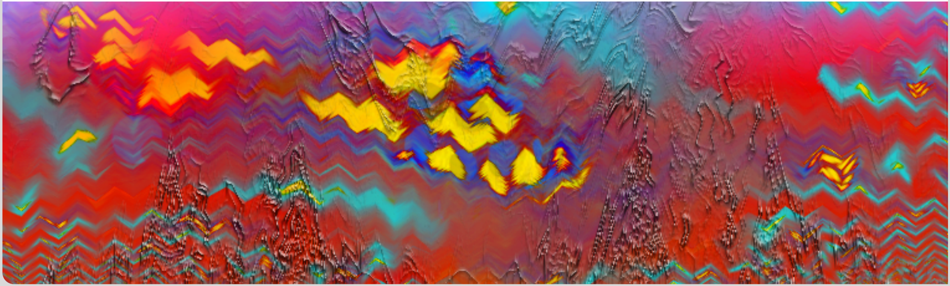
Karlsruhe Institute of Technology

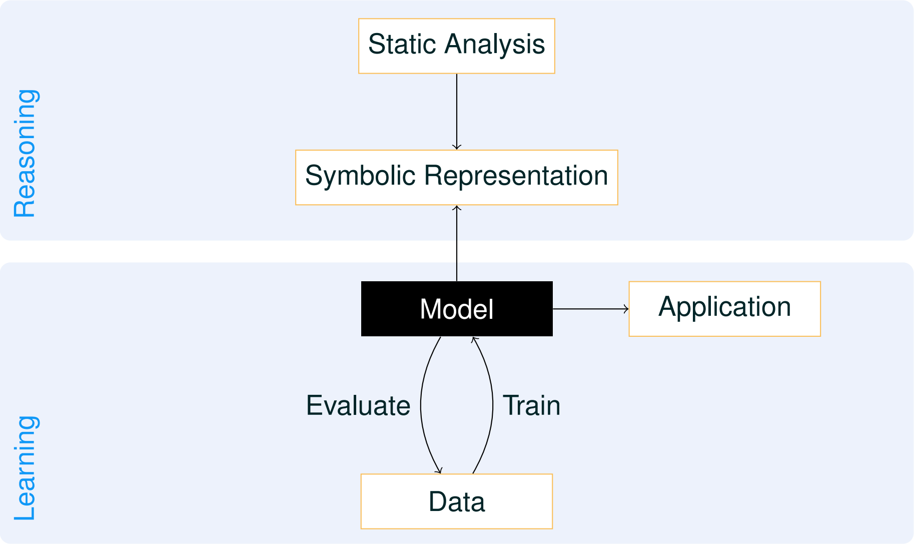
Calculating Sufficient Reasons for Random Forest Classifiers

Markus Iser

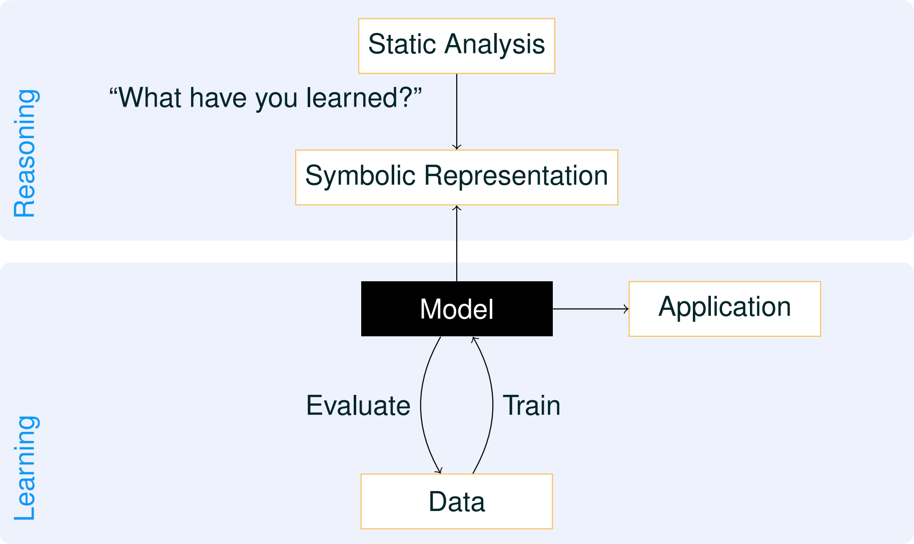
Algorithm Engineering · Institute of Theoretical Informatics



