Relational Learning and Feature Extraction by Querying over Heterogeneous Information Networks

Parisa Kordjamshidi+ Sameer Singh++ Daniel Khashabi* Christos Christodoulopoulos Mark Summons* Saurabh Sinha* Dan Roth***

*+Tulane University *University of Illinois at Urbana-Champaign ++University of California Irvine **Amazon Research Cambridge UK*

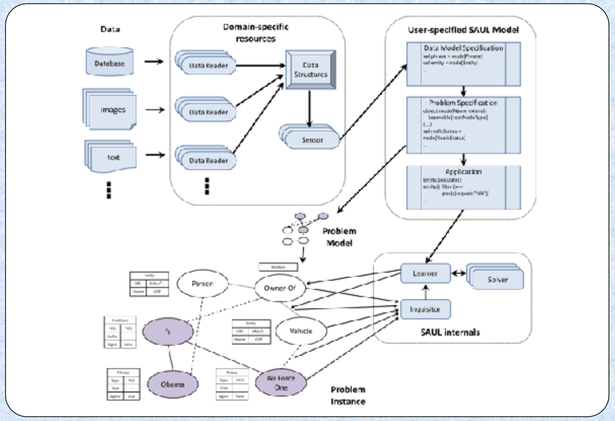
Motivation

Many real-world systems use **heterogeneous information networks** that consist of numerous interacting components of different types. Examples are Biological Networks, Social Networks or general Knowledge Graphs with arbitrary complex structures. Previous research extensively addresses the challenges of working with such networks for various mining tasks. However, a general solution for **easily constructing** such networks or **systematically manipulating** them by **various analysis units** has not yet been worked out. Mostly specialized approaches perform specific types of analysis over a network design with **task specific implementations**. Relational data representation, learning and flexible intelligent data analysis, as well as the evolution of these networks based on the analysis outcomes, need to be placed in a well-defined framework.

Saul Language Components

Saul is a declarative Learning based programming language that targets **designing relation learning models** that consider **global dependencies** and **domain knowledge in** learning and inference**.**

- Data Modeling
- Relational Feature Engineering
- Knowledge Representation
- Designing Learning and Inference Paradigms



Selected references:

[1] P. Kordjamshidi, C. Christodoulopoulos, D. Khashabi, B. Mangipudi, S. Singh, D. Roth. Better call Saul: Flexible Programming for Learning and Inference in NLP, COLING-2016.

[2] P. Kordjamshidi, D. Roth and H. Wu. Saul: Towards Declarative Learning based programming, IJCAI-2015.

[3] D. Roth. Learning based programming. Innovations in Machine Learning: Theory and Applications, 2005.

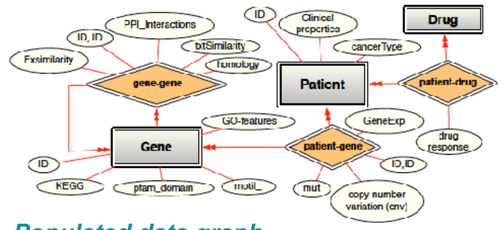
[4] N. Rizzolo and D. Roth. Learning Based Java for rapid development of NLP systems. LREC 2010.

Data Modeling

A **datamodel graph** is defined and populated, yielding a **data graph**. We provide a **language** for **specifying the schema of a typed graph;** and **interaction with messy, naturally occurring data** using **Readers, Domain BaseTypes** and **Domain Sensors.**

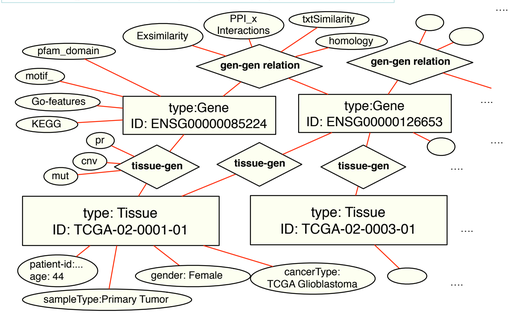
```
val patients = node[Patient]
val genes = node[Gene]
val patientGene = node[PatientGene]
val patientDrug = node[PatientDrug]
val geneGene = node[GeneGene]
val geneGenes = edge(geneGene, genes)
val drugResponse = property(patientDrug) {x: PatientDrug => x.response }
…
```

◆*Datamodel graph example of biology domain*



◆ *Populated data graph*

KnowEngDataModel.populate(inputCollections)



