Better call Saul: Flexible Programming for Learning and Inference in NLP

Parisa Kordjamshidi, Tulane University, pkordjam@tulane.edu
Daniel Khashabi, University of Illinois at Urbana-Champaign
Christos Christodouloupolos, University of Illinois at Urbana-Champaign
Bhargav Mangipudi, University of Illinois at Urbana-Champaign
Sameer Singh, University of California, Irvine
Dan Roth, University of Illinois at Urbana-Champaign

COLING-2016
December 2016, Osaka, Japan
Better Call Saul:...
WHAT

• Introducing Saul which is a declarative Learning based programming language. [Kordjamshidi et. al. IJCAI-2015]
WHAT

• Introducing Saul which is a declarative Learning based programming language. [Kordjamshidi et. al. IJCAI-2015]

• Particularly, the way we have augmented it with the abstraction levels and facilities for designing various NLP tasks with arbitrary output structure with various linguistic granularities.
WHAT

- Introducing Saul which is a declarative Learning based programming language. [Kordjamshidi et. al. IJCAI-2015]

- Particularly, the way we have augmented it with the abstraction levels and facilities for designing various NLP tasks with arbitrary output structure with various linguistic granularities.

  - Word level
  - Phrase level
  - Sentence level …
WHY

Better Call Saul:...
WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.
WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

- Syntactic => Part of speech tagging
- Information Extraction => Entity mention / Relation extraction
- Semantic => Semantic role labeling
WHY

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

- Syntactic => Part of speech tagging
- Information Extraction => Entity mention / Relation extraction
- Semantic => Semantic role labeling

We often need to make a lot of programming effort and hard code to:
**WHY**

Most of the NLP learning tasks seek for a mapping from input structures to output structures.

- Syntactic => Part of speech tagging
- Information Extraction => Entity mention / Relation extraction
- Semantic => Semantic role labeling

We often need to make a lot of programming effort and hard code to:

- Benefit from a specific structure
- (Relational) Feature extraction (Even when using representation learning techniques)
Example Tasks (1)

Semantics: Semantic role labeling

INTRODUCTION CMPS-3240/6240 Fall ’16
Example Tasks (2)

Information extraction: Entity mention relation extraction

SENTENCE

WASHINGTON COVERS SEATTLE FOR THE ASSOCIATED PRESS

ENTITIES

WASHINGTON  
Seattle  
the Associated Press

RELATION

WASHINGTON-SEATTLE  
Seattle-the Associated Press

Washington covers Seattle for the Associated Press.

Better Call Saul:...
Figure 1: An instantiation of the data-model for the NLP domain. The colored ovals are some observed properties, while the white ones show the unknown labels. For the POS and Entity Recognition tasks, the boxes represent candidates for single labels; for the SRL and Relation Extraction tasks, they represent candidates for linked labels.

These constraints are added as a part of Saul’s objective, so we have the following objective form, which is in fact a constrained conditional model (Chang et al., 2012),

\[ g = h_w, f(x, y) \]

\[ \implies c(x, y) \]

where \( c \) is the constraint function and \( \implies \) is the vector of penalties for violating each constraint. This representation corresponds to an integer linear program, and thus can be used to encode any MAP problem. Specifically, the \( g \) function is written as the sum of local joint feature functions which are the counterparts of the probabilistic factors:

\[ g(x, y; w) = \sum_l p_l \sum_k p_k \{ \tau \} h_{wp, fp}(x_k, l_p) + |C| \sum_m \implies m c_m(x, y) \]

\( C \) is a set of global constraints that can hold among various types of nodes. \( g \) can represent a general scoring function rather than the one corresponding to the likelihood of an assignment. The constraints are used during training for loss-augmented inference as well as during prediction.

4 Calling Saul: Case Studies

For programming global models in Saul the programmer needs to declare a) the data-model which is a global structure of the data and b) the templates for learning an inference decompositions. The templates are declared intuitively in two forms of classifiers using \texttt{Learnable} construct and first order constraints using \texttt{ConstrainedClassifier} construct. With these components have been specified, the programmer can easily choose which templates to use for learning (training) and inference (prediction). In this way the global objective is generated automatically for different training and testing paradigms in the spectrum of local to global models.

