pomegranate
fast and flexible probabilistic modelling in python

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

@jmschreiber91
@jmschrei
@jmschreiber91
Acknowledgements
pomegranate is more flexible than other packages, faster, is intuitive to use, and can do it all in parallel
Overview: supported models

Six Main Models:

1. Probability Distributions
2. General Mixture Models
3. Markov Chains
4. Hidden Markov Models
5. Bayes Classifiers / Naive Bayes
6. Bayesian Networks

Two Helper Models:

1. k-means++/kmeans||
2. Factor Graphs
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 made in two ways...

...from data

\[ d = \text{NormalDistribution}.\text{from}_\text{samples}(X) \]

...from known values

\[ d = \text{NormalDistribution}(5, 2.3) \]
Models can be made in two ways...

...from data

d = BayesianNetwork.from_samples(X)

...from known values

n1 = Node(...)  
n2 = Node(...)  
model = BayesianNetwork()  
model.add_nodes(n1, n2...)  
model.add_edges(...)

The API is common to all models

```python
model.log_probability(X) / model.probability(X)
model.sample()
model.fit(X, weights, inertia)  # All models have these methods!
model.summarize(X, weights)
model.from_summaries(inertia)
Model.from_samples(X, weights)  # All models composed of distributions (like GMM, HMM...) have these methods too!
model.predict(X)
model.predict_proba(X)
model.predict_log_proba(X)
```
Overview: model stacking in pomegranate

```python
GeneralMixtureModel.from_samples(NormalDistribution, n_components=3, X=X)

GeneralMixtureModel.from_samples(ExponentialDistribution, n_components=3, X=X)

BayesClassifier.from_samples(MultivariateGaussianDistribution, X, y)

d1 = GeneralMixtureModel.from_samples...
d2 = GeneralMixtureModel.from_samples...
model = BayesClassifier([d1, d2])
```
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(axis=0), numpy.cov(data, rowvar=False, bias=True)
print "\n" "pomegranate time:"
%timeit -n 10 MultivariateGaussianDistribution.from_samples(data)

numpy time:
10 loops, best of 3: 3.52 s per loop

pomegranate time:
10 loops, best of 3: 2.87 s per loop
pomegranate uses additive summarization

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

$$
\begin{align*}
\mu &= \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \\
\sigma^2 &= \frac{\sum_{i=1}^{n} w_i x_i^2}{\sum_{i=1}^{n} w_i} - \left( \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \right)^2
\end{align*}
$$
pomegranate supports out-of-core learning

Batches from a dataset can be reduced to additive summary statistics, enabling exact updates from data that can’t fit in memory.
Instead of going through the full dataset before updating parameters, one could update parameters at each step.

Dataset

On Disk

In Memory

Extract summaries

Sufficient Statistics

New Parameters
Multiple batches can be loaded at the same time and processed by different threads using `n_jobs` in either fitting or prediction methods.
Training a mixture of HMMs in parallel

```python
model.fit(X, n_jobs=n)
```
For many tasks, there is limited labeled data but a deluge of unlabeled data, and one wants to utilize both.
pomegranate supports semisupervised learning

Summaries from MLE on the labeled data can be added to summaries from EM on the unlabeled data.

- Labeled Data
- Unlabeled Data
- MLE Summaries
- EM Summaries
- Sufficient Statistics
- New Parameters

17
pomegranate supports semisupervised learning

Supervised Accuracy: 0.93
Test Data, Supervised Boundaries

Semisupervised Accuracy: 0.96
Test Data, Semi-supervised Boundaries
Many real world tasks involve data sets with missing data. The next version of pomegranate will include handling for all models by ignoring missing data, mean imputation, and EM imputation.
pomegranate uses Cupy for GPU support
pomegranate can be faster than scipy

```python
d = MultivariateGaussianDistribution(mu, cov)
X = numpy.random.randn(2000, 2000)
print "scipy time: ",
%timeit multivariate_normal.logpdf(X, mu, cov)
print "pomegranate time: ",
%timeit MultivariateGaussianDistribution(mu, cov).log_probability(X)
print "pomegranate time (w/ precreated object): ",
%timeit d.log_probability(X)
```

scipy time: 1 loop, best of 3: 1.67 s per loop
pomegranate time: 1 loop, best of 3: 801 ms per loop
pomegranate time (w/ precreated object): 1 loop, best of 3: 216 ms per loop
pomegranate uses aggressive caching

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

\[ \log P(X|\mu, \sigma) = -\log(\sqrt{2\pi\sigma}) - \frac{(x - \mu)^2}{2\sigma^2} \]

\[ \log P(X|\mu, \sigma) = \alpha - \frac{(x - \mu)^2}{\beta} \]
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.
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’?

