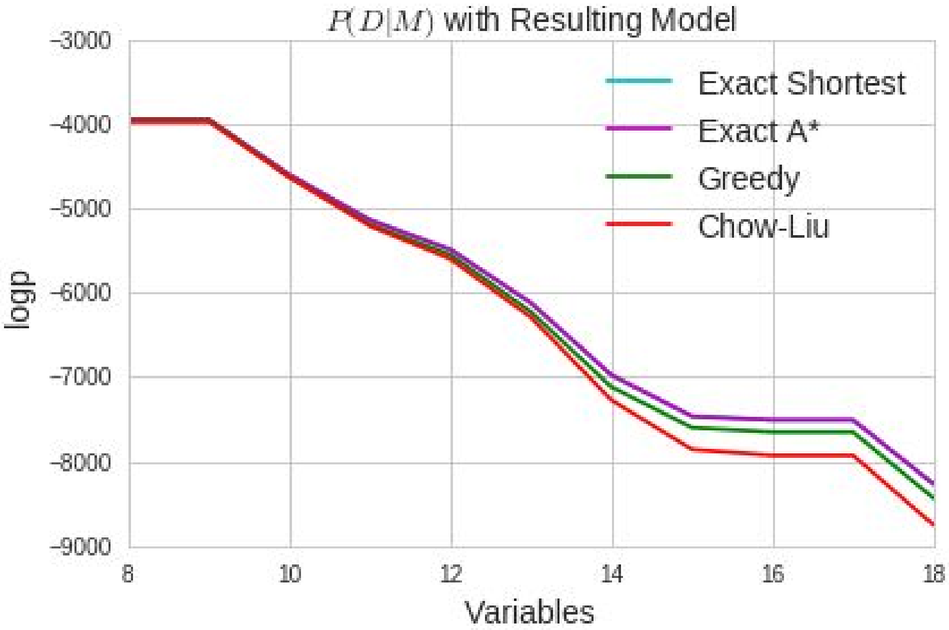
- “Search and score” / Exact
- “Constraint Learning” / PC
- Heuristics