One advantage of programming in Saul is that one can define a generic data-model for various tasks in each application domain. In this paper, we enrich Saul with an NLP data-model based on E\textsc{DISON}, a recently-introduced NLP library which contains raw data readers, data structures and feature extractors (Sammons et al., 2016) and use it as a collection of Sensors to easily generate the data-model from the raw data. In Saul, a Sensor is a ‘black-box’ function that can generate nodes, edges and properties in the graph. An example of a sensor for generating nodes and edges is a sentence tokenizer which receives a sentence and generates its tokens. Here, we will provide some examples of data-model declaration language but more details are available on-line.

In the rest of the paper, we walk through the tasks of Semantic Role Labeling (SRL), Part-of-Speech (POS) tagging and Entity-Relation (ER) extraction and show how we can design a variety of local to global models by presenting the related code.

\[ \text{https://github.com/IllinoisCogComp/saul/blob/master/saul-core/doc/DATAMODELING.md} \]

For extraction and representation of features as well as for exploiting global output structure we do
Structured Output Models: Common Practice

For extraction and representation of features as well as for exploiting global output structure we do

- Task Specific Programming for Data Structures
- Model Specific Programming for Inference and Learning
- It will be hard to generalize
- It will be hard to Reuse and Reproduce results
Structured Output Models:
Learning and Inference Paradigms
Structured Output Models: Learning and Inference Paradigms

- Local Models: Local classifiers trained/output components predicted independently (LO)
Structured Output Models:
Learning and Inference Paradigms

- Local Models: Local classifiers trained/output components predicted independently (LO)
- Pipelines
Structured Output Models: Learning and Inference Paradigms

- Local Models: Local classifiers trained/output components predicted independently (LO)
- Pipelines
- Global Models
  - L+I: Training LO, global prediction
  - IBT: Global training and global prediction
Saul and NLP

Better Call Saul:...
Saul and NLP

Idea: High level abstraction for programming various configurations from local to global learning as well as building pipelines over NLP data structures.
Idea: High level abstraction for programming various configurations from local to global learning as well as building pipelines over NLP data structures.

- Data Model
  - Graph (typed nodes, edges and properties)
  - Sensors (black box functions that operate on graph’s base types)
Saul and NLP

**Idea:** High level abstraction for programming various configurations from local to global learning as well as building pipelines over NLP data structures.

- **Data Model**
  - Graph (typed nodes, edges and properties)
  - Sensors (black box functions that operate on graph’s base types)
- **Templates for learning and inference decomposition**
  - Classifiers
  - Constraints
Underlying Computational Model
Underlying Computational Model

Learning:

\[ h : \mathcal{X} \rightarrow \mathcal{Y} \]
Underlying Computational Model

Learning:

Structured output learning:

\[ h : \mathcal{X} \rightarrow \mathcal{Y} \]

\[ g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \]
Underlying Computational Model

Learning:

Structured output learning:

\[ h : \mathcal{X} \rightarrow \mathcal{Y} \]

\[ g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \]

\[ h(x; W) = \arg \max_{y \in \mathcal{Y}} g(x, y; W) \]
Underlying Computational Model

Learning:

Structured output learning:

\[ h : \mathcal{X} \rightarrow \mathcal{Y} \]
\[ g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \]
\[ h(x; W) = \arg \max_{y \in \mathcal{Y}} g(x, y; W) \]

Inference
Underlying Computational Model

Learning:

Structured output learning:

Decoding/ Prediction
time inference

\[ h : \mathcal{X} \rightarrow \mathcal{Y} \]
\[ g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \]

\[ h(x; W) = \arg \max_{y \in \mathcal{Y}} g(x, y; W) \]
Underlying Computational Model

Learning:

Structured output learning:

\[
\begin{align*}
  h &: \mathcal{X} \rightarrow \mathcal{Y} \\
  g &: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \\
  h(x; W) &= \text{arg} \max_{y \in \mathcal{Y}} g(x, y; W) \\
  g(x, y; W) &= \langle W, f(x, y) \rangle
\end{align*}
\]
Underlying Computational Model

