Formulas of propositional logic as interpretable classifiers

Reijo Jaakkola
reijo.jaakkola@tuni.fi

Tampere University

April 9, 2024

Joint work with Tomi Janhunen, Antti Kuusisto, Masood Feyzbakhsh Rankooh and Miikka Vilander.

Propositional logic recap

- Fix a set $\tau := \{p_1, \ldots, p_d\}$ of propositional variables. The set of formulas of $\mathrm{PL}[\tau]$ is generated via the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi).$$

  Here $p \in \tau$.

Propositional logic recap

- Fix a set $\tau := \{p_1, \ldots, p_d\}$ of propositional variables. The set of formulas of $\mathrm{PL}[\tau]$ is generated via the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi).$$

  Here $p \in \tau$.

- Given an **assignment**

$$s : \tau \to \{0, 1\},$$

  we define recursively $s(\varphi)$ as follows:

  - $s(\neg\varphi) = 1$ iff $s(\varphi) = 0$.
  - $s(\varphi \wedge \psi) = 1$ iff $s(\varphi) = 1$ and $s(\psi) = 1$.
  - $s(\varphi \vee \psi) = 1$ iff $s(\varphi) = 1$ or $s(\psi) = 1$.

Propositional logic recap

- Fix a set $\tau := \{p_1, \ldots, p_d\}$ of propositional variables. The set of formulas of $\mathrm{PL}[\tau]$ is generated via the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi).$$

  Here $p \in \tau$.

- Given an **assignment**

$$s : \tau \to \{0, 1\},$$

  we define recursively $s(\varphi)$ as follows:

  - $s(\neg\varphi) = 1$ iff $s(\varphi) = 0$.
  - $s(\varphi \wedge \psi) = 1$ iff $s(\varphi) = 1$ and $s(\psi) = 1$.
  - $s(\varphi \vee \psi) = 1$ iff $s(\varphi) = 1$ or $s(\psi) = 1$.

- The **size** of a formula φ is defined recursively as follows:

  - $\mathrm{size}(p) = 1$
  - $\mathrm{size}(\neg\varphi) = 1 + \mathrm{size}(\varphi)$
  - $\mathrm{size}(\varphi \wedge \psi) = \mathrm{size}(\varphi \vee \psi) = 1 + \mathrm{size}(\varphi) + \mathrm{size}(\psi)$

Learning formulas of propositional logic

- Using propositional (Boolean) features from a given vocabulary τ we want to predict the value that a given target feature q will receive.

Learning formulas of propositional logic

- Using propositional (Boolean) features from a given vocabulary τ we want to predict the value that a given target feature q will receive.
- As an example, consider the following Boolean data set.

p_1	p_2	p_3	q
1	1	1	1
1	1	0	1
1	0	0	0
0	1	0	0

Based on this data, the formula $(p_1 \wedge p_2)$ seems to predict quite well the value of q.

Learning formulas of propositional logic

- Let
$$T_{\tau,q} := \{s : \tau \cup \{q\} \to \{0,1\}\}.$$
There is an unknown probability distribution $\mu : T_{\tau,q} \to [0,1]$.

Learning formulas of propositional logic

- Let
$$T_{\tau,q} := \{s : \tau \cup \{q\} \to \{0,1\}\}.$$
There is an unknown probability distribution $\mu : T_{\tau,q} \to [0,1]$.
- Think τ as the set of features and the assignments s as datapoints.

Learning formulas of propositional logic

- Let
$$T_{\tau,q} := \{s : \tau \cup \{q\} \to \{0,1\}\}.$$
There is an unknown probability distribution $\mu : T_{\tau,q} \to [0,1]$.
- Think τ as the set of features and the assignments s as datapoints.
- **Goal** is to learn a propositional formula φ over τ with small **error**:
$$\mathrm{err}_\mu(\varphi) := \Pr_{s \in T_{\tau,q}}[s(\varphi) \neq s(q)]$$

Learning formulas of propositional logic

- Let
$$T_{\tau,q} := \{s : \tau \cup \{q\} \to \{0,1\}\}.$$
There is an unknown probability distribution $\mu : T_{\tau,q} \to [0,1]$.
- Think τ as the set of features and the assignments s as datapoints.
- **Goal** is to learn a propositional formula φ over τ with small **error**:
$$\mathrm{err}_\mu(\varphi) := \Pr_{s \in T_{\tau,q}}[s(\varphi) \neq s(q)]$$

