Uncertainty in Artificial Intelligence
Jul 21, 2025
Rio de Janeiro

Tutorial 1

Principled Hyperparameter Optimization and Algorithm Selection

Practical Techniques, Theory, and New Frontiers

Dravyansh (Dravy) Sharma
IDEAL Postdoctoral Researcher
(joint TTIC+Northwestern)

Roadmap

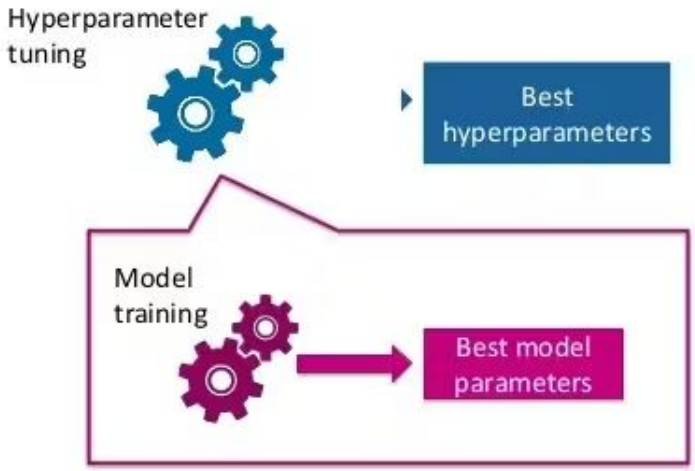
- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

Roadmap

- **Algorithm design for machine learning (aka HP tuning)**
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

What is a hyperparameter?

HP tuning is a special case of algorithm selection in Machine Learning



Why so common in ML? Hard problems + role of data

Hyperparameter tuning and transfer

HP tuning is important across ML

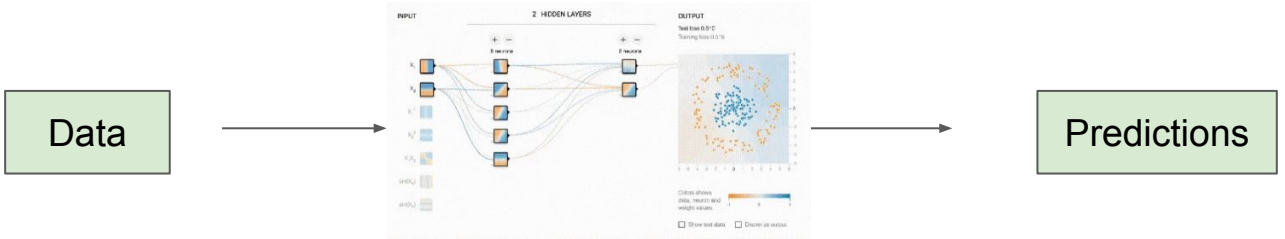
- Data prep + HP tuning take up most of the applied ML researcher hours
- Takes up to 90% of the compute
- Critical in high-stakes and large-scale applications

HP transfer is crucial today!

- Unavoidable in LLMs where each of the above is magnified multifold!

Algorithm design for machine learning

- **Hyperparameter tuning** is poorly understood and yet of critical importance
  - why? ML works on data



Machine learning

  - There is NO single best algorithm+hyperparameter!
  - Must tune/configure for the best performance on **domain-specific data**

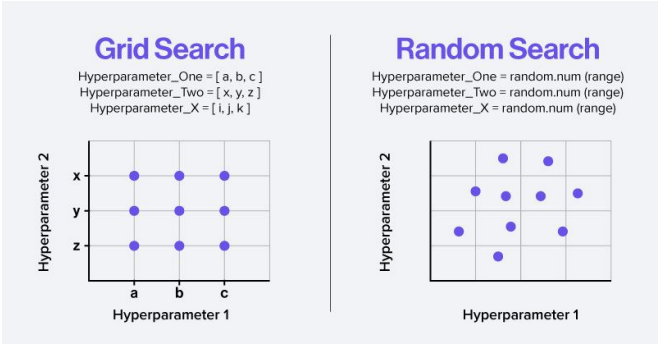
- Current practices require incredible amounts of **compute** and **engineering** efforts, and yet with no guarantees!
- Understanding how the performance actually varies with the hyperparameter is crucial for **principled tuning**