Learning:

Structured output learning:

\[
\begin{align*}
    h : \mathcal{X} & \rightarrow \mathcal{Y} \\
    g : \mathcal{X} \times \mathcal{Y} & \rightarrow \mathbb{R}
\end{align*}
\]

\[
h(x; W) = \arg \max_{y \in \mathcal{Y}} g(x, y; W)
\]

\[
g(x, y; W) = \langle W, f(x, y) \rangle
\]
Underlying Computational Model

Learning:

Structured output learning:

Decoding/ Prediction time inference

Joint feature function

Weight vector

Inference

\[ h : \mathcal{X} \rightarrow \mathcal{Y} \]

\[ g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \]

\[ h(x; W) = \arg \max_{y \in \mathcal{Y}} g(x, y; W) \]

\[ g(x, y; W) = \langle W, f(x, y) \rangle \]
Underlying Computational Model

Learning:

Structured output learning:

Decoding/ Prediction

time inference

Inference

Weight vector

Joint feature function

$h : \mathcal{X} \rightarrow \mathcal{Y}$

$g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$

$h(x; W) = \arg \max_{y \in \mathcal{Y}} g(x, y; W)$

$g(x, y; W) = \langle W, f(x, y) \rangle$

$l(W) = \sum_{i=1}^{N} \max_{y \in \mathcal{Y}} (g(x^i, y; W) - g(x^i, y^i; W) + \Delta(y^i, y))$
Underlying Computational Model: Input/Output

\[ g(x, y; W) = \langle W, f(x, y) \rangle \]

\[ \{x_1 \ldots x_K\} \quad I = \{l_1, \ldots, l_P\} \]

\[ g(x, y; W) = \sum_{l_p \in I} \sum_{x_k \in C_{l_p}} \langle W_p, f_p(x_k, l_p) \rangle = \sum_{l_p \in I} \sum_{x_k \in C_{l_p}} \langle W_p, \phi_p(x_k) \rangle l_{pk} = \]

\[ \sum_{l_p \in I} \langle W_p, \sum_{x_k \in C_{l_p}} (\phi_p(x_k) l_{pk}) \rangle \]
Underlying Computational Model: Input/Output

\[ g(x, y; W) = \langle W, f(x, y) \rangle \]

\[ \{x_1 \ldots x_K\} = \{l_1, \ldots, l_P\} \]

\[ g(x, y; W) = \sum_{l_p \in L} \sum_{x_k \in C_{l_p}} \langle W_p, f_p(x_k, l_p) \rangle = \sum_{l_p \in L} \sum_{x_k \in C_{l_p}} \langle W_p, \phi_p(x_k) \rangle l_{pk} = \]

\[ \sum_{l_p \in L} \sum_{x_k \in C_{l_p}} \langle W_p, \phi_p(x_k) l_{pk} \rangle \]

in addition to the constraints between labels!
Underlying Computational Model:
Global Constraints

Constrained Conditional Models (CCM)

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Underlying Computational Model: 
Global Constraints

Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[ h(x) = \arg \max_{y \in Y(x)} g(x, y; W) \]

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[
h(x) = \arg \max_{y \in Y(x)} g(x, y; W)
\]

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[ h(x) = \arg \max_{y \in Y(x)} g(x, y; W) \]

[Rothe & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[
h(x) = \arg \max_{y \in Y(x)} g(x, y; W)
\]

- Objective is linear in features and constraints

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[ h(x) = \arg \max_{y \in Y(x)} g(x, y; W) \]

- Objective is linear in features and constraints

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[ h(x) = \arg \max_{y \in Y(x)} g(x, y; W) \]

- Objective is linear in features and constraints

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[ h(x) = \arg \max_{y \in Y(x)} g(x, y; W) \]

- Objective is linear in features and constraints

\[ g = \langle W, f(x, y) \rangle - \langle \rho, c(x, y) \rangle \]

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]

Better Call Saul:...
Constrained Conditional Models (CCM)

- Prediction function: assign values that maximize objective

\[ h(x) = \arg \max_{y \in Y(x)} g(x, y; W) \]

- Objective is linear in features and constraints

\[ g = \langle W, f(x, y) \rangle - \langle \rho, c(x, y) \rangle \]

Compile everything in an Integer Linear Program: expressive enough to support decision making in the context of any probabilistic modeling.