Graph Queries

- Graph Querying from heterogeneous resources in one unified datamodel:
  - Basic Relational Algebraic Operations
    - Filtering, Projecting, Explicit Joins (see c1)
  - Graph Matching Queries
    - Path, Neighborhood, Window (see c1)
  - Queries for learning and inference
    - **Preparing learning examples**
    - **Feature extraction (c3)**
    - **Querying over the outcome of learning and inference under global constraints expressed with logical expressions.**
- Evolving the graph based on the outcomes of learning and inference.
- Performing meta analysis on the graph

Learning and Inference

◆Local Models

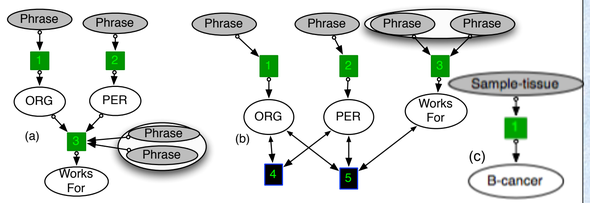
Defining classical learning models which do classification is one line of code given the feature declaration language. See example codes C2 and C4**.**

◆Global Models

The structure of the dependencies in the global models is shown using an extended notion of **factor graphs**. Each local learner represents a factor in the graph and each constraint is **constraint factor** relating factors to each other. In factor graph b, the factor 5 corresponds to the constraint c5 and factor 4 corresponds to constraint c6. We can indicate that the global constraints are used only during prediction time (L+I model) as in code c7 or we can make use of the global constraints during training and train jointly (IBT) as in code c8.

◆Pipeline Models

A pipeline model uses the prediction of other classifiers as features.



a) The FG of a pipeline model.
b) The FG of a joint model in which the labels are related with constraint factors.
c) The FG of a basic classifier.

Relational Feature Engineering

The data graph is used as a global structure via which various local or contextual features can be extracted. See (**c1** and **c3**) codes, in c3, a set of **propeerties** are defined for a node of type **Gene** in the model graph. These features include a number of **local properties** of a gene such as the KEGG pathway of it, in addition to some **contextual properties** such as the KEGG features of the genes in a window with length one which are connected to that gene in the data graph.

Knowledge Representation

To represent higher level information about the problem that are hard to represent in a graph, we use a first order like language using the **logical operators** and **quantifiers** to represent how various variables interact with each other. These are **first order constraints**, see examples in **C5** and **C6** codes. A query can ask for the prediction of an unknown variable that satisfies the global constraints.

Code Excerpts-two case studies

◆ Graph Declaration and queries, NLP

```
val sentences = node[Sentence]
val phrases = node[Phrase]
val contains_s_ph = edge(sentences,phrases)
val surface = property(phrases) {
  x => surface_sensor(x) }
contains_s_ph.addSensor(chunker)
sentences(x) ~> contains_s_ph ~> prop surface
```

◆ Basic operations (c1)

```
words().filter(x=>pos-tag(x).equals("NN"))
val joinNode = join(node1,node2)(/*body*/)
mynode(x).neighborAt(n)
mynode(x).neighborWithin(n)
mynode(x).path(y)
def feature = using(query1, query2, …)
```

◆ Local Learner Declaration (c2)

```
object CancerClassifierG extends Learnable[Phrase] {
def label = cancerType
def features=TissueFeatures}
```

◆ Relational Feature Declaration (c3)

```
val GeneFeatures = property(Gene)
{x=>{ KEGG(x)::motif_(x)::GoFeatures(x)::
KEGG(x.windo(1))::Nil}}
```

◆ Pipeline Model- ER (c4)

```
Object WorkForPipe extends Learnable[Pair]{
def label = retype is "Work-For"
def features=RelationFeatures}
```

◆Constraint Declaration (c5)- ER

```
val WorkFor=constraintOf[Pairs] {
x:Pairs => {
((WorkFor on x) is true) ==>
((PER on x.firstArg) is true) and
((WorkFor on x) is true) ==>
((ORG on x.secondArg) is true) } }
```

◆ Constraint Declaration (c6) -ER

```
val SingleEntityLabel=constraintOf[Phrase]{
x:Phrase=>{
((PER on x) is true)==>((ORG on x) is false)
and
((ORG on x) is true)==>((PER on x) is false)}}
```

◆ L+I model (c7) -ER

```
object WorkForCCM extends ConstraintClassifier[Pairs]
{def subjectTo= WorkFor}
```

◆ IBT Model (c8) -ER

```
object WorkForJoint extends ConstraintLearner
[Pairs]{def subjectTo= WorkFor }
```

Supported by grant 1U54GM114838, trans-NIH Big Data to Knowledge (BD2K) and by DARPA under agreement number FA8750-13-2-0008.

Tulane University

COGNITIVE COMPUTATION GROUP UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

knoweng BIG DATA TO KNOWLEDGE CENTER OF EXCELLENCE