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 work well
Beta distributions can model uncertainty
Beta distributions can model uncertainty
Beta distributions can model uncertainty
Overview: this talk

Overview

Major Models/Model Stacks

1. Hidden Markov Models
2. Bayes Classifiers
3. Bayesian Networks
CG enrichment detection HMM

GACTACGACTCGCGCTCGCACGTCGCTCGACATCATCGACA
CG enrichment detection HMM

GACTACGACTCGCGCTCGCACGTCGCTCGACATCATCGACA

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()
### Algorithms

<table>
<thead>
<tr>
<th>Feature</th>
<th>pomegranate</th>
<th>hmmlearn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent States</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Optional Explicit End State</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Sparse Implementation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Arbitrary Emissions Allowed on States</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Discrete/Gaussian/GMM Emissions</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Large Library of Other Emissions</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Build Model from Matrices</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Build Model Node-by-Node</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Serialize to JSON</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Serialize using Pickle/Joblib</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **Prior**
- **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**
GMM-HMM easy to define

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()
HMMs are faster than hmmlearn
Overview: this talk

Overview

Major Models/Model Stacks

1. Hidden Markov Models
2. Bayes Classifiers
3. Bayesian Networks
Naive Bayes produces ellipsoid boundaries

model = NaiveBayes.from_samples(NormalDistribution, X, y)
Naive Bayes assumes independent features

\[
Posterior = \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalization}}
\]

\[
P(M|D) = \frac{\prod_{i=1}^{d} P(D_i|M)P(M)}{\sum_{M} \prod_{i=1}^{d} P(D_i|M)P(M)}
\]
Data can fall under different distributions
Data can fall under different distributions
Data can fall under different distributions
Using appropriate distributions is better

dists = [LogNormalDistribution, PoissonDistribution, ExponentialDistribution, PoissonDistribution]

model1 = NaiveBayes.from_samples(NormalDistribution, X, y)
model2 = NaiveBayes.from_samples(dists, X, y)
model3 = GaussianNB().fit(X, y)

Gaussian Naive Bayes: 0.711
sklearn Gaussian Naive Bayes: 0.711
Heterogeneous Naive Bayes: 0.726
This additional flexibility is just as fast.
Bayes classifiers don’t require independence

naive accuracy: 0.929
bayes classifier accuracy: 0.966
Gaussian mixture model Bayes classifier
Overview:

Major Models/Model Stacks

1. Hidden Markov Models
2. Bayes Classifiers
3. Bayesian Networks
Bayesian networks are powerful inference tools which define a dependency structure between variables.

- **Sprinkler**
- **Rain**
- **Wet Grass**
Two main difficult tasks:
(1) Inference given incomplete information
(2) Learning the dependency structure from data
Three primary ways:

- “Search and score” / Exact
- “Constraint Learning” / PC
- Heuristics
Bayesian network structure learning

pomegranate supports:
- “Search and score” / Exact
- “Constraint Learning” / PC
- Heuristics
pomegranate supports four algorithms
**Global Parameter Independence:** The parents of some variable $A$ are independent of the parents of some variable $B$ given that they don’t form a cycle in the resulting graph.

*Easy! Tractable!*

*Hard! Exponential Time!*
Medical diagnosis Bayesian network

BRCA 2  →  BRCA 1  →  LCT

OC  ←  LE  ↔  BLOAT

LI  ←  LOA  ↔  VOM  ↔  AC

PREG
Constraint graphs merge data + knowledge

genetic conditions

diseases

symptoms
Global Parameter Independence: The parents of some variable A are independent of the parents of some variable B given that they don’t form a cycle in the resulting graph.
Constraint graphs merge data + knowledge

The parents of some variable A are independent of the parents of some variable B
Constraint graphs merge data + knowledge

- Genetic conditions
- Diseases
- Symptoms
Modeling the global stock market

A

Time

TSE

FTSE

NYSE

Opening

Closing

B

VED-open

VZ-open

DEB-close

BRKA-close

CVX-close

GE-close

FB-close

T-close

GOOGL-close

ORCL-close

AAPL-open

KESEI-open

GOOG-open

JNJ-open

GWIR-open

RTN-close

XOM-close
Finding the optimal Bayesian network given a constraint graph

Jacob M. Schreiber\textsuperscript{1} and William S. Noble\textsuperscript{2}

\textsuperscript{1} Department of Computer Science, University of Washington, Seattle, WA, United States of America
\textsuperscript{2} Department of Genome Science, University of Washington, Seattle, WA, United States of America

\section*{ABSTRACT}

Despite recent algorithmic improvements, learning the optimal structure of a Bayesian network from data is typically infeasible past a few dozen variables. Fortunately, domain knowledge can frequently be exploited to achieve dramatic computational savings, and in many cases domain knowledge can even make structure learning tractable. Several methods have previously been described for representing this type of structural prior...
pomegranate is more flexible than other packages, faster, is intuitive to use, and can do it all in parallel
What’s next?

