Probabilistic Circuits: Tractable Representations for Learning and Reasoning

Lecture 5 – Probabilistic Circuits Everywhere

Robert Peharz,[1] **Antonio Vergari**[2]
[1]Graz University of Technology
[2]University of Edinburgh

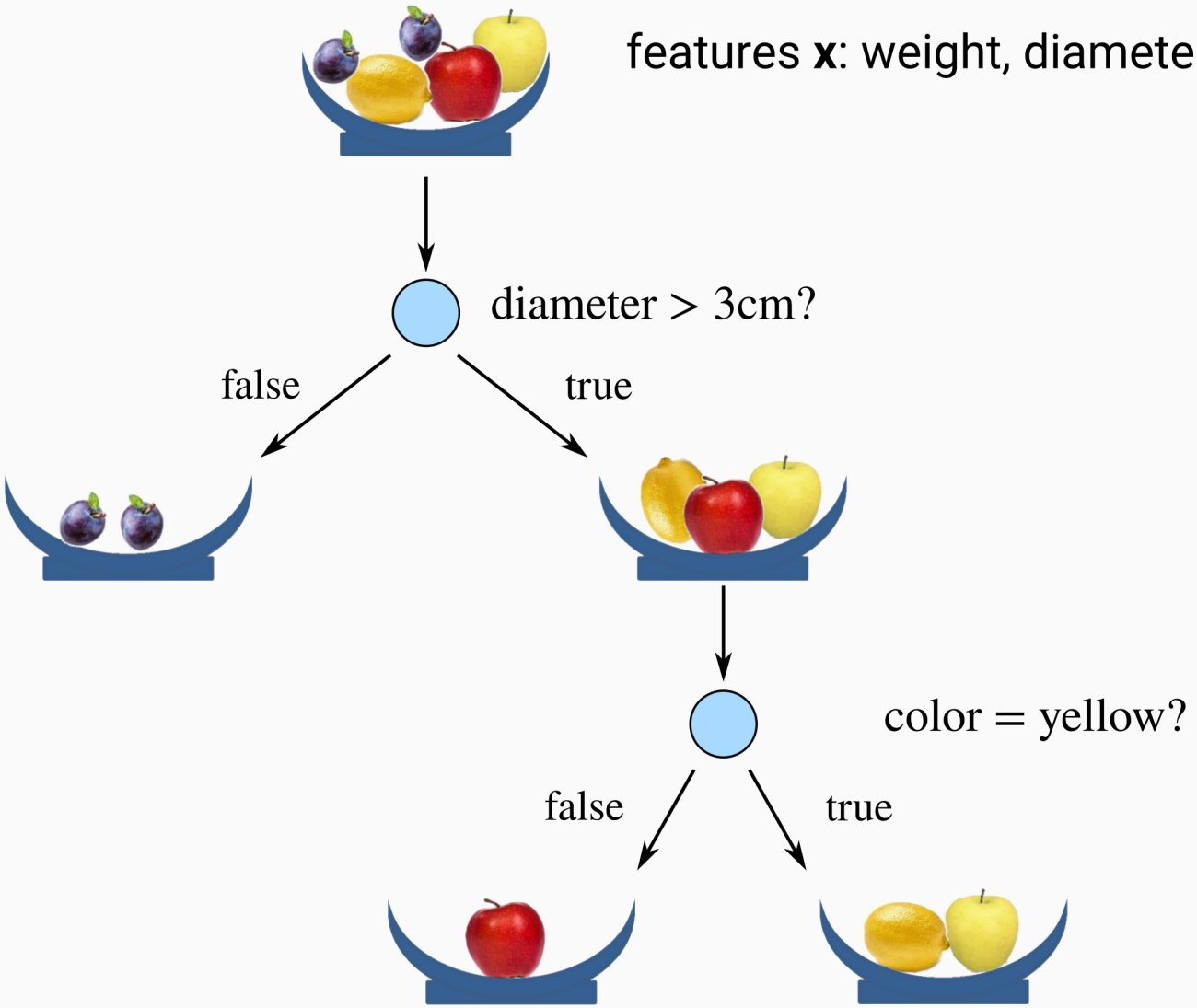
European Summer School on Artificial Intelligence
Athens, 19th July 2024

PCs as Decision Trees

- Correia et al., Joints in Random Forests
- Khosravi et al., Handling Missing Data in Decision Trees: A Probabilistic Approach
- Ventola et al., Generative Clausal Networks: Relational Decision Trees as Probabilistic Circuits

`https://github.com/smatmo/ESSAI24-PCs`

features **x**: weight, diameter, color, acid, ...



Given a set of features $X_1, X_2, \ldots, X_D$ (discrete or continuous) and a class variable Y, a Decision Tree is a **directed binary tree** with two types of nodes:

- **Decision Nodes** (internal nodes)
  Decision nodes are associated with a feature X_i and a **decision** (**test**) function selecting one of the child nodes.
  E.g., for binary decisions

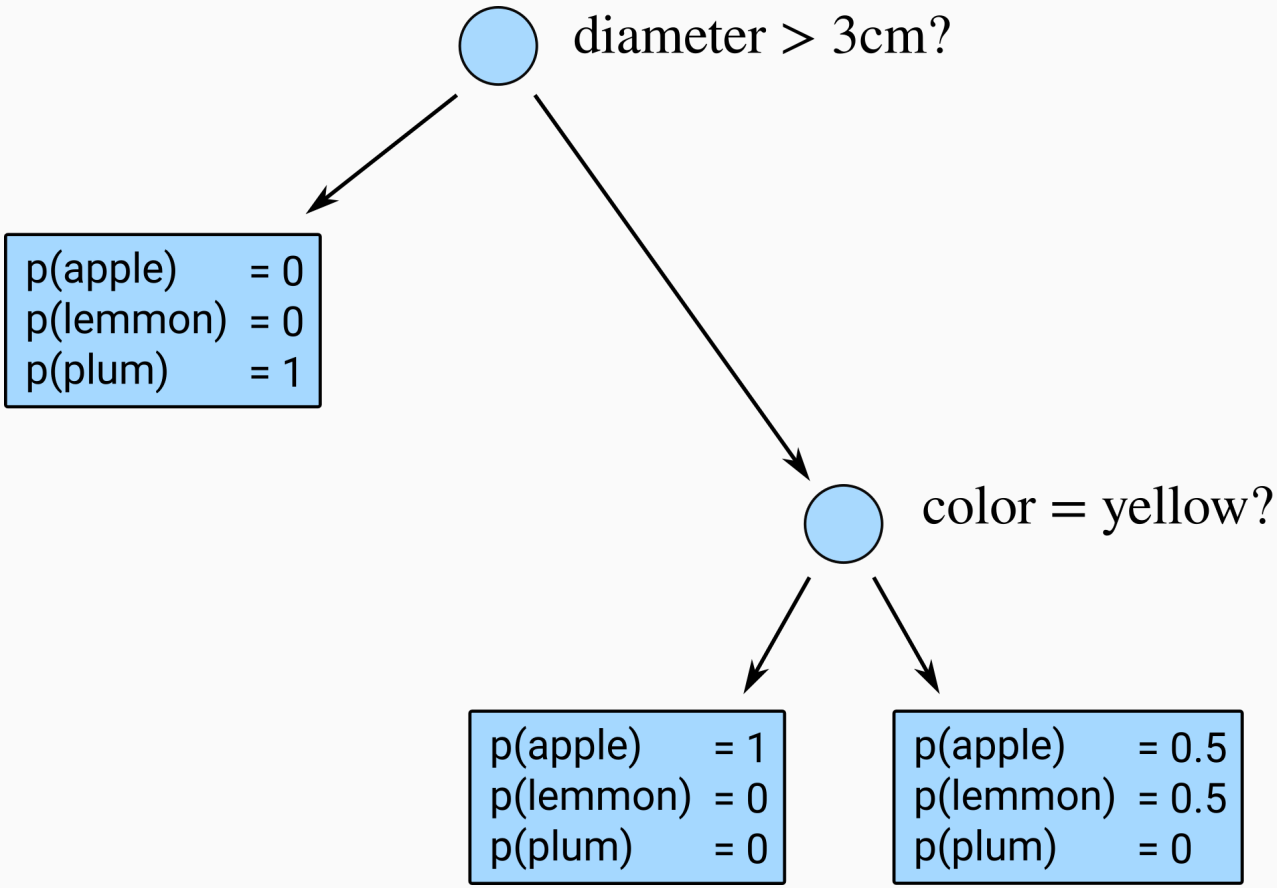
$$f : X_i \mapsto \{true, false\}$$

  selects one of 2 children labeled *true* and *false*.
- **Prediction Nodes** (leaves)
  Prediction nodes contain a distribution over labels $p(Y)$

Decision Tree is a Conditional Distribution

Example

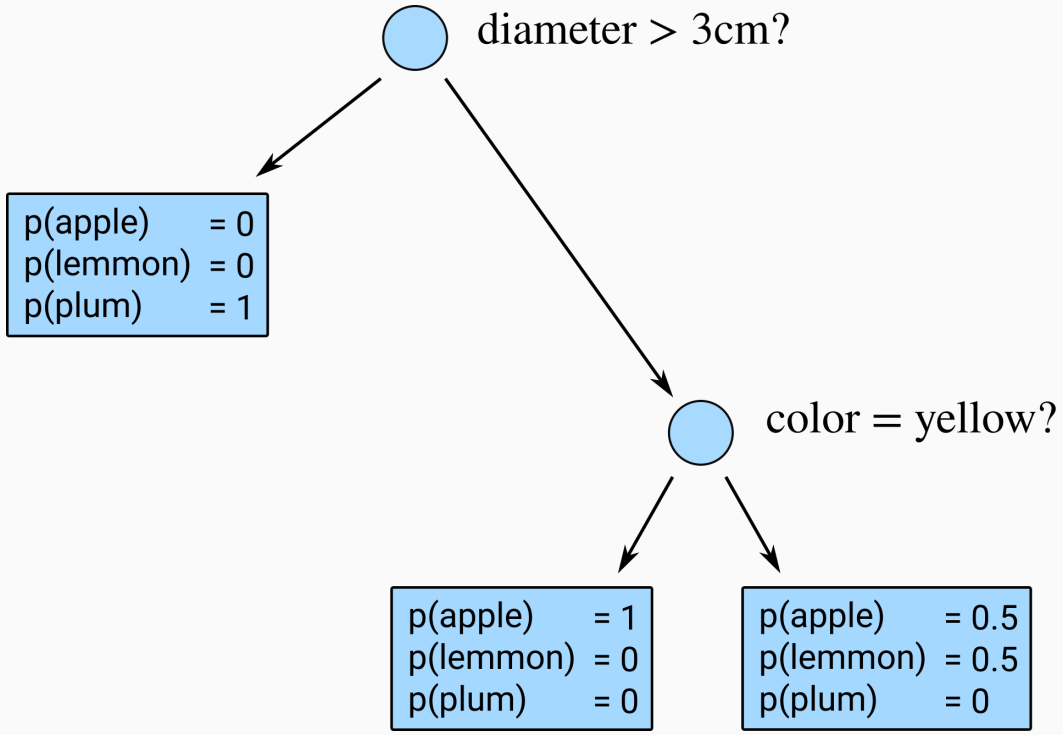
Following the decisions dictated by input $\boldsymbol{x}$ retrieves a distribution over Y. **Hence, a decision tree is a conditional distribution** $p(\boldsymbol{Y} \mid \boldsymbol{x})$.



Decision Trees as Circuits?

