FACTORIE: Probabilistic Programming with Imperatively Factor Graphs

Karl Schultz

http://factorie.googlecode.com/

Department of Computer Science
University of Massachusetts Amherst

Joint work with Andrew McCallum, Sameer Singh, Michael Wick, Sebastian Riedel.
Outline

• Motivation + Some Basics
• Overview of Conditional Random Fields
• Probabilistic Programming
• FACTORIE
• Joint Model of Segmentation and Entity Resolution (and other results)
• Future Work
Outline

- Motivation + Some Basics
- Overview of Conditional Random Fields
- Probabilistic Programming
- FACTORIE
- Joint Model of Segmentation and Entity Resolution (and other results)
- Future Work
Motivation

• There are many problems that are too difficult to solve by hand or too expensive
  – E.g., document classification, genetic sequence alignment, object detection in images

• Most problems are high-dimensional with a lot of unknown variables, in addition to inherent ambiguity
  – E.g., images have thousands of pixels; some documents belong to several classes
Motivation (cont.)

• We want methods that are applicable to a wide range of problems

• Probability Theory
  – Need a method that is flexible with regard to the types of probabilistic relations it can capture
  – We also want an easy and natural way to inject domain knowledge

• Graphical models to the rescue …
Graphical Models

• Basic framework:
  – A set of observed/input variables: \( x \)
  – A set of predicted/output variables: \( y \)
  – A set of edges connecting these variables to form a graphical structure

• Why a graph?
  – Many problems have are naturally represented as a graph
    • E.g., part-of-speech tagging, image segmentation
  – We can easily capture conditional independence!
Types of Graphical Models

Directed

Undirected
Recall: Conditional Independence

• $x$ is conditionally independent of $y$ given $z$
  
  \[ x \perp y \mid z \iff p(x \mid z) = p(x \mid y, z) \]

• This generalizes to sets of variables
  
  \[ x \perp y \mid z \iff p(x \mid z) = p(x \mid y, z) \]

• We can get this information directly from the graphical structure
Why Conditional Independence?

- Reduces the space of probabilistic distributions that we will consider to model the data
  - Many computational advantages to this
- We often know, given a problem, which variables have dependencies and which are independent
  - snow today \(\perp\) weather 2 week’s ago | last 5 days
  - child genes \(\perp\) grandparent’s genes | parent’s genes
Notation

• We are interested in modeling probability distributions over sets of random variables: \( V = X \cup Y \)
  – \( X \) are the input/observed variables
  – \( Y \) are the predicted/output variables
• An assignment to \( X \) is denoted by \( x \)
• An assignment to a set \( A \subseteq X \) by \( x_A \)
Undirected Graphical Models

• Given a collection of subsets $A \subset X$ we define an undirected graphical model as the set of distributions that can be written as:

$$p(x, y) = \frac{1}{Z} \prod_A \Psi_A(x_A, y_A)$$

• $\Psi_A$ is called a factor, and it computes a scalar value over its inputs $x_A$ and $y_A$
  – Often called local or compatibility functions
Undirected Graphical Models

\[ p(x, y) = \frac{1}{Z} \prod_A \Psi_A(x_A, y_A) \]

- The constant \( Z \) is called the normalization factor, and is defined as:

\[ Z = \sum_{x,y} \prod_A \Psi_A(x_A, y_A) \]

- This guarantees that we have defined a valid probability distribution that sums to 1
Visualizing Factor Graphs

Diagram of a factor graph with nodes and connections.
Technical Note: Factor Ambiguity in Undirected Graphs
Factors and Exponential Family

- We assume that factors have the form:

\[
\Psi_A(x_A, y_A) = \exp \left\{ \sum_k \theta_{Ak} f_{Ak}(x, y) \right\}
\]

- \( \theta_{Ak} \) is a real-valued parameter vector
- \( \{f_{Ak}\} \) is a set of feature functions or sufficient statistics
Factors and Exponential Family

\[
\Psi_A(x_A, y_A) = \exp \left\{ \sum_k \theta_{Ak} f_{Ak}(x, y) \right\}
\]

• Why this form?
  – Easier to parameterize the model according to relevant feature functions
    • often results in many less parameters, reducing computational complexity
  – We are guaranteed that the family of distributions which factorize over \( \mathcal{V} \), parameterized by \( \theta \), is an exponential family
  – Mathematically convenient for many reasons
  – Does not alter the representational power
Outline

• Motivation + Some Basics
• Overview of Conditional Random Fields
• Probabilistic Programming
• FACTORIE
• Joint Model of Segmentation and Entity Resolution (and other results)
• Future Work
Conditional Random Fields (CRF)

• Traditionally graphical models have been used to represent the joint distribution $p(x,y)$
• This is problematic because it requires knowledge of $p(x)$
  – E.g., if the task is named entity recognition, where we need features such as capitalization, prefixes, suffixes
  – We don’t want to model these types of dependencies in $x$
• Denominator of $p(x,y)$ has a summation over $x$
  – This translates into summing over all possible combinations of input variables
CRFs (cont.)

- CRFs address this by modeling the conditional distribution $p(y|x)$ directly.
- By modeling $p(y|x)$ directly we are able to include a much richer set of features. This means our models can be much more complicated, without incurring the cost of calculating $p(x)$.
(Linear Chain) Conditional Random Fields

Undirected graphical model, trained to maximize conditional probability of outputs given inputs

\[ p(y|x) = \frac{1}{Z(x)} \prod_{i=1}^{T} \psi_i(y_i, y_{i+1}, x) \]

where

\[ \psi_i(y_i, y_{i+1}, x) = \exp\{\sum_k \lambda_k f_k(y_i, y_{i+1}, x)\} \]

Fast-growing, wide-spread interest, many positive experimental results.