[Roth & Yih ‘04, 07; Chang, et.al.,’08,’12]
Semantic Role Labeling: Data Model

```scala
case class node[TextAnnotation]
case class Constituent
val sentences = node[TextAnnotation]
case class Relations
val predicates = node[Constituent]
case class Arguments
val arguments = node[Constituent]
case class Relations
val pairs = node[Relations]
case class Property
val pos-tag = property(arguments)
case class WordForm
val word-form = property(arguments)
case class Relations
val relationsToArguments = edge(relations, arguments)
case class Relations
relationsToArguments.addSensor(relToArgument _)
```

A graph in terms of typed nodes, edges and Properties

Better Call Saul:...
Semantic Role Labeling: Classifiers

\[ x = \{x_1, \ldots, x_4\} \]
\[ l = \{l_{isPred}, l_{isArg}, l_{argType}\} \]
\[ \phi_{constituent}(x_i) \]
\[ \phi_{pair}(x_i, x_j) \]

object ArgTypeLearner extends Learnable(pairs) {
    def label = argumentLabelGold
    def feature = using(containsMOD, containsNEG, clauseFeatures, chunkPathPattern, chunkEmbedding, chunkLength, constituentLength, argPOSWindow, argWordWindow, headwordRelation, syntacticFrameRelation, pathRelation, phraseTypeRelation, predPosTag, predLemmaR, linearPosition)
}

a Learning Template for argument types

Better Call Saul...
Semantic Role Labeling: Combined Feature Functions

- Single or composed components of the input are represented with typed nodes in the graph.
- All features are defined as the properties of the nodes.
- Labels also applied to single components or composed components of the input (called link labels in the latter case).
- Edges are established between nodes.
- The edges and properties are defined and computed based on a set of given NLP sensors.
Semantic Role Labeling: Constraints

Only legal arguments of a predicate could be assigned as a type to the candidate arguments. The legality is checked according to the Propbank frames.

```scala
val legalArgumentsConstraint = constraint(sentences) { x =>
  val constraints = for {
    predicate <- sentences(x) ~> sentenceToPredicates
    candidateRelations = (predicates(y) ~> -relationsToPredicates)
    argLegalList = legalArguments(y)
    relation <- candidateRelations
  } yield classifierLabelIsLegal(argumentTypeLearner, relation, argLegalList)
  or (argumentTypeLearner on relation is "none")
}

def classifierLabelIsLegal(classifier, relation, legalLabels) = {
  legalLabels._exists { l => (classifier on relation is l) }
}
```
This Constrained Classifier now applies on a given pair candidate, but it uses the global constraints at the sentence level. The sentence is accessed via the edges defined in the data model that connect the relation to its original sentence.

```java
object ArgTypeConstraintClassifier extends ConstrainedClassifier(ArgTypeLearner)
{
    def subjectTo = srlConstraints
}
```
val srlDataModelObject = PopulateSRLDataModel(...)
val AllMyConstrainedClassifiers = List(argTypeConstraintClassifier, ...)
JointTrain(sentences, AllMyConstrainedClassifiers)
ClassifierUtils.TestClassifiers(AllMyConstrainedClassifiers)
Program Structure

```scala
val srlDataModelObject = PopulateSRLDataModel()
val AllMyConstrainedClassifiers = List(argTypeConstraintClassifier, ...)
JointTrain(sentences, AllMyConstrainedClassifiers)
ClassifierUtils.TestClassifiers(AllMyConstrainedClassifiers)
```

Same amount of code for other paradigms!
Other tasks
Other tasks

```scala
val labelTwoBefore = property(tokens) { x: Constituent =>
    // Use edges to jump to the previous constituent
    val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
    if (POSTaggerKnown.isTraining)
        POSLabel(cons)
    else POSTaggerKnown(cons)
}
```

```
// A1 ⇠ constituentBefore
// A2 ⇠ constituentBefore
// A3 ⇠ constituentBefore
// A4 ⇠ constituentBefore
// A5 ⇠ constituentBefore
```
Other tasks

```scala
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```

Pos-Tag Contextual Feature

Better Call Saul:...
Other tasks