Structure learning can be super-exponential in time

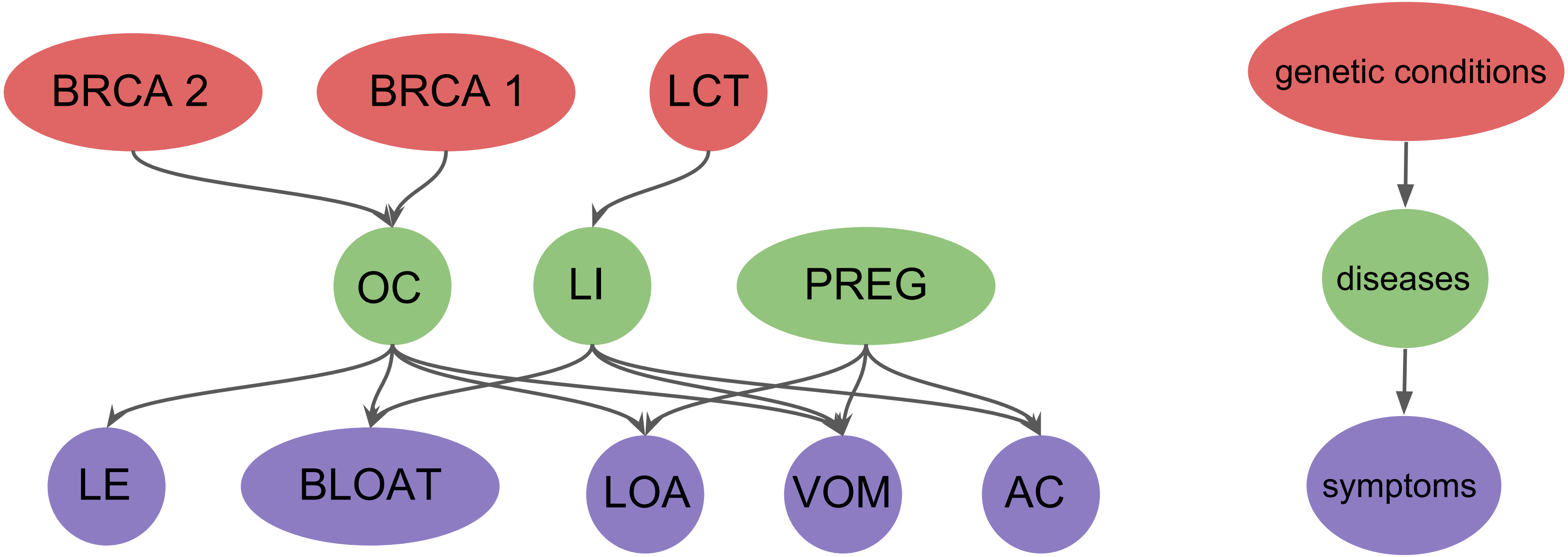


pomegranate supports 4 structure learning algos

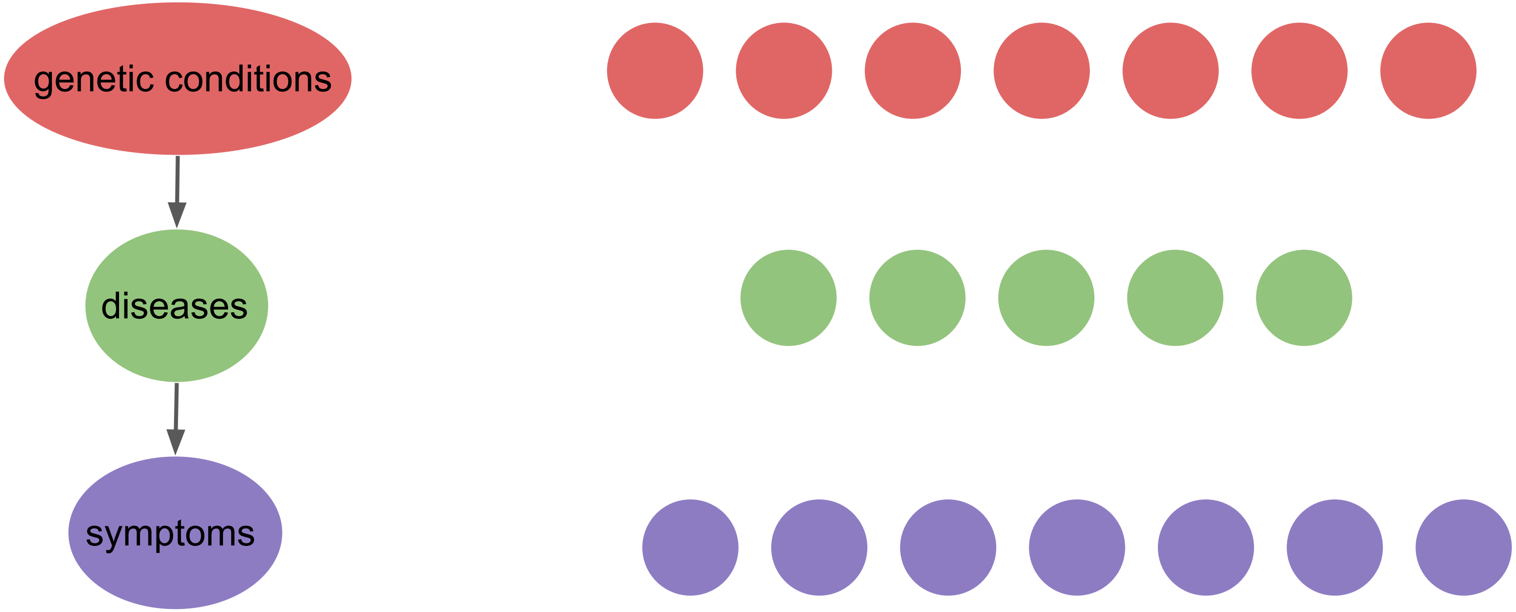




Constraint graphs can merge expert knowledge with data

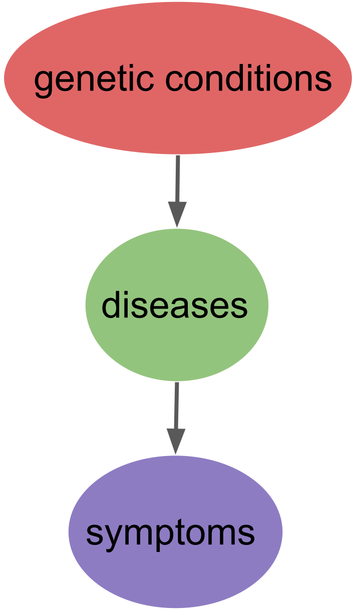


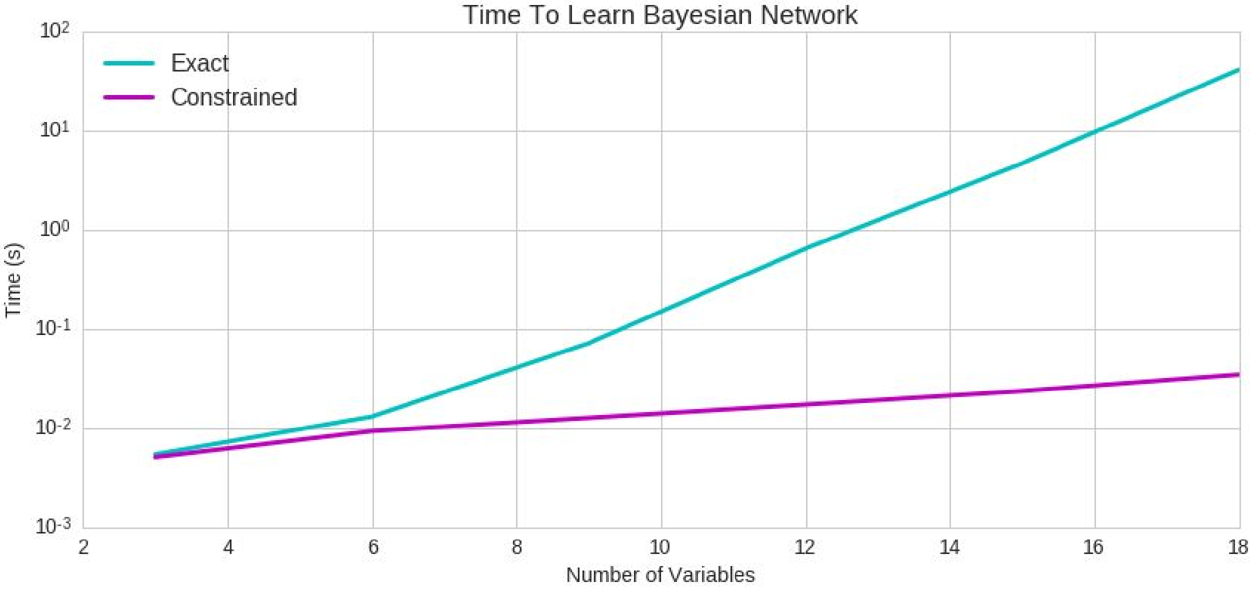
Structure learning with Constraint Graphs



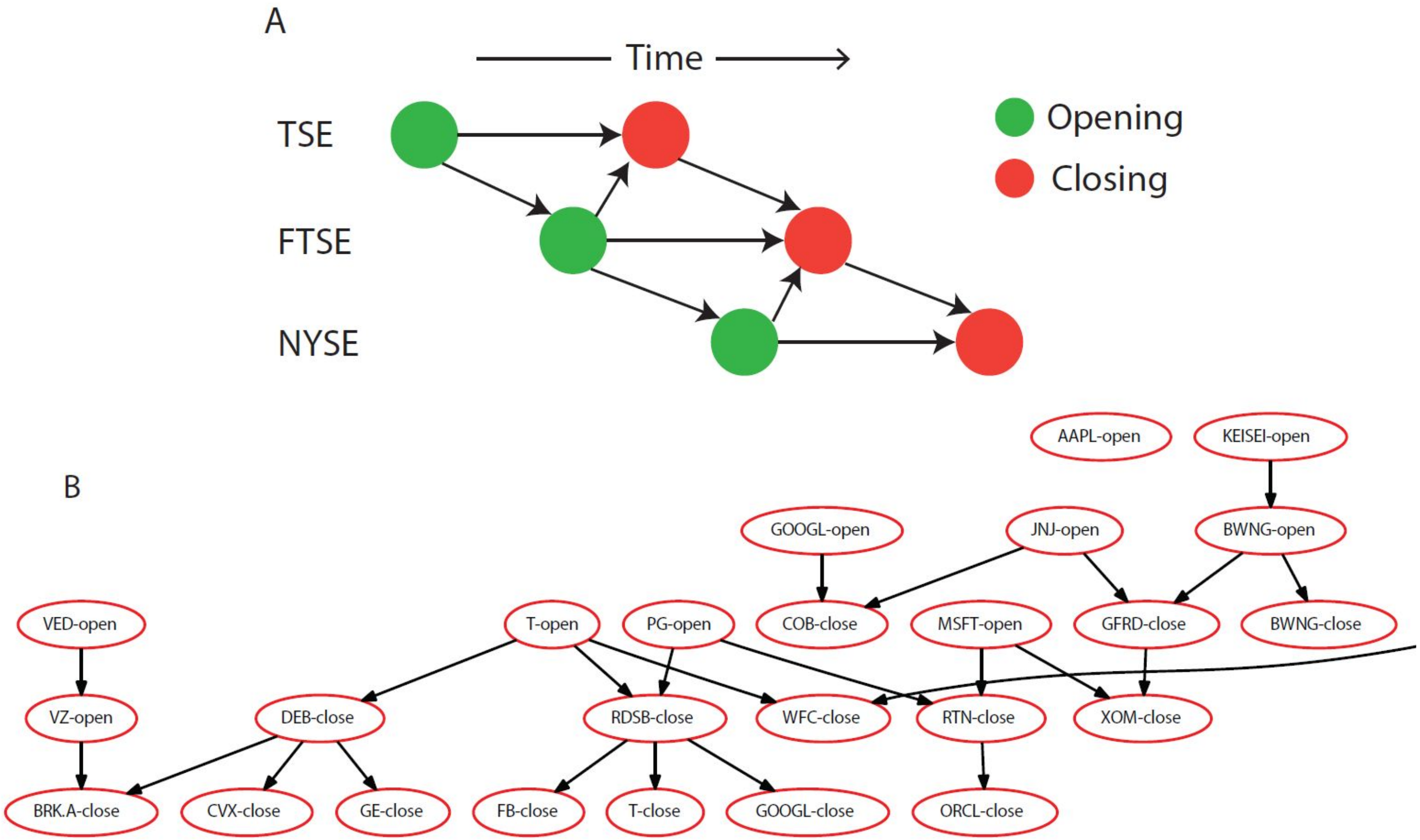
Structure learning with constraint graphs

Constraint graphs can also encode possible dependencies as layers.





Constraint graphs can model the global stock market



Overview

The API

Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

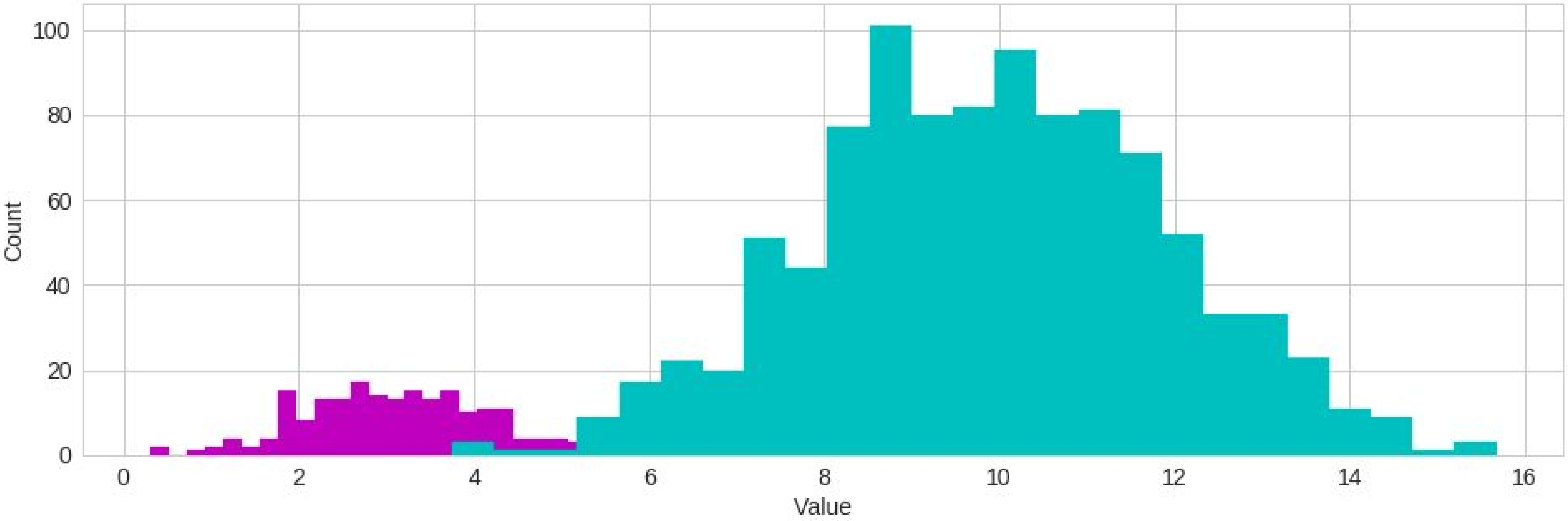
Finale: Train a mixture of hidden markov models in parallel