Noun phrase, Named entity
Protein structure prediction
IE from Bioinformatics text

Asian word segmentation
IE from Research papers
Object classification in images
Factorial CRFs
Jointly labeling cascaded sequences

Named-entity tag
Noun-phrase boundaries
Part-of-speech
English words

[Sutton, Rohanimanesh, McCallum, 2004]
Factorial CRFs
Jointly labeling cascaded sequences

Named-entity tag
Noun-phrase boundaries
Part-of-speech
English words

[Sutton, Rohanimanesh, McCallum, 2004]
Factorial CRFs
Jointly labeling cascaded sequences

But errors cascade--must be perfect at every stage to do well.

[Sutton, Rohanimanesh, McCallum, 2004]
Factorial CRFs
Jointly labeling cascaded sequences

Joint prediction of part-of-speech and noun-phrase in newswire, matching accuracy with only 50% of the training data.

[Sutton, Rohanimanesh, McCallum, 2004]
Skip-chain CRFs
Jointly labeling distant mentions

Dependency among similar, distant mentions ignored.

[Sutton & McCallum, 2004]
Senator Joe Green said today.

Green ran for ...

14% reduction in error on most repeated field in email seminar announcements.

[Sutton & McCallum, 2004]
General CRFs

\[ p(y|x) = \frac{1}{Z(x)} \prod_{\Psi_i \in G} \exp \left[ \sum_{k=1}^{K_i} \theta_{ik} f_{ik}(x_i, y_i) \right] \]
General CRFs

\[
p(y|x) = \frac{1}{Z(x)} \prod_{\psi_i \in G} \exp \left[ \sum_{k=1}^{K_i} \theta_{ik} f_{ik}(x_i, y_i) \right]
\]

Train to maximize \( \ell(\theta) = \log p(y|x) \)

\[
\frac{\partial \ell}{\partial \theta_i} = f(x_i, y_i) - \sum_{y_i} p(y_i|x_i; \theta) f(x_i, y_i)
\]
Parameter Estimation

• Optimization problem
  – E.g., maximize the log likelihood

• Perceptron style
  – Make a prediction, adjust parameters as necessary to agree
    • Most similar to Factorie’s SampleRank
Inference in CRFs

• Exact Inference
  – Variable elimination
  – Message passing in trees (or poly-trees)

• Optimization Based Methods
  – Message passing in loopy graphs
  – Approximating distributions

• Particle Methods
  – Gibbs sampling
  – Importance sampling
  – Markov Chain Monte Carlo
    • Metropolis Hastings
Parameter Tying in CRFs

- Often the same parameters are used for several factors
  - Factor template \( T_j \); parameters \( \{f_{jk}\} \); sufficient statistics functions \( \{\theta_{jk}\} \); and a description of what variables tuples \( \{(x_i, y_i)\} \) fulfill the arbitrary relationship the factor template represents
  - E.g., in practice the same weights are used for each time step in a linear-chain CRF

\[
p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{T_j \in T} \prod_{(x_i, y_i) \in T_j} \exp \left[ \sum_{k=1}^{K_j} \theta_{jk} f_{jk}(x_i, y_i) \right]
\]
Outline

• Motivation + Some Basics
• Overview of Conditional Random Fields
• Probabilistic Programming
• FACTORIE
• Joint Model of Segmentation and Entity Resolution (and other results)
• Future Work
Probabilistic Modeling in the Last Few Years

• Models ever growing in richness and variety
  — hierarchical
  — recursive
  — spatio-temporal
  — relational
  — infinite

• Developing the representation, reasoning and learning for a new model is a significant task.
Probabilistic Programming Languages

• Make it easy to represent rich, complex models, using the full power of programming languages
  — data structures
  — control mechanisms
  — abstraction
• Inference and learning come for free (or sort of)
• Gives you a language to think of and create new models
Small Sampling of Probabilistic Programming Languages

• Logic-based
  — Markov logic, BLOG, PRISM
• Functional
  — IBAL, Church
• Object Oriented
  — Figaro
Markov Logic
First-Order Logic as a Template to Define CRF Parameters

[Richardson & Domingos 2005]
[Paskin & Russell 2002]
[Taskar et al 2003]

<table>
<thead>
<tr>
<th>English</th>
<th>First-Order Logic</th>
<th>Clausal Form</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friends of friends are friends.</td>
<td>$\forall x \forall y \forall z \text{Fr}(x, y) \land \text{Fr}(y, z) \Rightarrow \text{Fr}(x, z)$</td>
<td>$\neg \text{Fr}(x, y) \lor \neg \text{Fr}(y, z) \lor \text{Fr}(x, z)$</td>
<td>0.7</td>
</tr>
<tr>
<td>Friendless people smoke.</td>
<td>$\forall x \left( \neg \left( \exists y \text{Fr}(x, y) \right) \Rightarrow \text{Sm}(x) \right)$</td>
<td>$\text{Fr}(x, g(x)) \lor \text{Sm}(x)$</td>
<td>2.3</td>
</tr>
<tr>
<td>Smoking causes cancer.</td>
<td>$\forall x \text{Sm}(x) \Rightarrow \text{Ca}(x)$</td>
<td>$\neg \text{Sm}(x) \lor \text{Ca}(x)$</td>
<td>1.5</td>
</tr>
<tr>
<td>If two people are friends, either both smoke or neither does.</td>
<td>$\forall x \forall y \text{Fr}(x, y) \Rightarrow (\text{Sm}(x) \Leftrightarrow \text{Sm}(y))$</td>
<td>$\neg \text{Fr}(x, y) \lor \text{Sm}(x) \lor \neg \text{Sm}(y)$, $\neg \text{Fr}(x, y) \lor \neg \text{Sm}(x) \lor \text{Sm}(y)$</td>
<td>1.1, 1.1</td>
</tr>
</tbody>
</table>

**ground Markov network**

$$P(X = x) = \frac{1}{Z} \exp \left( \sum w_i n_i(x) \right)$$

**grounding Markov network requires space $O(n'r)$**

$n$ = number constants
$r$ = highest clause arity
BLOG

• Generative model of objects and relations.
• Handles unknown number of objects
• Inference by MCMC.

