ENIGMA Given Clause Guidance

Karel Chvalovský[1] Jan Jakubův[1] Martin Suda[1] Josef Urban[1]

Machine Learning and Reasoning, 22nd March 2019

[1]Czech Technical University in Prague, Czech Republic

Outline

Outline

Recap & General Intro

ATPs & Given Clauses

Enigma Models

Enhancing Enigma

Experiments

Internal Guidance of Reasoning – General Setting

- How do we generally guide reasoning by ML?
- We have a *reasoning task* and a *reasoning engine* (algorithm)
- The engine is capable of drawing (many) correct logical inferences in a *proof state*
- We want to guide the application of the inferences by ML

Saturation-style Theorem Proving

- Previous lecture: strong inference engine called Vampire
- Produces proofs by working in a *refutational setting*:
- Turning $T \vdash C$ into $T, \neg C \vdash \bot$
- Then saturating the resulting set of clauses in a fair way
- Using the *given clause loop* (ANL loop - Argonne, Otter)
- We have Processed (P) and Unprocessed (U) clauses
  1. Pick a good *given clause* from U
  2. Do all its inferences with the clauses in P
  3. Put the resulting clauses into U
  4. GOTO 1

Remarks on ML in Saturation-style Theorem Proving

- Inference guidance done by learning given clause selection
- We learn what are good/bad given clauses for the problem
- Clauses characterized by engineered or learned features (NNs)
- The problem characterization can be *fixed* (or even implicit) for all steps, e.g.:
  - just the (feature/neural) characterization of the conjecture
  - or also an important axiom (assumption/hypothesis)
  - or all axioms (if we believe they are all important)
  - or all weighted by their predicted importance
- Or the problem characterization can be *dynamic*
- E.g. based on various characteristics of the inferred clauses
- Or using partial matching of previous proofs
- Not in this talk, easier to do for tableau provers (coming next)

Outline

The Given Clause Loop Paradigm

Problem representation

- first order clauses (ex. "$x = 0 \lor \neg P(f(x, x))$")
- posed for proof by contradiction

Given an initial set C of clauses and a set of inference rules, find a derivation of the *empty clause* (for example, by the resolution of clauses with conflicting literals L and $\neg L$).

Basic Loop

```
Proc = {}
Unproc = all available clauses
while (no proof found)
{
   select a given clause C from Unproc
   move C from Unproc to Proc
   apply inference rules to C and Proc
   put inferred clauses to Unproc
}
```

Clause Selection Heuristics in E Prover

- E Prover has several pre-defined clause weight functions.
  (and others can be easily implemented)
- Each weight function assigns a real number to a clause.
- Clause with the smallest weight is selected.

E Prover Strategy

- E strategy = E parameters influencing proof search (term ordering, literal selection, clause splitting, ...)
- Weight function gives the priority to a clause.
- Selection by several priority queues in a round-robin way

```
(10  *  ClauseWeight1(10,0.1,...),
  1  *  ClauseWeight2(...),
 20  *  ClauseWeight3(...))
```

Outline

Enigma Basics

- **Idea:** Use fast linear classifier to guide given clause selection!
- **ENIGMA** stands for. . .

Enigma Basics

- **Idea:** Use fast linear classifier to guide given clause selection!
- **ENIGMA** stands for...

Efficient learNing-based Inference Guiding MAchine

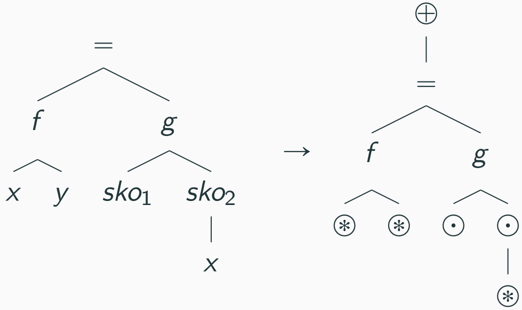
LIBLINEAR: Linear Classifier

- LIBLINEAR: open source library[1]
- **input:** positive and negative examples (float vectors)
- **output:** model ($\sim$ a vector of weights)
- **evaluation** of a generic vector: dot product with the model

[1] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`

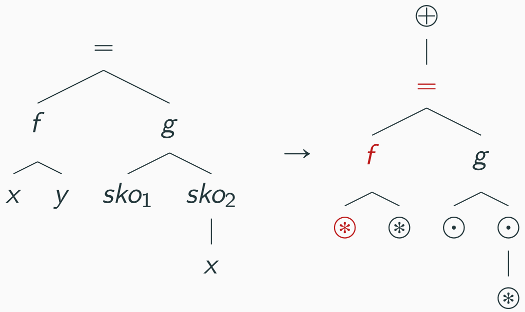
Clauses as Feature Vectors

Consider the literal as a tree and simplify (sign, vars, skolems).



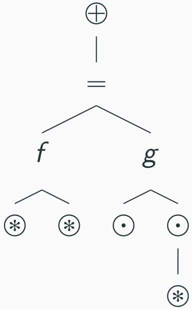
Clauses as Feature Vectors

Features are descending paths of length 3 (triples of symbols).