Bayes' Rule

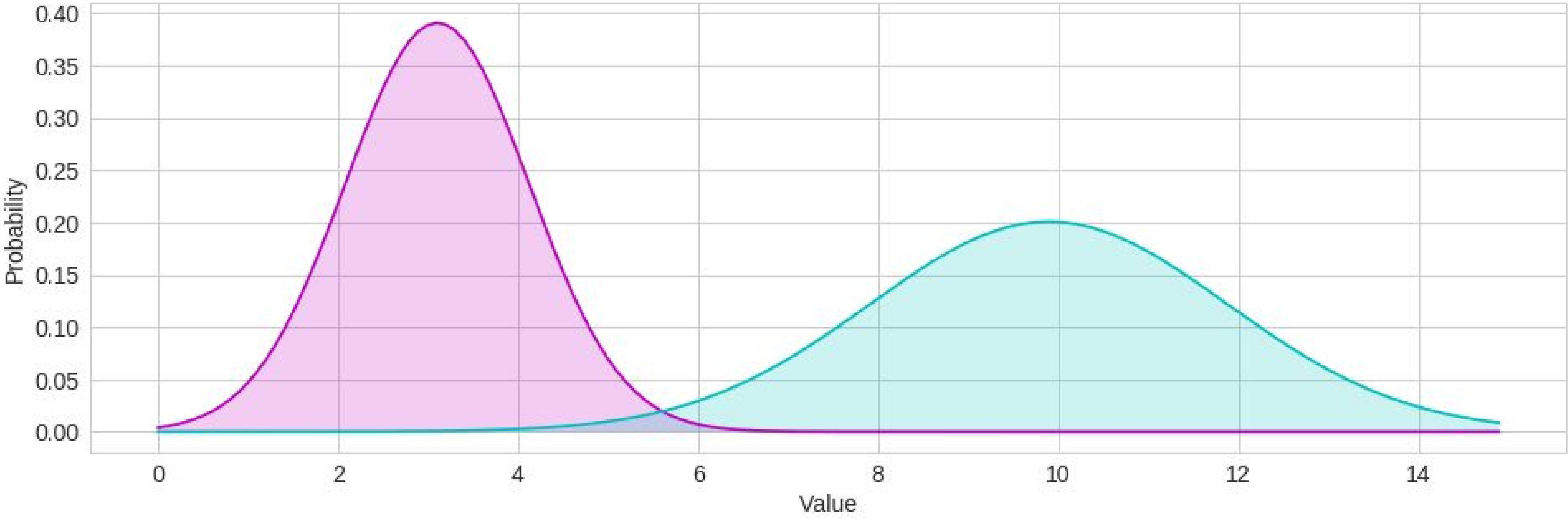
$P(M|D) = P(D|M)P(M) / P(D)$

Posterior = Likelihood * Prior / Normalization

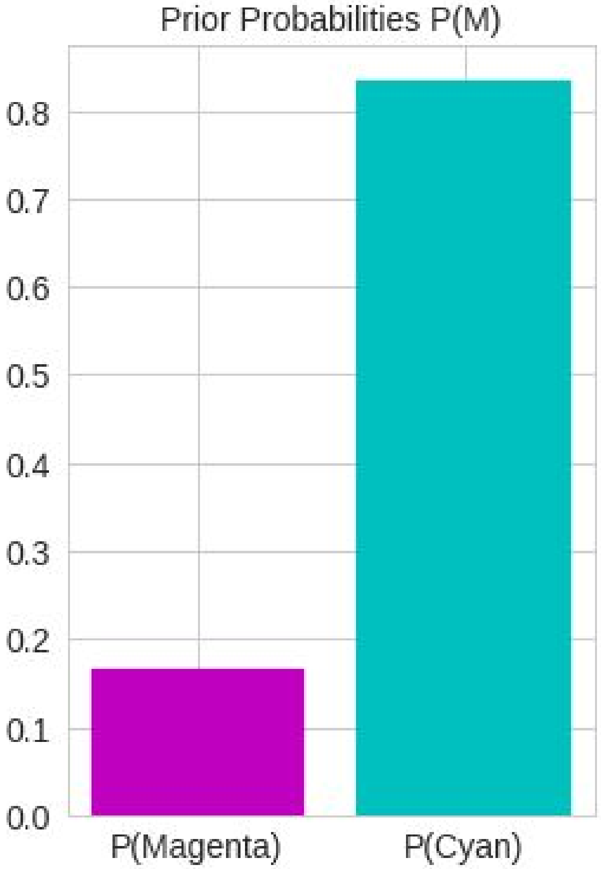
Let’s build a simple classifier on this data



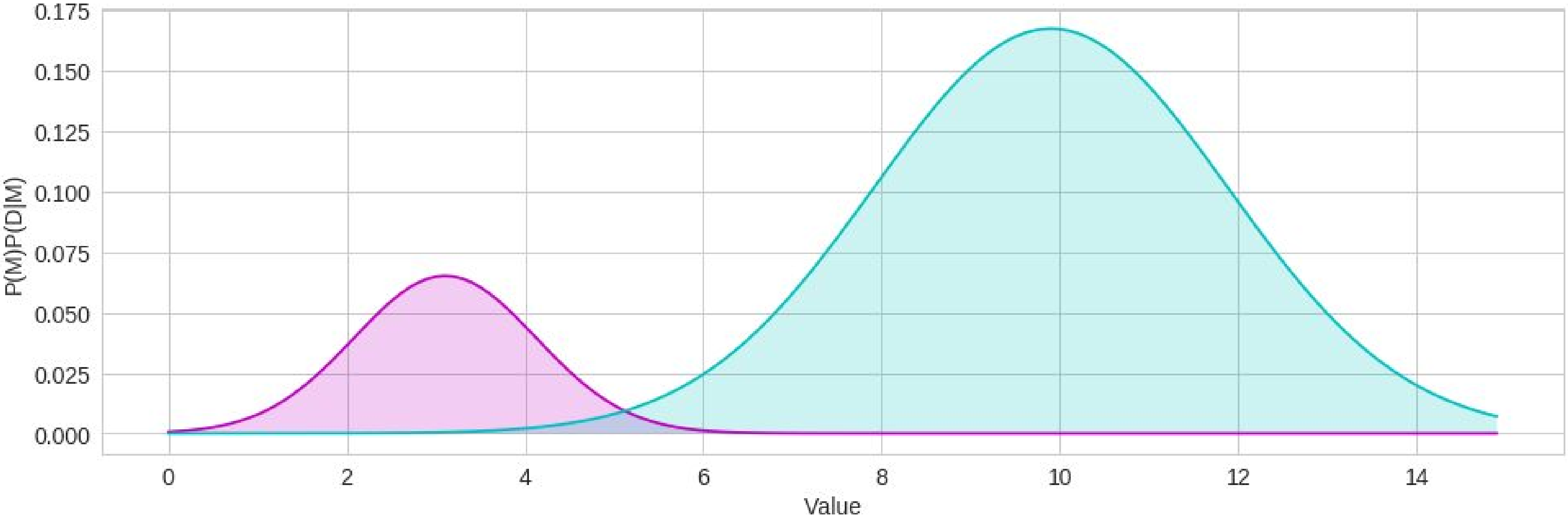
The likelihood function itself ignores class imbalance



