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

Parisa Kordjamshidi[*†] **Daniel Khashabi**[‡] **Christos Christodoulopoulos**[‡]
Bhargav Mangipudi[‡] **Sameer Singh**[§] **Dan Roth**[‡]
[†] Tulane University [‡] University of Illinois at Urbana-Champaign [§] University of California, Irvine
[†] pkordjam@tulane.edu [§] sameer@uci.edu
[‡] {khashab2,christod,mangipu2,danr}@illinois.edu

Abstract

We present a novel way for designing complex joint inference and learning models using *Saul* (Kordjamshidi et al., 2015), a recently-introduced declarative learning-based programming language (DeLBP). We enrich *Saul* with components that are necessary for a broad range of learning based Natural Language Processing tasks at various levels of granularity. We illustrate these advances using three different, well-known NLP problems, and show how these generic learning and inference modules can directly exploit *Saul*'s graph-based data representation. These properties allow the programmer to easily switch between different model formulations and configurations, and consider various kinds of dependencies and correlations among variables of interest with minimal programming effort. We argue that *Saul* provides an extremely useful paradigm both for the design of advanced NLP systems and for supporting advanced research in NLP.

1 Introduction

Most of the problems in natural language processing domain can be viewed as a mapping from an input structure to an output structure that represents lexical, syntactical or semantic aspects of the text. For example, Part-of-Speech (POS) tagging provides a syntactic representation, Semantic Role Labeling (SRL) is a (shallow) semantic representation, and all variations of information extraction such as Entity-Relation (ER) extraction provide a lightweight semantic representation of unstructured textual data. Even though the text data looks initially unstructured, solving such problems requires one to consider various kinds of relationships between linguistic components at multiple levels of granularity.

However, designing machine learning models that deal with structured representations is a challenging problem. Using such representations is challenging in that designing a new learning model or tackling a new task requires a significant amount of task-specific and model-specific programming effort for learning and inference. Additionally, global background knowledge in these models is usually hard-coded, and changing or augmenting it is extremely time-consuming. There are several formal frameworks (Tsochantaridis et al., 2004; Chang et al., 2013) for training structured output models but these provide no generic solution for doing inference on arbitrary structures. Likewise, probabilistic programming languages (Pfeffer (2009), McCallum et al. (2009) *inter alia*) try to provide generic probabilistic solutions to arbitrary inference problems but using them in the context of training arbitrary structured output prediction models for real world problems is a challenge; in addition, encoding structured features and high-level background knowledge necessary for these problems is not a component of those frameworks.

In this paper we build on the abstractions made available in *Saul* programming language (Kordjamshidi et al., 2015), in order to create a unified and flexible machine learning programming framework using the generic Constrained-Conditional-Model (CCM) paradigm (Chang et al., 2012).

Specifically, we build upon *Saul*'s graph-based data representation and enrich it with primitive structures and *sensors* for the NLP domain that facilitate operating at arbitrary levels of 'globality' for learning and inference. For example, a programmer can easily decide if he or she needs to operate at document,

[*] Most of the work was performed at the University of Illinois.

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 3030–3040, Osaka, Japan, December 11-17 2016.

sentence, or phrase level. In other words, without any programming effort the user can specify at which level of granularity the context should be considered, and how 'global' should learning and inference be. This ability also better supports the declarative specification of the structured features and is useful in a wide range of tasks that involve mapping the language to its syntactic or semantic structure. Using this new framework, we show how one can design structured output prediction models in an easy and flexible way for several well-known and challenging NLP tasks and achieve comparable results to the existing state-of-the-art models.

The paper is structured as follows: Section 2, describes the general formulation used for structured output prediction and its possible configurations. Section 3 explains how the structured input and output spaces are represented as a graph, the form of the objective and the way arbitrary dependencies can be represented in *Saul*. In Section 4, various model configurations are presented for our case study tasks, alongside experimental results. Section 5 provides a brief overview of related work. Section 6 concludes.

2 Structured Output Prediction

Punyakanok et al. (2005) describe three fundamentally different and high level solutions towards designing structured output prediction models,

(a) **Learning Only (LO):** Local classifiers are trained to predict each output component independently;

(b) **Learning plus inference (L+I):** Training is performed locally as in the LO model, but global constraints/dependencies among components are imposed during prediction (Chang et al., 2012). In the context of training probabilistic graphical models this is referred to as *piecewise training* (Sutton and McCallum, 2009);

(c) **Inference based training (IBT):** Here, during the training phase, predictions are made globally so that constraints and dependencies among the output variables are incorporated into the training process (Collins, 2004; Taskar et al., 2002; Tsochantaridis et al., 2004).