```scala
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```

Pos-Tag Contextual Feature

(WorkFor(x) is true then PER(x.firstArg) is true and ORG(x.secondArg) is true)
Other tasks

```scala
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```

Pos-Tag Contextual Feature

(WorkFor(x) is true then PER(x.firstArg) is true and ORG(x.secondArg) is true)

ER constraint imposed on mentioned and relations
Other tasks

```scala
val labelTwoBefore = property(tokens) { x: Constituent =>
  // Use edges to jump to the previous constituent
  val cons = (tokens(x) ~> constituentBefore ~> constituentBefore).head
  if (POSTaggerKnown.isTraining)
    POSLabel(cons)
  else POSTaggerKnown(cons)
}
```

Pos-Tag Contextual Feature

(WorkFor(x) is true then PER(x.firstArg) is true else G(x.secondArg) is true)

ER constraint imposed on mentioned and relations
### Results: Semantic Role Labeling

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgTypeLearner^{G}(GOLDPREDs)</td>
<td>85.35</td>
<td>85.35</td>
<td>85.35</td>
</tr>
<tr>
<td>ArgTypeLearner^{G}(GOLDPREDs) + C</td>
<td>85.35</td>
<td>85.36</td>
<td>85.35</td>
</tr>
<tr>
<td>ArgTypeLearner^{Xue}(GOLDPREDs)</td>
<td>82.32</td>
<td>80.97</td>
<td>81.64</td>
</tr>
<tr>
<td>ArgTypeLearner^{Xue}(GOLDPREDs) + C</td>
<td>82.90</td>
<td>80.70</td>
<td>81.79</td>
</tr>
<tr>
<td>ArgTypeLearner^{Xue}(PREDPREDs)</td>
<td>82.47</td>
<td>80.79</td>
<td>81.62</td>
</tr>
<tr>
<td>ArgTypeLearner^{Xue}(PREDPREDs) + C</td>
<td>83.62</td>
<td>80.54</td>
<td>82.05</td>
</tr>
<tr>
<td>ArgIdentifier^{Xue}</td>
<td>ArgTypeLearner^{Xue}(PREDPREDs)</td>
<td>82.55</td>
<td>81.59</td>
</tr>
<tr>
<td>ArgIdentifier^{G}(PREDPREDs)</td>
<td>95.51</td>
<td>94.19</td>
<td>94.85</td>
</tr>
</tbody>
</table>

Table 1: Evaluation of SRL various labels and configurations. The superscripts over the different Learners refer to the whether gold argument boundaries (G) or the Xue-Palmer heuristics (Xue) were used to generate argument candidates as input. GOLD/PREDPREDs refers to whether the Learner used gold or predicted predicates. ‘C’ refers to the use of constraints during prediction and |denotes the pipeline architecture.
Results: PoS-Tagging and ER

Table 2: The performance of the POStagger, tested on sections 22–24 of the WSJ portion of the Penn Treebank (Marcus et al., 1993).

<table>
<thead>
<tr>
<th>Setting</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count-based baseline</td>
<td>91.80%</td>
</tr>
<tr>
<td>Unknown Classifier</td>
<td>77.09%</td>
</tr>
<tr>
<td>Known Classifier</td>
<td>94.92%</td>
</tr>
<tr>
<td>Combined Known-Unknown</td>
<td>96.69%</td>
</tr>
</tbody>
</table>

Table 3: 5-fold CV performance of the fine-grained entity (E) and relation (R) extraction on Newswire and Broadcast News section of ACE-2005.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mention Coarse-Label</td>
<td>77.14</td>
<td>70.62</td>
<td>73.73</td>
</tr>
<tr>
<td>Mention Fine-Label</td>
<td>73.49</td>
<td>65.46</td>
<td>69.24</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td>54.09</td>
<td>43.89</td>
<td>50.48</td>
</tr>
<tr>
<td>+ Sampling</td>
<td>52.48</td>
<td>56.78</td>
<td>54.54</td>
</tr>
<tr>
<td>+ Sampling + Brown</td>
<td>54.43</td>
<td>54.23</td>
<td>54.33</td>
</tr>
<tr>
<td>+ Sampling + Brown + HCons</td>
<td>55.82</td>
<td>53.42</td>
<td>54.59</td>
</tr>
</tbody>
</table>
Conclusion

Better Call Saul:....

https://github.com/IllinoisCogComp/saul
Conclusion

- Saul language facilitates modeling NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances
Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code**.

https://github.com/IllinoisCogComp/saul
Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances

- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a few lines of declarative code.

- Saul increases the **reusability** of codes
  - when data and knowledge about the problem increases
  - when we get to use new emerging algorithms

https://github.com/IllinoisCogComp/saul
Conclusion