- Since μ is unknown, this is done using samples $S \sim \mu^n$ and by minimizing **empirical error**:
$$\mathrm{err}_S(\varphi) := \frac{1}{n} \sum_{\substack{s \in S \\ s(\varphi) \neq s(q)}} 1$$

Example: learning conjunctions [Valiant, 1984]

- An algorithm for learning a conjunction over $\tau = \{p_1, \ldots, p_d\}$:

Example: learning conjunctions [Valiant, 1984]

- An algorithm for learning a conjunction over $\tau = \{p_1, \ldots, p_d\}$:
  1. Start with the conjunction
  $$\varphi := p_1 \wedge \neg p_1 \wedge \cdots \wedge p_d \wedge \neg p_d$$

Example: learning conjunctions [Valiant, 1984]

- An algorithm for learning a conjunction over $\tau = \{p_1, \ldots, p_d\}$:
  1. Start with the conjunction
  $$\varphi := p_1 \wedge \neg p_1 \wedge \cdots \wedge p_d \wedge \neg p_d$$
  2. When you encounter an assignment s such that $s(\varphi) = 0$ and $s(q) = 1$, then remove each literal ℓ from φ for which $s(\ell) = 0$.

Example: learning conjunctions [Valiant, 1984]

- An algorithm for learning a conjunction over $\tau = \{p_1, \ldots, p_d\}$:
  1. Start with the conjunction
  $$\varphi := p_1 \wedge \neg p_1 \wedge \cdots \wedge p_d \wedge \neg p_d$$
  2. When you encounter an assignment s such that $s(\varphi) = 0$ and $s(q) = 1$, then remove each literal ℓ from φ for which $s(\ell) = 0$.

  Works well **if** q is assumed to be equivalent to a conjunction of symbols from τ.

Example: learning conjunctions [Valiant, 1984]

- An algorithm for learning a conjunction over $\tau = \{p_1, \ldots, p_d\}$:
  1. Start with the conjunction
  $$\varphi := p_1 \wedge \neg p_1 \wedge \cdots \wedge p_d \wedge \neg p_d$$
  2. When you encounter an assignment s such that $s(\varphi) = 0$ and $s(q) = 1$, then remove each literal ℓ from φ for which $s(\ell) = 0$.

  Works well **if** q is assumed to be equivalent to a conjunction of symbols from τ.
- Can be also made to work if q is equivalent to a conjunction plus "noise".

Learning general propositional formulas is in general hard

- Learning an arbitrary propositional formula over τ requires in general exponentially many samples, because the VC-dimension of $\mathrm{PL}[\tau]$ is $2^{|\tau|}$.

Learning general propositional formulas is in general hard

- Learning an arbitrary propositional formula over τ requires in general exponentially many samples, because the VC-dimension of $\mathrm{PL}[\tau]$ is $2^{|\tau|}$.
- Restricting attention to formulas of $\mathrm{PL}[\tau]$ which are of polynomial size w.r.t. $|\tau|$ seems to make the problem feasible. Indeed, the VC-dimension of this class is only polynomial w.r.t. $|\tau|$. Thus, in principle, one needs only polynomially many samples.

Learning general propositional formulas is in general hard

- Learning an arbitrary propositional formula over τ requires in general exponentially many samples, because the VC-dimension of $\mathrm{PL}[\tau]$ is $2^{|\tau|}$.
- Restricting attention to formulas of $\mathrm{PL}[\tau]$ which are of polynomial size w.r.t. $|\tau|$ seems to make the problem feasible. Indeed, the VC-dimension of this class is only polynomial w.r.t. $|\tau|$. Thus, in principle, one needs only polynomially many samples.
- However, [Kearns and Valiant, 1994] established that — under a standard cryptographic assumption — polynomial size propositional formulas can not be learned in polynomial time.

What about real-world?

What about real-world?

- The previous hardness results are asymptotic in nature.

What about real-world?

- The previous hardness results are asymptotic in nature.
- In [Jaakkola et al., 2023] we implemented an algorithm in ASP which we used to learn formulas from **tabular** data sets.

What about real-world?

- The previous hardness results are asymptotic in nature.
- In [Jaakkola et al., 2023] we implemented an algorithm in ASP which we used to learn formulas from **tabular** data sets.
- The input of the algorithm is a parameter k and a sample S. Output is a propositional formula φ of size at most k which has a minimal empirical error among all formulas of size at most k.