When training structured output models there is a spectrum of *configurations* (model compositions) between the two extremes – only local training as in the LO and L+I schemes and the full global training as in the IBT scheme (Samdani and Roth, 2012). The key here is choosing the best decomposition of the variables/structures which is largely an empirical question; having an expressive and flexible machinery for modeling the data and for learning from it is thus useful and eases in designing, assessing, decomposing and improving the models (Kordjamshidi and Moens, 2013). With this as motivation, we aim here to enrich *Saul* with components that facilitate these analyses in the NLP domain (see Section 4).

We first introduce the notation and the formal framework for designing global (IBT) models in *Saul*. In supervised *structured* learning we are given a set of examples i.e. pairs of input and output, $E = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) \in \mathcal{X} \times \mathcal{Y} : i = 1 \ldots N\}$, where both inputs ($\mathcal{X}$) and outputs ($\mathcal{Y}$) can be complex structures; the goal is learning the mapping, $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ (Bakir et al., 2007). Making predictions in this formulation requires an inference procedure over g, that finds the best $\hat{\boldsymbol{y}}$ for a given $\boldsymbol{x}$. Thus the prediction function h is, $h(\boldsymbol{x}; \boldsymbol{w}) = \arg\max_{\hat{\boldsymbol{y}} \in \mathcal{Y}} g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$. In the generalized linear models (Tsochantaridis et al., 2005), the function g is assumed to be a linear model of the input and output features $f(\boldsymbol{x}, \boldsymbol{y})$, i.e. $g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \langle \boldsymbol{w}, f(\boldsymbol{x}, \boldsymbol{y}) \rangle$, where $\boldsymbol{w}$ denotes the parameters of the model (weight vector). A commonly used discriminative training method is to minimize the following convex upper bound of the loss function over the training data (Tsochantaridis et al., 2004):

$$L = \sum_{i=1}^{N} \max_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \Big[g(\boldsymbol{x}^i, \hat{\boldsymbol{y}}; \boldsymbol{w}) - g(\boldsymbol{x}^i, \boldsymbol{y}^i; \boldsymbol{w}) + \Delta(\boldsymbol{y}^i, \hat{\boldsymbol{y}}) \Big]$$

The inner maximization is called loss-augmented inference and quantifies the most violated output per training example. This is a crucial inference problem to be solved during training of such models. Here we assume that the distance function Δ is decomposed in the same way as the feature function. During training and at prediction time, there is a need to solve the same inference problem to find the best $\hat{\boldsymbol{y}}$

given the objective $g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$. This will be the focus of the paper: to show how we can write high level specifications of g and model it in a generic, efficient and flexible fashion.

3 Graph Representation

Saul is a powerful programming paradigm which uses graphs to explicitly declare the structure of the data that serves as the model of the domain. This graph is called *data-model*[1] and is comprised of nodes, edges and properties that describe nodes. The *data-model* is a global structure that facilitates designing a wide range of learning and inference *configurations*, based on arbitrary model decompositions.

Inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$ are sub-graphs of the *data-model* and each learning model can pick specific correlations and substructures from it. In other words, $\boldsymbol{x}$ is a set of nodes $\{x_1, \ldots, x_K\}$ and each node has a *type* p. Each $x_k \in \boldsymbol{x}$ is described by a set of properties; this set of properties will be converted to a feature vector ϕ_p. For instance, an input type can be a word (*atomic node*) or a pair of words (*composed node*), and each type is described by its own properties (e.g. a single word by its part of speech, the pair by the distance of the two words). The output $\boldsymbol{y}$ is represented by a set of *labels* $\boldsymbol{l} = \{l_1, \ldots, l_P\}$ each of which is a *property* of a node. The labels can have semantic relationships. We conceptually (not technically) distinguish between two types of labels: the *single labels* and *linked labels* that refer to an independent concept and to a configuration of a number of related single labels respectively. *Linked label*s can represent different types of semantic relationships between single labels.