Reasoning about Learned Models



Reasoning about Learned Models



Outline

Prime Implicants of CNF Encoding of Random Forest Classifiers

Contributions

- Monotonic CNF *encoding* for random forest classifiers
- *Implementation* for `sklearn.ensemble.RandomForestClassifier`
- *Incremental method* to generate all prime implicants
- First initial *results*
- Plenty of ideas for future work (paper is WIP)

Decision Tree Classifiers

Samples	Classes	Ground-truth
$T \subset \mathbb{R}^n$	K	$G \subset T \to K$

Classification Problem

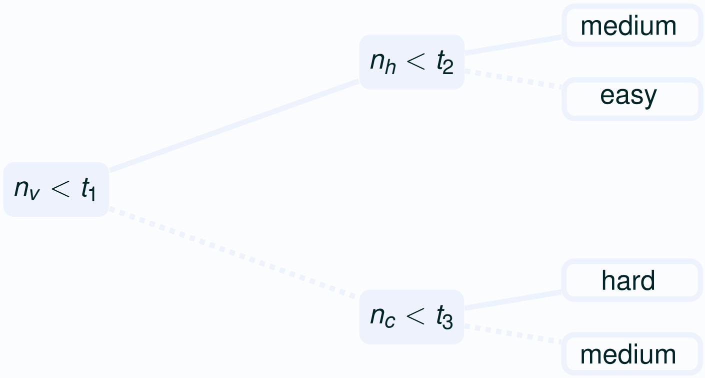
Devise a prediction function $c : \mathbb{R}^n \to K$ maximizing the cardinality of correctly classified samples

Decision Tree Classifier

A decision tree $\mathcal{D} = (V, E, f, t)$ is a *binary tree* (V, E) with features $f : V \to \{1, 2, \ldots, n\}$ and thresholds $t : V \to \mathbb{R}$

CNF Encoding Decision Tree Classifiers

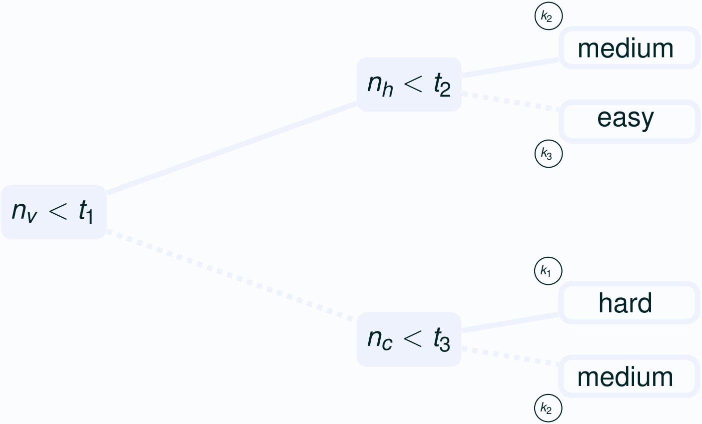
Example `sklearn.tree.DecisionTreeClassifier`



CNF Encoding Decision Tree Classifiers

Karlsruhe Institute of Technology

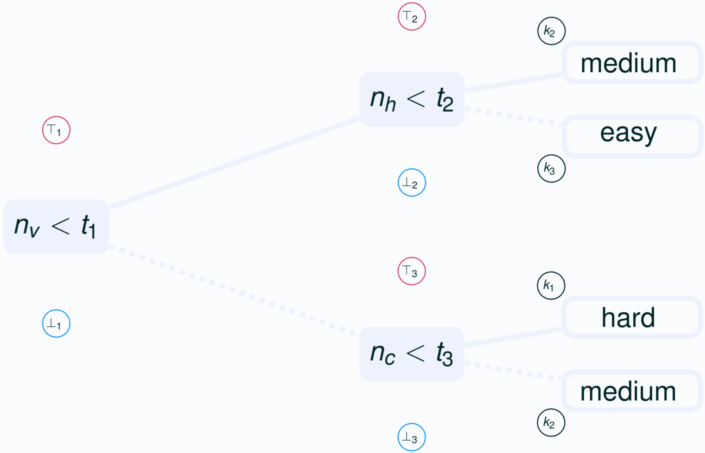
Example `sklearn.tree.DecisionTreeClassifier`