#Researcher ~ NumResearchersPrior();
Name(r) ~ NamePrior();
#Paper ~ NumPapersPrior();
FirstAuthor(p) ~ Uniform({Researcher r});
Title(p) ~ TitlePrior();
PubCited(c) ~ Uniform({Paper p});
Text(c) ~ NoisyCitationGrammar
   (Name(FirstAuthor(PubCited(c))), Title(PubCited(c)));
Figaro

- Generative model of objects and relations.
- Object oriented (also in Scala!)
  - “Models” are basic building block, composed of other models, derived by inheritance.
  - Models are objects with conditions, constraints and relations to other objects.
  - Model = data + factors; they are highly intertwined.

[Pfeffer, 2009]
Figaro

• People smoke with probability 0.6:
  • Smoke(x) 1.5

• Friends are 3 times as likely to have the same smoking habit than different:
  • ← Friends(x,y) v ← Smoke(x) v Smoke(y) 3
  • ← Friends(x,y) v Smoke(x) v ← Smoke(y) 3

```java
class Person { val smokes = Flip(0.6) }
val alice, bob, clara = new Person
alice.smokes.condition(true)
val friends = List((alice, bob), (bob, clara))
def constraint(pair: (Boolean, Boolean)) =
  if (pair._1 == pair._2) 3.0; else 1.0
for { (p1,p2) ← friends }
  Pair(p1.smokes, p2.smokes).constrain(constraint)
```
• Tell generative story-line in Scheme.
• Do MCMC inference over execution paths.

(define (DP alpha proc)
  (let ((sticks (mem (lambda x (beta 1.0 alpha))))
        (atoms (mem (lambda x (proc))))
        (lambda () (atoms (pick-a-stick sticks 1)))))

(define (pick-a-stick sticks J)
  (if (< (random) (sticks J))
      J
      (pick-a-stick sticks (+ J 1))))

(define (DPmem alpha proc)
  (let ((dps (mem (lambda args
                    (DP alpha (lambda () (apply proc args))
                    )))))
   (lambda argsin ((apply dps argsin))) )

[Goodman, Mansinghka, Roy, Tenenbaum, 2009]
Outline

• Motivation + Some Basics
• Overview of Conditional Random Fields
• Probabilistic Programming
• FACTORIE
• Joint Model of Segmentation and Entity Resolution (and other results)
• Future Work
Factorie Approach

• We want a single framework for combining declarative and procedural domain knowledge
  – By leveraging imperative constructs (snippets of procedural code)

• A model written as an IDF is a factor graph with all the traditional semantics of factor graphs
  – E.g., factors, variables, possible worlds, scores, partition functions
Logic + Probability

- Significant interest in this combination
  - Poole, Muggleton, DeRaedt, Sato, Domingos,...

- We now hypothesize that in much of this previous work the “logic” aspect is not crucial to the ultimate goal of accurate and expressive modeling
  - Power: repeated relational structures and tied parameters
  - Logic is one way to specify these structures, but not the only one, and perhaps not the best (problems with sets, graph reachability).
  - In deterministic programming, Prolog replaced by imperative lang’s
    - programmers have to keep imperative solver in mind afterall
    - much domain knowledge is procedural anyway
  - Logical inference replaced by probabilistic inference.
Declarative Model Specification

- One of biggest advances in AI & ML

- Gone too far?
  Much domain knowledge is also procedural.

- Logic + Probability $\rightarrow$ Imperative + Probability
  - Rising interest: Church, Csoft,...

- Our approach
  - Preserve the *declarative* statistical semantics of factor graphs
  - “*Imperatively-Defined Factor Graphs*” (IDFs)
Our Design Goals

- Represent factor graphs
  - emphasis on discriminative undirected models
- Scalability
  - input data, output configuration, factors, tree-width
  - observed data that cannot fit in memory
  - exponential number of factors
- Efficient discriminative parameter estimation
  - sensitive to the expense of inference
- Leverage object-oriented benefits
  - Modularity, encapsulation, inheritance,...
- Integrate declarative & procedural knowledge
  - natural, easy-to-use
  - upcoming slides:
    4 examples of injecting imperativ-ism into factor graphs
• Factor Graphs, Imperative, Extensible
• Implemented as a library in Scala [Martin Odersky]
  — object oriented & functional
  — type inference
  — lazy evaluation
  — everything an object (int, float,...)
  — nice syntax for creating “domain-specific languages”
  — runs in JVM (complete interoperation with Java)
  — “Haskell++ in a Java style”

• Library, not new “little language”
  — all familiar Java constructs & libraries available to you
  — integrate data pre-processing & eval. w/ model spec
  — Scala makes syntax not too bad.
  — But not as compact as a dedicated language (BLOG, MLN)
Stages of Factorie Programming

1. Identify a natural representation of the data (variables)
   - Use data structures just like in deterministic programming.
   - Only special requirement: provide “undo” capability for changes.