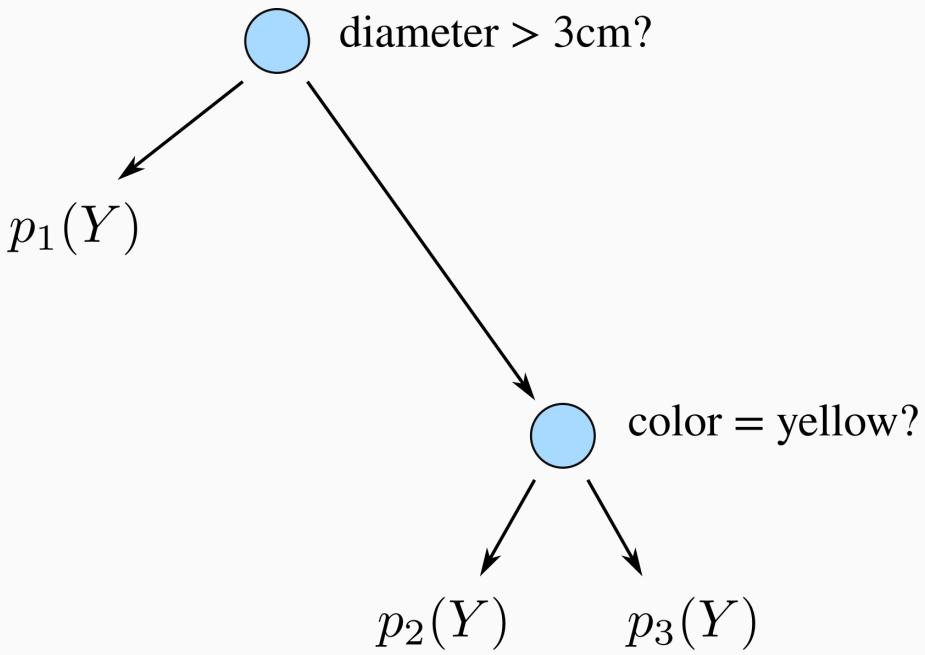
- **decision trees** are one of the most widely used type of classifiers and regression models
- considered highly **interpretable**
- many algorithms exist since the 80's, e.g. the **CART algorithm** grows decision trees top-down by constructing decisions minimizing some **impurity measure**
- **but really, we have been working with PCs all along...**

Converting Decision Trees to PCs



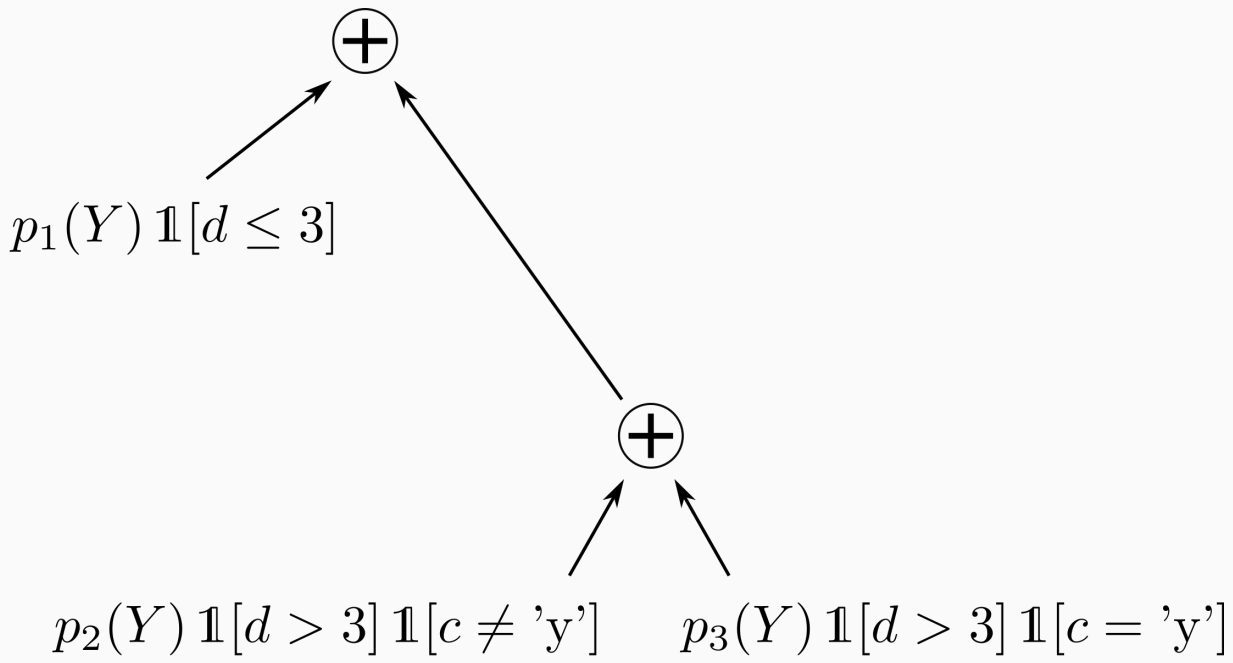
Starting from a decision tree with distributions over Y as leaves.

Converting Decision Trees to PCs



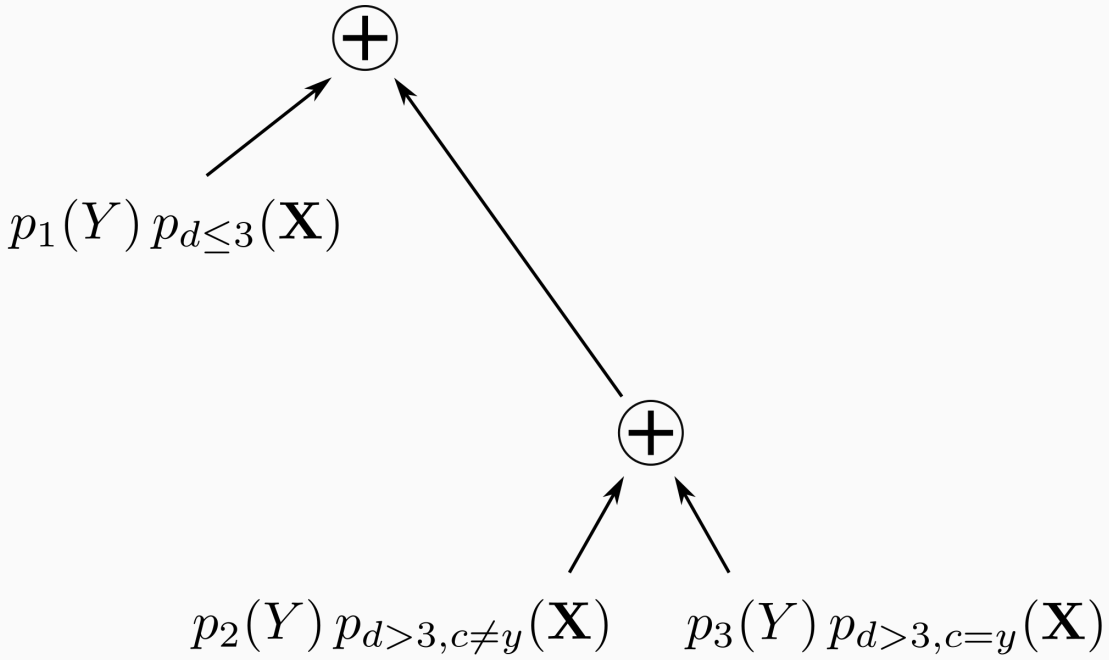
Starting from a decision tree with distributions over Y as leaves.

Converting Decision Trees to PCs



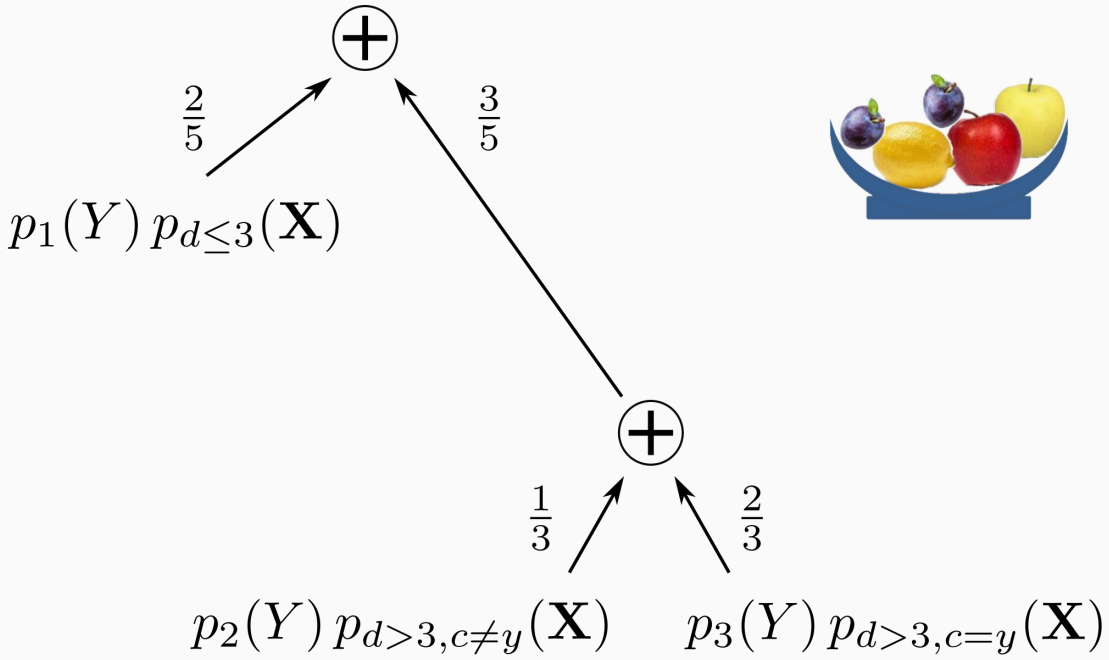
We can replace decision nodes with sum nodes, when we multiply the leaves with **indicators** of all decisions above.

Converting Decision Trees to PCs



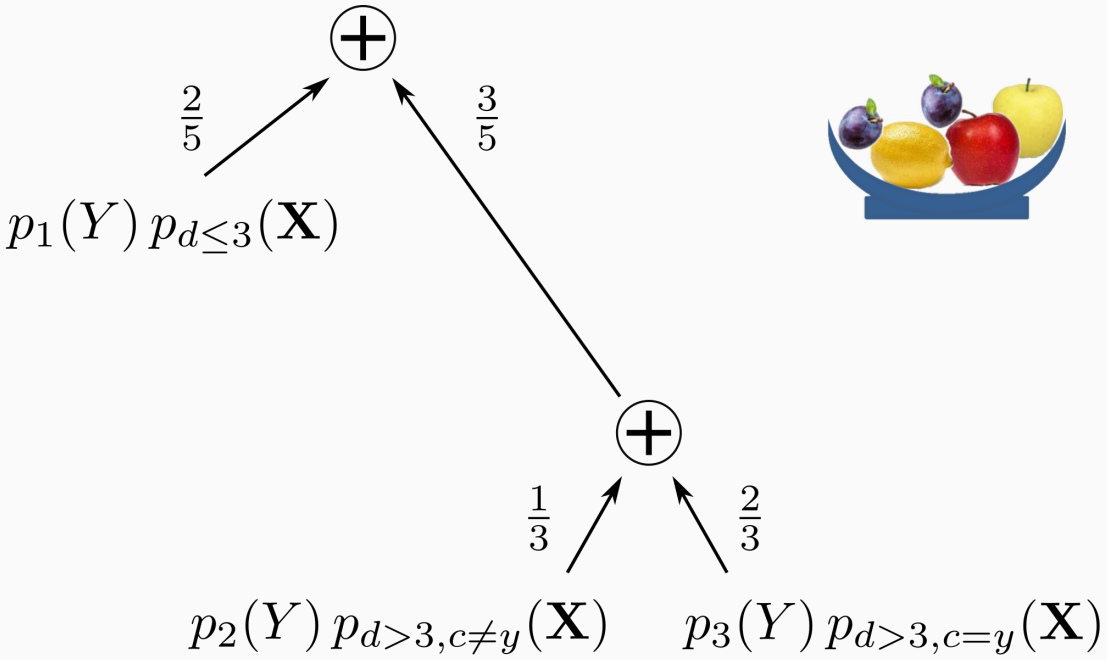
Rather than multiplying with indicators of decisions, we might multiply with a distribution over $\boldsymbol{X}$, where support is restricted by the decisions above (e.g. **truncated Gaussians**, PCs, . . .). They can be learned with maximum likelihood on training samples consistent with these decisions.

Converting Decision Trees to PCs



Finally, let's equip the sum weights also learned with maximum likelihood (just "counting how many samples go each way").