CNF Encoding Decision Tree Classifiers

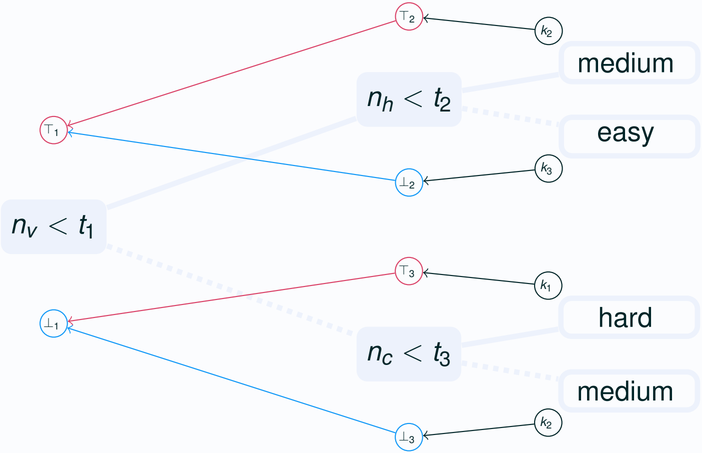
Karlsruhe Institute of Technology

Example `sklearn.tree.DecisionTreeClassifier`



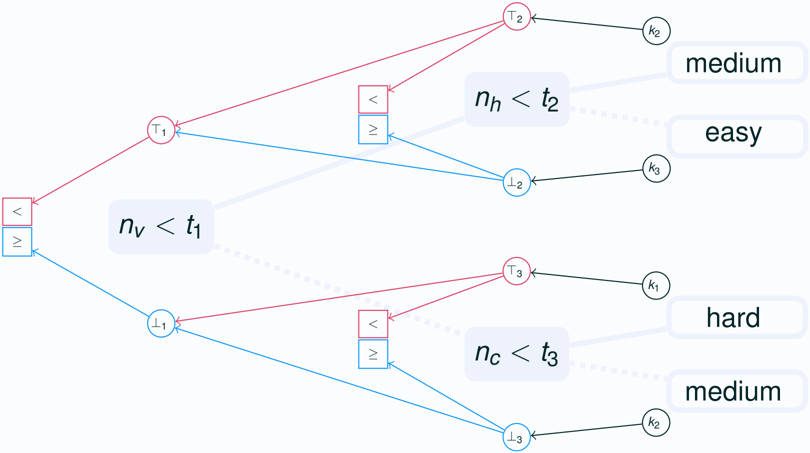
CNF Encoding Decision Tree Classifiers

Example `sklearn.tree.DecisionTreeClassifier`



CNF Encoding Decision Tree Classifiers

Example `sklearn.tree.DecisionTreeClassifier`



Encoding Feature Constraints

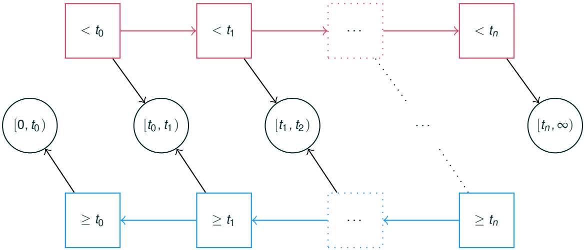
Multiple thresholds per feature

- per feature collect thresholds and sort: $t_0 < t_1 < \cdots < t_n$
- one Boolean variable per interval (induced by thresholds)
- $\mathcal{O}(n)$ encoding variables and clauses, only one clause per node
- without encoding variables: quadratic growth