2. Create factor templates to capture dependencies between variables
   - Distinct from above data representation; makes it easy to modify model scores independently.
   - Use & transform data’s natural relations to define factors’ relations.
   - Design your set of features (parameterization)

3. Optionally, define MCMC proposal functions that leverage domain knowledge.
Scala

- New variable
  ```scala
  var myHometown : String
  var myAltitude = 10523.2
  ```

- New constant
  ```scala
  val myName = "Karl"
  ```

- New method
  ```scala
  def climb(increment:double) = myAltitude += increment
  ```

- New class
  ```scala
  class Skier extends Person
  ```

- New trait (like Java interface with implementations)
  ```scala
  trait FirstAid { def applyBandage = ... }
  ```

- New class with trait
  ```scala
  class BackcountrySkier extends Skier with FirstAid
  ```

- New static object [generic]
  ```scala
  object GlobalSkierTable extends ArrayList[Skier]
  ```
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg)

class Token(word:String) extends EnumVariable(word)
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq

class Token(word:String) extends EnumVariable(word) with VarSeq

label.prev
label.next

Labels:

Words:

Bill  loves  skiing  Tom  loves  snowshoeing
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq

class Token(word:String) extends EnumVariable(word) with VarSeq
Example: Linear-Chain CRF for Segmentation

class Label(isBeg: boolean) extends Bool(isBeg) with VarSeq {
  val token : Token
}
class Token(word: String) extends EnumVariable(word) with VarSeq {
  val label : Label
}

Avoid representing relations by indices.
Do it directly with members, pointers... arbitrary data structure.
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
  val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
  val label : Label
  def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
  val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
  val label : Label
  def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label,Token]

Labels  
| T | F | F | T | F | F |
---|---|---|---|---|---|
Words  
| Bill | loves | skiing | Tom | loves | snowshoeing |
Example: Linear-Chain CRF for Segmentation

```scala
class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
  val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
  val label : Label
  def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label,Token]
object StateTransitionTemplate extends Template2[Label,Label]
```

Labels

```
T  F  F  T  F  F  F
```

Words

```
Bill    loves    skiing    Tom    loves    snowshoeing
```
Markov Chain Monte Carlo

- Only represent one configuration of the variables at a time
- Use a stochastic proposal distribution ("jump function") to generate new samples
- Allows us to avoid unrolling the entire network
  - Can therefore have much more complicated models
Key Operation: Scoring a Proposal

• Acceptance probability ~ ratio of model scores. Scores of factors that didn’t change cancel.

• To efficiently score:
  – Proposal method runs.
Key Operation: Scoring a Proposal

• Acceptance probability $\sim$ ratio of model scores. Scores of factors that didn’t change cancel.

• To efficiently score:
  – Proposal method runs.
  – Automatically build a list of variables that changed.
Key Operation: Scoring a Proposal

- Acceptance probability ~ ratio of model scores. Scores of factors that didn’t change cancel.
- To efficiently score:
  - Proposal method runs.
  - Automatically build a list of variables that changed.
  - Find factors that touch changed variables
Key Operation: Scoring a Proposal

- Acceptance probability ~ ratio of model scores. Scores of factors that didn’t change cancel.
- To efficiently score:
  - Proposal method runs.
  - Automatically build a list of variables that changed.
  - Find factors that touch changed variables
  - Find other (unchanged) variables needed to calculate those factors’ scores
Key Operation: Scoring a Proposal

• Acceptance probability ~ ratio of model scores. Scores of factors that didn’t change cancel.

• To efficiently score:
  – Proposal method runs.
  – Automatically build a list of variables that changed.
  – Find factors that touch changed variables.
  – Find other (unchanged) variables needed to calculate those factors’ scores.
Key Operation: Scoring a Proposal

- Acceptance probability ~ ratio of model scores. Scores of factors that didn’t change cancel.
- To efficiently score:
  - Proposal method runs.
  - Automatically build a list of variables that changed.
  - Find factors that touch changed variables
  - Find other (unchanged) variables needed to calculate those factors’ scores

- How to find factors from variables & vice versa?
  - In BLOG, rich, highly-indexed data structure stores mapping variables ↔ factors
  - But complex to maintain as structure changes
Imperativ-ism #1: Jump Function

• Proposal “jump function”
  – Make changes to world state
• Sometimes simple, sometimes not
  – Sample Gaussian with mean at old value
  – Sample cluster to split, run stochastic greedy agglomerative clustering

• Gibbs sampling, one variable at a time
  – poor mixing
• Rich jump function
  – Natural place to embed domain knowledge about what variables should change in concert.
Imperativ-ism #1: Jump Function

• Proposal “jump function”
  – Make changes to world state

• Sometimes simple, sometimes not
  – Sample Gaussian with mean at old value
  – Sample cluster to split,
    run stochastic greedy agglomerative clustering

• Gibbs sampling, one variable at a time
  – poor mixing

• Rich jump function
  – Natural place to embed domain knowledge about what variables
    should change in concert.
  – Avoid some expensive deterministic factors with
    property-preserving jump functions (e.g. coref transitivity,
    dependency parsing projectivity)
Imperativ-ism #2: Variable Coordination

- Given the set of variables that change as a result of a proposal by the jump function
  - Can generate a 2nd set of variables that should change in concert with the first set
  - Model authors can provide this imperatively via the set() method in variable class declaration

- Helps reshape the feasible region by automatically making changes rather than relying on sampling to discover the valid configurations

- Examples:
  - Given a named-entity label change, coreferent mentions should change as well
  - Recalculating canonical values in a set-based variable
Imperativ-ism #3: Model Structure

- Maintain a map structure between factors and variables

- Finding factors is easy. Usually # instantiations < 50.

  **Primitive operation:**
  Given factor template and one changed variable, find other variables

  In factor Template object, define *imperative methods* that do this.
  - `unroll1(v1)` returns `(v1,v2,v3)`
  - `unroll2(v2)` returns `(v1,v2,v3)`
  - `unroll3(v3)` returns `(v1,v2,v3)`
    - I.e., use Turing-complete language to determine structure on the fly.
    - If you want to use a data structure instead, access it in the method.
    - If you want a higher-level language for specifying structure, write it terms of this primitive.