Clauses as Feature Vectors

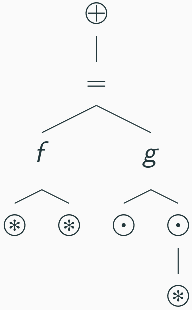
Collect and enumerate all the features. Count the clause features.



#	feature	count
1	$(\oplus,=,\mathrm{a})$	0
⋮	⋮	⋮
11	$(\oplus,=,\mathrm{f})$	1
12	$(\oplus,=,\mathrm{g})$	1
13	$(=,\mathrm{f},\circledast)$	2
14	$(=,\mathrm{g},\odot)$	2
15	$(\mathrm{g},\odot,\circledast)$	1
⋮	⋮	⋮

Clauses as Feature Vectors

Take the counts as a feature vector.



#	feature	count
1	$(\oplus,=,a)$	0
⋮	⋮	⋮
11	$(\oplus,=,f)$	1
12	$(\oplus,=,g)$	1
13	$(=,f,\circledast)$	2
14	$(=,g,\odot)$	2
15	$(g,\odot,\circledast)$	1
⋮	⋮	⋮

Enigma Model Construction

1. Collect training examples from E runs (useful/useless clauses).
2. Enumerate all the features (π :: feature $\rightarrow$ int).
3. Translate clauses to feature vectors.
4. Train a LIBLINEAR classifier (w :: float$^{|\mathrm{dom}(\pi)|}$).
5. Enigma model is $\mathcal{E} = (\pi, w)$.

Given Clause Selection by Enigma

We have Enigma model $\mathcal{E} = (\pi, w)$ and a generated clause C.

1. Translate C to feature vector Φ_C using π.
2. Compute prediction:

$$\text{weight}(C) = \begin{cases} 1 & \text{iff } w \cdot \Phi_C > 0 \\ 10 & \text{otherwise} \end{cases}$$

Enigma Given Clause Selection

- We have implemented `Enigma` weight function in E.
- Enigma model can be used alone to select a given clause:

$$(1 \quad * \quad \texttt{Enigma}(\mathcal{E},\delta))$$

- or in combination with other E weight functions:

$$\begin{aligned}(23 \quad &* \quad \texttt{Enigma}(\mathcal{E},\delta),\\ 3 \quad &* \quad \texttt{StandardWeight(...)},\\ 20 \quad &* \quad \texttt{StephanWeight(...)})\end{aligned}$$

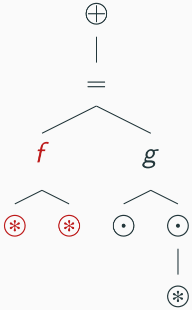
Outline

Conjecture Features

- Enigma classifier $\mathcal{E}$ is independent on the goal conjecture!
- Improvement: Extend Φ_C with goal conjecture features.
- Instead of vector Φ_C take vector (Φ_C, Φ_G).

Horizontal Features

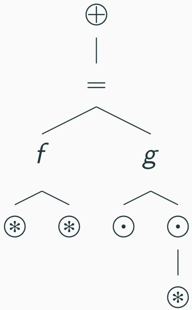
Function applications and arguments top-level symbols.



#	feature	count
1	$(\oplus,=,\mathrm{a})$	0
$\vdots$	$\vdots$	$\vdots$
100	$=(f,g)$	1
101	$f(\circledast,\circledast)$	1
102	$g(\odot,\odot)$	1
103	$\odot(\circledast)$	1
$\vdots$	$\vdots$	$\vdots$

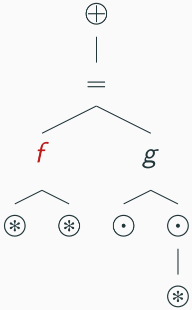
Static Clause Features

For a clause, its length and the number of pos./neg. literals.



#	feature	count/val
103	$\odot(\circledast)$	1
$\vdots$	$\vdots$	$\vdots$
200	len	9
201	pos	1
202	neg	0
$\vdots$	$\vdots$	$\vdots$

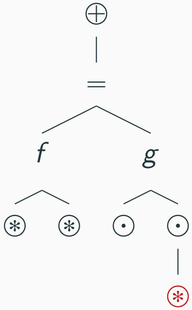
For each symbol, its count and maximum depth.



#	feature	count/val
202	neg	0
$\vdots$	$\vdots$	$\vdots$
300	$\#_{\oplus}(f)$	1
301	$\#_{\ominus}(f)$	0
$\vdots$	$\vdots$	$\vdots$
310	$\%_{\oplus}(\circledast)$	4
311	$\%_{\ominus}(\circledast)$	0
$\vdots$	$\vdots$	$\vdots$

Static Symbol Features

For each symbol, its count and maximum depth.