What about real-world?

- The previous hardness results are asymptotic in nature.
- In [Jaakkola et al., 2023] we implemented an algorithm in ASP which we used to learn formulas from **tabular** data sets.
- The input of the algorithm is a parameter k and a sample S. Output is a propositional formula φ of size at most k which has a minimal empirical error among all formulas of size at most k.
- Hope was that it would be sufficient to consider small values of k.

What about real-world?

- The previous hardness results are asymptotic in nature.
- In [Jaakkola et al., 2023] we implemented an algorithm in ASP which we used to learn formulas from **tabular** data sets.
- The input of the algorithm is a parameter k and a sample S. Output is a propositional formula φ of size at most k which has a minimal empirical error among all formulas of size at most k.
- Hope was that it would be sufficient to consider small values of k.
- Most real-world data sets are not Boolean, so one needs to first booleanize them.

What about real-world?

- The previous hardness results are asymptotic in nature.
- In [Jaakkola et al., 2023] we implemented an algorithm in ASP which we used to learn formulas from **tabular** data sets.
- The input of the algorithm is a parameter k and a sample S. Output is a propositional formula φ of size at most k which has a minimal empirical error among all formulas of size at most k.
- Hope was that it would be sufficient to consider small values of k.
- Most real-world data sets are not Boolean, so one needs to first booleanize them.
  - Categorical features can be one-hot encoded.

What about real-world?

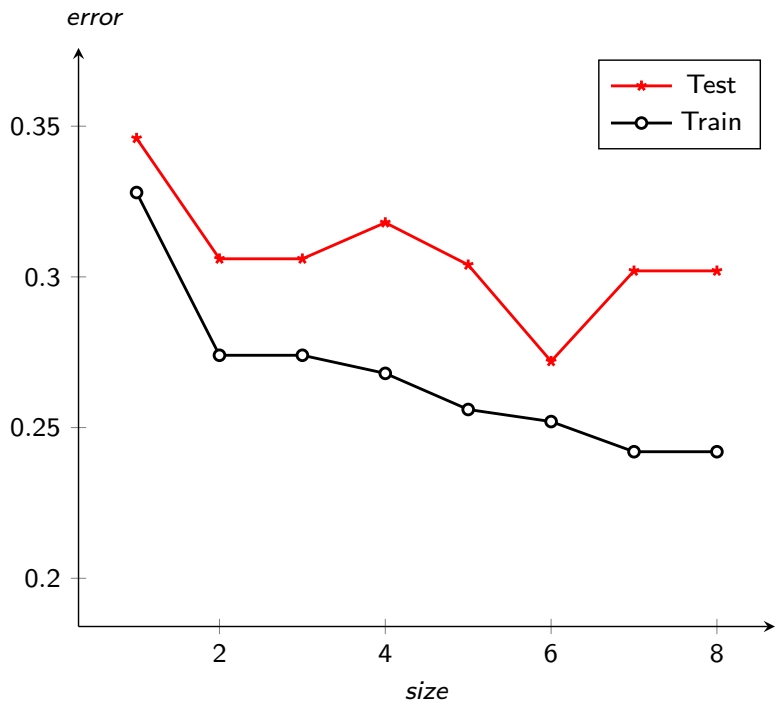
- The previous hardness results are asymptotic in nature.
- In [Jaakkola et al., 2023] we implemented an algorithm in ASP which we used to learn formulas from **tabular** data sets.
- The input of the algorithm is a parameter k and a sample S. Output is a propositional formula φ of size at most k which has a minimal empirical error among all formulas of size at most k.
- Hope was that it would be sufficient to consider small values of k.
- Most real-world data sets are not Boolean, so one needs to first booleanize them.
  - Categorical features can be one-hot encoded.
  - For continuous features we simply split them at the median.

Empirical results from [Jaakkola et al., 2023]

- The first data set was the **Statlog-German credit** data set, classifies persons based on whether or not it is "risky" to give them a loan.
- 1000 data points and 68 attributes. The data was split 50-50 to training and testing sets.
- The following formula of size six

  $$\neg\,(\text{negative\_balance} \wedge \text{above\_median\_loan\_duration}) \vee \text{employment\_on\_managerial\_level}$$

  had 73% accuracy on the test set.
- In the literature e.g. neural networks had obtained 76% test accuracy.