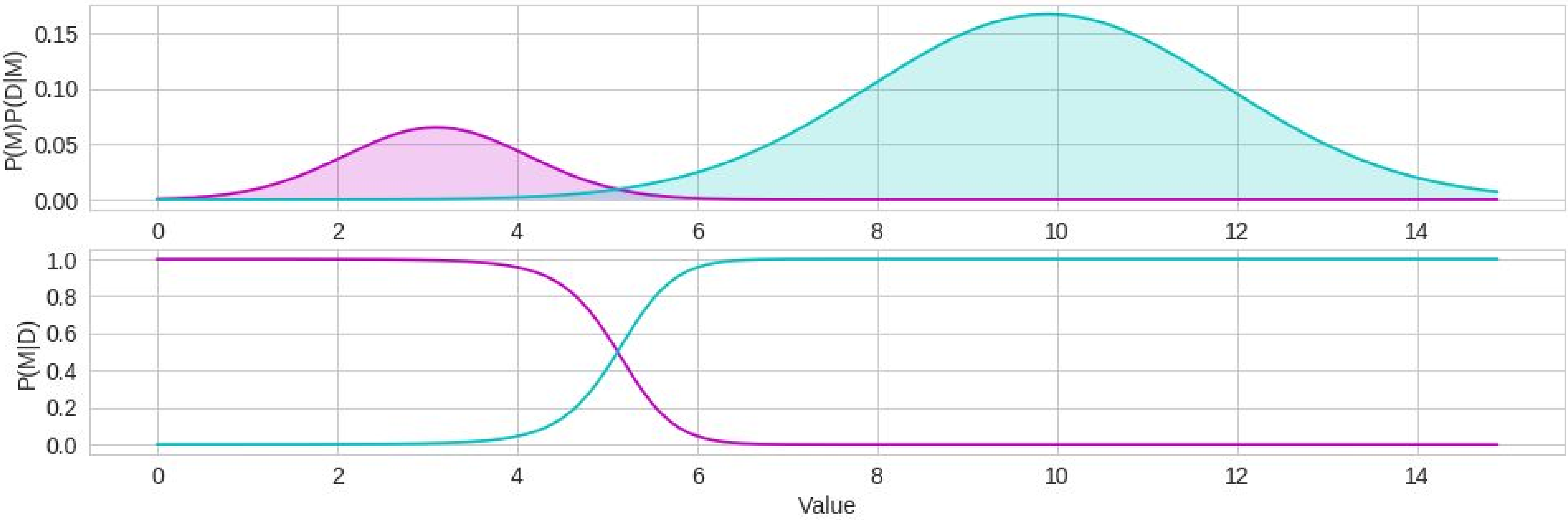
The prior probabilities can model class imbalance



The posterior models the original data more faithfully



The ratio of the posterior is a good classifier



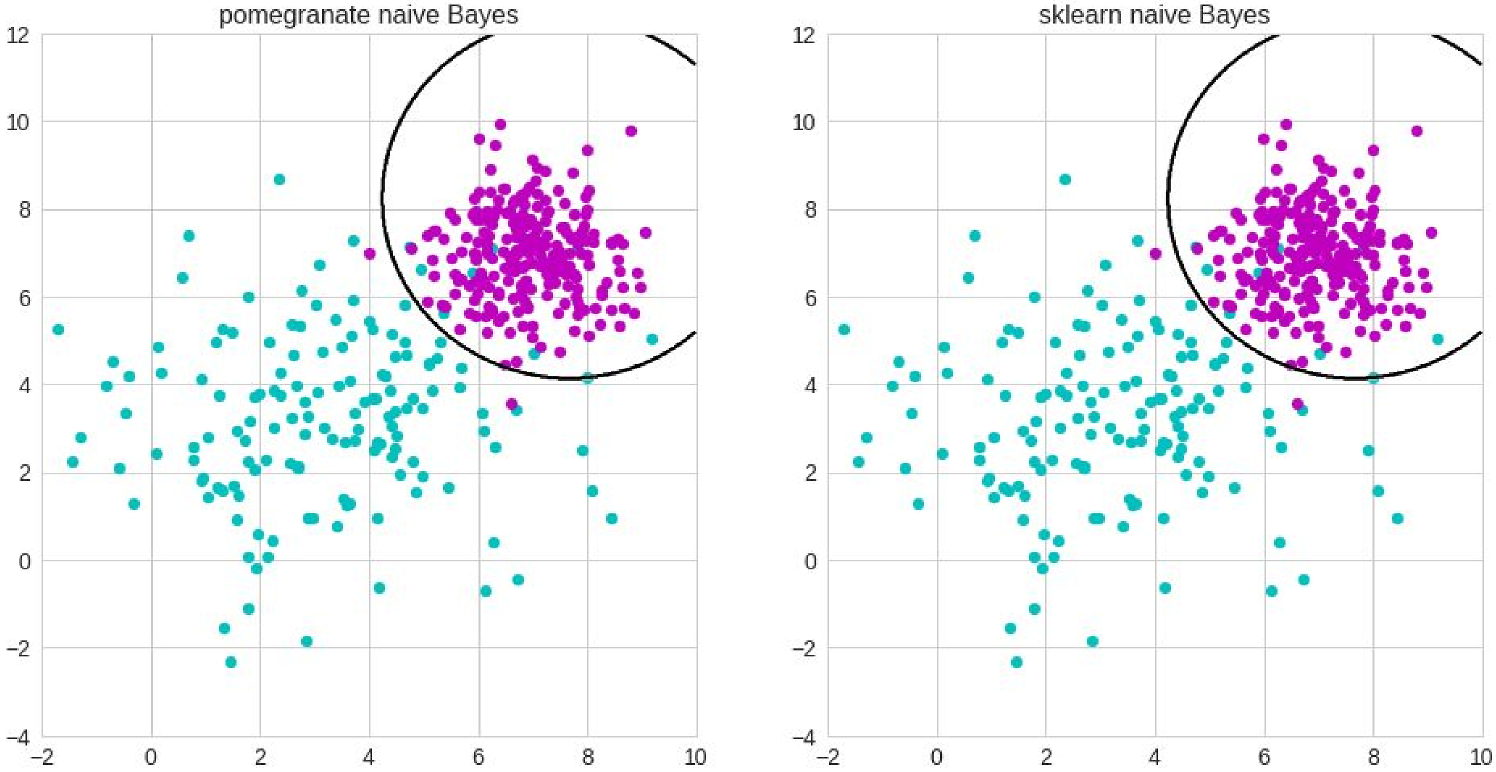
```
model = NaiveBayes.from_samples(NormalDistribution, X, y)
posteriors = model.predict_proba(idxs)
```

Naive Bayes assumes all dimensions are independent

$P(M|D) = \prod P(D|M)\ P(M)\ /\ P(D)$

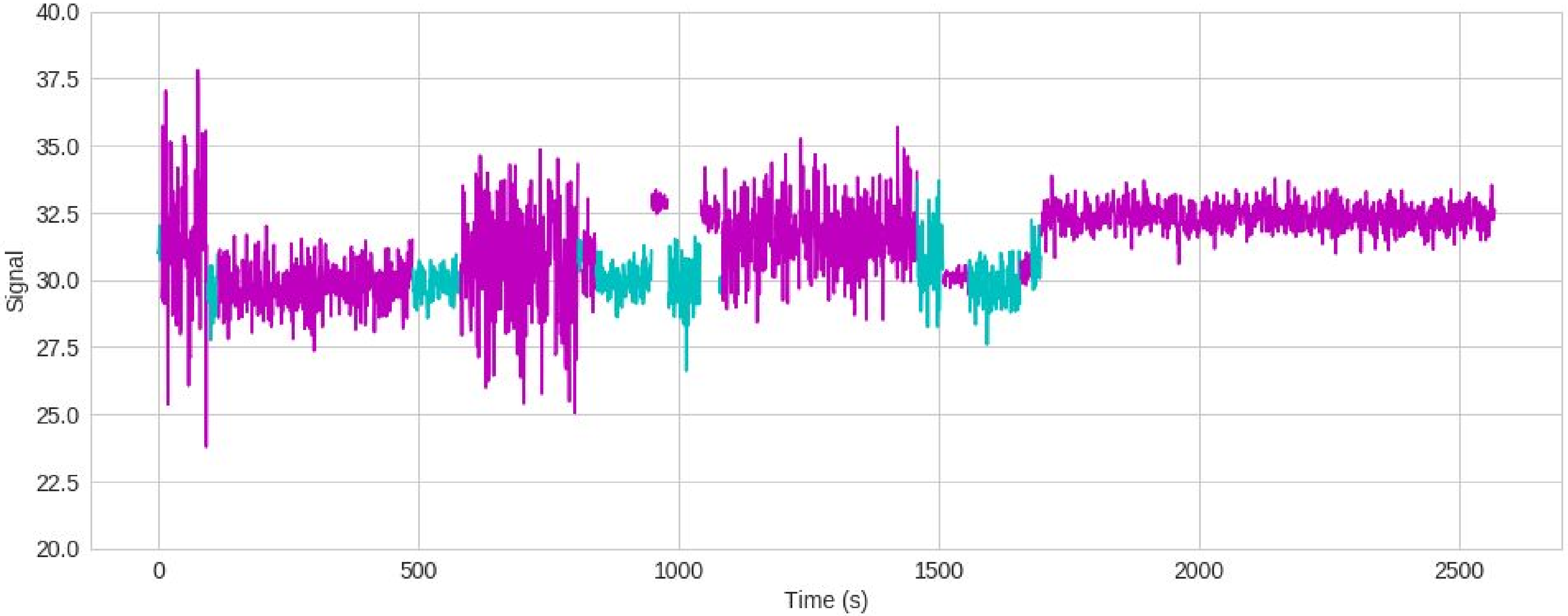
Posterior = Likelihood * Prior / Normalization