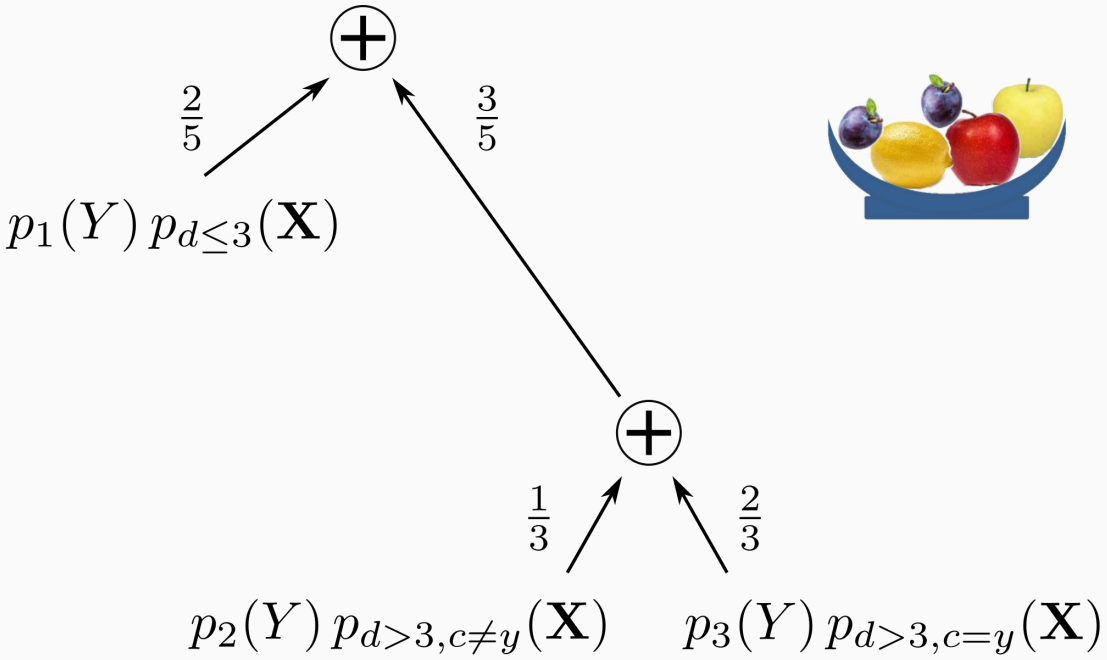
Converting Decision Trees to PCs



Ok, this is now a **smooth**, **decomposable** and **deterministic** PC!
It is now a full **joint distribution** $p(Y, \boldsymbol{X})$, called **generative decision tree**.

Does it still represent the same classifier as the decision tree? Does it represent the same $p(Y \mid \boldsymbol{X})$?

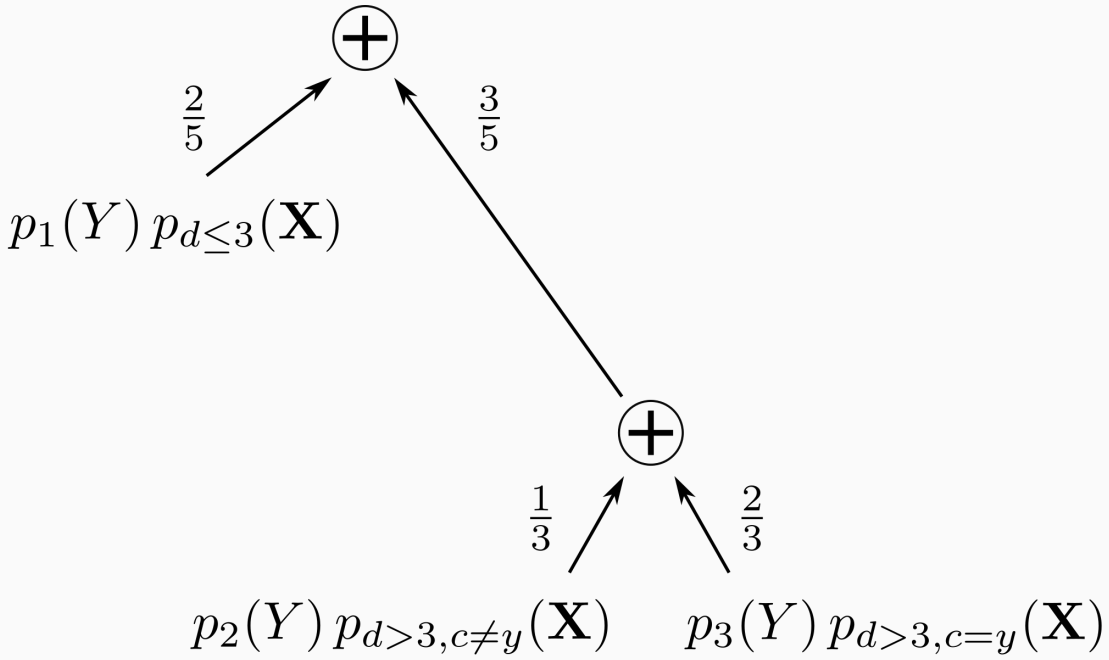
Converting Decision Trees to PCs



Ok, this is now a **smooth**, **decomposable** and **deterministic** PC!
It is now a full **joint distribution** $p(Y, \boldsymbol{X})$, called **generative decision tree**.

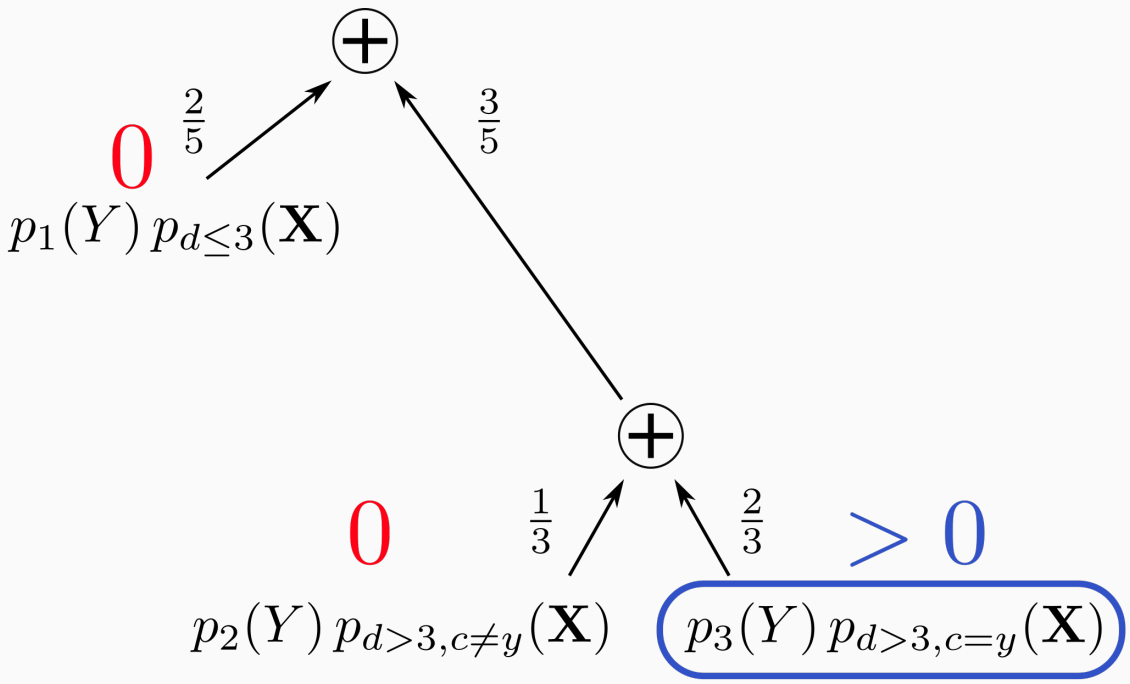
Does it still represent the same classifier as the decision tree? Does it represent the same $p(Y \mid \boldsymbol{X})$? **Yes! It is exactly the same!**

Converting Decision Trees to PCs



For each sample $\boldsymbol{x}$, at most one input will be larger than 0.

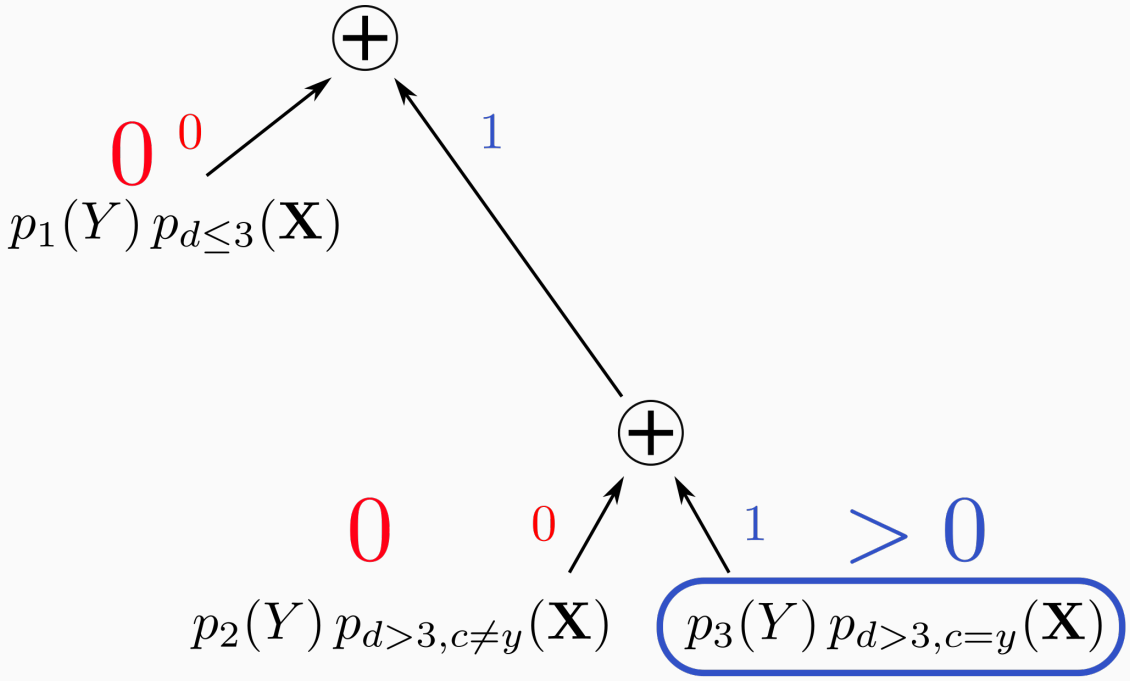
Converting Decision Trees to PCs



For each sample $\boldsymbol{x}$, at most one input will be larger than 0. For example, consider an input sample where

- diameter > 3
- color $=$ 'yellow'