Empirical results from [Jaakkola et al., 2023]

- The second data set was **Breast cancer Wisconsin** data set, classifies tumors based on whether or not they are benign.
- 683 data points ja 9 attributes. The data was split 50-50 to training and testing sets.
- The following formula of size eight

$$\neg(((p \wedge q) \vee r) \wedge s)$$

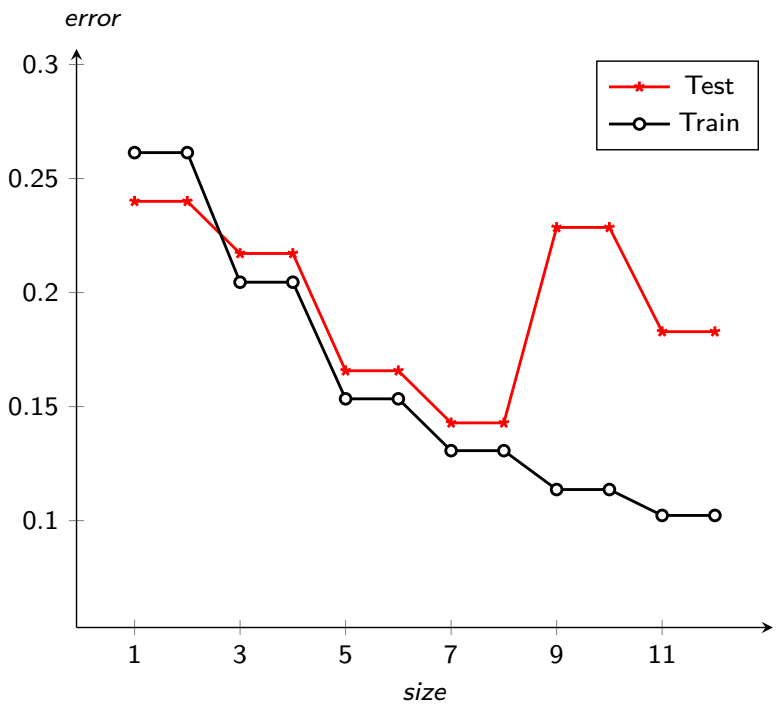
  had 95.3% accuracy on the test set.
- In the literature e.g. naive Bayes classifiers had obtained 97.4% test accuracy.

Empirical results from [Jaakkola et al., 2023]

- The final data set was the **Ionosphere** data set, classifies radar signals based on whether or not they are "good".
- 351 data points and 34 attributes. The data was split 50-50 to training and testing sets.
- The following formula of size seven

$$((p \wedge q) \vee r) \wedge s$$

  had 86% accuracy on the test set.
- In the literature e.g. neural networks had obtained 96% test accuracy.



What we learned from these experiments?

What we learned from these experiments?

- Crude discretization already works quite well.

What we learned from these experiments?

- Crude discretization already works quite well.
- Already short and interpretable formulas obtain good accuracies.

Importance of feature selection

- Short formulas use only a few number of features.

Importance of feature selection

- Short formulas use only a few number of features.
- Instead of regularizing via size, maybe one could just limit the number of features the formulas are allowed to use?

Importance of feature selection

- Short formulas use only a few number of features.
- Instead of regularizing via size, maybe one could just limit the number of features the formulas are allowed to use?
- In fact, already [Holte, 1993] observed that for several real-world datasets one can get pretty good accuracies by using a single feature (comparable to ones obtained using decision trees).

Importance of feature selection

- Short formulas use only a few number of features.
- Instead of regularizing via size, maybe one could just limit the number of features the formulas are allowed to use?
- In fact, already [Holte, 1993] observed that for several real-world datasets one can get pretty good accuracies by using a single feature (comparable to ones obtained using decision trees).
- In [Jaakkola et al., 2024] we investigated an approach based on feature selection in more detail.

The approach of [Jaakkola et al., 2024]

- Given a sample S of assignments over $\tau \cup \{q\}$, we first select a **small** set $\sigma \subseteq \tau$ of "promising" features. We then compute a DNF-formula over σ which has a minimal empirical error w.r.t. S.

The approach of [Jaakkola et al., 2024]

- Given a sample S of assignments over $\tau \cup \{q\}$, we first select a **small** set $\sigma \subseteq \tau$ of "promising" features. We then compute a DNF-formula over σ which has a minimal empirical error w.r.t. S.
- DNF-formulas are formulas of the form

$$t_1 \vee \cdots \vee t_m,$$

where $t_1, \ldots, t_m$ are conjunctions of literals. Note that each assignment corresponds to a conjunction of literals.