Roadmap

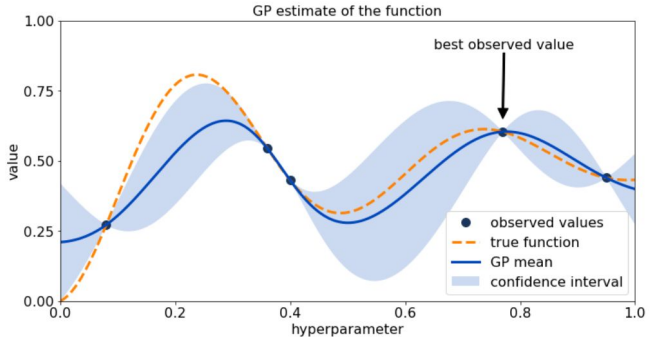
- Algorithm design for machine learning (aka HP tuning)
- **Current approaches in practice**
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

Existing approaches and their (theoretical) limitations

- Manual tuning, grid search, random search:
  - inefficient
  - unprincipled
  - no transfer across tasks
  - data-independent grids can be highly suboptimal
    [Balcan et al. BNVW (COLT'17), BDDKSV (JACM'24)]



- State-of-the-art:
  - Bayesian Optimization (BO);
    [e.g. Snoek et al. 2012]
  - Gradient-based;
  - Bandit-based



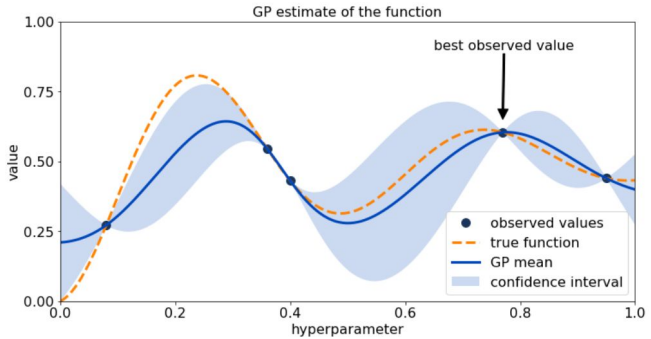
Gap: Limited <u>theoretical understanding</u>,
no guarantees for tuning <u>continuous hyperparameters</u>,
typically <u>no transfer</u> across tasks

But how does the model performance depend on its hyperparameters?

- **Short answer**: we don't really understand it!

- BO works with a crude approximation: Noisy evaluation of function with certain smoothness properties?
  - But how do we know what are the right smoothness priors?
  - Assumptions needed on noise correlations (kernel function)
  - How to search? (acquisition fns)

- But what is the **actual dependence**? Even on a fixed data instance?



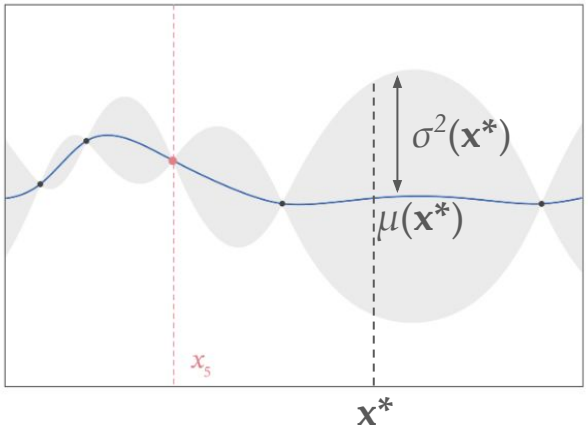
Bayesian Optimization

- **Gaussian Process**:
  - a collection of (infinitely many) random variables that are jointly Gaussian.
  - a distribution over functions – models noisy evaluation of some $f(\mathbf{x})$.
  - given by a mean function $m(\mathbf{x})$ and covariance $k(\mathbf{x}, \mathbf{x}')$.

$$\mathrm{E}[f(\mathbf{x})] = m(\mathbf{x}).$$
$$\mathrm{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] = k(\mathbf{x}, \mathbf{x}').$$