For convenience, to show which labels are connected by a *linked label*, we represent the *linked labels* by a concatenation of the labels' names that are linked together and construct a bigger semantic part of the whole output. For example an SRL predicate-argument relation (see Figure 1 in Section 4) can be denoted by *pred-arg* meaning that it is *composed-of* the two single labels, *pred* (predicate), and *arg* (argument). The labels are defined with a graph query that extracts a property from the *data-model*. The $l_p(x_k)$ or shorter l_{pk} denotes an indicator function indicating whether component x_k has the label l_p. Kordjamshidi et al. (2015) introduced the term *relational constrained factor graph* to represent each possible learning and inference configuration (Taskar et al., 2002; Taskar et al., 2004; Bunescu and Mooney, 2007; Martins et al., 2011).

The structure of the learning/inference i.e. the relational constraint factor graph is specified with a set of *templates* which can be constraint templates or feature templates, $\mathcal{C} = \{\mathcal{C}_1, .., \mathcal{C}_P\}$. Each template $\mathcal{C}_p \in \mathcal{C}$ is specified by three main components: 1) *A subset of joint features*, denoted by $f_p(\boldsymbol{x}_k, l_p)$, where $\boldsymbol{x}_k$ is an input component that is a node in the *data-model* graph, and l_p is a single/linked label (a property in the *data-model*). In the case of constraint templates, l_p is a Boolean label denoting the satisfiability of the constraint. 2) *A candidate generator*, that generates candidate components upon which the specified subset of joint features is applicable, the set of candidates for each template is denoted by C_{l_p}. For constraint templates the candidate generator is the propositionalization of the constraint's first-order logical expression. 3) *A block of weights* $\boldsymbol{w}_p$, which is a part of the main weight vector $\boldsymbol{w}$ of the model and is associated to the local joint feature function of $\mathcal{C}_p$. In general, $\boldsymbol{w}_p$ can also be defined for the constraint templates. The main objective g is written in terms of the instantiations of the (feature) templates and their related blocks of weights $\boldsymbol{w}_p$ in $\boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_P]$,

$$g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \sum_{l_p \in \boldsymbol{l}} \sum_{x_k \in C_{l_p}} \langle \boldsymbol{w}_p, f_p(x_k, l_p) \rangle = \sum_{l_p \in \boldsymbol{l}} \sum_{x_k \in C_{l_p}} \langle \boldsymbol{w}_p, \phi_p(x_k) \rangle \, l_{pk} = \sum_{l_p \in \boldsymbol{l}} \left\langle \boldsymbol{w}_p, \sum_{x_k \in C_{l_p}} (\phi_p(x_k) l_{pk}) \right\rangle, \quad (1)$$

where the local joint feature vector $f_p(\boldsymbol{x}_k, l_p)$, is an instantiation of the template $\mathcal{C}_{l_p}$ for candidate $\boldsymbol{x}_k$. This feature vector is computed by scalar multiplication of the input feature vector of $\boldsymbol{x}_k$ (i.e. $\phi_p(\boldsymbol{x}_k)$), and the output label l_{pk}.

Given this objective, we can view the inference task as a *combinatorial constrained optimization* given the polynomial g which is written in terms of labels, subject to the constraints that describe the relationships between the labels (either single or linked labels). For example, the *is-a* relationships can be defined as the following constraint, $(l(\boldsymbol{x}_c) \; is \; 1) \Rightarrow (l'(\boldsymbol{x}_c) \; is \; 1)$, where l and l' are two distinct

[1] Kordjamshidi et al. (2015) used the term *model-graph*.