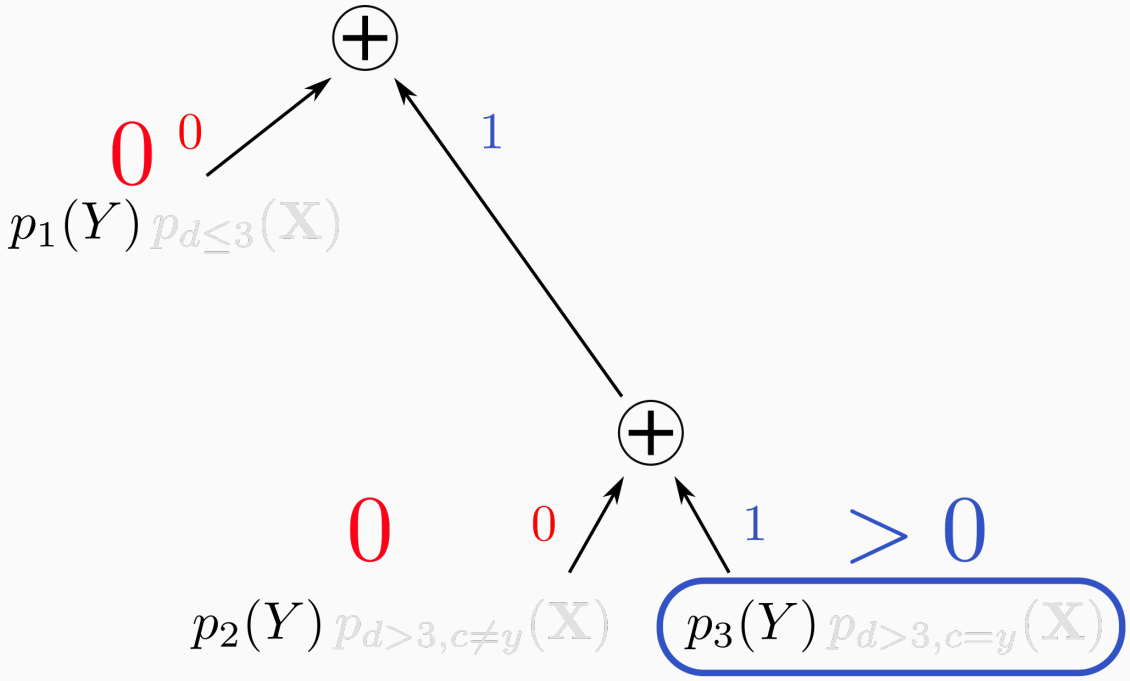
Converting Decision Trees to PCs



To compute the conditional PC $p(Y \mid \boldsymbol{x})$, recall that

- the sum weights get multiplied with the messages below and re-normalized

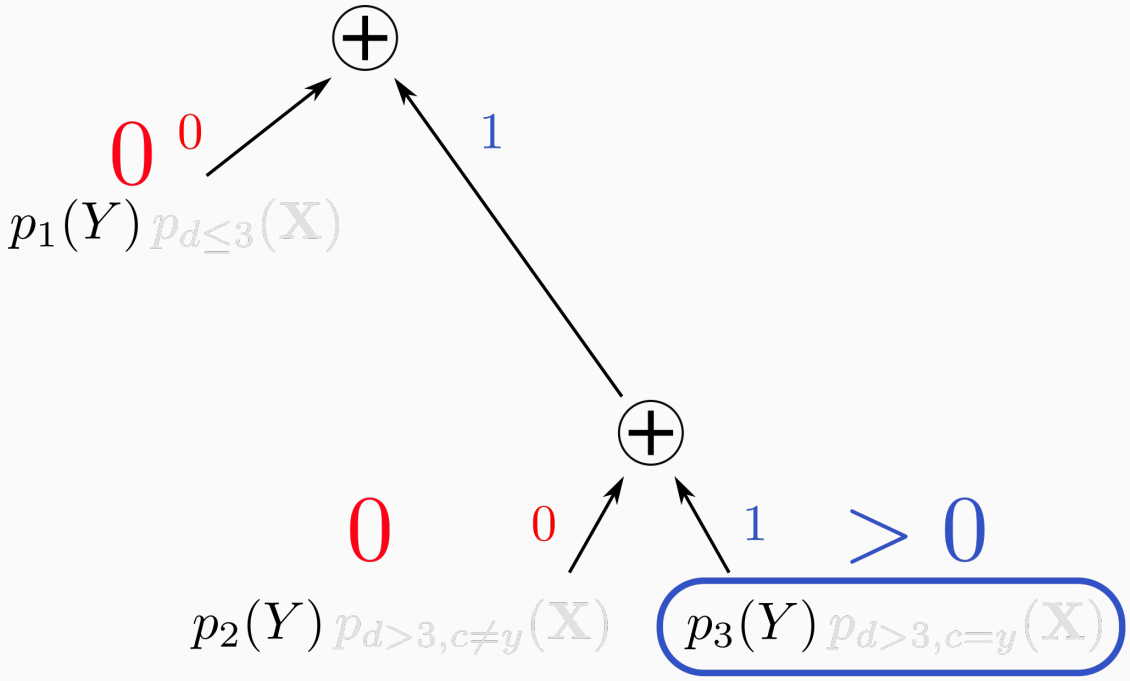
Converting Decision Trees to PCs



To compute the conditional PC $p(Y \mid \boldsymbol{x})$, recall that

- the sum weights get multiplied with the messages below and re-normalized
- the input distributions are replaced by conditionals over Y—here simply the original $p(Y)$'s due to the factorization $p(Y)p(\boldsymbol{X})$

Converting Decision Trees to PCs



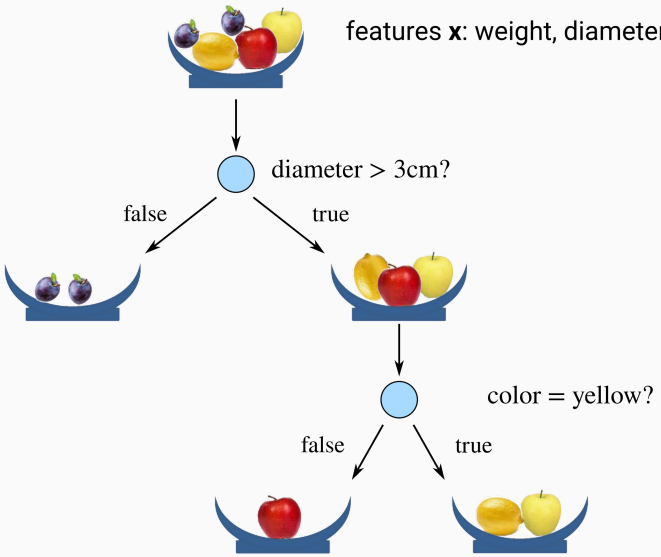
This delivers **exactly** the same prediction node the original decision tree would select!

The Use of a Full Generative Model

- we can take any classical decision tree learner from the literature and use it as a PC structure (**discriminative structure learning**)
- we just need to additionally
  - learn input distributions over $\boldsymbol{X}$
  - learn sum weights
- the classifier stays exactly the same
- having a full joint $p(Y, \boldsymbol{X})$ is useful:
  - we can naturally treat **missing data**
  - we have a natural **outlier detector** on board

Missing Data

- many prediction systems (e.g. classifiers) require us that **all input features are observed**
- what if they are missing?
- how to deal with this in neural networks, decision trees etc.?
- for example, what if the feature *diameter* is simply not present in our test sample?



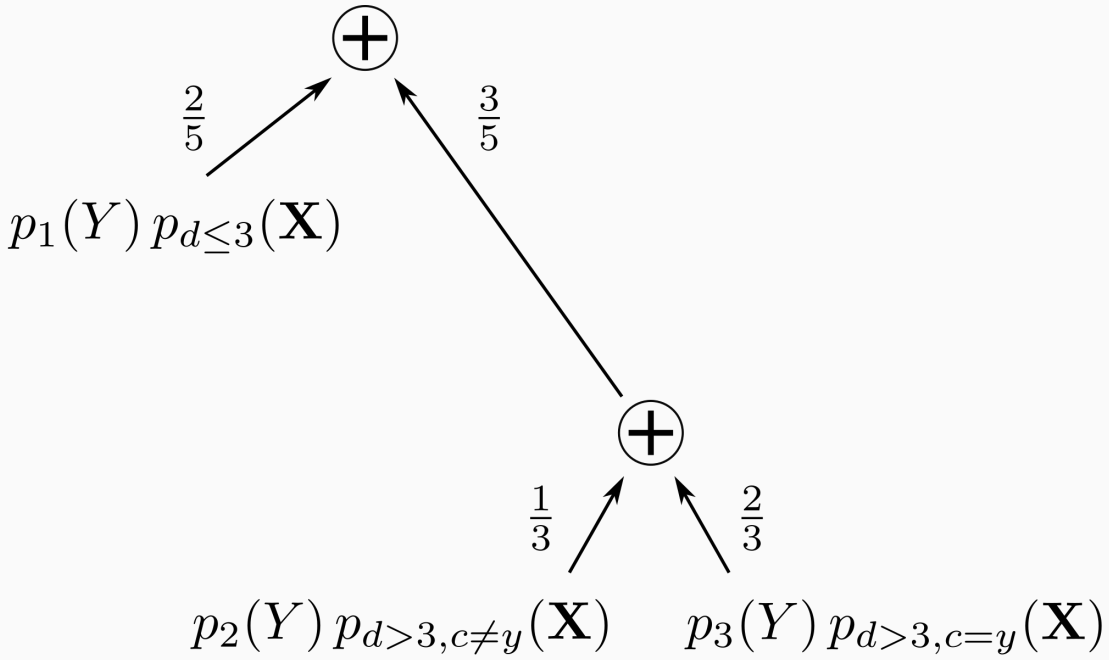
Missing Data: The Probabilistic (hence Correct) Answer

- let $p(Y, \boldsymbol{X})$ be the joint distributions generating data
- the **Bayes optimal classifier** is

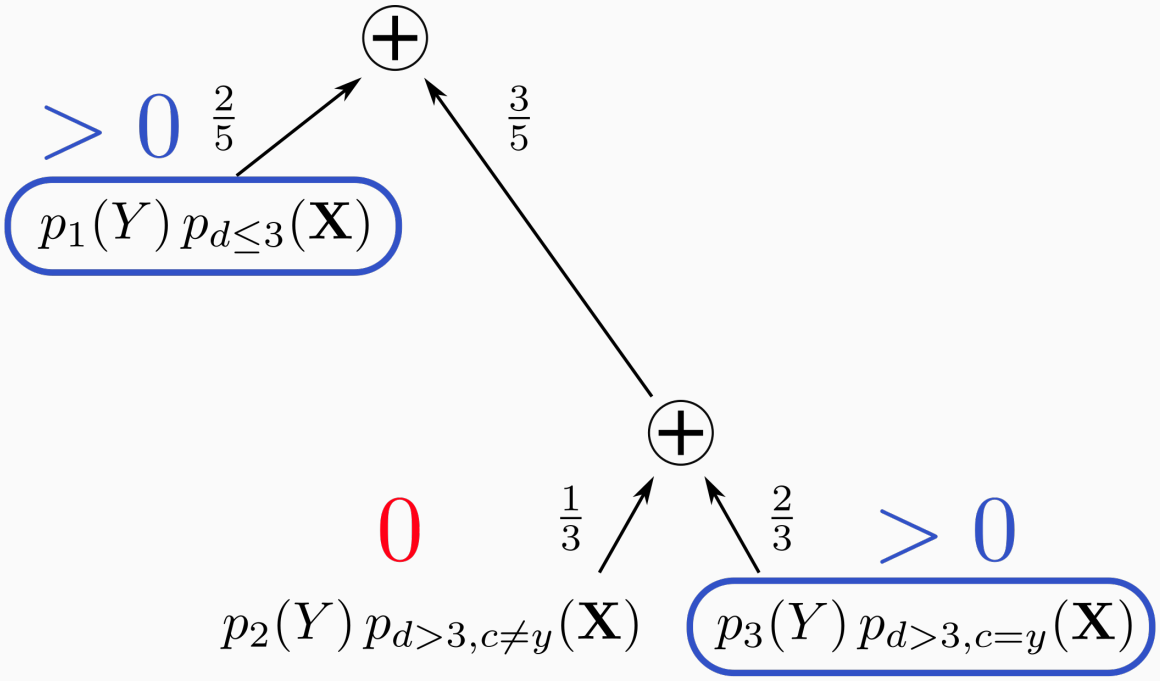
$$\arg\max_y p(y \mid \boldsymbol{x})$$

- let $\boldsymbol{X}_m \subset \boldsymbol{X}$ be features **missing at random** (meaning, the missingness is uncorrelated with Y)
- the **observed** variables are $\boldsymbol{X}_o = \boldsymbol{X} \setminus \boldsymbol{X}_m$
- the Bayes optimal classifier for $\boldsymbol{X}_o$ is

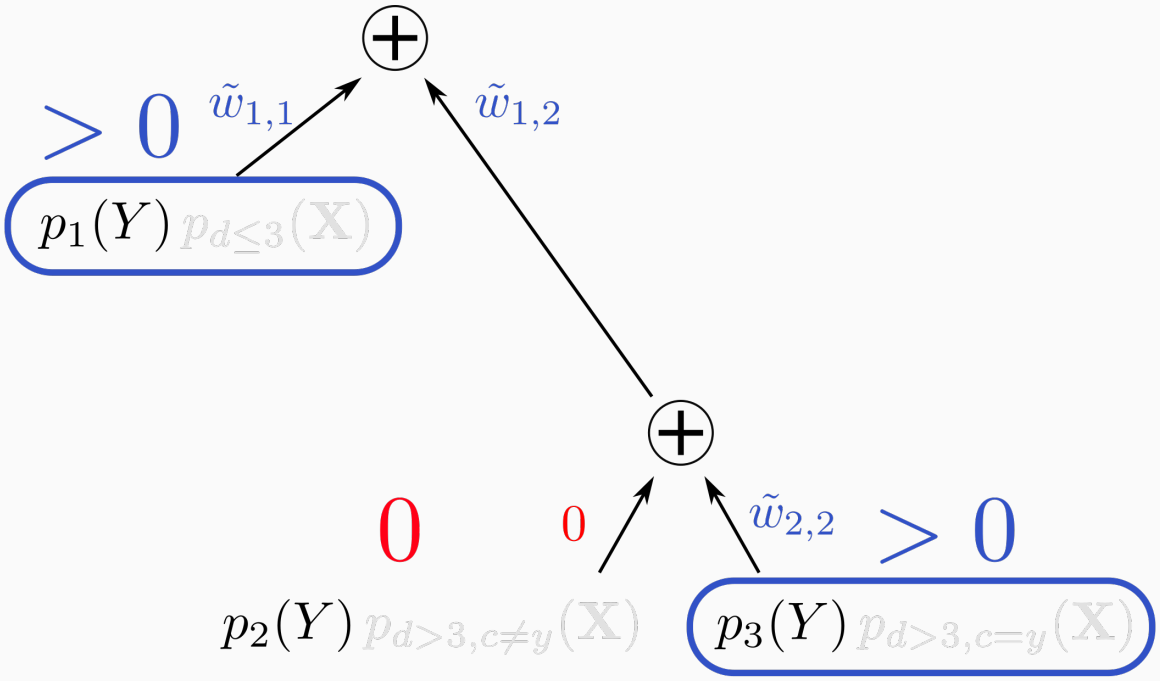
$$\arg\max_y p(y \mid \boldsymbol{x}_o) = \arg\max_y \frac{\overbrace{\int p(y, \boldsymbol{x}_o, \boldsymbol{x}_m) \, \mathrm{d}\boldsymbol{x}_m}^{p(y, \boldsymbol{x}_o),\ \boldsymbol{X}_m \text{ marginalized}}}{\underbrace{\sum_y \int p(y, \boldsymbol{x}_o, \boldsymbol{x}_m) \, \mathrm{d}\boldsymbol{x}_m}_{p(\boldsymbol{x}_o),\ \boldsymbol{X}_m \text{ and } Y \text{ marginalized}}}$$



In our example, consider a sample where *diameter* is missing, but we observe *color* is 'yellow'.