Gaussian naive Bayes produces spherical distributions

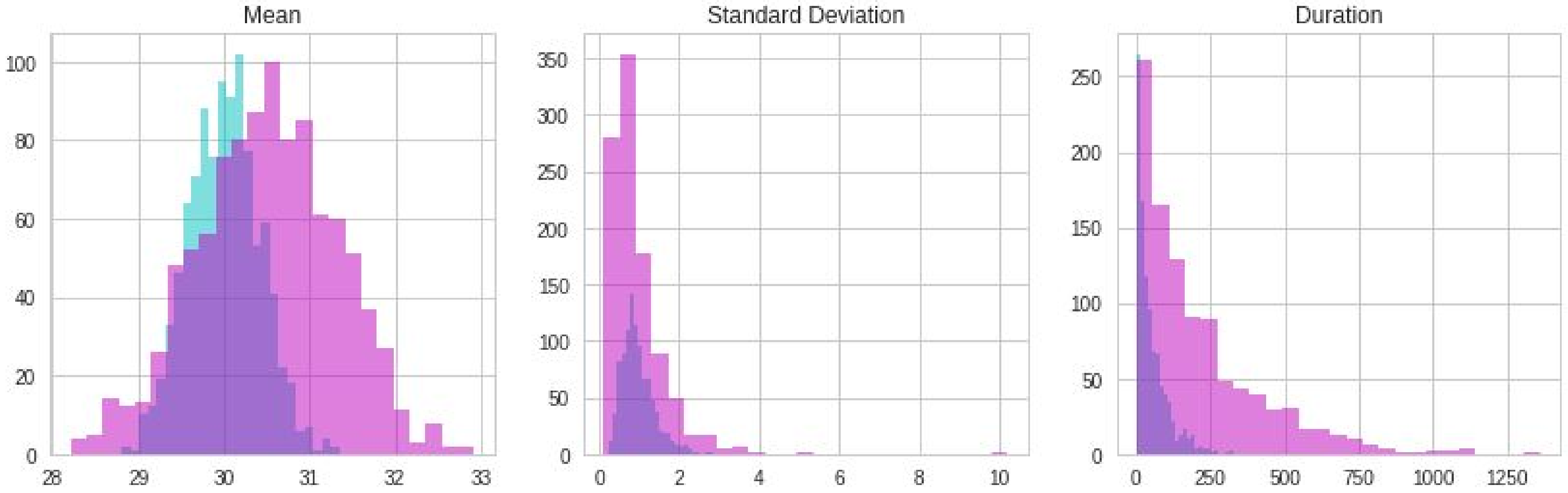


model = NaiveBayes.from_samples(NormalDistribution, X, y)

Naive Bayes does not need to be homogenous



Different features fall under different distributions

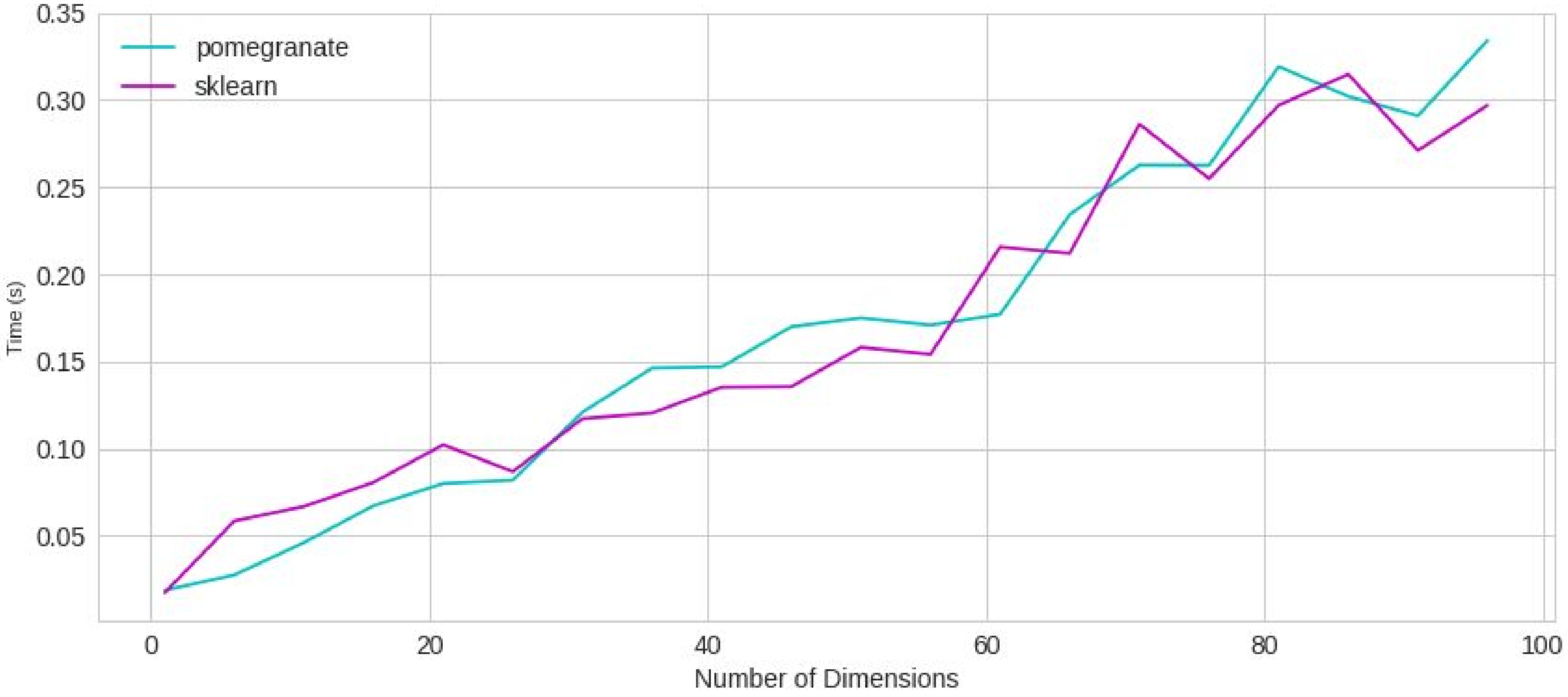


Explicitly modeling these distributions yields better classifiers

```
model = NaiveBayes.from_samples(NormalDistribution, X_train, y_train)
print "Gaussian Naive Bayes: ", (model.predict(X_test) == y_test).mean()

clf = GaussianNB().fit(X_train, y_train)
print "sklearn Gaussian Naive Bayes: ", (clf.predict(X_test) == y_test).mean()

model = NaiveBayes.from_samples([NormalDistribution, LogNormalDistribution,
ExponentialDistribution], X_train, y_train)
print "Heterogeneous Naive Bayes: ", (model.predict(X_test) == y_test).mean()
```

```
Gaussian Naive Bayes:  0.798
sklearn Gaussian Naive Bayes:  0.798
Heterogeneous Naive Bayes:  0.844
```