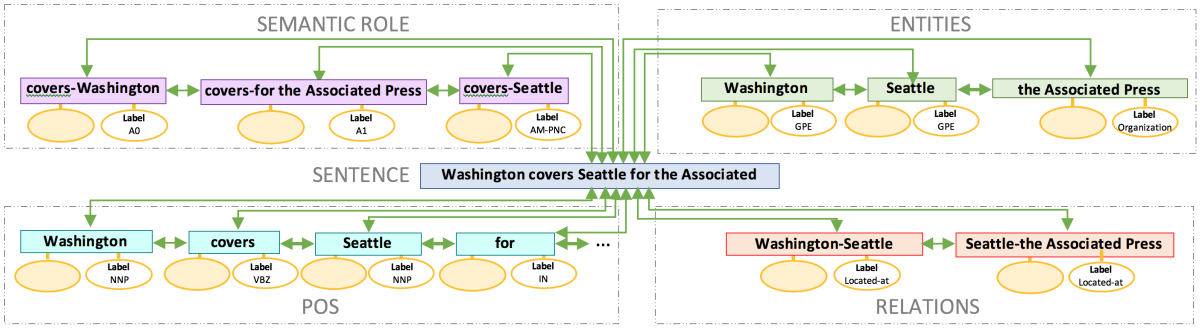


Figure 1: An instantiation of the *data-model* for the NLP domain. The colored ovals are some observed properties, while the white ones show the unknown labels. For the POS and Entity Recognition tasks, the boxes represent candidates for *single labels*; for the SRL and Relation Extraction tasks, they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

labels that are applicable on the node with the same type of $\boldsymbol{x}_c$. 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), $g = \langle \boldsymbol{w}, f(\boldsymbol{x}, \boldsymbol{y}) \rangle - \langle \rho, c(\boldsymbol{x}, \boldsymbol{y}) \rangle$, where c is the constraint function and ρ 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:

$$g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \sum_{l_p \in \boldsymbol{l}} \sum_{\boldsymbol{x}_k \in \{\tau\}} \langle \boldsymbol{w}_p, f_p(\boldsymbol{x}_k, l_{pk}) \rangle + \sum_{m=1}^{|C|} \rho_m c_m(\boldsymbol{x}, \boldsymbol{y}), \tag{2}$$

where 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 `Learnable` construct and first order constraints using `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 EDISON, 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[2].

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[3].

[2]https://github.com/IllinoisCogComp/saul/blob/master/saul-core/doc/DATAMODELING.md
[3]https://github.com/IllinoisCogComp/saul/tree/master/saul-examples/src/main/scala/edu/illinois/cs/cogcomp/saulexamples/nlp

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

Figure 2: An Example of *data-model* declarations for nodes, edges, properties and using sensors. The `sentences` nodes are of type `TextAnnotation` class, which is a part of *Saul*'s underlying NLP library; many predefined sensors can be applied on it to generate various nodes of type `Constituent` and `Relations`, properties of those nodes and establish edges between them.

4.1 Semantic Role Labeling

SRL (Carreras and Màrquez, 2004) is a shallow semantic analysis framework, whereby a sentence is analysed into multiple propositions; each one consisting of a predicate and one or more core arguments, labeled with protosemantic roles (agents [Arg0], patient/theme [Arg1], beneficiary [Arg2], etc.), and zero or more optional arguments, labeled according to their semantic function (temporal, locative, manner, etc.). See Figure 1 for an example annotation.

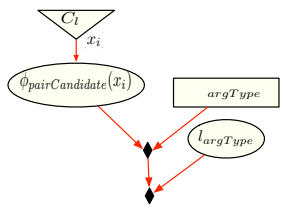
4.1.1 Input-Output Spaces

Each sentence is a node in the *data-model*, comprised of *constituents* (derived from a tokenizer *Sensor*). These constituents are atomic components of $\boldsymbol{x}$ (see Figure 1) and are identified as $\boldsymbol{x} = \{x_1, \ldots, x_4\}$, where x_i is the identifier of the ith constituent in the sentence. Each constituent is described by a number of properties (word-form, pos-tag, ...) and the corresponding feature vector representation of these properties is denoted by $\phi_{constituent}(x_i)$. There are also composed components – pairs of constituents; their descriptive vectors are referred to as $\phi_{pair}(x_i, x_j)$. The feature vector of a composed component such as a pair, $\phi_{pair}(x_1, x_2)$ is usually described by the local features of x_1, x_2 and the relational features between them, such as the order of their position, etc.

The main labels set for the SRL model is $\boldsymbol{l} = \{l_{isPred}, l_{isArg}, l_{argType}\}$ which indicate whether a constituent is a predicate, whether it is an argument and the argument role respectively. l_{isArg} and $l_{argType}$ are *linked labels*, meaning that they are defined with respect to another constituent (the predicate). Depending on the type of correlations we aim to capture, we can introduce new *linked labels* in the model. These labels are not necessarily the target of the predictions but they help to capture the dependencies among labels. For example, to capture the long distance dependencies between two different arguments of same predicate we can introduce a *linked label* linking the label of two pairs and impose consistency constraints between this new *linked label* and the label of each pair. Figure 2 shows some declarations of the *data-model*'s graph representing input and output components of the learning models. The graph can be queried using our invented graph traversal language and the queries are directly used as structured machine learning features later.