#	feature	count/val
202	neg	0
$\vdots$	$\vdots$	$\vdots$
300	$\#_{\oplus}(f)$	1
301	$\#_{\ominus}(f)$	0
$\vdots$	$\vdots$	$\vdots$
310	$\%_{\oplus}(\circledast)$	4
311	$\%_{\ominus}(\circledast)$	0
$\vdots$	$\vdots$	$\vdots$

Balancing Training Data

- Training data are uneven.
- Usually we have more negative examples (cca 10 times).
- Previously: Repeat positive examples 10 times.

Accuracy-based Balancing

1. Collect training data.
2. Create classifier $\mathcal{E} = (\pi, w)$.
3. Compute prediction accuracy on the training data (using w).
4. **If** $(acc^+ > acc^-)$ **then** finish.
5. Repeat misclassified positive clauses in the training data.
6. **Goto** 2.

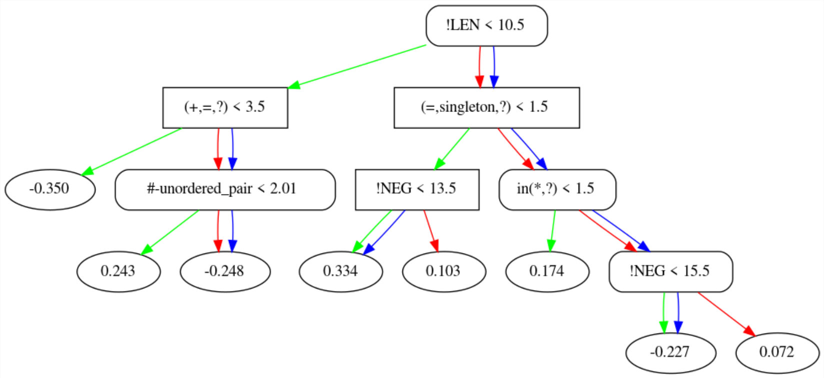
XGBoost Tree Boosting System

- **Idea:** Use decision trees instead of linear classifier.
- Gradient boosting library XGBoost.[2]
- Provides C/C++ API and Python (and others) interface.
- Uses exactly the same training data as LIBLINEAR.
- We use the same Enigma features.
- No need for training data balancing.

[2] `http://xgboost.ai`

XGBoost Models

- An XGBoost model consists of a set of decision trees.
- Leaf scores are summed and translated into a probability.



Outline

Experiments Setup

- MPTP 2078: FOL translation of selected articles from Mizar Mathematical Library (MML)
- Leaf scores are summed and translated into a probability.
- Fix E strategy $\mathcal{S}$.

TPR/TNR: True Positive/Negative Rates

- Training Accuracy:

	$\mathcal{M}_{\mathrm{lin}}$	$\mathcal{M}_{\mathrm{tree}}$	$\mathcal{M}_{\mathrm{nn}}$
TPR	90.54 %	99.36 %	97.82 %
TNR	83.52 %	93.32 %	94.69 %

- Testing Accuracy:

	$\mathcal{M}_{\mathrm{lin}}$	$\mathcal{M}_{\mathrm{tree}}$	$\mathcal{M}_{\mathrm{nn}}$
TPR	80.54 %	83.35 %	82.00 %
TNR	62.28 %	72.60 %	76.88 %

Models ATP Performance

- $\mathcal{S}$ with model $\mathcal{M}$ alone ($\odot$) or combined 50-50 ($\oplus$) in 10s

	$\mathcal{S}$	$\mathcal{S} \odot \mathcal{M}_{\mathrm{lin}}$	$\mathcal{S} \odot \mathcal{M}_{\mathrm{tree}}$	$\mathcal{S} \odot \mathcal{M}_{\mathrm{nn}}$
solved	1086	1115	1231	1167
unique	0	3	10	3
$\mathcal{S}+$	0	+119	+155	+114
$\mathcal{S}-$	0	-90	-10	-33

	$\mathcal{S}$	$\mathcal{S} \oplus \mathcal{M}_{\mathrm{lin}}$	$\mathcal{S} \oplus \mathcal{M}_{\mathrm{tree}}$	$\mathcal{S} \oplus \mathcal{M}_{\mathrm{nn}}$
solved	1086	1210	**1256**	1197
unique	0	7	**15**	2
$\mathcal{S}+$	0	+138	**+173**	+119
$\mathcal{S}-$	0	-14	**-3**	-8

Some References

- K. Chvalovsky, J. Jakubuv, M. Suda, J. Urban: ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E. CoRR abs/1903.03182 (2019)
- Jan Jakubuv, Josef Urban: Enhancing ENIGMA Given Clause Guidance. CICM 2018: 118-124
- J. Jakubuv, J. Urban: ENIGMA: Efficient Learning-Based Inference Guiding Machine. CICM 2017: 292-302
- S. M. Loos, G. Irving, C. Szegedy, C. Kaliszyk: Deep Network Guided Proof Search. LPAR 2017: 85-105
- Christoph Goller: Learning search-control heuristics for automated deduction systems with folding architecture networks. ESANN 1999: 45-50
- Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research, 9:18711874, 2008.
- Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In KDD, pages 785794. ACM, 2016.
- R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. EMNLP-CoNLL 2012: 12011211

Thank you.

Questions?