- Other nice attribute
  - Easy to do value-conditioned structure. Case Factor Diagrams, etc.
  - Not only avoid unrolling, don’t even allocate all factors for current config.
  - Avoid object look-up by index! (Hairy when objects created/destroyed)
class Label(isBeg: boolean) extends Bool(isBeg) with VarSeq {
  val token : Token
}
class Token(word: String) extends EnumVariable(word) with VarSeq {
  val label : Label
  def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label, Token]
object StateTransitionTemplate extends Template2[Label, Label]

Example: Linear-Chain CRF for Segmentation
Example: Linear-Chain CRF for Segmentation

class Label(isBeg: boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word: String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label, Token]

object StateTransitionTemplate extends Template2[Label, Label]
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
}

object StateTransitionTemplate extends Template2[Label,Label]
**Example: Linear-Chain CRF for Segmentation**

```scala
class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
  val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
  val label : Label
  def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label] 
object StateTokenTemplate extends Template2[Label,Token] {
  def unroll1(label:Label) = Factor(label, label.token)
  def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends Template2[Label,Label]
```

---

**Diagram:**
- **Labels**
- **Words**
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
    def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends Template2[Label,Label] {
    def unroll1(label:Label) = Factor(label, label.next)
}
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
    def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends Template2[Label,Label] {
    def unroll1(label:Label) = Factor(label, label.next)
    def unroll2(label:Label) = Factor(label.prev, label)
}
Imperativ-ism #4: Neighbor-Sufficient Map

- “Neighbor Variables” of a factor
  - Variables touching the factor
- “Sufficient Statistics” of a factor
  - Vector, dot product with weights of log-linear factor → factor’s score

- Usually confounded. Separate them.
- Skip-chain NER. Instead of 5x5 parameters, just 2.
  \((\text{label1}, \text{label2}) \rightarrow \text{label1} == \text{label2}\)
Example: Linear-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends Template1[Label]
object StateTokenTemplate extends Template2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
    def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends Template2[Label,Label] {
    def unroll1(label:Label) = Factor(label, label.next)
    def unroll2(label:Label) = Factor(label.prev, label)
}
Example: Skip-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends TemplateWithStatistics1[Label]
object StateTokenTemplate extends TemplateWithStatistics2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
    def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends TemplateWithStatistics2[Label,Label] {
    def unroll1(label:Label) = Factor(label, label.next)
    def unroll2(label:Label) = Factor(label.prev, label)
}
object SkipTemplate extends Template1[Label,Label] with Statistics1[Bool]
Example: Skip-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends TemplateWithStatistics1[Label]
object StateTokenTemplate extends Template2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
    def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends TemplateWithStatistics2[Label,Label] {
    def unroll1(label:Label) = Factor(label, label.next)
    def unroll2(label:Label) = Factor(label.prev, label)
}
object SkipTemplate extends Template1[Label,Label] with Statistics1[Bool] {
    def unroll1(label:Label) = for (other <- label.seq)
        if (label.token == other.token)) yield Factor (label,other)
}
Example: Skip-Chain CRF for Segmentation

class Label(isBeg:boolean) extends Bool(isBeg) with VarSeq {
    val token : Token
}
class Token(word:String) extends EnumVariable(word) with VarSeq {
    val label : Label
    def longerThanSix = word.length > 6
}

// Factor templates
object StateTemplate extends TemplateWithStatistics1[Label]
object StateTokenTemplate extends Template2[Label,Token] {
    def unroll1(label:Label) = Factor(label, label.token)
    def unroll2(token:Token) = new Error // Tokens shouldn’t change
}
object StateTransitionTemplate extends TemplateWithStatistics2[Label,Label] {
    def unroll1(label:Label) = Factor(label, label.next)
    def unroll2(label:Label) = Factor(label.prev, label)
}
object SkipTemplate extends Template2[Label,Label] with Statistics1[Bool]{
    def unroll1(label:Label) = for (other <- label.seq;
        if (label.token == other.token)) yield Factor (label,other)
    def statistics(label1:Label,label2:Label) = Stat(label1 == label2)
}
class Word(str: String) extends EnumVariable(word)
class Node(word: Word, parent: Node) extends PrimitiveVariable(parent)

    def statistics(n: Node) = Stat(n.word, n.parent.word)
}

    def statistics(n: Node) = Stat(n.word, closestVerb(n).word)
    def closestVerb(n: Node) = if (isVerb(n.word)) n else closestVerb(n.parent)
    def unroll1(n: Node) = n.selfAndDescendants
}
SampleRank: Learning by Ranking

How do we make one million updates?
IDEA: rank configuration during a random walk

Each jump produces a config pair $y \rightarrow y'$
Objective signal ($\Omega$) yields score for all $y$ in $F$ (e.g. $B^3$ F1-score in a clustering problem)

WANT: model to agree with signal
SOLUTION: set $\theta$ to rank $(y,y')$ in same order as signal
SampleRank (algorithm)

Input:
1. Factor graph $G_\theta = \langle X, Y, \Psi \rangle$
2. Proposal distribution $Q : Y \times Y \rightarrow [0,1]$
3. Preference function $P(y, y') = 1$ if $y$ is preferred over $y'$, 0 otherwise

Output:
Parameters $\theta$

1. Initialize $y = y_0$ in $F$
2. Propose a change $y' \sim Q(-|y)$
3. Compute gradient: $\nabla = \phi(y') - \phi(y)$
4. Update if misranked: $\theta = \theta + \eta \nabla$
5. Probabilistically accept: $y = y'$
6. Repeat steps 2-5
SampleRank

Length is magnitude of score

Truth signal

model’s preferences should agree with signal’s preferences.

model score

Correction: $\theta_{t+1} = \theta_t + \eta_t (\phi(\text{Truth signal}) - \phi(\text{model score}))$
Correction: $\theta_{t+1} = \theta_t + \eta_t (\phi(\text{truth}) - \phi(\text{score}))$

Correction: $\theta_{t+2} = \theta_{t+1} + \eta_{t+1} (\phi(\text{truth}) - \phi(\text{score}))$
SampleRank

Correction: $\theta_{t+1} = \theta_t + \eta_t (\phi(\text{signal}) - \phi(\text{model score}))$

Correction: $\theta_{t+2} = \theta_{t+1} + \eta_{t+1} (\phi(\text{signal}) - \phi(\text{model score}))$
FACTORIE Summary

http://factorie.googlecode.com/

- Factor graphs,
- ...object-oriented
  - data types and factor template types, with inheritance
- ...scalable
  - factors created on demand, only score diffs
- ...with imperative hooks
  - jump function, override variable.set() for coordination
  - model structure
  - neighbor variables $\rightarrow$ sufficient statistics
- ...discriminative
  - efficient online training by SampleRank
- Combine declarative & procedural knowledge
Extensibility

• Many variables types provided:
  — boolean, int, float, String, categorical,...