4.1.2 Classifiers and Constraints

As mentioned in Section 3, the structure of a learning model is specified by templates which are defined as classifiers (feature templates) and global constraints (constraint templates) in *Saul*. SRL has four main feature templates: 1) *Predicate* template: connects an input constituent to a single label l_{isPred}. The input features of this template are generated based on the properties of the constituents $\phi_{constituent}$. The candidate generator of this template is a filter that takes all constituents whose pos-tag is *VP*; 2) *Argument* template: connects a pair of constituents to a *linked label* l_{isArg}. The candidate generator of this template is a set of rules that are suggested by Xue and Palmer (2004); 3) *ArgumentType* template: connects a pair of constituents to the linked label $l_{argType}$. Same Xue-Palmer heuristics are used; 4) *ArgumentTypeCorrelations* template: connects two pairs of pairs of constituent (i.e. relations between relations) to their join linked label. The candidates are the pairs of Xue-Palmer candidates.




```
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)
}
```

Figure 3: Left: shows the components of the *ArgumentType* feature template. $l_{argType}$ is one *linked label* as a part of the objective in Equation 1, along with the corresponding block of weights and the `pair` candidates (diamonds show dot products). Right: shows the code for the template. `label` and `feature` are respectively one property and a list of properties of `pair` nodes declared in the *data-model*, these serve as the output and input parts of this template. This template can be used as a local classifier or as a part of the objective of a global model, depending on the indicated learning paradigm by the programmer.

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

def classiferLabelIsLegal(classifier, relation, legalLabels) = {
    legalLabels._exists { l => (classifier on relation is l) }
}
```

Figure 4: Given a predicate, some argument types are illegal according to PropBank Frames (e.g. the verb ‘cover‘ with sense 03 can take only Arg0 or Arg1), which means that they should be excluded from the inference. The `legalArguments(y)` returns the predefined list of legal arguments for a predicate y. In line 3, graph traversal queries (using the $\sim>$ operator) are applied to use an edge and go from a *sentence* node to all contained predicate nodes in the sentence and then apply the constraint to all of those predicates. Each constraint imposes the `argumentTypeLearner` to assign a legal argument type to each candidate argument or does not count it as an argument at all, i.e., to assign `none` value to the argument type.

The feature templates are instances of `Learnable` in *Saul* and in fact they are treated as local classifiers. The script of Figure 3 shows the *ArgumentType* template. The *Constraints* are specified by means of first-order logical expressions. We use the constraints specified in Punyakanok et al. (2008) in our models. The script in Figure 4, shows an example expressing the *legal argument* constraints for a sentence.

4.1.3 Model Configurations

Programming for learning and inference configurations in *Saul* is simply composing the basic building blocks of the language, that is, feature and constraint templates in different ways.

Local models. Training local models is as easy as calling the train function over each specified feature template separately (e.g. `ArgTypeLearner.train()`). The test on these models also is simply done by calling test for each template (e.g. `ArgTypeLearner.test()`). In addition, `TestClassifeirs(/*a list of classifiers*/)` and `TrainClassifiers(/*a list of classifiers*/)` can be used to train/test a number of classifiers independently by passing a list of classifier's names to these functions. The training algorithm can be specified when declaring the `Learnable`; here we have used averaged perceptrons in the experiments which is the default model.

Model	Precision	Recall	F1
`ArgTypeLearner`^{G}(GOLDPREDS)	85.35	85.35	85.35
`ArgTypeLearner`^{G}(GOLDPREDS) + C	85.35	85.36	85.35
`ArgTypeLearner`^{Xue}(GOLDPREDS)	82.32	80.97	81.64
`ArgTypeLearner`^{Xue}(GOLDPREDS) + C	82.90	80.70	81.79
`ArgTypeLearner`^{Xue}(PREDPREDS)	82.47	80.79	81.62
`ArgTypeLearner`^{Xue}(PREDPREDS) + C	83.62	80.54	82.05
`ArgIdentifier`^{Xue} | `ArgTypeLearner`^{Xue}(PREDPREDS)	82.55	81.59	82.07
`ArgIdentifier`^{G}(PREDPREDS)	95.51	94.19	94.85

Table 1: Evaluation of SRL various labels and configurations. The superscripts over the different `Learner`s 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.

Pipeline. Previous research on SRL (Punyakanok et al., 2008) shows that a good working model is the one that first decides on argument identification and then takes those arguments and decides about their roles. This configuration is made with a very minor change in the templates of the local models. Instead of using Xue-Palmer candidates, we can use the identified arguments by a `isArgument` classifier as input candidates for the `ArgTypeLearner` model. The rest of the model is the same.

L+I model. This is simply a locally trained classifier that uses a number of constraints on prediction time. We define a constrained argument predictor based on a previously trained local *Learnable* as follows:

```
object ArgTypeConstraintClassifier extends ConstrainedClassifier(ArgTypeLearner)
{
    def subjectTo = srlConstraints
}
```

where the `srlConstraints` is a constraint template. Having this definition we only need to call the `ArgTypeConstraintClassifier` constraint predictor during the test time as `ArgTypeConstraintClassifier(x)` which decides for the label of `x` in a global context.

IBT model. The linguistic background knowledge about SRL that is described in Section 4.1.2 provides the possibility of designing a variety of global models. The constraints that limit the argument arrangement around a specific predicate help to make sentence level decisions for each predicate during training phase and/or prediction phase. To train the global models we simply call the joint train function and provide the list of all declared constraint classifiers as parameters.

The results of some versions of these models are shown in Table 1. The experimental settings, the data and the train/test splits are according to (Punyakanok et al., 2008) and the results are comparable. As the results show the models that use constraints are the best performing ones. For SRL the global background knowledge on the arguments in IBT setting did not improve the results.

4.2 Part-Of-Speech Tagging

This is perhaps the most often used application in ML for NLP. We use the setting proposed by Roth and Zelenko (1998) as the basis for our experiments. The graph of an example sentence is shown in Figure 1. We model the problem as a single-node graph representing constituents in sentences. We make use of context window features and hence our graph has edges between each token and its context window. This enables us to define contextual features by traversing the relevant edges to access tokens in the context. The following code uses the gold POS-tag label (`POSLabel`) of the two tokens before the current token during training and POS-tag classifier's prediction (`POSTaggerKnown`) of the two tokens before the current token during the test.

```
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)
}
```

4.2.1 Model configurations

Here, we point to a few interesting settings for this problem and report the results we obtained by *Saul* in Table 2.

Count-based baseline. The simplest scenario is to create a simple count-based baseline: for each constituent choose the most popular label. This is trivial to program in *Saul*.

Independent classifiers. We train independent classifiers for known and unknown words. Though both classifiers use similar sets of features, the unknown classifier is trained only on tokens that were seen fewer than 5 times in the training data. Here the 'Learnable' is defined as exampled in Section 4.1.2.

Classifier combination. Given the known and unknown classifiers, one easy extension is to combine them, depending whether the input instance is seen during the training phase or not. To code this, the defined Learnables for the two classifiers are simply reused in an 'if' construct.

Sequence tagging. One can extend the previous configurations by training higher-order classifiers, i.e. classifiers trained on pair/tuple of neighboring constituents (similar to HMM or chain-CRF). At the prediction time one needs to choose the best structure by doing constrained inference on the predictions of the local classifiers. The following snippet shows how one can simply write a consistency constraint, given a pairwise classifier `POSTaggerPairwise` which scores two consecutive constituents.

