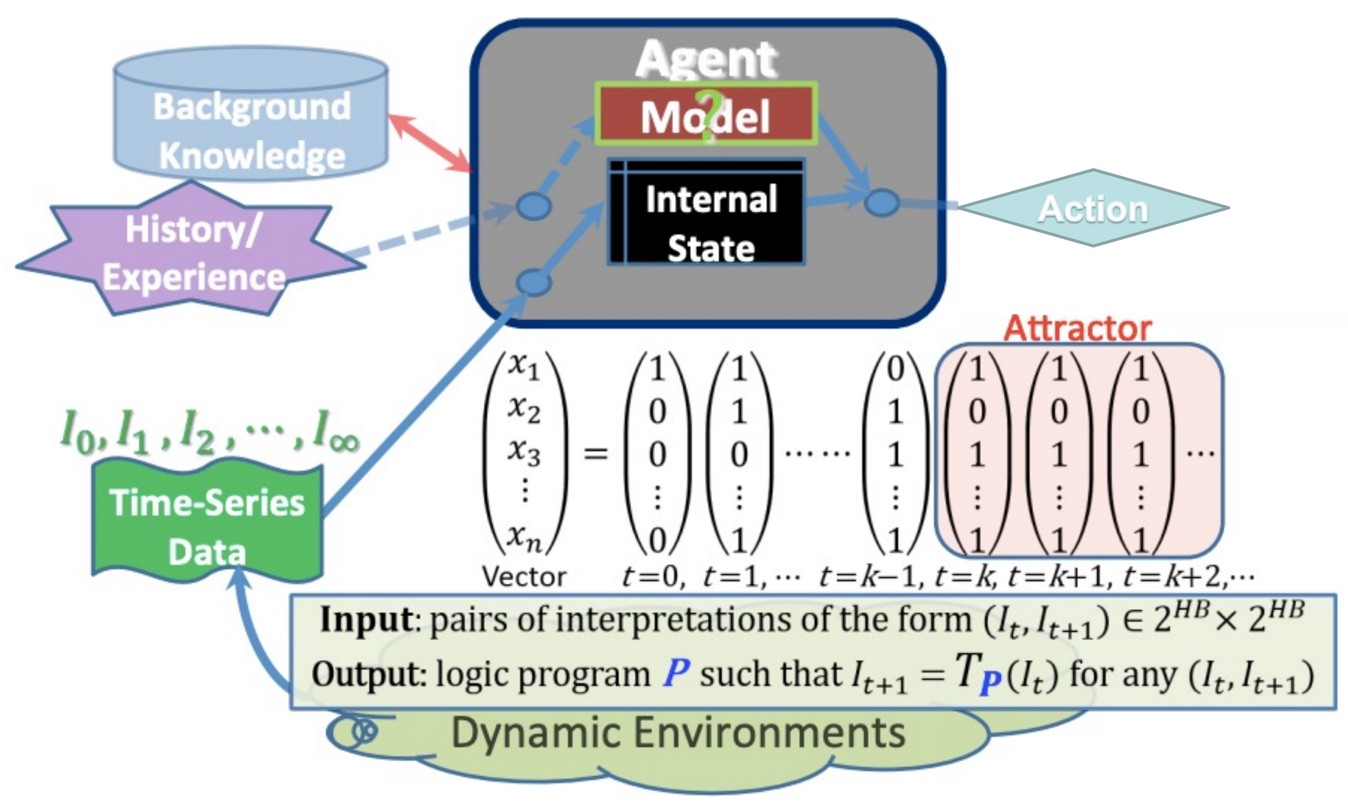
Learning from interpretation transition using differentiable logic programming semantics

Kun Gao[1], Hanpin Wang[1,2], Yongzhi Cao[1], Katsumi Inoue[3]
[1] Peking University [2] Guangzhou University [3] National Institute of Informatics

PEKING UNIVERSITY
北京大學

大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

Summary



The description of learning from interpretation transition (LFIT)