• Create new ones!
  — set-valued variable
  — finite-state machine as a variable [JHU]

• Create new factor types
  — Poisson, Dirichlet,
Factorie: In-Progress

- Support for first-order logic
- Belief propagation
- Generative modeling
Conclusion: Some Reasons To Use Probabilistic Programming

• Simple
  — Save time & avoid debugging complex, hand-built ML code.
  — Say exactly what you want in the way you want to say it.

• Flexible
  — Encourage research exploration by making it easier to try new modeling ideas
  — The language provides the right “hinge-points” to provide the flexibility you want, without the underlying cruft.

• Glue that binds many reasoning paradigms together.

• Allows probabilistic modeling to be integrated with all the other traditional deterministic programming.

Some of this text from Pfeffer
Key Questions for Probabilistic Programming

- What are good design patterns for probabilistic programming?
- What are the skills required to be an effective probabilistic programmer?
- How can probabilistic programmers work well with domain experts and end users?
- What kind of tools can we develop to support probabilistic programming (debuggers, profilers etc.)?
- How can probabilistic programs be learned (especially structure)?
- How can we make inference more efficient (especially memory) to scale up to even large domains?

Some of this text from Pfeffer
Outline

• Motivation + Some Basics
• Overview of Conditional Random Fields
• Probabilistic Programming
• FACTORIE
• Joint Model of Segmentation and Entity Resolution (and other results)
• Future Work
Citation Data

- drucker h., schapire r., and simard r. improving performance in neural networks using a boosting algorithm. advances in neural information processing systems 5, san mateo, ca. morgan kaufmann.1993 pages 42-49, in hanson, s. j., cowan, j. d., and giles, c. l., editors,

- yoav freund, and robert e. schapire. experiments with a new boosting algorithm. in proceedings of the 13th international conference on machine learning. morgan kaufmann, 1996

- freund y., schapire r.e. experiments with a new boosting algorithm, in saittta l.(ed.), proc of the thirteenth international conference on machine learning, san francisco, ca, pp.148-156, morgan kauf-mann, 1996

Segmentation


Entity Resolution

- drucker h., schapire r., and simard r. improving performance in neural networks using a boosting algorithm. advances in neural information processing systems 5, san mateo, ca. morgan kaufmann.1993 pages 42-49, in hanson, s. j., cowan, j. d., and giles, c. l., editors,

- yoav freund, and robert e. schapire. experiments with a new boosting algorithm. in proceedings of the 13th international conference on machine learning. morgan kaufmann, 1996

- freund y., schapire r.e. experiments with a new boosting algorithm, in saitta l.(ed.), proc of the thirteenth international conference on machine learning, san francisco, ca, pp.148-156, morgan kauf-mann, 1996

Entity Resolution


Pipeline Approach

- Each of the tasks can be solved independently
Pipeline Approach

- Each of the tasks can be solved independently
- However, sharing information can help
  - drucker h., schapire r., and simard r. improving performance in neural networks using a boosting algorithm. advances in neural information processing systems 5, san mateo, ca. morgan kaufmann. 1993 pages 42-49, in hanson, s. j., cowan, j. d., and giles, c. l., editors,
Pipeline Approach

- Each of the tasks can be solved independently
- However, sharing information can help
  - drucker h., schapire r., and simard r. improving performance in neural networks using a boosting algorithm. advances in neural information processing systems 5, san mateo, ca. morgan kaufmann.1993 pages 42-49, in hanson, s. j., cowan, j. d., and giles, c. l., editors,
- Thus, entity resolution should use segmentation
Pipeline Approach

- Cascading error through the stages
  - Can be reduced by using N-best lists*, or sampling†

---

* Sutton & McCallum CoNLL 2005
† Finkel et al. EMNLP 2006
Pipeline Approach

- Cascading error through the stages
  - Can be reduced by using N-best lists*, or sampling†
- Unidirectional information flow

---

* Sutton & McCallum CoNLL 2005
† Finkel et.al. EMNLP 2006
Pipeline Approach

- Cascading error through the stages
  - Can be reduced by using N-best lists\(^*\), or sampling\(^\dagger\)
- Unidirectional information flow
  - drucker harris, schapire, robert, and simard patrice 1993.

\(^*\)Sutton & McCallum CoNLL 2005
\(^\dagger\)Finkel et.al. EMNLP 2006
Iterated Pipeline Approach

- Close the loop of the pipeline
  - Both tasks use information from each other
- Reduces cascading error
  - However, still not eliminated
  - N-best lists can be used to further reduce this error†

†Wellner et al. UAI 2004
Integrate models in a single, unified, "fully-joint" factor graph

- To **decrease cascading error** inference is performed simultaneously over both tasks
- **Increased complexity** is handled efficiently by using procedural hooks in model specification and inference
Bi-directional Joint Inference for Segmentation and Entity Resolution

**Objective:**
- **Input:** a set of mention strings (e.g., bibliographic citations)
- **Output:**
  - A set of fields for each mention string (*segmentation*)
  - A clustering of the mention strings (*entity resolution*)