pomegranate is just as fast as sklearn

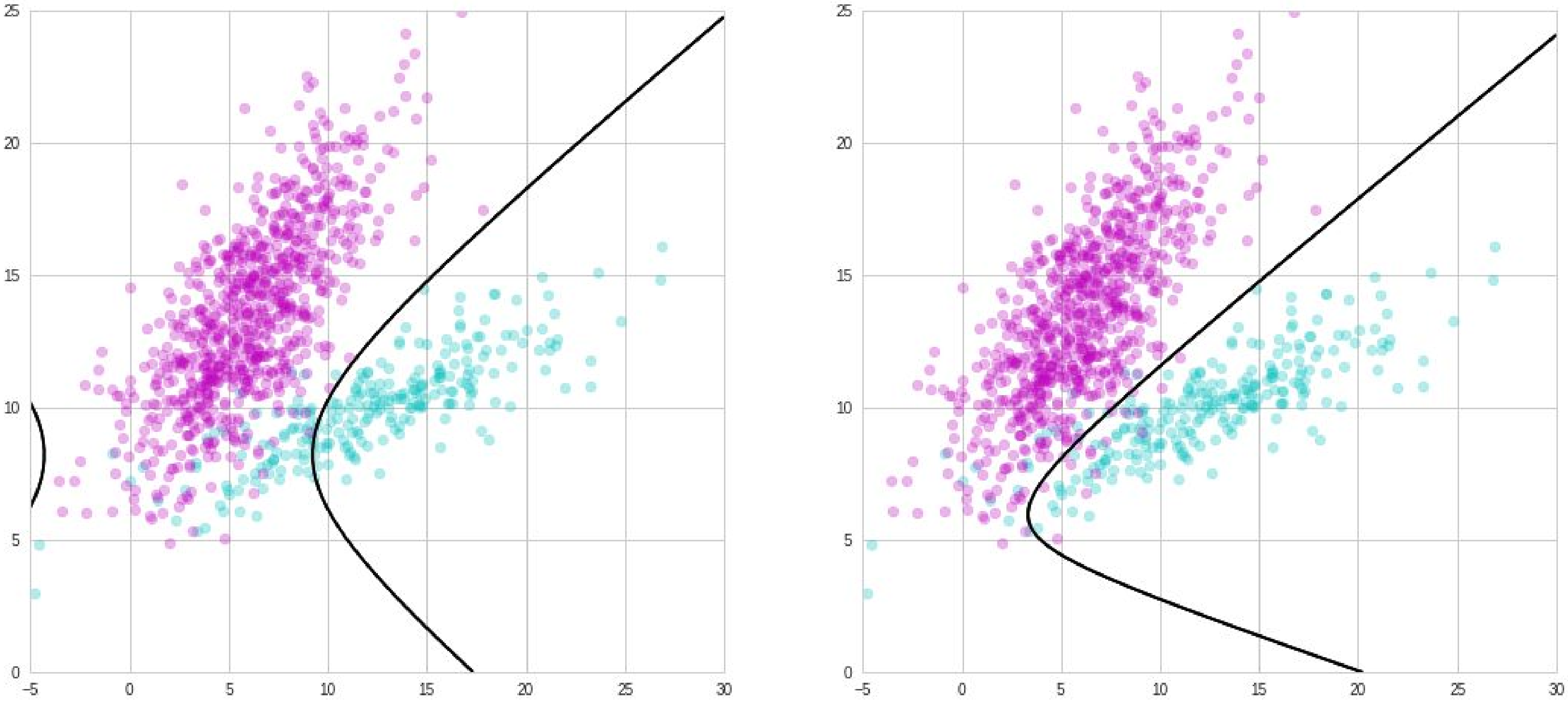


Bayes Classifiers are more general than naive Bayes

$P(M|D) = P(D|M)\ P(M)\ /\ P(D)$

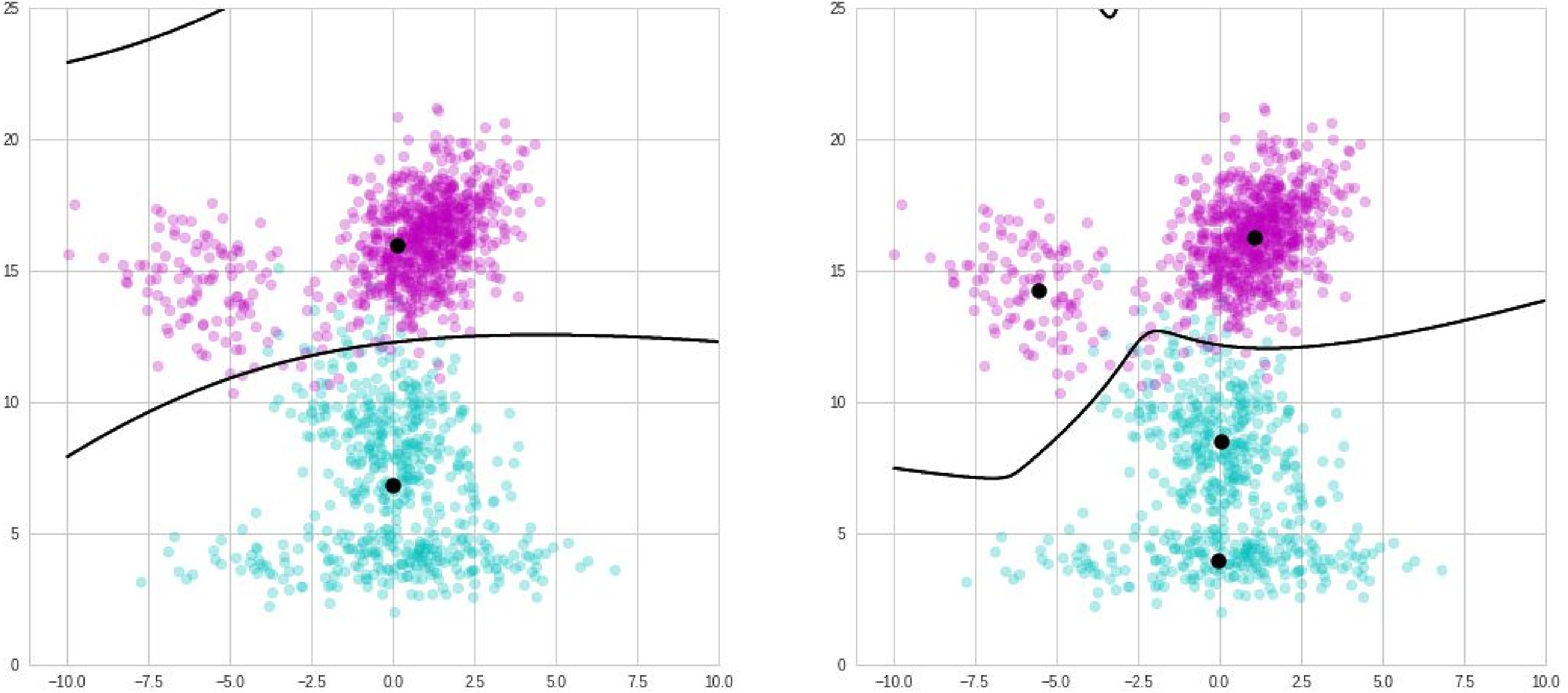
Posterior = Likelihood * Prior / Normalization

Gaussian Bayes Classifiers model the full covariance



naive training accuracy: 0.9286
bayes classifier training accuracy: 0.9657

Real data isn’t as clean (which is why we get paid)



Creating mixture model Bayes classifiers is simple

```
gmm_a = GeneralMixtureModel.from_samples(MultivariateGaussianDistribution, 2, X[y == 0])
gmm_b = GeneralMixtureModel.from_samples(MultivariateGaussianDistribution, 2, X[y == 1])
model_b = BayesClassifier([gmm_a, gmm_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

Creating any Bayes classifiers is simple

```
mc_a = MarkovChain.from_samples(X[y == 0])
mc_b = MarkovChain.from_samples(X[y == 1])
model_b = BayesClassifier([mc_a, mc_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

```
hmm_a = HiddenMarkovModel…
hmm_b = HiddenMarkovModel...
model_b = BayesClassifier([hmm_a, hmm_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

```
bn_a = BayesianNetwork.from_samples(X[y == 0])
bn_b = BayesianNetwork.from_samples(X[y == 1])
model_b = BayesClassifier([bn_a, bn_b], weights=numpy.array([1-y.mean(), y.mean()]))
```

Overview

The API

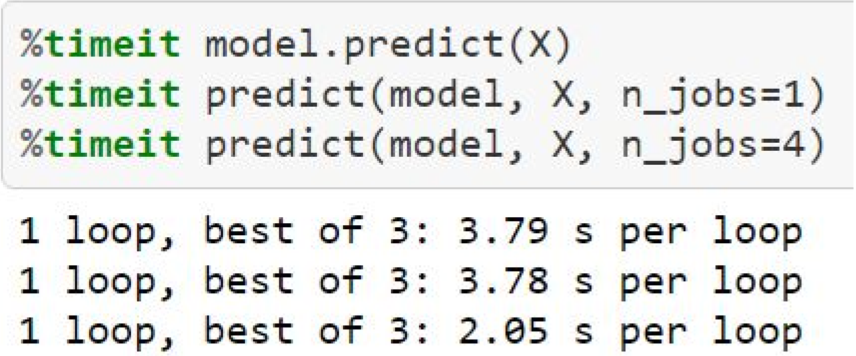
Major Models/Model Stacks

1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

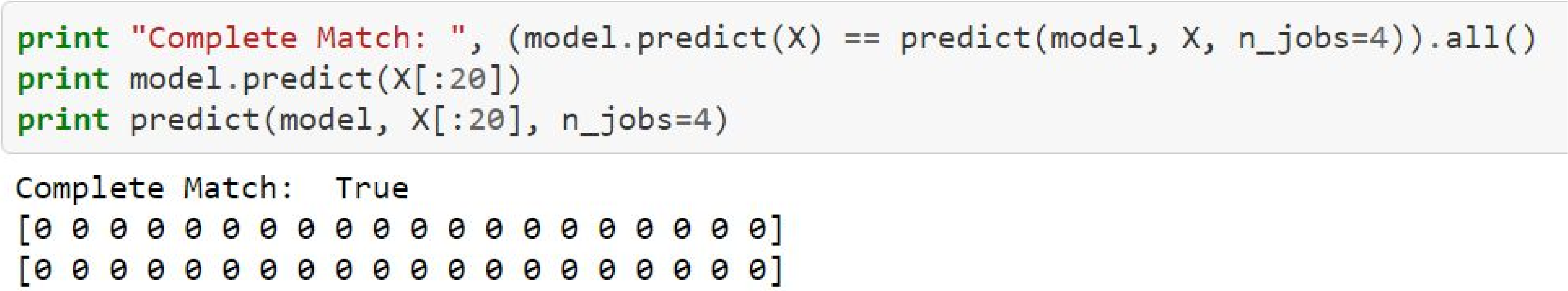
Finale: Train a mixture of hidden markov models in parallel

pomegranate has built in parallelization