Encoding Feature Constraints

Multiple thresholds per feature

- per feature collect thresholds and sort: $t_0 < t_1 < \cdots < t_n$
- one Boolean variable per interval (induced by thresholds)
- $\mathcal{O}(n)$ encoding variables and clauses, only one clause per node
- without encoding variables: quadratic growth



Properties of the Encoding

Observations

- 2-SAT
- Horn
- Monotonic Circuit:
  - Classes are roots
  - Feature Intervals are leafs
  - Leafs are purely positive
- Select a class by adding a unit-clause of the class variable, or select multiple classes by adding a disjunction of class variables

Random Forest Classifiers

Samples	Classes	Ground-truth
$T \subset \mathbb{R}^n$	K	$G \subset T \to K$

Classification Problem

Devise a prediction function $c : \mathbb{R}^n \to K$ maximizing the cardinality of correctly classified samples

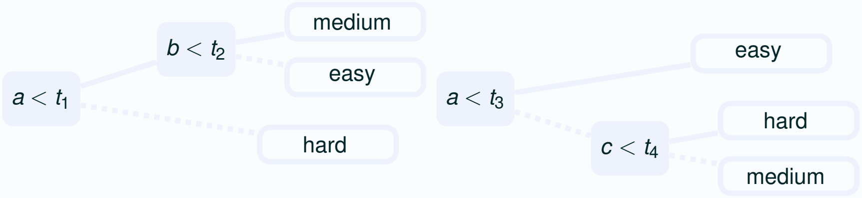
Random Forest Classifier

A *random forest* $\mathcal{R}^d = \{(T_i, \mathcal{D}_i) \mid 1 \leq i \leq d\}$ combines a set of d decision trees. Each decision tree $\mathcal{D}_i$ is independently trained on randomly selected subsets of the training samples $T_i \subset T$

CNF Encoding Random Forest Classifiers

Example `sklearn.ensemble.RandomForestClassifier`

- each leaf ℓ has class probabilities $p_\ell(\text{easy}), p_\ell(\text{medium}), p_\ell(\text{hard})$
- each sample belongs to exactly one leaf in each of the trees
- class is determined by $\underset{k \in \{\text{eas.},\text{med.},\text{h.}\}}{\arg\max} \sum_{\ell \in L} p_\ell(k)$

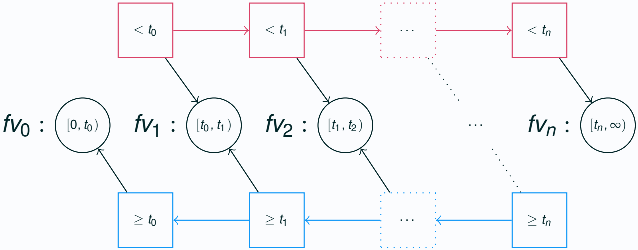


Encoding an Auxiliary SAT Instance

Number of leaf combinations exponential in number of trees, but not all leaf combinations are *possible*.

Generate possible leaf combinations

- Encoding all decision trees as described
- For each feature f add constraint: $\neg fv_0 \vee \neg fv_1 \vee \dots \vee \neg fv_n$
- Generate all solutions, projected to leaf variables



Final Encoding of Random Forest Classifiers

- Encoding all decision trees as described
- Determine class $\kappa(M) \in K$ for each model M of the auxiliary SAT instance (each solution is a *possible* leaf combination)
- $S_k := \{M \mid \kappa(M) = k\}$

Monolithic Approach

For each class $k \in K$, encode $k \rightarrow \bigvee_{M \in S_k} \bigwedge_{m \in M} m$

Incremental Approach

Incrementally build formula equisatisfiable to $k \rightarrow \bigvee_{M \in S_k} \bigwedge_{m \in M} m$

- add models in S_k one by one and solve
- needs additional encoding variables and use of an assumption literal

Algorithm Engineering · Institute of Theoretical Informatics

Determine all Prime Implicants of Classifier

Given a formula F, a model $P \models F$ is a prime implicant of F iff it is subset minimal, i.e., $\nexists P' \subset P, P' \models F$.