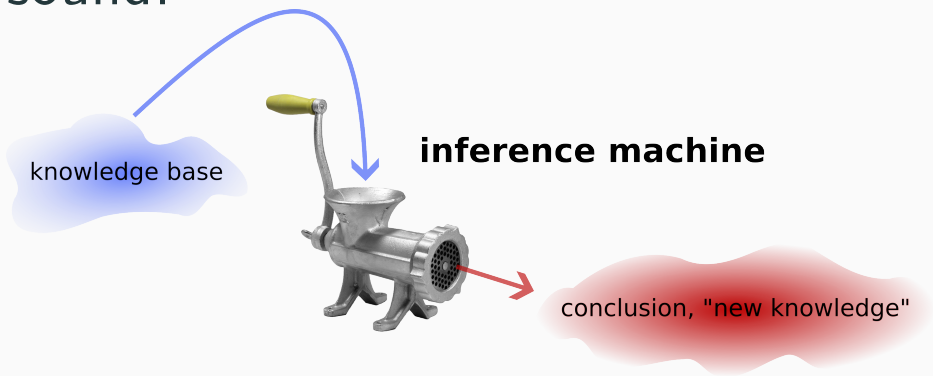
Since the input distributions marginalize now over *diameter*, several of them are non-zero.



The conditional PC delivers some weight updates, which are not just 0 or 1 now. The conditional PC computes a weighted combination of distributions over Y now!

Missing Data with Generative Decision Trees

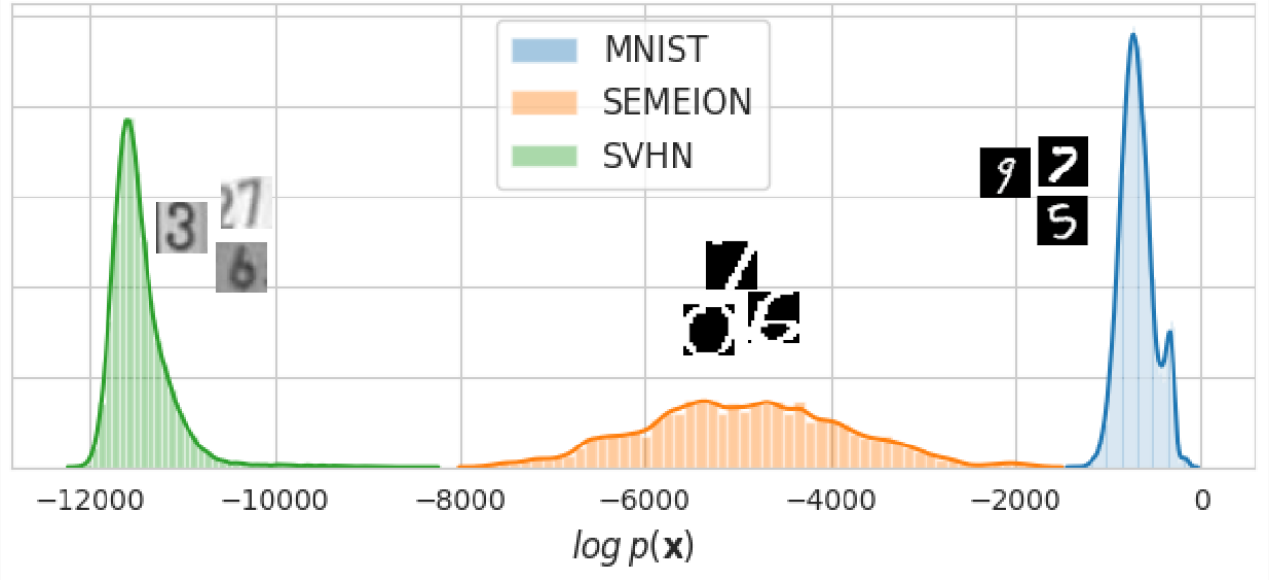
- **missing data should simply be marginalized!**
- **an easy exercise for PCs, in particular generative decision trees!**
- of course, PCs are only an approximation to the real data-generating distribution, but the reasoning principle is sound!



- other approaches in literature are based on **imputation** and theoretically less well-justified or even flawed

Outlier Detection

Another use of having a full joint $p(Y, \boldsymbol{X})$ is that we can detect outliers with the marginal $p(\boldsymbol{X})$. If we get an input $\boldsymbol{x}^*$, simply query $p(\boldsymbol{x}^*)$ and check whether it is much lower than the typical $p(\boldsymbol{x})$ seen during training:



Peharz et al., Einsum Networks, ICML 2020.

Random Forests

- decision trees are nice and interpretable, but they usually underperform in practice
- **random forests** are ensembles of decision trees and some of the strongest predictive models on tabular data!
- learned by **bagging** (bootstrapping and aggregating):
  - **bootstrapping**: generate K random variations of the training data, by drawing N samples **with replacement**
  - **aggregating**: learn an individual decision tree on each bootstrap and aggregate the results, such as majority vote or averaging $p(Y \mid \boldsymbol{X})$ over trees

Generative Forests

- convert each decision tree in a random forest into a generative decision tree, yielding a **generative forest** (GeF)
- how to aggregate though?
- **the backward compatible way:**

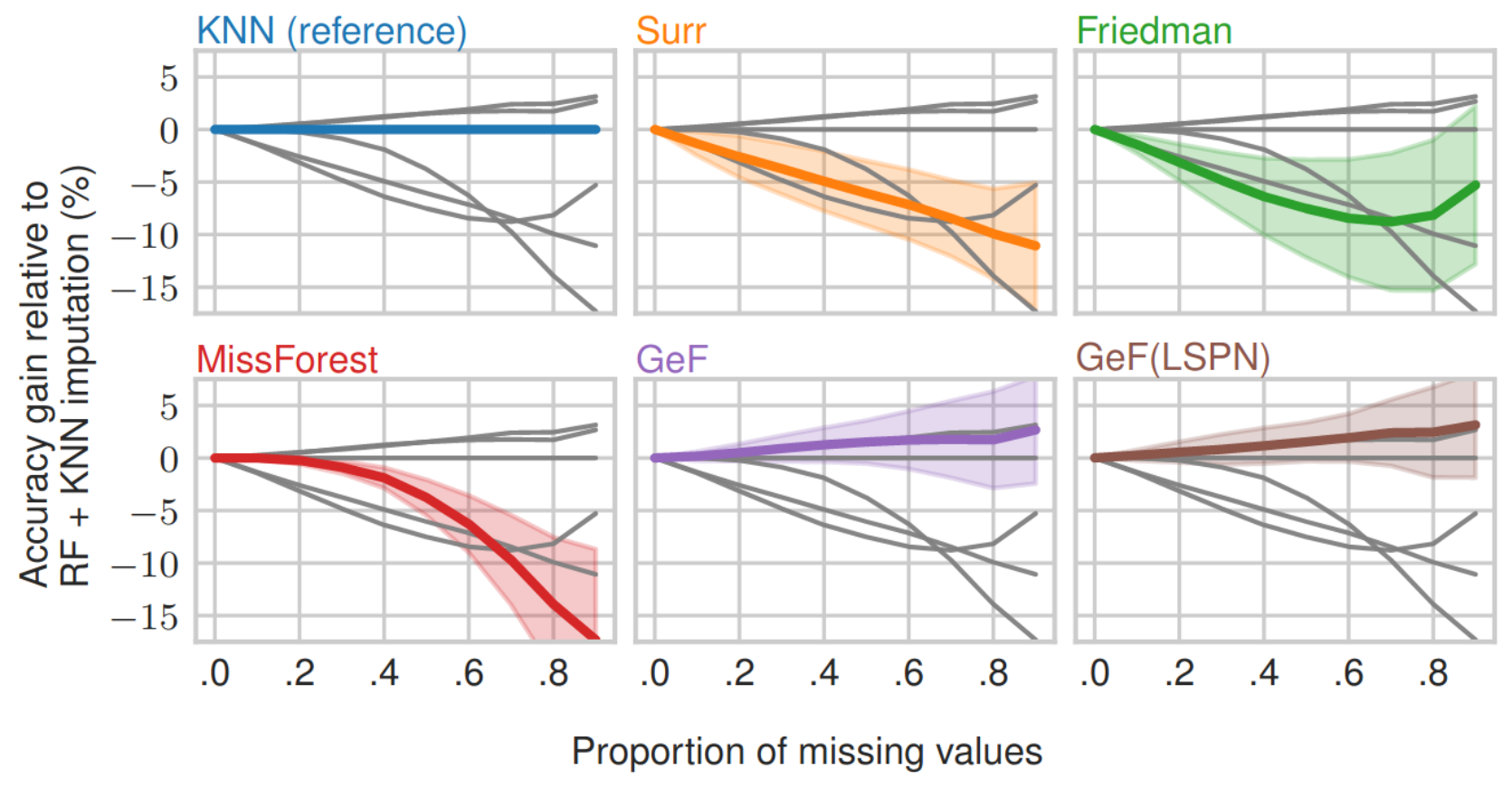
$$\frac{1}{K}\sum_{k=1} p_k(Y \mid \boldsymbol{X})$$

- **the novel PC way (GeF$^+$):**

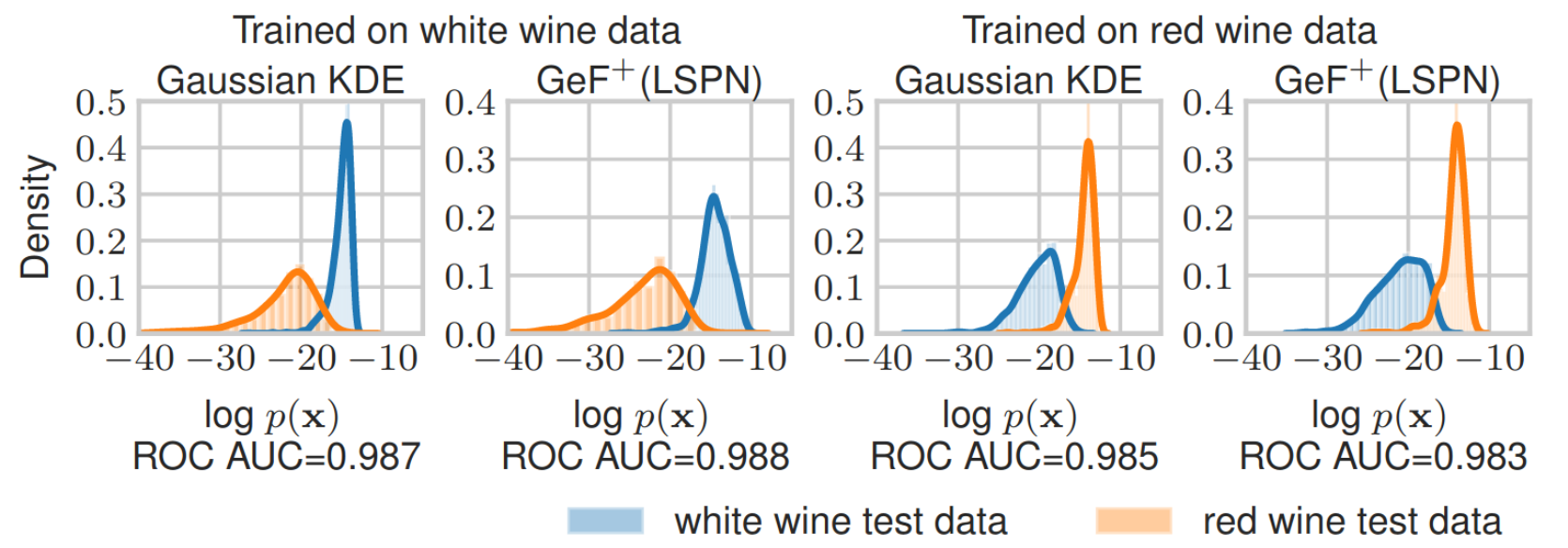
$$\frac{1}{K}\sum_{k=1} p_k(Y, \boldsymbol{X})$$

- the latter is a mixture of joints, hence again a PC and a tractable joint
- it sometimes delivers even a better classifier than the original forest (but sometimes also not...)