```
def sentenceLabelsMatch = constraint(sentences) {
    t: TextAnnotation =>
    val constituents = t.getView(ViewNames.TOKENS).getConstituents
    // Go through a sliding window of tokens
    constituents.sliding(3)._forall { cons: List[Constituent] =>
      POSTaggerPairwise on (cons(0), cons(1)).second === POSTaggerPairwise on (
    cons(1), cons(2)).first }
}
```

4.3 Entity-Relation extraction

This task is for labeling entities and recognizing semantic relations among them. It requires making several local decisions (identifying named entities in the sentence) to support the relation identification. The models we represent here are inspired some well-known previous work (Zhou et al., 2005; Chan and Roth, 2010). The nodes in our models consists of `Sentences`, `Mentions` and `Relations`.

4.3.1 Features and Constraints

For the entity extraction classifier, we define various lexical features for each mention – head word, POStags, words and POStags in a context window. Also, we incorporate some features based on gazetteers for organization, vehicle, weapons, geographic locations, proper names and collective nouns. The relation extraction classifier uses lexical, collocation and dependency-based features from the baseline implementation in Chan and Roth (2010). We also use features from the brown word clusters (Brown et al., 1992). The features for each word are based on a path from the root in its Brown clustering representation. These features are easily available in our NLP *data-model*. We also use a decayed down-sampling of negative examples between training iterations.

Setting	Accuracy
Count-based baseline	91.80%
Unknown Classifier	77.09%
Known Classifier	94.92 %
Combined Known-Unknown	96.69%

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

	Scenario	Precision	Recall	F1
E	Mention Coarse-Label	77.14	70.62	73.73
	Mention Fine-Label	73.49	65.46	69.24
R	Basic	54.09	43.89	50.48
	+ Sampling	52.48	56.78	54.54
	+ Sampling + Brown	54.43	54.23	54.33
	+ Sampling + Brown + HCons	55.82	53.42	54.59

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.

Relation hierarchy constraint. Since the coarse and fine labels follow a strict hierarchy, we leverage this information to boost the prediction of the fine-grained classifier by constraining its prediction upon the (more reliable) coarse-grained relation classifier.

4.3.2 Model Configuration

Entity type classifier. For the entity type task, we train two independent classifiers - one for coarse-label and the second for the fine-grained entity type. We generate the candidates for entities by taking all nouns and possessive pronouns, base noun phrases, selective chunks from the shallow parse and named entities annotated by the NE tagger of Ratinov and Roth (2009).

Relation type classifier. For the relation types, we train two independent classifiers - coarse-grained relation type label and fine-grained relation type label. We use features from our unified *data-model* which are `properties` defined on the `relations` node in the data-model graph. We also incorporate the Relation Hierarchy constraint during inference so that the predictions of both classifiers are coherent. We report some of our results in Table 3.

5 Related Work

This work has been done in the context of *Saul*, a recently developed declarative learning based programming language. DeLBP is a new paradigm (Roth, 2005; Rizzolo, 2011; Kordjamshidi et al., 2015) which is related to probabilistic programming languages (PPL) (Pfeffer, 2009; McCallum et al., 2009) *(inter alia)*, sharing the goal of facilitating the design of learning and inference models. However, compared to PPL, it is aimed at non-expert users of machine learning, and it is a more generic framework that is not limited to probabilistic models. It focuses on learning over complex structures where there are global correlations between variables, and where first order background knowledge about the data and domain could be easily considered during learning and inference. The desideratum of this framework is the conceptual representation of the domain, data and knowledge, in a way that is suitable for non-experts in machine learning and, it considers the aspect of relational feature extraction; this is different also from the goals of Searn (Hal et al., 2009) and Wolf (Riedel et al., 2014). DeLBP focuses on data-driven learning and reasoning for problem solving and handling collections of data from heterogeneous resources, unlike Dyna (Eisner, 2008) which is a generic declarative problem solving paradigm based on dynamic programming. This paper exhibits the capabilities and flexibility of Saul for solving problems in the NLP domain. Specifically, it shows how a unified predefined NLP *data-model* can help performing various tasks at various granularity levels.

6 Conclusion

We presented three examples of NLP applications as defined in the declarative learning-based programming language *Saul*. The main advantage of our approach compared to traditional, task-specific, systems is that *Saul* allows one to define all the components of the models declaratively, from feature extraction to learning and inference with arbitrary structures. This allows designers and researchers a way to explore different way to decompose, learn and do inference and easily gain insights into the impact of these on the