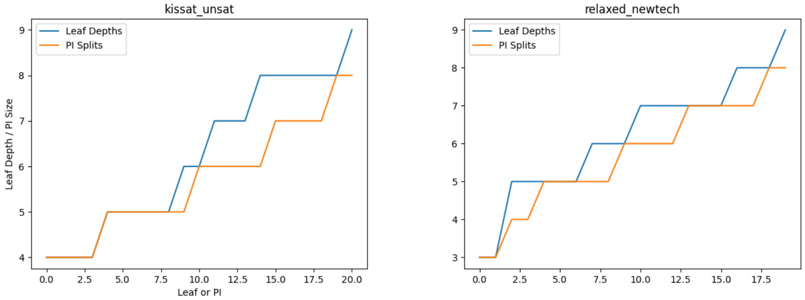
Prime Implicants of our Random Forest Encoding

- Minimal number of excluded value intervals for selected class(es)
- NOT minimal number of case-distinctions for selected class(es)
- Largest connected feature subspaces for selected class(es)

Results: Decision Tree (SC 2020 Data)

Predict fastest solver in {kissat-unsat, relaxed-newtech} from 56 features,
Accuracy: 79%

Numbers of Case Distinctions: Leaf Depths vs. Prime Implicants

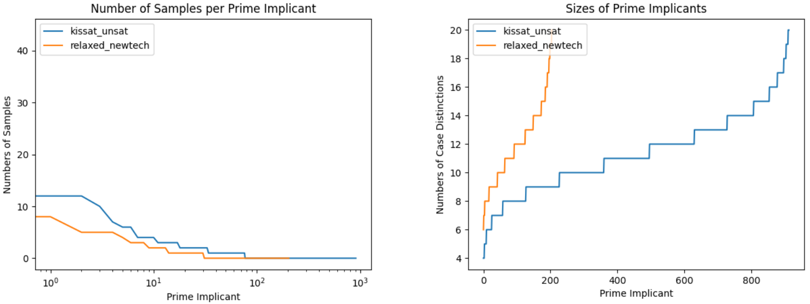


Results: Random Forest, 2 Trees (SC 2020 Data)

Karlsruhe Institute of Technology

Predict fastest solver in {kissat-unsat, relaxed-newtech} from 56 features, Accuracy: 74%

Prime Implicant: Numbers of Samples (left) Sizes (right)

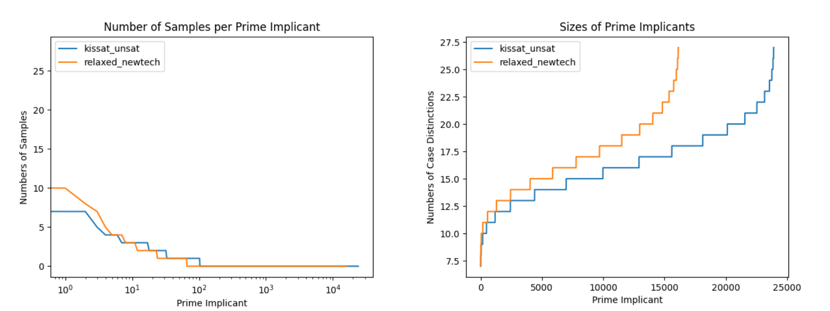


Leaf Comb.	Possible Comb.	PI computation
1554	1118	2 seconds

Results: Random Forest, 3 Trees (SC 2020 Data)

Accuracy: 79%

Numbers of Samples



Leaf Comb.	Possible Comb.	PI computation	code version
77700	40047	9518 seconds	original
		50 seconds	C++
		5 seconds	incremental

Future Work

Parallelize and Approximate

- Parallel and incremental enumeration of possible leaf combinations
- Determine leaf combinations which are backed by samples
- Analyze evolution of prime implicants while adding leaf combinations
- Analyze evolution of accuracy through generalization

Empirical Classes of SAT Instances

- Analyze prediction models for algorithm portfolios
- Feedback for algorithm engineers
- Recurrent ISAC-like approach without unsupervised learning