Missing Data with GeFs



Outlier Detection with GeF$^+$s



complex
probabilistic reasoning & modeling

antonio vergari (he/him)

@tetraduzione

robert peharz

19th July 2024 - **ESSAI 2024** - *Athens*

Reasoning about ML models

q_1 *"What is the probability of a treatment for a patient with* ***unavailable records****?"*

q_2 *"How* ***fair*** *is the prediction is a certain protected attribute changes?"*

q_3 *"Can we certify no* ***adversarial examples*** *exist?"*

Reasoning about ML models

$\mathbf{q}_1$ $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

$\mathbf{q}_2$ $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

$\mathbf{q}_3$ $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

...in the language of probabilities

Reasoning about ML models

q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

q3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

it is crucial we compute them exactly and in polytime!

Reasoning about ML models

q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

q3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

it is crucial we compute them tractably!

Which structural properties

for complex reasoning

smooth + decomposable

Which structural properties

for complex reasoning

smooth + decomposable

???????

decomposable

Which structural properties

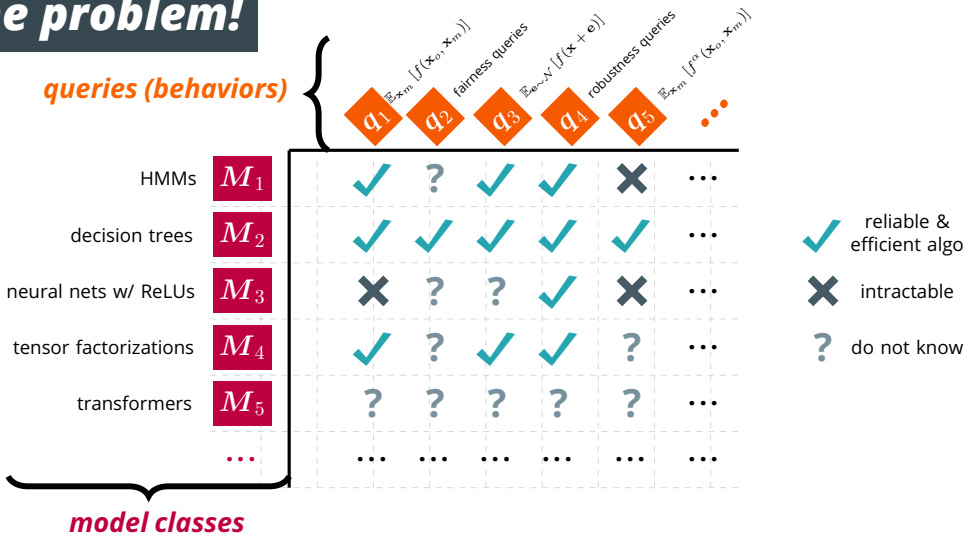
for complex reasoning

smooth + decomposable

compatibility

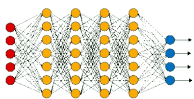
decomposable

The problem!


queries (behaviors)
q_1 $\mathbb{E}_{\mathbf{x}_m}[f(\mathbf{x}_o, \mathbf{x}_m)]$
q_2 fairness queries
q_3 $\mathbb{E}_{\mathbf{e}\sim\mathcal{N}}[f(\mathbf{x}+\mathbf{e})]$
q_4 robustness queries
q_5 $\mathbb{E}_{\mathbf{x}_m}[f^n(\mathbf{x}_o, \mathbf{x}_m)]$
HMMs M_1
decision trees M_2
neural nets w/ ReLUs M_3
tensor factorizations M_4
transformers M_5
model classes
reliable & efficient algo
intractable
do not know

"How uncertain will the classifier be if some input is missing?"

behaviors
to be inspected



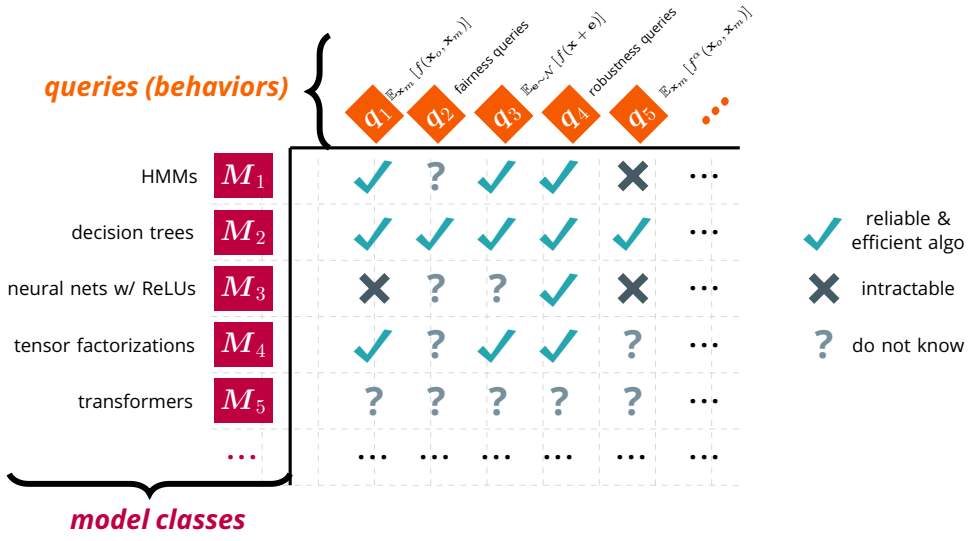
ML models
(classifiers, generative models, ...)

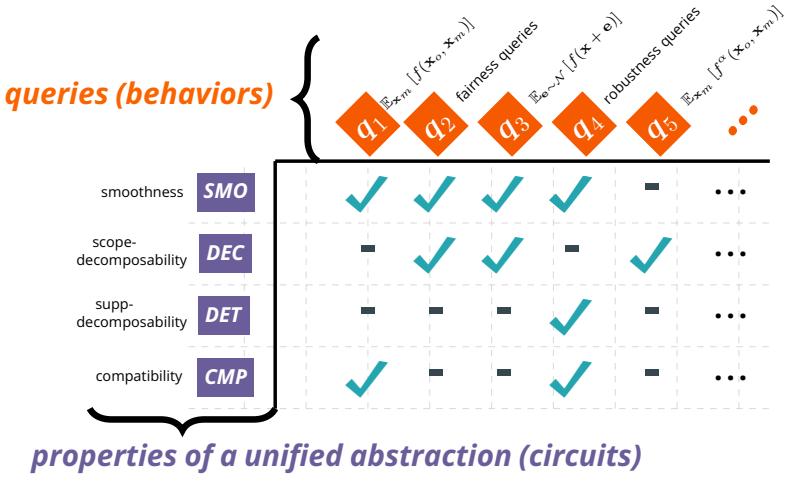
???

```
def var(p, f):
    r = pow(f, 2)
    t = mult(r, p)
    return integrate(t)
```

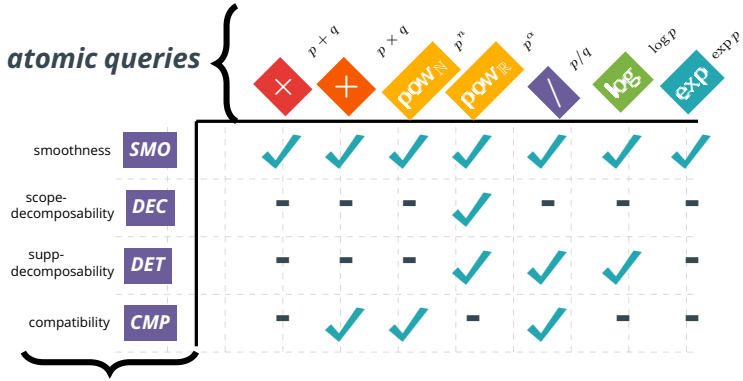
efficient & reliable complex reasoning routines
(fast routines to inspect and guarantee a ML system's behavior)

How can we solve this *engineering bottleneck*?


queries (behaviors)
q_1 $\mathbb{E}_{x_m}[f(x_o, x_m)]$
q_2 fairness queries
q_3 $\mathbb{E}_{e\sim\mathcal{N}}[f(x+e)]$
q_4 robustness queries
q_5 $\mathbb{E}_{x_m}[f^{a_1}(x_o, x_m)]$
HMMs M_1
decision trees M_2
neural nets w/ ReLUs M_3
tensor factorizations M_4
transformers M_5
model classes
reliable & efficient algo
intractable
do not know



necess. conditions for reliability and efficiency



✓ necess. conditions for reliability and efficiency

(De)composing queries

*"What is the **expected prediction** for a patient with unavailable records?"*

$$\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times f(\mathbf{x}_o, \mathbf{x}_m) \right]$$

(De)composing queries

*"What is the **expected prediction** for a patient with unavailable records?"*

$$\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times f(\mathbf{x}_o, \mathbf{x}_m) \right]$$

*"How **fair** is the prediction is a certain protected attribute changes?"*

$$\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times \text{pow}\left(f(\mathbf{x}_o, \mathbf{x}_m), 2\right) \right] - \text{pow}\left(\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times f(\mathbf{x}_o, \mathbf{x}_m) \right], 2 \right)$$

(De)composing queries

*"What is the **expected prediction** for a patient with unavailable records?"*

$$\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times f(\mathbf{x}_o, \mathbf{x}_m) \right]$$

*"How **fair** is the prediction is a certain protected attribute changes?"*

$$\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times \text{pow}(f(\mathbf{x}_o, \mathbf{x}_m), 2) \right] - \text{pow}\left(\int \left[p(\mathbf{x}_m \mid \mathbf{x}_o) \times f(\mathbf{x}_o, \mathbf{x}_m) \right], 2 \right)$$

*"Can we certify no **adversarial examples** exist?"*

$$\int \left[p(\mathbf{x}_c \mid X_s = 0) \times f(\mathbf{x}_c, 0) \right] - \int \left[p(\mathbf{x}_c \mid X_s = 1) \times f(\mathbf{x}_c, 1) \right]$$

A language for queries

Integral expressions that can be formed by composing these operators

$+$, $\times$, **pow**, **log**, **exp** and $/$

$\Rightarrow$ *many divergences and information-theoretic queries*

A language for queries

Integral expressions that can be formed by composing these operators

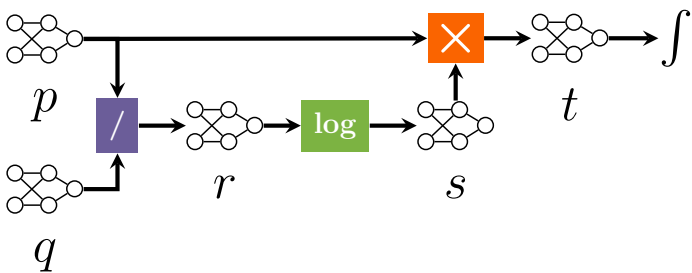
$+$, $\times$, **pow**, **log**, **exp** and $/$

$\Rightarrow$ *many divergences and information-theoretic queries*

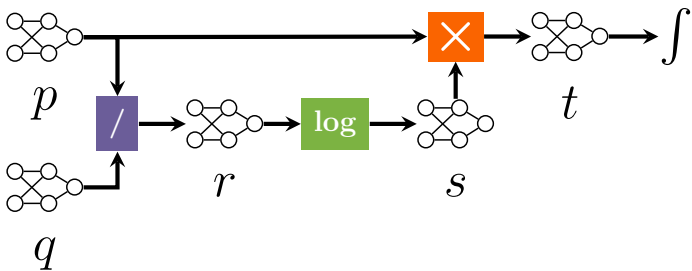
Represented as ***higher-order computational graphs***—pipelines—operating over circuits!