task. We enriched *Saul* with an extensive NLP *data-model* that enables users to perform various tasks at different levels of granularity and eventually to perform multiple tasks jointly. This work will help pave the way for more learning-based programming applications which will allow both practitioners and researchers in the field to develop quick solutions to advanced NLP tasks and to focus on exploring the tasks while staying at a sufficient level of abstraction from the component's implementation.

Acknowledgments

The authors would like to thank all the students who have helped in the implementation of this project, as well as the anonymous reviewers for helpful comments. This research is supported by NIH grant U54-GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov), and Allen Institute for Artificial Intelligence (`allenai.org`). This material is based on research sponsored by DARPA under agreement number FA8750-13-2-0008. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government nor NIH.

References

G. H. Bakir, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan. 2007. *Predicting Structured Data (Neural Information Processing)*. The MIT Press.

P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, pages 467–479.

R. Bunescu and R. J. Mooney. 2007. Statistical relational learning for natural language information extraction. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*, pages 535–552. MIT Press.

X. Carreras and L. Màrquez. 2004. Introduction to the CoNLL-2004 shared tasks: Semantic role labeling. In *Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL)*, pages 89–97. Boston, MA, USA.

Y. Chan and D. Roth. 2010. Exploiting background knowledge for relation extraction. In *Proc. of the International Conference on Computational Linguistics (COLING)*, Beijing, China.

M. Chang, L. Ratinov, and D. Roth. 2012. Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431, 6.

K.-W. Chang, V. Srikumar, and D. Roth. 2013. Multi-core structural svm training. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

M. Collins. 2004. Parameter estimation for statistical parsing models: theory and practice of distribution-free methods. In *New Developments in Parsing Technology*, pages 19–55. Kluwer.

J. Eisner. 2008. Dyna: A *non*-probabilistic programming language for probabilistic AI. Extended abstract for talk at the NIPS*2008 Workshop on Probabilistic Programming.

D. Hal, J. Langford, and D. Marcu. 2009. Search-based structured prediction. *Machine Learning*, pages 297–325, June.

P. Kordjamshidi and M-F. Moens. 2013. Designing constructive machine learning models based on generalized linear learning techniques. In *NIPS Workshop on Constructive Machine Learning*.

P. Kordjamshidi, D. Roth, and H. Wu. 2015. Saul: Towards declarative learning based programming. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.

M. P. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June.

A. FT. Martins, M. AT. Figeuiredo, P. MQ. Aguiar, N. A. Smith, and E. P. Xing. 2011. An augmented Lagrangian approach to constrained MAP inference. In *International Conference on Machine Learning (ICML)*.

A. McCallum, K. Schultz, and S. Singh. 2009. FACTORIE: Probabilistic Programming via Imperatively Defined Factor Graphs. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*.

A. Pfeffer. 2009. Figaro: An object-oriented probabilistic programming language. Technical report, Charles River Analytics.

V. Punyakanok, D. Roth, W. Yih, and D. Zimak. 2005. Learning and inference over constrained output. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1124–1129.

V. Punyakanok, D. Roth, and W. Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*.

L. Ratinov and D. Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*.

S. Riedel, S. Singh, V. Srikumar, T. Rocktäschel, L. Visengeriyeva, and J. Noessner. 2014. WOLFE: strength reduction and approximate programming for probabilistic programming. *Statistical Relational Artificial Intelligence*.

N. Rizzolo. 2011. *Learning Based Programming*. Ph.D. thesis, University of Illinois, Urbana-Champaign. http://cogcomp.cs.illinois.edu/papers/Rizzolo11.pdf.

D. Roth and D. Zelenko. 1998. Part of speech tagging using a network of linear separators. In *The 17th International Conference on Computational Linguistics (COLING-ACL)*, pages 1136–1142.

D. Roth. 2005. Learning based programming. *Innovations in Machine Learning: Theory and Applications*.

R. Samdani and D. Roth. 2012. Efficient decomposed learning for structured prediction. In *Proc. of the International Conference on Machine Learning (ICML)*.

M. Sammons, C. Christodoulopoulos, P. Kordjamshidi, D. Khashabi, V. Srikumar, P. Vijayakumar, M. Bokhari, X. Wu, and D. Roth. 2016. Edison: Feature Extraction for NLP, Simplified. In *LREC*.

C. Sutton and A. McCallum. 2009. Piecewise training for structured prediction. *Machine Learning*, pages 165–194.

B. Taskar, P. Abbeel, and D. Koller. 2002. Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI, pages 485–492. Morgan Kaufmann Publishers Inc.

B. Taskar, M. Fai Wong, P. Abbeel, and D. Koller. 2004. Link prediction in relational data. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning (ICML)*.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. 2005. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research (JMLR)*, pages 1453–1484.

N. Xue and M. Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 88–94, Barcelona, Spain.

G. Zhou, J. Su, J. Zhang, and M. Zhang. 2005. Exploring various knowledge in relation extraction. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 427–434. Association for Computational Linguistics.