Missing value support

Conversion from Cython to numba

Linear Gaussian Bayesian networks

Research in ancestral constraints for Bayesian network structure learning
pomegranate: fast and flexible probabilistic modeling in python

Jacob Schreiber
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98195
jmschr@cs.washington.edu

Abstract

We present pomegranate, an open source machine learning package for probabilistic modeling in Python. Probabilistic modeling encompasses a wide range of methods that explicitly describe uncertainty using probability distributions. Three widely used probabilistic models implemented in pomegranate are general mixture models, hidden Markov models, and Bayesian networks. A primary focus of pomegranate is to abstract away the complexities of training models from their definition. This allows users to focus on specifying the correct model for their
pomegranate is now NumFOCUS affiliated

---

**pomegranate**

pomegranate is a Python module for fast and flexible probabilistic modeling inspired by the design of scikit-learn. A primary focus of pomegranate is to abstract away the intricacies of a model from its definition, allowing users to easily prototype with complex models and training strategies. Its modular implementation allows for probability distributions to be swapped in or out for each other with ease and for models to be stacked within each other, yielding such delights as a mixture of Bayesian networks or a Gaussian mixture model Bayes classifier.

pomegranate is a python package which implements fast, efficient, and extremely flexible probabilistic models ranging from probability distributions to Bayesian networks to mixtures of hidden Markov models. The most basic level of probabilistic modeling is the a simple probability distribution. If we're modeling language, this may be a simple distribution over the frequency of all possible words a person can say.

1. Probability Distributions

The next level up are probabilistic models which use the simple distributions in more complex ways. A markov chain can extend a simple probability distribution to say that the probability of a certain word depends on the word(s) which have been said previously. A hidden Markov model may say that the probability of a certain words depends on the latent/hidden state of the previous word.
Tutorials available on github

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

<table>
<thead>
<tr>
<th>Branch: master</th>
<th>pomegranate / tutorials</th>
<th>Latest commit 72453ed 10 hours ago</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmschrei</td>
<td>ADD bayes backend</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GGBlasts.xlsx</td>
<td>PyData Chicago 2016</td>
<td>8 months ago</td>
</tr>
<tr>
<td>PyData_2016_Chicago_Tutorial.ipynb</td>
<td>FIX markov chain notebooks</td>
<td>3 months ago</td>
</tr>
<tr>
<td>README.md</td>
<td>Update README.md</td>
<td>2 years ago</td>
</tr>
<tr>
<td>Tutorial_0_pomegranate_overview.ipynb</td>
<td>Minor typos</td>
<td>3 months ago</td>
</tr>
<tr>
<td>Tutorial_1_Distributions.ipynb</td>
<td>ENH tutorials</td>
<td>2 years ago</td>
</tr>
<tr>
<td>Tutorial_2_General_Mixture_Models.ipynb</td>
<td>FIX hmm dimensionality</td>
<td>11 months ago</td>
</tr>
<tr>
<td>Tutorial_3_Hidden_Markov_Models.ipynb</td>
<td>edit tutorial 3 to remove deprecated bake</td>
<td>7 months ago</td>
</tr>
<tr>
<td>Tutorial_4_Bayesian_Networks.ipynb</td>
<td>ENH pomegranate vs libpgm tutorial</td>
<td>7 months ago</td>
</tr>
<tr>
<td>Tutorial_4b_Bayesian_Network_Structure_Learning.ipynb</td>
<td>ENH a* search</td>
<td>28 days ago</td>
</tr>
<tr>
<td>Tutorial_5_Bayes_Classifiers.ipynb</td>
<td>ADD bayes backend</td>
<td>10 hours ago</td>
</tr>
<tr>
<td>Tutorial_6_Markov_Chain.ipynb</td>
<td>FIX markov chain notebooks</td>
<td>3 months ago</td>
</tr>
<tr>
<td>Tutorial_7_Parallelization.ipynb</td>
<td>ADD tutorial 7 parallelization</td>
<td>8 months ago</td>
</tr>
</tbody>
</table>
Pomegranate implements probabilistic models that do not require samplers perform inference with, whereas these packages focus on the implementation of efficient samplers

Model hyperparameters in pomegranate are numbers, whereas they are typically distributions in these other packages. This allows uncertainty in model parameters to be explicitly captured.

Pomegranate focuses on discrete latent state (but discrete/continuous observed state) whereas these focus on continuous latent state