The approach of [Jaakkola et al., 2024]

- Given a sample S of assignments over $\tau \cup \{q\}$, we first select a **small** set $\sigma \subseteq \tau$ of "promising" features. We then compute a DNF-formula over σ which has a minimal empirical error w.r.t. S.
- DNF-formulas are formulas of the form

$$t_1 \vee \cdots \vee t_m,$$

  where $t_1, \ldots, t_m$ are conjunctions of literals. Note that each assignment corresponds to a conjunction of literals.
- Given $t : \sigma \to \{0, 1\}$, we let

$$\Pr_S[q|t] := \frac{\text{number of assignments } s \text{ in } S \text{ for which } s \restriction \sigma = t \text{ and } s(q) = 1}{\text{number of assignments } s \text{ in } S \text{ for with } s \restriction \sigma = t}$$

The approach of [Jaakkola et al., 2024]

- Given a sample S of assignments over $\tau \cup \{q\}$, we first select a **small** set $\sigma \subseteq \tau$ of "promising" features. We then compute a DNF-formula over σ which has a minimal empirical error w.r.t. S.
- DNF-formulas are formulas of the form

$$t_1 \vee \cdots \vee t_m,$$

where $t_1, \ldots, t_m$ are conjunctions of literals. Note that each assignment corresponds to a conjunction of literals.

- Given $t : \sigma \to \{0, 1\}$, we let

$$\Pr_S[q|t] := \frac{\text{number of assignments } s \text{ in } S \text{ for which } s \restriction \sigma = t \text{ and } s(q) = 1}{\text{number of assignments } s \text{ in } S \text{ for with } s \restriction \sigma = t}$$

The following DNF-formula minimizes empirical error w.r.t. S

$$\bigvee \left\{ t \mid \Pr_S[q|t] > 1/2 \right\}.$$

Calculating $\Pr_S[q|t]$ from S is trivial.

Related approach

- [Angelino et al., 2018] investigate learning sparse rule lists which are optimized with respect to their error and size.

Related approach

- [Angelino et al., 2018] investigate learning sparse rule lists which are optimized with respect to their error and size.
- Example of a rule list from [Angelino et al., 2018]:

  if (priors$>$ 3) **then predict** `yes`

  else if (sex is male) **and** (juvenile crimes$>$ 0) **then predict** `yes`

  else predict `no`

Related approach

- [Angelino et al., 2018] investigate learning sparse rule lists which are optimized with respect to their error and size.
- Example of a rule list from [Angelino et al., 2018]:

  if (priors>3) **then predict** `yes`

  else if (sex is male) **and** (juvenile crimes>0) **then predict** `yes`

  else predict `no`

- Certain type of rule lists are essentially DNF-formulas. E.g., the above rule list corresponds to the following DNF-formula

$$(\text{priors} > 3) \vee (\text{sex is male} \wedge \text{juvenile crimes} > 0)$$

Related approach

- [Angelino et al., 2018] investigate learning sparse rule lists which are optimized with respect to their error and size.
- Example of a rule list from [Angelino et al., 2018]:

  if (priors$>$ 3) **then predict** `yes`

  else if (sex is male) **and** (juvenile crimes$>$ 0) **then predict** `yes`

  else predict `no`

- Certain type of rule lists are essentially DNF-formulas. E.g., the above rule list corresponds to the following DNF-formula

$$(\text{priors} > 3) \vee (\text{sex is male} \wedge \text{juvenile crimes} > 0)$$

- Since rule lists are optimized w.r.t. their size, the approach is also similar in spirit to that of [Jaakkola et al., 2023].

Some experiments using the approach of [Jaakkola et al., 2024]

Data set	Our method	Random forests	XGBoost
BankMarketing	89.5%	89.7%	89.6%
BreastCancer	95.3%	97.1%	96.4%
CongressionalVoting	95.6%	96.3%	96.3%
GermanCredit	71.5%	76.2%	72%
HeartDisease	82.8%	84.2%	81.2%
Hepatitis	85.0%	83.6%	79.7%
StudentDropout	81.0%	87.3%	87.3%

- Results from 10-fold cross validation for 7 different data sets. Table contains the average test accuracies.