```
%timeit model.predict(X)
%timeit predict(model, X, n_jobs=1)
%timeit predict(model, X, n_jobs=4)
```

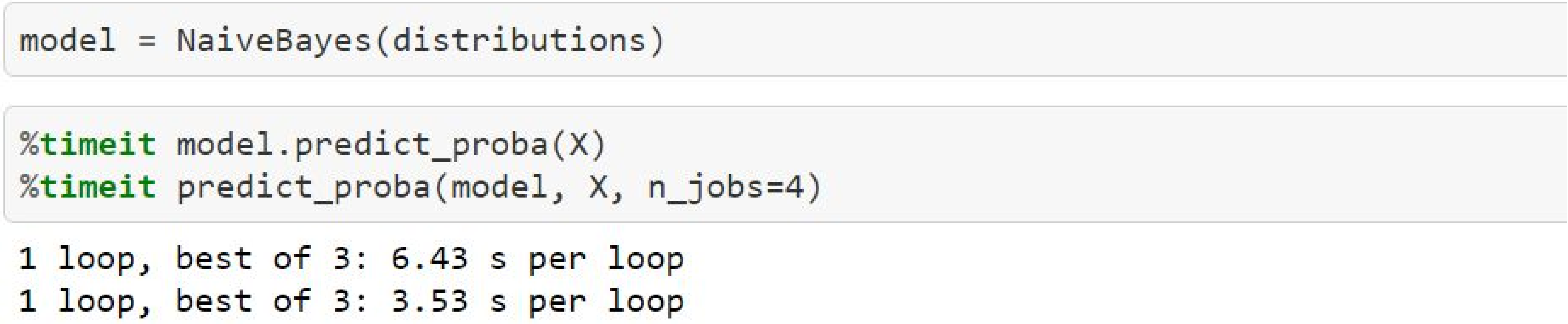
```
1 loop, best of 3: 3.79 s per loop
1 loop, best of 3: 3.78 s per loop
1 loop, best of 3: 2.05 s per loop
```



```
print "Complete Match: ", (model.predict(X) == predict(model, X, n_jobs=4)).all()
print model.predict(X[:20])
print predict(model, X[:20], n_jobs=4)
```

```
Complete Match:  True
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

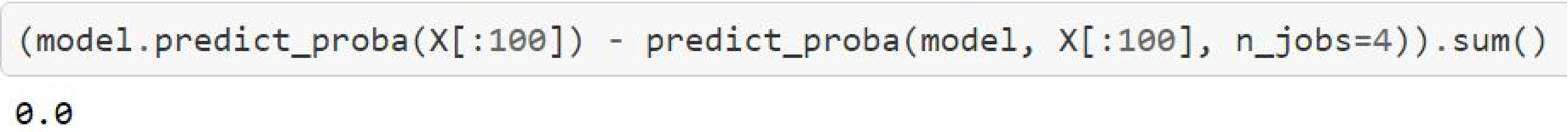
pomegranate has built in parallelization



```
model = NaiveBayes(distributions)
```

```
%timeit model.predict_proba(X)
%timeit predict_proba(model, X, n_jobs=4)
```

```
1 loop, best of 3: 6.43 s per loop
1 loop, best of 3: 3.53 s per loop
```



```
(model.predict_proba(X[:100]) - predict_proba(model, X[:100], n_jobs=4)).sum()
```

```
0.0
```

Overview

The API

Major Models/Model Stacks

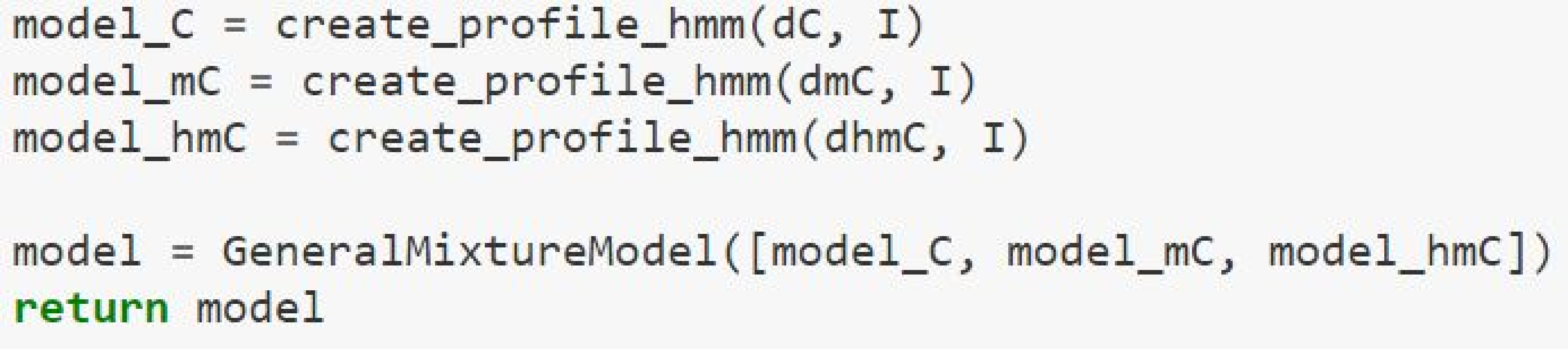
1. General Mixture Models
2. Hidden Markov Models
3. Bayesian Networks
4. Bayes Classifiers

Parallelization

Finale: Train a mixture of hidden markov models in parallel

Mixture of Hidden Markov Models

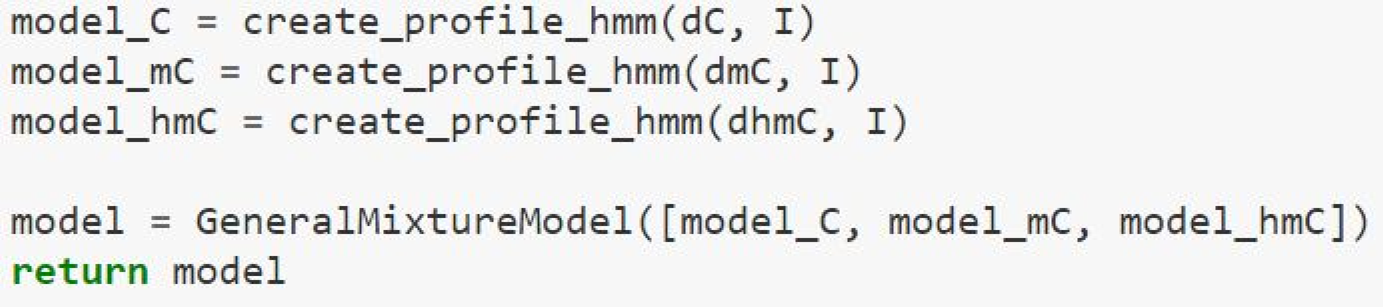
Creating a mixture of HMMs is just as simple as passing the HMMs into a GMM as if it were any other distribution



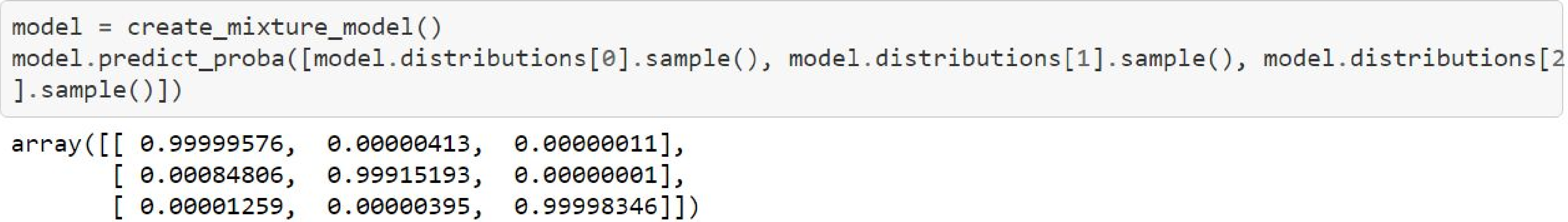
```
model_C = create_profile_hmm(dC, I)
model_mC = create_profile_hmm(dmC, I)
model_hmC = create_profile_hmm(dhmC, I)

model = GeneralMixtureModel([model_C, model_mC, model_hmC])
return model
```

parallel training of a mixture of hmms

Creation is just as simple as passing the HMMs into the GMM object. In this case, each model has 307 edges and 39 states to train