$\Rightarrow$ *re-using intermediate transformations across queries*

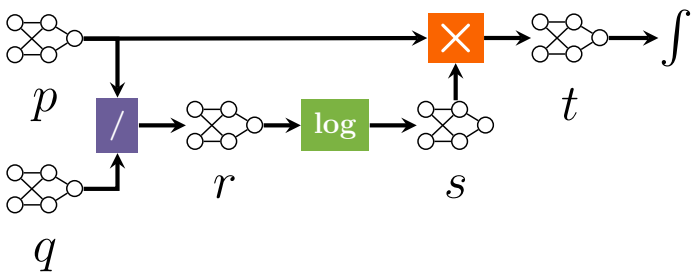
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \log\left(p(\mathbf{x})/q(\mathbf{x})\right)\ d\mathbf{X}$$



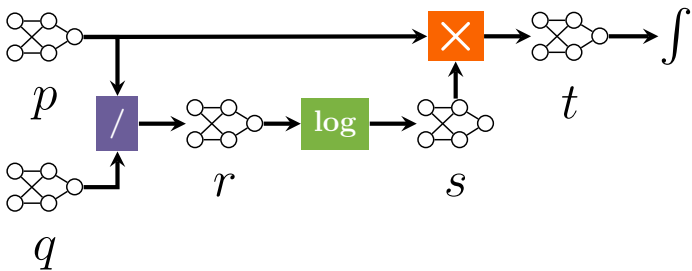
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \log\left(p(\mathbf{x}) \,/\, q(\mathbf{x})\right)\, d\mathbf{X}$$



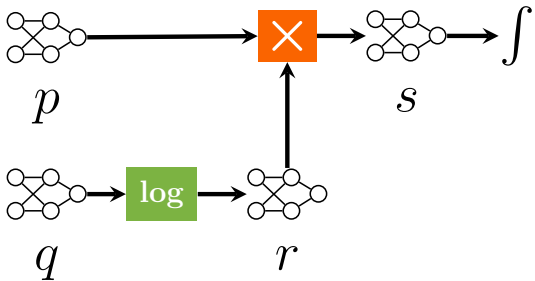
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \log\left(p(\mathbf{x})/q(\mathbf{x})\right)\, d\mathbf{X}$$



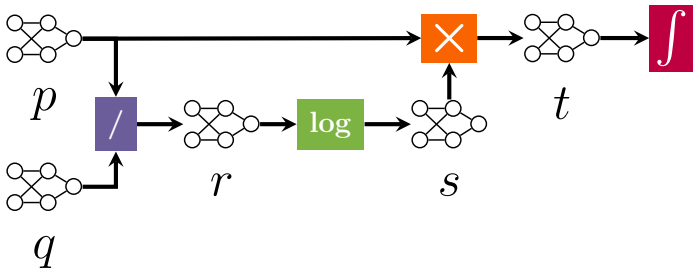
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \log\left(p(\mathbf{x})/q(\mathbf{x})\right)\ d\mathbf{X}$$



$$\mathbb{XENT}(p \mid\mid q) = \int p(\mathbf{x}) \times \log q(\mathbf{x})\, d\mathbf{X}$$

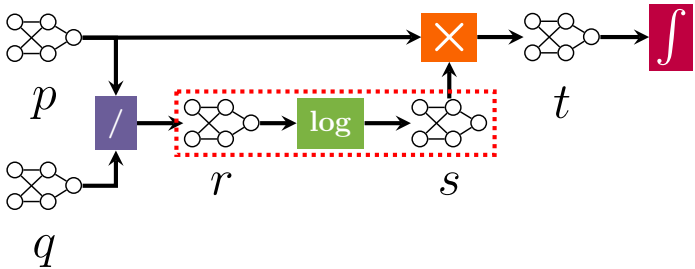


$$\int p(\mathbf{x}) \times \log \left(p(\mathbf{x}) \,/\, q(\mathbf{x}) \right) \, d\mathbf{X}$$



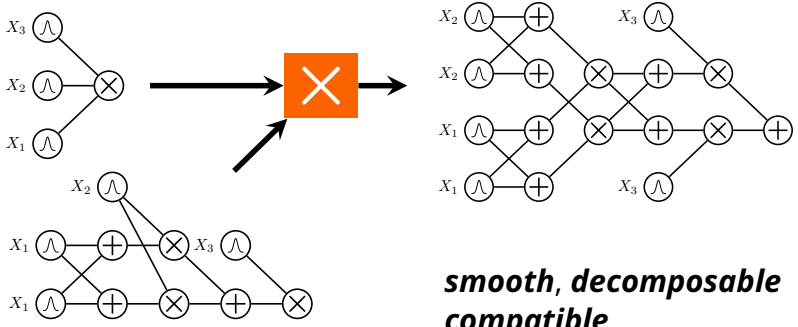
build a LEGO-like query calculus...

$$\int p(\mathbf{x}) \times \log\left(p(\mathbf{x}) / q(\mathbf{x})\right) \, d\mathbf{X}$$



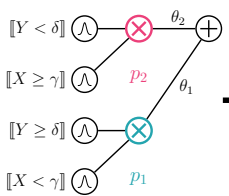
build a LEGO-like query calculus...

Tractable operators

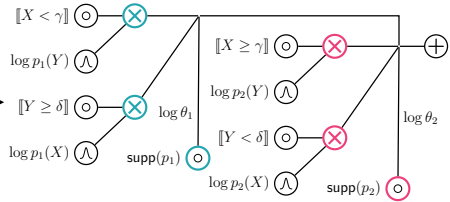


smooth*, *decomposable
compatible

Tractable operators

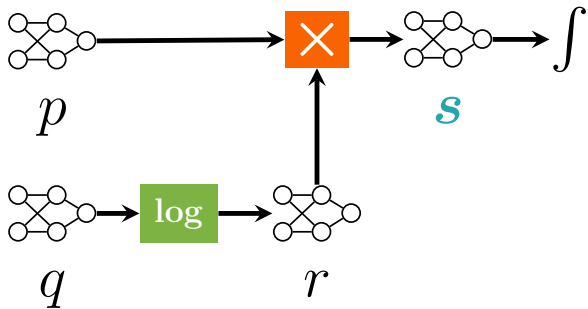


log

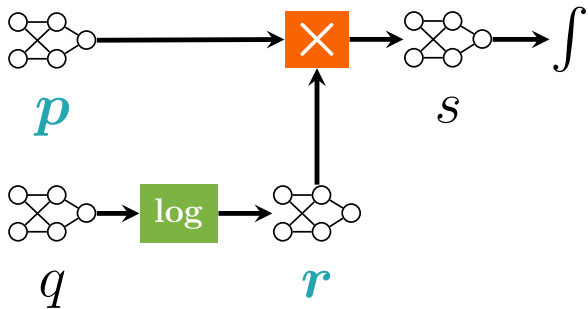


smooth, decomposable deterministic

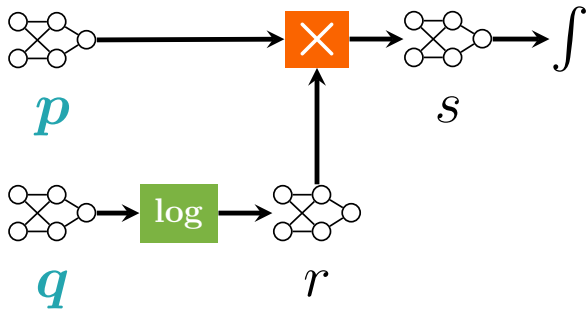
smooth, decomposable



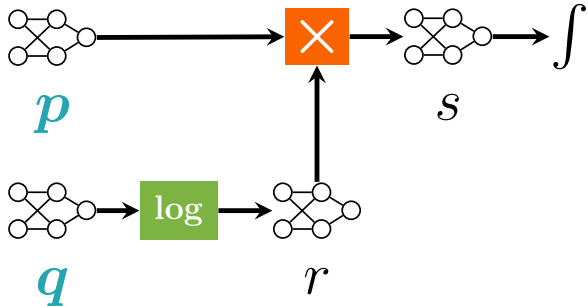
To perform tractable integration we need s to be ***smooth and decomposable...***



hence we need p and r to be smooth, decomposable and ***compatible***...



therefore q must be smooth, decomposable and ***deterministic***...



we can compute $\mathbb{XENT}$ tractably if $\boldsymbol{p}$ and $\boldsymbol{q}$ are smooth, decomposable, compatible and $\boldsymbol{q}$ is deterministic...

	Query	Tract. Conditions	Hardness
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x}) \, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det	#P-hard w/o Det
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x}))d\mathbf{X}$	Cmp, Det	#P-hard w/o Det
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp

compositionally *derive the tractability of many more queries*

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

	Query	Tract. Conditions	Hardness
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x}) \, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det	#P-hard w/o Det
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y}) / (p(\boldsymbol{x}) p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x}) / q(\boldsymbol{x})) d\mathbf{X}$	Cmp, Det	#P-hard w/o Det
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x}) \, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x}) / q(\boldsymbol{x}) - \log(p(\boldsymbol{x}) / q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp

*and **prove hardness** when some input properties are not satisfied*

***Vergari** et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

Composable tractable sub-routines

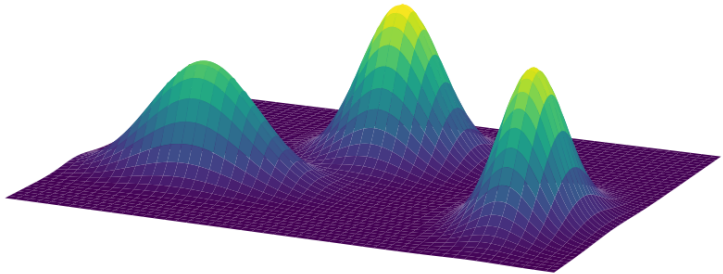
```
function kld(p, q)
    r = quotient(p, q)
    s = log(r)
    t = product(p, s)
    return integrate(t)
end
```

```
function xent(p, q)
    r = log(q)
    s = product(p, r)
    return -integrate(s)
end
```

```
function ent(p)
    q = log(p)
    r = product(p, q)
    return -integrate(s)
end
```

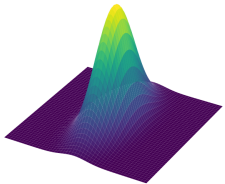
```
function alphadiv(p, q, alpha=1.5)
    r = real_pow(p, alpha)
    s = real_pow(q, 1.0-alpha)
    t = product(r, s)
    return log(integrate(t)) / (1.0-alpha)
end
```

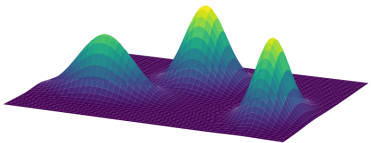
so far...



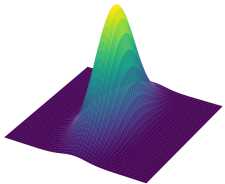
oh mixtures, you're so fine you blow my mind!