Some experiments using the approach of [Jaakkola et al., 2024]

Data set	Our method	Random forests	XGBoost
BankMarketing	89.5%	89.7%	89.6%
BreastCancer	95.3%	97.1%	96.4%
CongressionalVoting	95.6%	96.3%	96.3%
GermanCredit	71.5%	76.2%	72%
HeartDisease	82.8%	84.2%	81.2%
Hepatitis	85.0%	83.6%	79.7%
StudentDropout	81.0%	87.3%	87.3%

- Results from 10-fold cross validation for 7 different data sets. Table contains the average test accuracies.
- For our method we treated the number of used features as a hyperparameter. 10% of the training set was used as a validation set.

Some experiments using the approach of [Jaakkola et al., 2024]

Data set	Our method	Random forests	XGBoost
BankMarketing	89.5%	89.7%	89.6%
BreastCancer	95.3%	97.1%	96.4%
CongressionalVoting	95.6%	96.3%	96.3%
GermanCredit	71.5%	76.2%	72%
HeartDisease	82.8%	84.2%	81.2%
Hepatitis	85.0%	83.6%	79.7%
StudentDropout	81.0%	87.3%	87.3%

- Results from 10-fold cross validation for 7 different data sets. Table contains the average test accuracies.
- For our method we treated the number of used features as a hyperparameter. 10% of the training set was used as a validation set.
- For feature selection we tested F-test, χ^2-test and mutual information.

Some experiments using the approach of [Jaakkola et al., 2024]

- Results from Leave-One-Out Cross-Validation (LOOCV) for two high-dimensional data sets. Features related to gene expression profiles.

Some experiments using the approach of [Jaakkola et al., 2024]

- Results from Leave-One-Out Cross-Validation (LOOCV) for two high-dimensional data sets. Features related to gene expression profiles.
- Colon, 63 data points and 2000 features. All features were three-valued.
  1. [Jaakkola et al., 2024] with F-test and χ^2-test: 82.3% accuracy.
  2. Random forest with 1000 decision trees: 82.3% accuracy.
  3. Logistic regression with ℓ_1-regularization: 77.4% accuracy.
  4. Support vector machine with linear kernel: 85.5% accuracy.

Some experiments using the approach of [Jaakkola et al., 2024]

- Results from Leave-One-Out Cross-Validation (LOOCV) for two high-dimensional data sets. Features related to gene expression profiles.
- Colon, 63 data points and 2000 features. All features were three-valued.
  1. [Jaakkola et al., 2024] with F-test and χ^2-test: 82.3% accuracy.
  2. Random forest with 1000 decision trees: 82.3% accuracy.
  3. Logistic regression with ℓ_1-regularization: 77.4% accuracy.
  4. Support vector machine with linear kernel: 85.5% accuracy.
- Leukemia, 73 data points and 7070 features. All features were three-valued.
  1. [Jaakkola et al., 2024] with F-test and χ^2-test: 97.2% and 81.9% accuracies respectively.
  2. Random forest with 1000 decision trees: 98.6% accuracy.
  3. Logistic regression with ℓ_1-regularization: 95.8% accuracy.
  4. Support vector machine with linear kernel: 98.6% accuracy.

Limits of our method

- Limiting the number of used features gives you poor accuracy on tabular data sets where you actually need to use a large number of features.

Limits of our method

- Limiting the number of used features gives you poor accuracy on tabular data sets where you actually need to use a large number of features.
- **Example:** Covertype data set, which has 54 features and 423680 data points. Our method obtained on a 80-20 split an accuracy of 76% while e.g. random forests obtained an accuracy of 96%.

Limits of our method

- Limiting the number of used features gives you poor accuracy on tabular data sets where you actually need to use a large number of features.
- **Example:** Covertype data set, which has 54 features and 423680 data points. Our method obtained on a 80-20 split an accuracy of 76% while e.g. random forests obtained an accuracy of 96%.
  - Only 10 features turn out to be useful, but they are real-valued and seem to require very fine-grained discretization.

Limits of our method

- Limiting the number of used features gives you poor accuracy on tabular data sets where you actually need to use a large number of features.
- **Example:** Covertype data set, which has 54 features and 423680 data points. Our method obtained on a 80-20 split an accuracy of 76% while e.g. random forests obtained an accuracy of 96%.
  - Only 10 features turn out to be useful, but they are real-valued and seem to require very fine-grained discretization.
  - E.g. with a 80-20 split a decision tree of depth 10 obtained a test accuracy of 81.4% while a decision tree of depth 20 obtained a test accuracy of 91.9%.