- Saul language facilitates modeling NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances
- Saul saves Programming time
  - in designing various configurations and experimentation
    - each configuration is expressed in a few lines of declarative code.
- Saul increases the reusability of codes
  - when data and knowledge about the problem increases
  - when we get to use new emerging algorithms

The Novel Ideas include:

https://github.com/IllinoisCogComp/saul
Conclusion

- Saul language facilitates modeling NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances
- Saul saves Programming time
  - in designing various configurations and experimentation
    - each configuration is expressed in a few lines of declarative code.
- Saul increases the reusability of codes
  - when data and knowledge about the problem increases
  - when we get to use new emerging algorithms

The Novel Ideas include:
- Programming for decompositions of a global optimizations that are used in training and prediction
Conclusion

- Saul language facilitates modeling NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances
- Saul saves Programming time
  - in designing various configurations and experimentation
    - each configuration is expressed in a few lines of declarative code.
- Saul increases the reusability of codes
  - when data and knowledge about the problem increases
  - when we get to use new emerging algorithms

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unifying various formalisms for learning and reasoning which is an ongoing work

Better Call Saul:...
Conclusion

- Saul language facilitates modeling NLP applications that need:
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of expert knowledge beyond data instances
- Saul saves Programming time
  - in designing various configurations and experimentation
    - each configuration is expressed in a few lines of declarative code.
- Saul increases the reusability of codes
  - when data and knowledge about the problem increases
  - when we get to use new emerging algorithms

The Novel Ideas include:
- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unifying various formalisms for learning and reasoning which is an ongoing work
- A unified language for graph querying and structured learning
Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a **few lines of declarative code**.
- Saul increases the **reusability** of codes
  - when **data** and **knowledge** about the problem **increases**
  - when we get to use **new emerging algorithms**

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unifying various formalisms for learning and reasoning which is an ongoing work
- A unified language for graph querying and structured learning

Better Call Saul:...
Conclusion

- Saul language **facilitates modeling** NLP applications that need
  - Learning and reasoning over Structures
  - Relational Feature Extraction
  - Direct use of **expert knowledge** beyond data instances
- Saul saves **Programming time**
  - in designing various configurations and experimentation
    - each configuration is expressed in a few lines of declarative code.
- Saul increases the **reusability** of codes
  - when data and knowledge about the problem **increases**
  - when we get to use new emerging algorithms

The Novel Ideas include:

- Programming for decompositions of a global optimizations that are used in training and prediction
- An abstraction for unified expression of key ideas for learning and reasoning which is

I have two open PhD positions and one postdoc position, please contact me at pkordjam@tulane.edu, if you are interested!
def textAnnotationToTree(ta: TextAnnotation): Tree[Constituent]
def textAnnotationToStringTree(ta: TextAnnotation): Tree[String]
def getPOS(x: Constituent): String
def getLemma(x: Constituent): String
def getSubtreeArguments(currentSubTrees: List[Tree[Constituent]]): List[Tree[Constituent]]
def xuPalmerCandidate(x: Constituent, y: Tree[String]): List[Relation]
def fexContextFeats(x: Constituent, featureExtractor: WordFeatureExtractor): String
...