Probabilistic Programming Languages: *Bayesian Inference*

Holakou Rahmanian & Parameswaran Raman

University of California, Santa Cruz

(YET ANOTHER) HISTORY OF LIFE AS WE KNOW IT...
Bayesian Inference and PPLs

\[ y|\theta \sim \text{Binomial}(n, \theta) \]
\[ \theta \sim \text{Beta}(\alpha_0, \beta_0) \]

Model:

\[
\begin{array}{c|c}
\# & y \\
\hline
1 & 12 \\
2 & 10 \\
3 & 8 \\
4 & 17 \\
5 & 40 \\
6 & 8 \\
7 & 2 \\
\end{array}
\]

Data:

\[ \theta \sim \text{Beta}(\alpha_1, \beta_1) \]
Developed at MIT in 2008 - named for computation pioneer Alonzo Church

Universal language for describing stochastic generative processes

Based on the Lisp model of $\lambda$-calculus

Different implementations:  *Webchurch, Bher Church, MIT-Church, Cosh*
- FACTORIE a.k.a “Factor graphs, Imperative, Extensible”, Developed at University of Massachusetts Amherst in 2009
- Written in and uses Scala as the programming language.
- Creating **Factor graphs**, estimating parameters and performing inference

```scala
object TutorialSimpleChain extends App {
  // Imports Inference Methods, other required types here ...
  import cc.factorie.infer.{ GibbsSampler, InferByBPC } 
  implicit val random = new scala.util.Random(0)
  object LabelDomain extends CategoricalDomain[String]
  class Label(val token: Token, s: String) extends LabeledCategoricalVariable(s) {
    ...
  }
  object FeaturesDomain extends CategoricalVectorDomain[String]
  class Features(val token: Token) extends BinaryFeatureVectorVariable[String] {
    ...
  }
  object model extends ChainModel[Label, Features, Token] {
    ...
    // The Document class implements documents as sequences of sentences and tokens.
    val document = new Document("The quick brown fox jumped over the lazy dog."")
    val tokenizer = new app.nlp.segment.DeterministicTokenizer
    val segmenter = new app.nlp.segment.DeterministicSentenceSegmenter
    segmenter.process(document)
    assertStringEquals(document.tokenCount, "10")
    assertStringEquals(document.sentencesCount, "1")
    // Label the tokens and initialize features
    document.tokens.foreach(t => t.attr += new Label(t, "A"))
    document.tokens.foreach(t => {
      val features = t.attr += new Features(t)
      features += "W" = t.string.toLowerCase
      features += "toUpper" = t.string(0).isUpper.toString
    })
    val summary = InferByBPC.infer(document.tokens.toSeq.map(_.attr[Label]), model)
    assertStringEquals(summary.log2, "6.931471805599453")
    assertStringEquals(summary.marginal(document.tokens.head.attr[Label]), proportions,
                       "Proportions(0.499999999999994,0.499999999999994)")
  }
```

**Graphical Model Representation**

**Setup the model**

**Inference**

**Factor Graph Representation**
PPLs Comparison

**BUGS**
- Gibbs Sampling, Propositional Logic
- Pros: Simple
- Cons: Not scalable

**Infer.NET**
- Gibbs, EP, Variational Message Passing
- Pros: OOP, Scalable, Great Documentation
- Cons: Unrolling slows down inference

**Church**
- Metropolis-Hasting, Lisp, λ-calculus, generative
- Pros: Higher-order logic, Representational flexibility
- Cons: Inference complexity, inefficient implementations

**FACTORIE**
- Imperative, Discriminative Models, full support for NLP pipeline
- Pros: OOP, Scalable, Parallelizable
- Cons: No support for Contin Random Vars, Insufficient Documentation