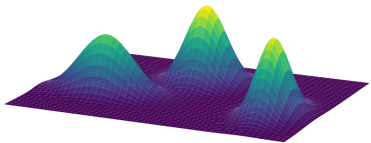
image taken from Hao Tang's course on ASR





$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X})$$

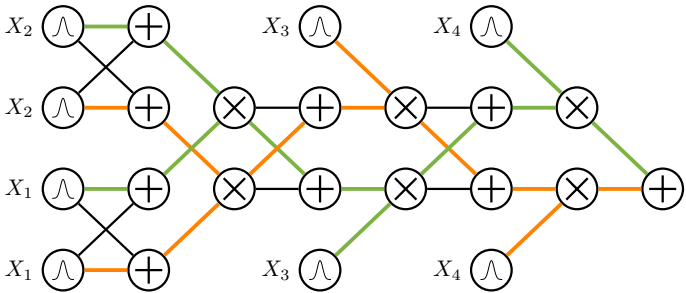




$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X})$$

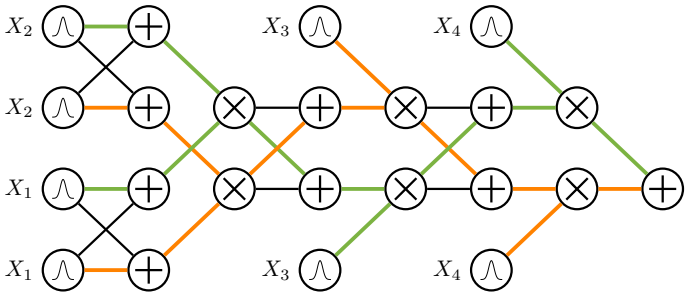
"if someone publishes a paper on model A, there will be a paper about mixtures of A soon with high probability" **A. Vergari**

Expressive efficiency

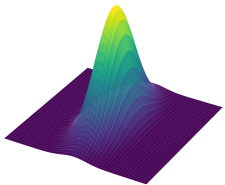


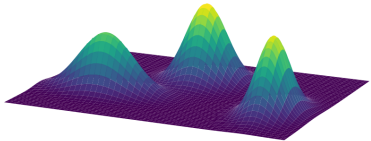
$$p(\mathbf{x}) = \sum_{\mathcal{T}} \left(\prod_{w_j \in \mathbf{w}_{\mathcal{T}}} w_j \right) \prod_{l \in \text{leaves}(\mathcal{T})} p_l(\mathbf{x})$$

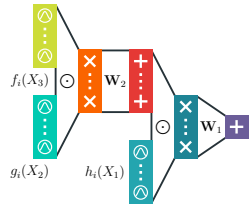
Expressive efficiency



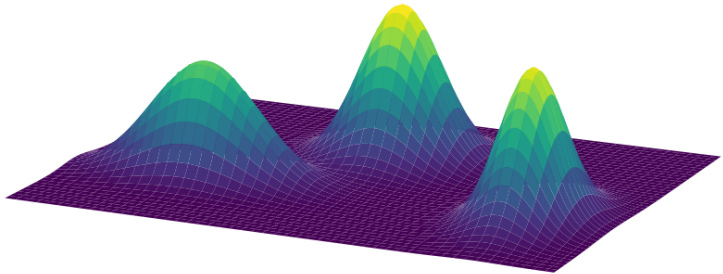
an exponential number of mixture components!





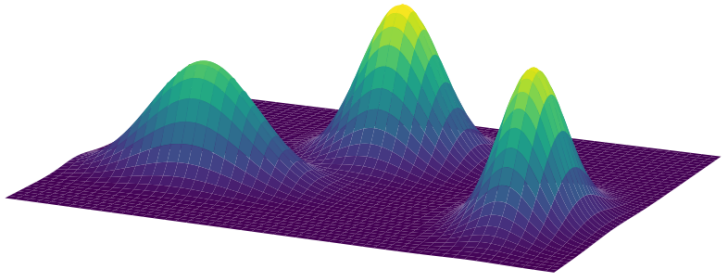


$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$



$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

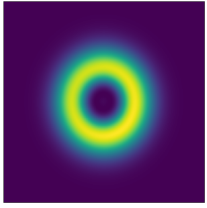
image taken from Hao Tang's course on ASR



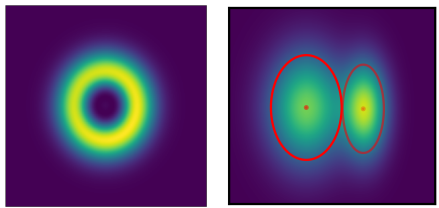
$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

however...

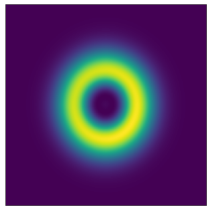


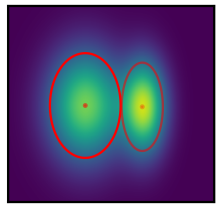
however...



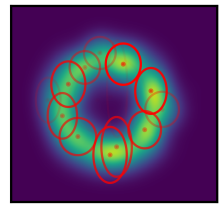
GMM ($K = 2$)

however...



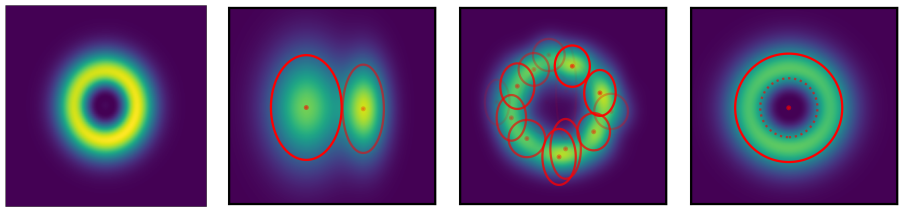


GMM ($K = 2$)



GMM ($K = 16$)

however...



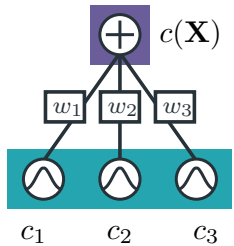
GMM ($K = 2$) GMM ($K = 16$) nGMM^2 ($K = 2$)

theorem

Shallow mixtures with negative parameters can be *exponentially more compact* than deep ones with positive ones.

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

subtractive MMs as circuits

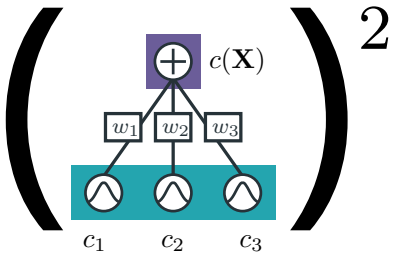


a **non-monotonic** smooth and (structured) decomposable circuit

$\Rightarrow$ *possibly with negative outputs*

$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \qquad w_i \in \mathbb{R},$$

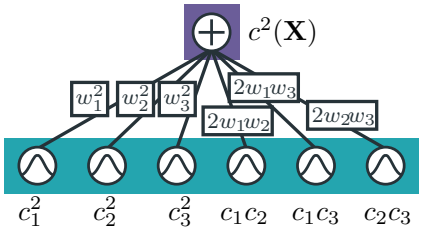
squaring shallow MMs



$$c^2(\mathbf{X}) = \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2$$

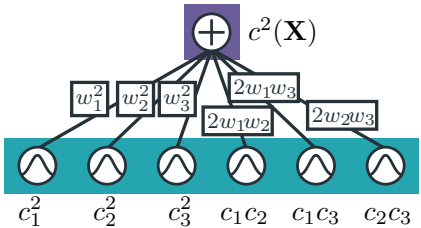
$\Rightarrow$ *ensure non-negative output*

squaring shallow MMs



$$c^2(\mathbf{X}) = \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2$$
$$= \sum_{i=1}^{K}\sum_{j=1}^{K} w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X})$$

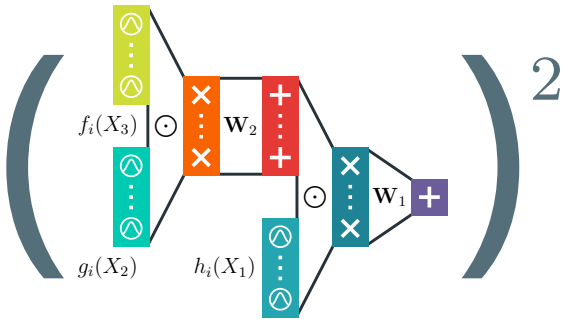
squaring shallow MMs



$$c^2(\mathbf{X}) = \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2 = \sum_{i=1}^{K}\sum_{j=1}^{K} w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X})$$

still a smooth and (str) decomposable PC with $\mathcal{O}(K^2)$ components!

$\Rightarrow$ *but still $\mathcal{O}(K)$ parameters*

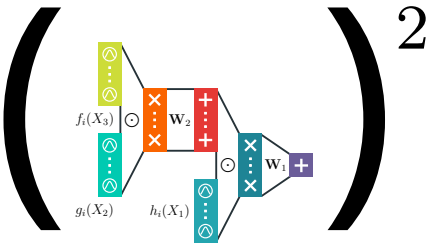


how to efficiently square (and renormalize) a deep PC?

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

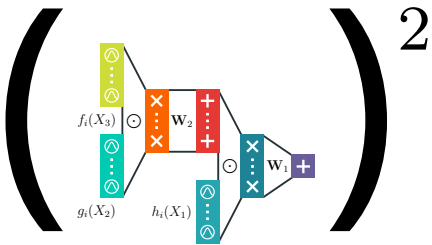
squaring deep PCs

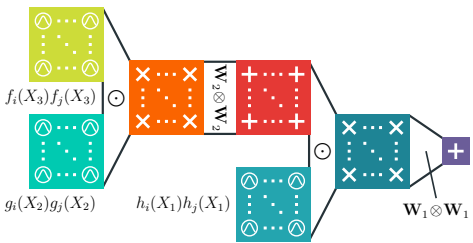
the tensorized way



squaring deep PCs

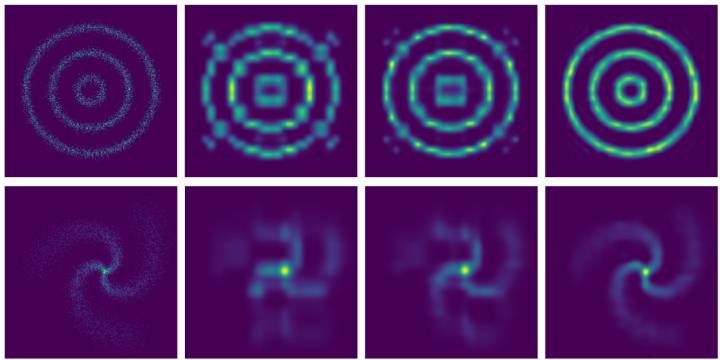
the tensorized way





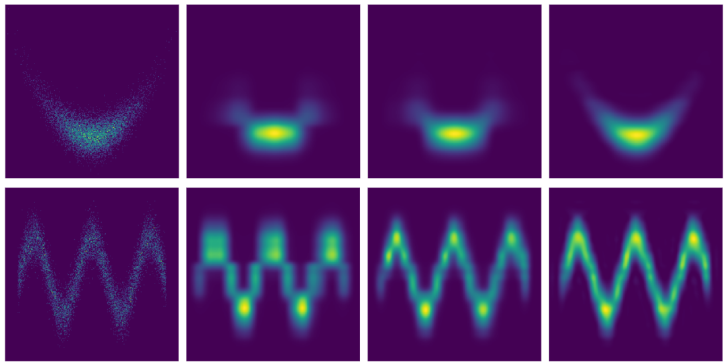
squaring reduces to square layers

more expressive?

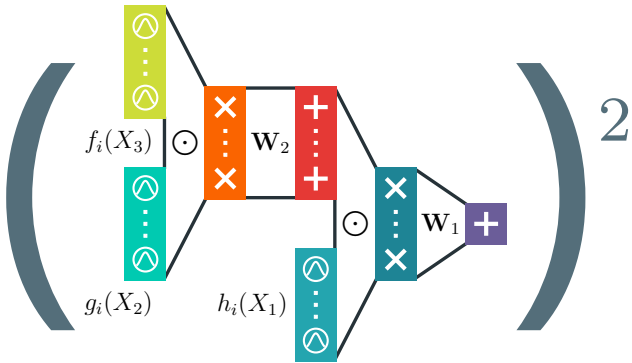


data | monoPC | monoPC^2 | non $-$ monoPC^2

more expressive?



data | monoPC | monoPC^2 | non $-$ monoPC^2



questions?

Hiring, please spread the word!

- 1 PhD **Probabilistic Circuits, Logic and Causality**
  FWF Cluster of Excellence on **Bilateral AI**

  robert.peharz@tugraz.at

- 1 PhD **Probabilistic Machine Learning for Wind Power**

  https://jobs.tugraz.at/en/jobs

- 1 PhD **Probabilistic Machine Learning for Enzyme Design**

  https://jobs.tugraz.at/en/jobs