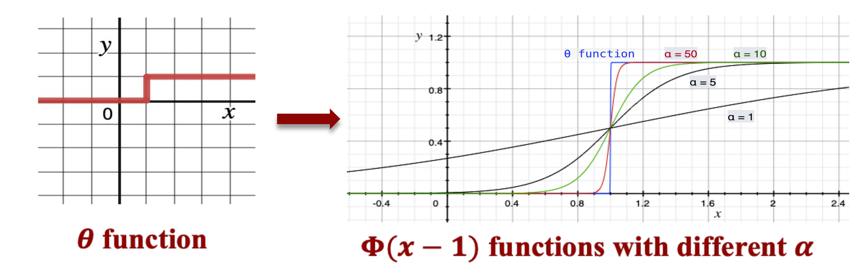
- We propose D-LFIT, an inductive logic programming (ILP) learner based on neural networks.
- D-LFIT fast learns logic programs from **interpretation transitions**.
- D-LFIT fast learn logic programs from incomplete data and mislabeled data.

Method

$$\mathbf{v}_o = \theta(\mathbf{M}_P^{NOR}\mathbf{v}_i),$$

$$\overline{\mathbf{v}}_o = \phi(\overline{\mathbf{M}}_P^{NOR}\mathbf{v}_i - 1), \quad \phi = \frac{1}{1+e^{-\alpha x}}$$

Inference in the meta-info learner



- $\mathbf{v}_o, \mathbf{v}_1, \dots, \mathbf{v}_n$ is the vectors representing the **interpretations**.
- We use Φ as the **activation** function to replace the **threshold** function θ, where α is a hyper parameter.

$$\text{loss} = \lambda_1 \cdot H(\mathbf{v}, \overline{\mathbf{v}}) + \lambda_2 \cdot \sum_{c=1}^{m}\sum_{b=1}^{2n} \overline{\mathbf{M}}_P^{NOR}[c,b].$$

The loss function in the meta-info learner. *H* is the cross-entropy function.

- Interpretation learner. For all k, learn the SHV matrix representing the k-th rule in the logic program.

$$P: \begin{array}{l} p \leftarrow p \wedge q \wedge \neg r \\ p \leftarrow r \wedge \neg p \\ p \leftarrow \end{array}, \quad \mathbf{M}_P^{SHV} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \mathbf{v}_i \xrightarrow{\text{Activation function}} \overline{\mathbf{v}}_o[1] \;\; \overline{\mathbf{v}}_o[2] \;\; \overline{\mathbf{v}}_o[3] \xrightarrow{\text{Fuzzy or operator: product t-norm}} \overline{\mathbf{v}_p}$$

$$\overline{\mathbf{v}}_o[k] = 1 - \prod_{c=1}^{C(L_{p_k}, \lfloor \frac{L_{p_k}}{2} \rfloor)} \left(1 - \phi\left(\sum_{b=1}^{2n} \overline{\mathbf{M}}_{P_k}^{SHV}[c,b] \cdot \mathbf{v}_i[b] - 1\right)\right), \; 1 \le k \le m.$$