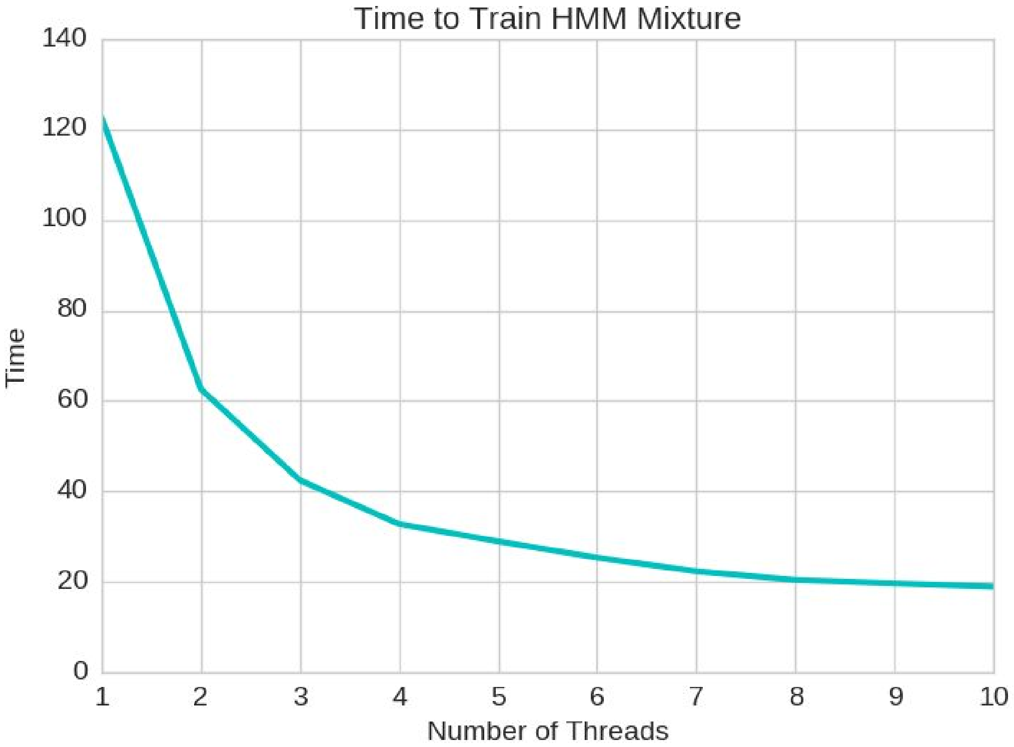
```
model_C = create_profile_hmm(dC, I)
model_mC = create_profile_hmm(dmC, I)
model_hmC = create_profile_hmm(dhmC, I)

model = GeneralMixtureModel([model_C, model_mC, model_hmC])
return model
```



```
model = create_mixture_model()
model.predict_proba([model.distributions[0].sample(), model.distributions[1].sample(), model.distributions[2
].sample()])
```

```
array([[ 0.99999576,  0.00000413,  0.00000011],
       [ 0.00084806,  0.99915193,  0.00000001],
       [ 0.00001259,  0.00000395,  0.99998346]])
```

Parallel Training of a Mixture of HMMs

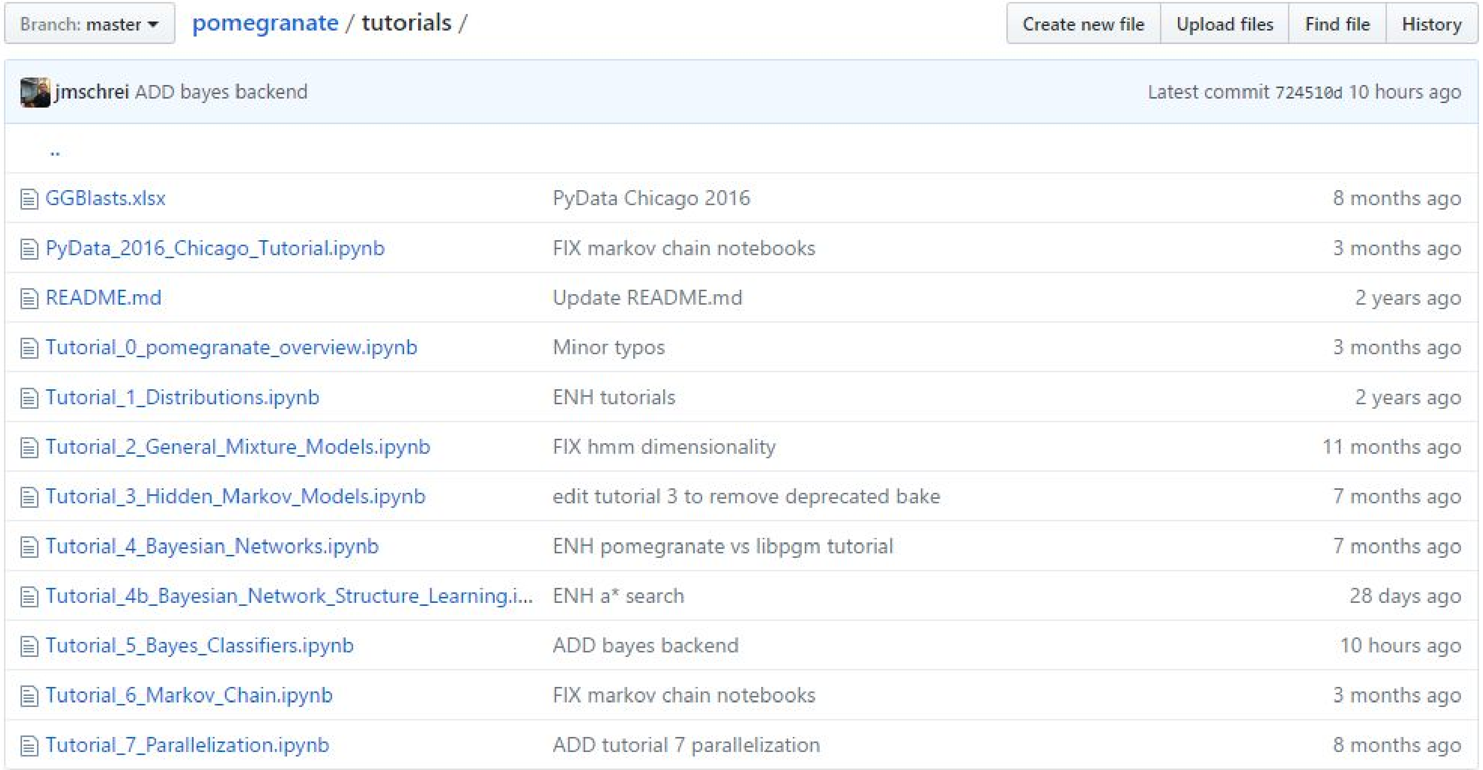


fit(model, X, n_jobs=n)

Overview

pomegranate can do more than other packages, faster, is intuitive to use, and can do it in parallel

Tutorials for each model are available on github



pomegranate / tutorials /

jmschrei ADD bayes backend Latest commit 724510d 10 hours ago

..

GGBlasts.xlsx	PyData Chicago 2016	8 months ago
PyData_2016_Chicago_Tutorial.ipynb	FIX markov chain notebooks	3 months ago
README.md	Update README.md	2 years ago
Tutorial_0_pomegranate_overview.ipynb	Minor typos	3 months ago
Tutorial_1_Distributions.ipynb	ENH tutorials	2 years ago
Tutorial_2_General_Mixture_Models.ipynb	FIX hmm dimensionality	11 months ago
Tutorial_3_Hidden_Markov_Models.ipynb	edit tutorial 3 to remove deprecated bake	7 months ago
Tutorial_4_Bayesian_Networks.ipynb	ENH pomegranate vs libpgm tutorial	7 months ago
Tutorial_4b_Bayesian_Network_Structure_Learning.i...	ENH a* search	28 days ago
Tutorial_5_Bayes_Classifiers.ipynb	ADD bayes backend	10 hours ago
Tutorial_6_Markov_Chain.ipynb	FIX markov chain notebooks	3 months ago
Tutorial_7_Parallelization.ipynb	ADD tutorial 7 parallelization	8 months ago

https://github.com/jmschrei/pomegranate/tree/master/tutorials

Thank you for your time.