Limits of our method

- Limiting the number of used features gives you poor accuracy on tabular data sets where you actually need to use a large number of features.
- **Example:** Covertype data set, which has 54 features and 423680 data points. Our method obtained on a 80-20 split an accuracy of 76% while e.g. random forests obtained an accuracy of 96%.
  - Only 10 features turn out to be useful, but they are real-valued and seem to require very fine-grained discretization.
  - E.g. with a 80-20 split a decision tree of depth 10 obtained a test accuracy of 81.4% while a decision tree of depth 20 obtained a test accuracy of 91.9%.
- Even if our method does not produce a good **classifier**, it is still able to identify interesting properties from the data set.

Bonus: estimating best possible accuracy

- Given a probability distribution, we might be interested in estimating

$$\mathrm{err}(\mu) = \mathsf{min}\{\mathrm{err}_\mu(\varphi) \mid \varphi \in \mathrm{PL}[\tau]\}.$$

Bonus: estimating best possible accuracy

- Given a probability distribution, we might be interested in estimating

$$\mathrm{err}(\mu) = \mathsf{min}\{\mathrm{err}_\mu(\varphi) \mid \varphi \in \mathrm{PL}[\tau]\}.$$

- Natural option is to take a sample S and look at

$$\mathrm{err}(S) := \mathsf{min}\{\mathrm{err}_S(\varphi) \mid \varphi \in \mathrm{PL}[\tau]\}.$$

Bonus: estimating best possible accuracy

- Given a probability distribution, we might be interested in estimating

$$\mathrm{err}(\mu) = \mathsf{min}\{\mathrm{err}_\mu(\varphi) \mid \varphi \in \mathrm{PL}[\tau]\}.$$

- Natural option is to take a sample S and look at

$$\mathrm{err}(S) := \mathsf{min}\{\mathrm{err}_S(\varphi) \mid \varphi \in \mathrm{PL}[\tau]\}.$$

- We have $\mathbb{E}(\mathrm{err}(S)) \leq \mathrm{err}(\mu)$. Unfortunately $\mathrm{err}(S)$ is not an unbiased estimator of $\mathrm{err}(\mu)$.

Bonus: estimating best possible accuracy

Theorem ([Jaakkola et al., 2023])

Suppose that $S \sim \mu^n$. *Then*

$$\mathbb{E}[\mathrm{err}(S)] \geq \mathrm{err}(\mu) - \frac{1}{\sqrt{n}} \sum_{t:\tau \to \{0,1\}} \sqrt{\mu(q|t)(1-\mu(q|t))\mu(t)}$$

Bonus: estimating best possible accuracy

Theorem ([Jaakkola et al., 2023])

Suppose that $S \sim \mu^n$. Then

$$\mathbb{E}[\mathrm{err}(S)] \geq \mathrm{err}(\mu) - \frac{1}{\sqrt{n}} \sum_{t:\tau \to \{0,1\}} \sqrt{\mu(q|t)(1-\mu(q|t))\mu(t)}$$

- We have the bound

$$\frac{1}{\sqrt{n}} \sum_{t:\tau \to \{0,1\}} \sqrt{\mu(q|t)(1-\mu(q|t))\mu(t)} \leq \frac{1}{2}\sqrt{\frac{2^{|\tau|}}{n}}.$$

  Very pessimistic: matching lower bound is realized by setting μ to be the uniform distribution over $\tau \cup \{q\}$.

Thanks!

References

Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M., and Rudin, C. (2018).

Learning certifiably optimal rule lists for categorical data.

Journal of Machine Learning Research, 18(234):1–78.

Holte, R. C. (1993).

Very simple classification rules perform well on most commonly used datasets.

Machine Learning, 11:63–91.

Jaakkola, R., Janhunen, T., Kuusisto, A., Rankooh, M. F., and Vilander, M. (2023).

Short boolean formulas as explanations in practice.

In Gaggl, S., Martinez, M. V., and Ortiz, M., editors, *Logics in Artificial Intelligence*, pages 90–105.

Jaakkola, R., Janhunen, T., Kuusisto, A., Rankooh, M. F., and Vilander, M. (2024).

Interpretable classifiers for tabular data via discretization and feature selection.

Kearns, M. and Valiant, L. (1994).

Cryptographic limitations on learning boolean formulae and finite automata.

J. ACM, 41(1):67–95.

Valiant, L. (1984).

A theory of the learnable.

Commun. ACM, 27(11):1134–1142.