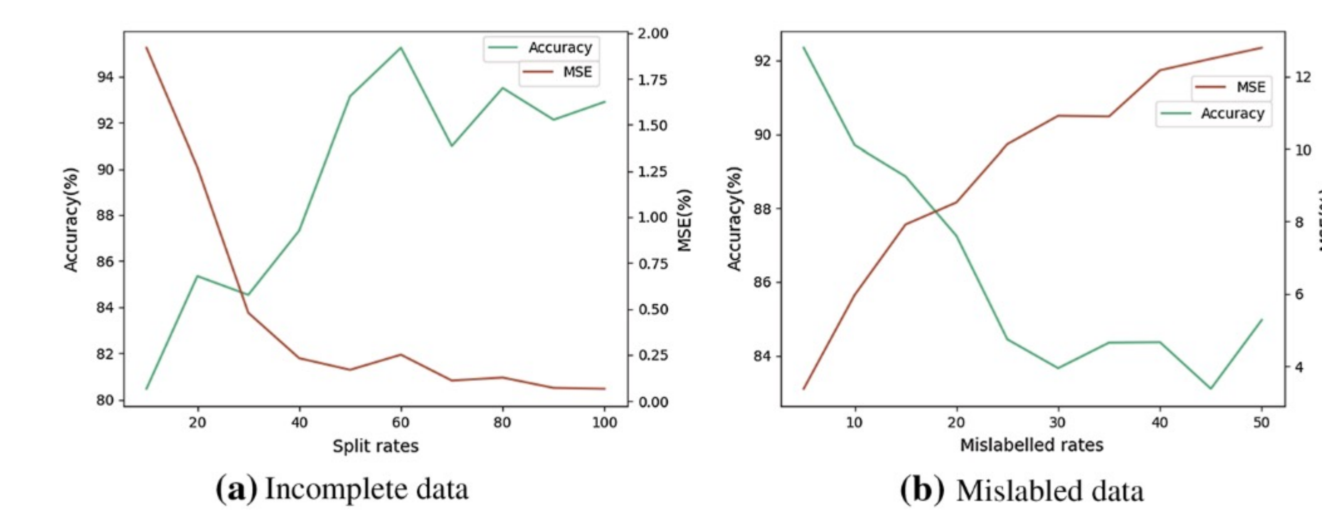
Inference in the interpretation learner

$$\text{loss} = \sum_{k=1}^{m}\left(\lambda_1 \cdot H(\mathbf{v}[k], \overline{\mathbf{v}}[k]) + \lambda_2 \cdot \sum_{c=1}^{C(L_{p_k}, \lfloor \frac{L_{p_k}}{2} \rfloor)} \left(1 - \sum_{b=1}^{2n} \overline{\mathbf{M}}_{P_k}^{SHV}[c,b]\right)\right).$$

Loss function (H is the cross-entropy function)

Results

1. Handling with **mislabeled** data
2. Handling with **incomplete** data



(a) Incomplete data (b) Mislabeled data

3. Handling with **first-order logic** program

- Using propositionalization method to generate first-order feature table. We use bottom clause propositionalization algorithm (**BCP**) to generate first-order features, and regard them as pairs of attribute-value. Then we use D-LFIT to learn first-order logic programs from these features.

Table 2 Comparison of the accuracy (%) of rules obtained by CILP++

Model	Mutagenesis	UW_CSE	Alzheimer-amine
CILP++	77.72	**81.98**	**78.70**
D-LFIT	**83.33**	79.44	67.75

D-LFIT generates first-order rules

Method

1. **Embeddings**: *same head variable (SHV) matrix* and *normal (NOR) matrix*.

- The matrices for propositional logic programs called NOR matrices.

$$P: \begin{array}{l} p \leftarrow (p \wedge q \wedge \neg r) \vee (p \wedge \neg q) \\ q \leftarrow p \wedge r \\ r \leftarrow \end{array} \longrightarrow \mathbf{M}_P^{NOR} = \begin{array}{c} \\ (p) \\ (q) \\ (r) \end{array} \begin{array}{c} \begin{array}{cccccc} p & q & r & \neg p & \neg q & \neg r \end{array} \\ \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The number of non zero **elements** in each line equals the number of different **literals** in the corresponding rule.

NOR matrix

- The number of rules m in a same head variable logic program P and the number of literals n in a disjunctive normal form rule with the same head with P hold: $m \le C(n, \lfloor \frac{n}{2} \rfloor)$.
- The matrices for same head variable logic programs called SHV matrices.

$$P: \begin{array}{l} p \leftarrow p \wedge q \wedge \neg r \\ p \leftarrow r \wedge \neg p \\ p \leftarrow \end{array} \longrightarrow \mathbf{M}_P^{SHV} = \begin{array}{c} \begin{array}{cccccc} p & q & r & \neg p & \neg q & \neg r \end{array} \\ \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ p \\ q \\ r \end{array}$$

According to the **index information** of non-zero **elements** in each line, we can extract the corresponding logic rule.