- Separate factor graphs are created for each task
- A unified factor graph is created to model both tasks
  - Contains variables for both tasks
  - Contains *joint factors*
    - Neighbor variables of different tasks
    - Capture dependencies between the tasks
- **Variables**
  - **Token**: Observed variable representing a word in the mention
  - **Label**: Variable that can take any of the field types as a value
**Variables**

- **Token**: Observed variable representing a word in the mention
- **Label**: Variable that can take any of the field types as a value
- **Field**: Consecutive Tokens that have the same label type
**Variables**
- **Token**: Observed variable representing a word in the mention
- **Label**: Variable that can take any of the field types as a value
- **Field**: Consecutive Tokens that have the same label type

**Factors**: LabelToken, LabelPrev/NextToken, FieldFactor
**Variables**

- **Mention**: Variable that takes a single **Entity** as its value
- **Entity**: Set of Mentions that are coreferent
Entity Resolution

- **Variables**
  - Mention: Variable that takes a single Entity as its value
  - Entity: Set of Mentions that are coreferent

- **Factors:** Affinity and Repulsion
Integrating the Two Tasks

- **Variables**
  - No additional variables are required
  - Field variables are added as members of Mention variables

- **Joint Factors**: connect variables of different tasks
  - **JointInfBased**:
    - Connect identical trigrams of Tokens between two Mentions where the trigram is preceded by punctuation in only one of the Mentions
    - Forms a weak connection between the tasks since it is sparse, and does not take the entire predicted Field into account
  - **JointAffinity, JointRepulsion**:
    - Connect corresponding Fields between pairs of Mentions
    - Utilize features computed over the full predicted Fields between Mention pairs (e.g., string similarity, number of matching Tokens)

---

Poon & Domingos AAAI 2007
Example Model
Example Model
Example Model
The Advantages of IDFs

Joint factor templates can make inference intractable

- JointAffinity and JointRepulsion factor templates have $O(m^2n^4)$ instances in a fully unrolled graph

---

$m =$ number of mentions, $n =$ number of tokens in a mention
The Advantages of IDFs

1. Joint factor templates can make inference intractable
   - `JointAffinity` and `JointRepulsion` factor templates have $O(m^2 n^4)$\footnote{\textit{m} = number of mentions, \textit{n} = number of tokens in a mention} instances in a fully unrolled graph
   - IDFs allow such factors through imperative structure definition and on-the-fly feature calculation
   - Evaluating a new sample requires re-scoring only \textit{m} such factors
The Advantages of IDF

1. Joint factor templates can make inference intractable
   - JointAffinity and JointRepulsion factor templates have $O(m^2 n^4)$ \( \| \) instances in a fully unrolled graph
   - IDF allows such factors through imperative structure definition and on-the-fly feature calculation
   - Evaluating a new sample requires re-scoring only \( m \) such factors

2. The proposal function utilizes domain knowledge to implicitly define and efficiently explore the feasible region

\[ \| ml = \text{number of mentions}, \ n = \text{number of tokens in a mention} \]
The Advantages of IDFs

1. Joint factor templates can make inference intractable
   - JointAffinity and JointRepulsion factor templates have $O(m^2 n^4)$ instances in a fully unrolled graph
   - IDFs allow such factors through imperative structure definition and on-the-fly feature calculation
   - Evaluating a new sample requires re-scoring only $m$ such factors

2. The proposal function utilizes domain knowledge to implicitly define and efficiently explore the feasible region

3. Factor templates leverage the flexible separation of data representation and parameterization provided by IDFs
   - E.g., a Field is most naturally represented as a range over Tokens, and the compatibility between Field pairs is easily parameterized by a JointAffinity factor

---

$\| m =$ number of mentions, $n =$ number of tokens in a mention
Experimental Setup

- Cora citation dataset**
  - 1,295 mentions, 134 clusters, 36,487 tokens
  - Evaluated using three-fold cross-validation

- **Isolated Models**
  - Each task is completely independent of the other
  - Learn with 5 loops of 100,000 MCMC samples each
  - Inference for 300,000 MCMC samples per task

- **Joint Models**
  - Single model over both the tasks
  - Learn with 5 loops of 250,000 MCMC samples each
  - Inference for 750,000 MCMC samples

- Results are compared to Poon and Domingos’ previous state-of-the-art isolated and joint Markov logic networks

**Available at http://alchemyl.cs.washington.edu/papers/poon07**
### Model Performance

**Table: Cora Entity Resolution:** Pairwise F1 and Cluster Recall

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec/Recall</th>
<th>F1</th>
<th>Cluster Rec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fellegi-Sunter</td>
<td>78.0/97.7</td>
<td>86.7</td>
<td>62.7</td>
</tr>
<tr>
<td>Joint MLN</td>
<td>94.3/97.0</td>
<td>95.6</td>
<td>78.1</td>
</tr>
<tr>
<td>Isolated IDF</td>
<td><strong>97.09/95.42</strong></td>
<td><strong>96.22</strong></td>
<td><strong>86.01</strong></td>
</tr>
<tr>
<td>Joint IDF</td>
<td><strong>95.34/98.25</strong></td>
<td><strong>96.71</strong></td>
<td><strong>94.62</strong></td>
</tr>
</tbody>
</table>

25.2%

**Table: Cora Segmentation:** Tokenwise F1

<table>
<thead>
<tr>
<th>Method</th>
<th>Author</th>
<th>Title</th>
<th>Venue</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolated MLN</td>
<td>99.3</td>
<td>97.3</td>
<td>98.2</td>
<td>98.2</td>
</tr>
<tr>
<td>Joint MLN</td>
<td>99.5</td>
<td>97.6</td>
<td>98.3</td>
<td>98.4</td>
</tr>
<tr>
<td>Isolated IDF</td>
<td>99.35</td>
<td>97.63</td>
<td>98.58</td>
<td>98.51</td>
</tr>
<tr>
<td>Joint IDF</td>
<td><strong>99.42</strong></td>
<td><strong>97.99</strong></td>
<td><strong>98.78</strong></td>
<td><strong>98.72</strong></td>
</tr>
</tbody>
</table>