SHV matrix

2. **Learning steps**: *Meta-info learner* and *interpretation learner*.

- Meta-info learner. The NOR matrices of target logic programs are learned in this process. Extract the number of non zero elements in every line l_{p_k} from the generated normal matrices.

Architecture & Results

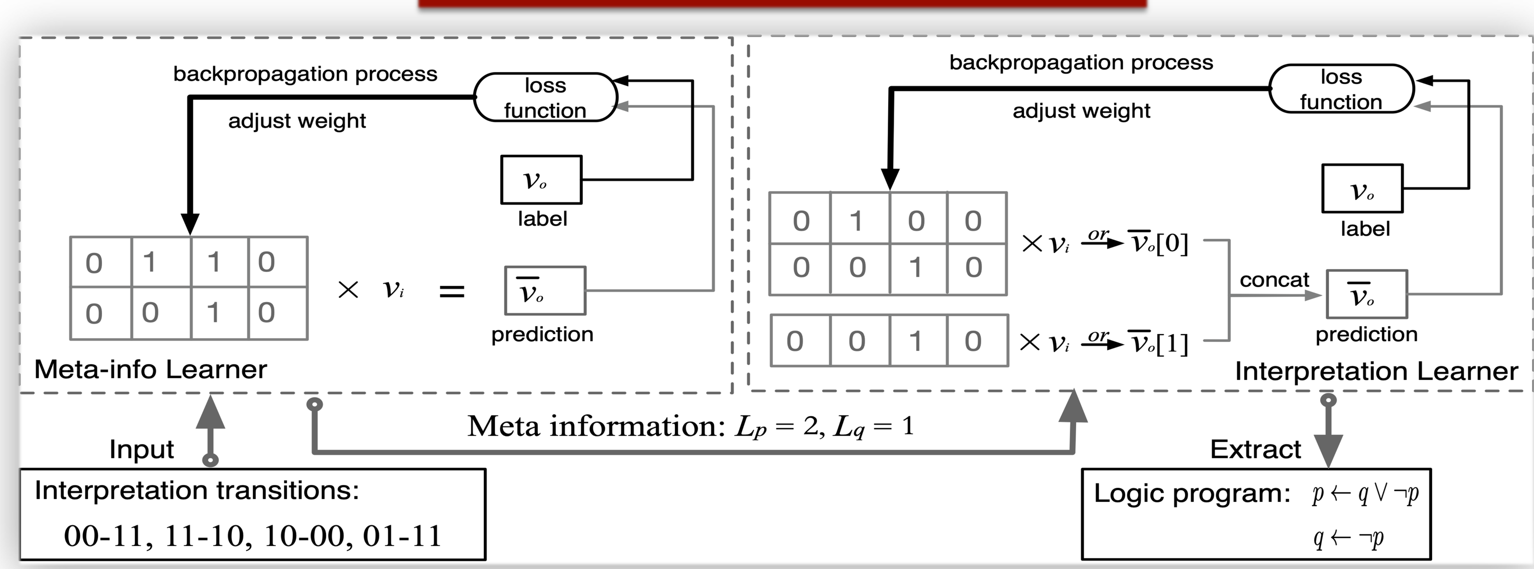


Table 1 Comparison of MSE (%) and accuracy (%) on partial datasets with different split rates