20.0%
# Model Performance

**Table: Cora Entity Resolution:** Pairwise F1 and Cluster Recall

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec/Recall</th>
<th>F1</th>
<th>Cluster Rec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fellegi-Sunter</td>
<td>78.0/97.7</td>
<td>86.7</td>
<td>62.7</td>
</tr>
<tr>
<td>Joint MLN</td>
<td>94.3/97.0</td>
<td>95.6</td>
<td>78.1</td>
</tr>
<tr>
<td>Isolated IDF</td>
<td>97.09/95.42</td>
<td>96.22</td>
<td>86.01</td>
</tr>
<tr>
<td>Joint IDF</td>
<td>95.34/98.25</td>
<td>96.71</td>
<td>94.62</td>
</tr>
</tbody>
</table>

50-90 mins  
~3 mins     
~18 mins

**Table: Cora Segmentation:** Tokenwise F1

<table>
<thead>
<tr>
<th>Method</th>
<th>Author</th>
<th>Title</th>
<th>Venue</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolated MLN</td>
<td>99.3</td>
<td>97.3</td>
<td>98.2</td>
<td>98.2</td>
</tr>
<tr>
<td>Joint MLN</td>
<td>99.5</td>
<td>97.6</td>
<td>98.3</td>
<td>98.4</td>
</tr>
<tr>
<td>Isolated IDF</td>
<td>99.35</td>
<td>97.63</td>
<td>98.58</td>
<td>98.51</td>
</tr>
<tr>
<td>Joint IDF</td>
<td>99.42</td>
<td>97.99</td>
<td>98.78</td>
<td>98.72</td>
</tr>
</tbody>
</table>

50-90 mins  
~3 mins     
~18 mins
Figure: F1 of the joint model as different types of factors are added, starting with the base model containing only isolated model factors. “Semi-Joint” refers to the model containing weakly joint factors while the “Fully-Joint” model consists of bi-directional highly-coupled factors.
Coreference in a PDB

Who’s who?

A: *incomplete* relational database
B: templated factor graph for coreference
C: graph unrolled on data
Query Evaluation Problem

- Have: Query Q, Database D
- Want: probability of tuple t being in Q(D)

**Problem:** each sample requires executing query over entire D

**Solution:** run modified query Q’ on the tuples that have changed

*Similar to materialized view maintenance (Blakely et al.)*
Results

Query 1: print tokens whose strings are the same, but labels are different (restrict to same doc)

```
SELECT t.string
FROM TOKEN t, TOKEN t2
WHERE t.string=t2.string AND t.label<>t2.label
AND t.label<>'O' AND t2.label<>'O'
AND t.doc_id=t2.doc_id
```

Query 2: print tokens whose strings are the same, but labels are different

```
SELECT t.string
FROM TOKEN t, TOKEN t2
WHERE t.string=t2.string AND t.label<>t2.label
AND t.label<>'O' AND t2.label<>'O'
```

TOKEN = <doc_id,sent_id,sent_pos, string, label>. |TOKEN|=168,000.
Reinforcement Learning

**MOTIVATION**

**Problem:** delayed credit assignment

![Diagram showing state transitions and F1 scores](image)

<table>
<thead>
<tr>
<th>Class A</th>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial State</td>
<td>...</td>
</tr>
<tr>
<td>F1 Score:</td>
<td>61%</td>
</tr>
</tbody>
</table>

SampleRank prefers this state over this state!

**Solution:** train factor graph with Q(lambda) [Wick, Rohanimanesh, Singh, McCallum 2008, 2009]

*Temporal difference (TD) has same diff-structure as SampleRank!*
## Results

### ConLL NER results (NEW in FACTORIE!!)

<table>
<thead>
<tr>
<th>Method</th>
<th>F1 Test A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reinforcement Learning</td>
<td>89.5</td>
</tr>
<tr>
<td>SampleRank</td>
<td>89.0</td>
</tr>
<tr>
<td>Max Likelihood</td>
<td>88.5</td>
</tr>
</tbody>
</table>

### Ontology alignment: ISIA data (taxonomy trees)

<table>
<thead>
<tr>
<th></th>
<th>Course Catalog</th>
<th>Company Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1</td>
<td>P</td>
</tr>
<tr>
<td>RL</td>
<td>94.3</td>
<td>96.1</td>
</tr>
<tr>
<td>MH-CD1</td>
<td>94.3</td>
<td>78.0</td>
</tr>
<tr>
<td>MH-SR</td>
<td>76.9</td>
<td>88.9</td>
</tr>
<tr>
<td>GA-PW</td>
<td>92.0</td>
<td>100</td>
</tr>
<tr>
<td>GLUE</td>
<td>80</td>
<td>80</td>
</tr>
</tbody>
</table>

[Wick, Rohanimanesh, Singh, McCallum 2008,2009]

- **RL**: train with reinforcement learning, test by following policy
- **MH-CD1**: train with contrastive divergence (1) and test with Metropolis-Hastings
- **MH-SR**: train with SampleRank and test with Metropolis-Hastings
- **GA-PW**: train with piecewise test with greedy agglomerative clustering
- **GLUE**: Doan et al. 2002 baseline system (1:1 assumption means P=R=F1)
Outline

• Motivation + Some Basics
• Overview of Conditional Random Fields
• Probabilistic Programming
• FACTORIE
• Joint Model of Segmentation and Entity Resolution (and other results)
• Future Work
Future Work

• Adding more methods for learning and inference
• Better support for generative models
• Joint models for more than two tasks
• Semi-supervised/unsupervised learning
Thanks!

http://factorie.googlecode.com/

kschultz@cs.umass.edu