Datasets (Complexity)	Model	The split rates: 10%	20%	50%	100%
Fission (10, 2^{10})	NN-LFIT	**1.2, 98.80**	**0.2, 99.80**	**0.00,99.92**	**0.00**,99.87
	D-LFIT	1.91, 80.45	1.27, 85.33	0.17, 93.13	0.06, 92.89
	LF1T	–, 76.85	–,77.03	–,77.15	– ,**100**
	JRip	–,79.14	–,78.05	–,80.04	–, 78.47
Mammalian (10, 2^{10})	NN-LFIT	**1.3, 96.6**	**0.4 , 94.35**	**0.00,99.89**	**0.00,99.91**
	D-LFIT	1.73, 71.67	1.21 ,75.9	0.79, 80.09	0.52,82.84
	LF1T	–,76.01	–,76.48	–,76.73	91.56
	JRip	– ,77.84	–,75.44	–,76.41	–, 74.66
Budding (12, 2^{12})	NN-LFIT	ROT	ROT	ROT	ROT
	D-LFIT	1.03, **71.96**	0.43,**71.39**	0.15,**70.5**	0.09,**76.52**
	LF1T	ROT	ROT	ROT	ROT
	JRip	–,67.97	–,68.55	–, 67.91	–,68.32
Arabidopsis (15, 2^{15})	NN-LFIT	ROT	ROT	ROT	ROT
	D-LFIT	0.57, **84.35**	0.51, **86.83**	0.48, **88.56**	0.45, **89.70**
	LF1T	ROT	ROT	ROT	ROT
	JRip	–, 68.84	–, 69.00	–, 68.79	–,68.67
Mutagenesis (1111, 188)	D-LFIT	**77.78**	**94.44**	**88.88**	**83.33**
	JRip	58.37	59.46	65.97	66.45
UW-CSE (601, 1614)	D-LFIT	**75.00**	**78.12**	**78.44**	**79.44**
	JRip	70.18	70.81	73.85	74.35
Alzheimers-amine (1084, 686)	D-LFIT	58.42	**60.29**	**63.24**	**67.65**
	JRip	**58.74**	57.66	57.14	54.81

D-LFIT handles incomplete data

Table 3 Comparison of the MSE (%) and accuracy (%) on the mislabeled data with different mislabeling rates

Datasets	Model name	The rates of mislabeled data: 5%	20%	35%	50%
Fission	NN-LFIT	3.25, **96.75**	10.56,**89.43**	15.14,**84.86**	17.74,82.25
	D-LFIT	**2.23**, 92.34	**8.53**, 87.25	**10.89**, 84.34	**12.08, 84.96**
	LF1T	–,77.25	–,77.16	–,77.38	–,77.32
	JRip	– ,78.91	– ,78.54	–,78.55	– ,78.15
Mammalian	NN-LFIT	4.77, 79.72	16,78.11	20.98,79.01	23.20,74.49
	D-LFIT	**3.45, 80.00**	**11.53,78.82**	**15.74, 80.29**	**16.35, 86.27**
	LF1T	–, 76.72	–,76,73	–,76.62	–,77.32
	JRip	–,74.21	–,74.45	– , 74.43	–,74.00
Budding	NN-LFIT	ROT	ROT	ROT	ROT
	D-LFIT	4.9, **76.42**	13.3, **74.28**	16.53,**73.41**	18.21,**74.71**
	LF1T	ROT	ROT	ROT	ROT
	JRip	– ,67.99	–,67.15	–,66.80	–,66.41
Arabidopsis	NN-LFIT	ROT	ROT	ROT	ROT
	D-LFIT	4.8 ,**81.46**	11.83,**76.59**	15.4 ,**70.28**	17.35,64.90
	LF1T	ROT	ROT	ROT	ROT
	JRip	–,68.27	–, 67.34	–,66.94	–,**66.65**
Mutagenesis	D-LFIT	**88.89**	**83.33**	**88.89**	**88.89**
	JRip	66.49	62.23	62.77	62.23
UW-CSE	D-LFIT	**73.49**	**72.67**	**73.29**	**73.29**
	JRip	72.80	67.35	66.04	66.23
Alzheimers-amine	D-LFIT	**63.24**	**63.24**	**60.29**	**58.83**
	JRip	48.35	49.42	49.27	50.87

D-LFIT handles mislabeled data

Discussions

1. D-LFIT is an inductive logic programming learner, which can learn propositional logic programs from **mislabeled** data or **incomplete** data.
2. Through adopting the BCP algorithm, we can learn **first-order logic programs** from relational databases.
3. D-LFIT is a robust, fast leaner, which can **curriculum learning strategy** to learn knowledge from data.
4. We will devise more constrains to make the generated logic programs meet the destinated format.
5. We will apply the D-LFIT on other dataset such like knowledge graph.