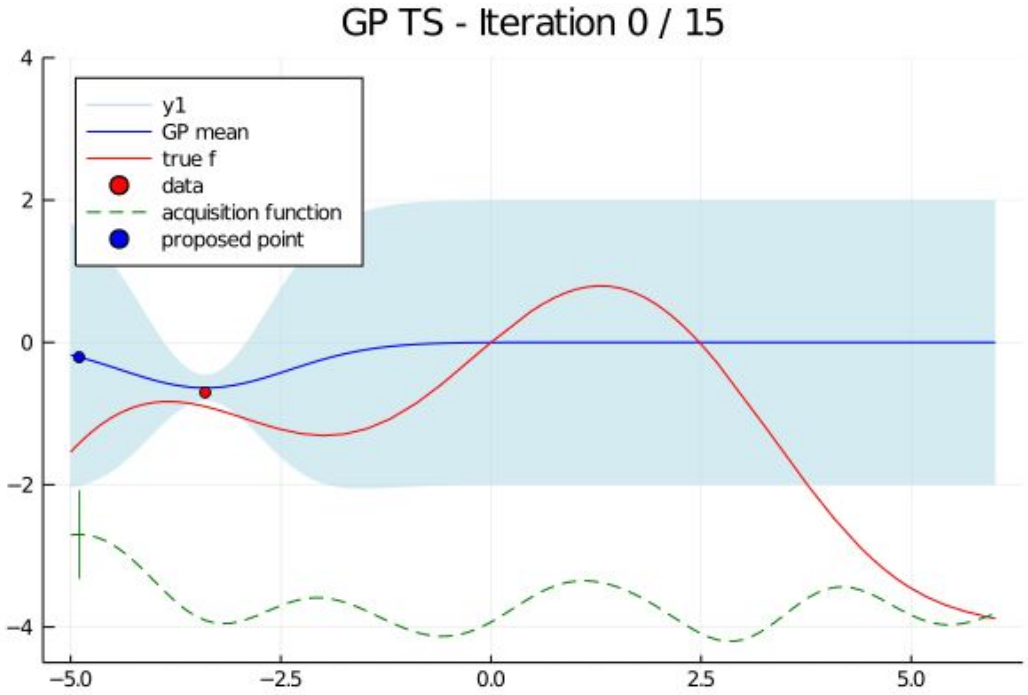
- Since all finite collections of function values are assumed jointly Gaussian, the conditional distribution of any new point given the observed points is also Gaussian, i.e. distribution of mean and variance at $\mathbf{x}^*$, given observed points $\mathbf{X}$ is

$$\mu(\mathbf{x}^*) = K(\mathbf{x}^*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}f(\mathbf{X}).$$
$$\sigma^2(\mathbf{x}^*) = K(\mathbf{x}^*, \mathbf{x}^*) - K(\mathbf{x}^*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{x}^*).$$



$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5]$$

Bayesian Optimization



[Timothy Wolodzko github]

BO has its own hyperparameters!

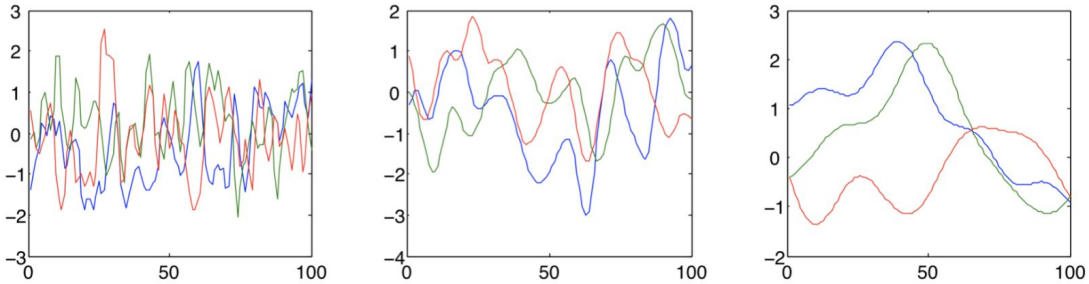


Figure 2: Random functions f drawn from a Gaussian process prior with a power exponential kernel. Each plot corresponds to a different value for the parameter α_1, with α_1 decreasing from left to right. Varying this parameter creates different beliefs about how quickly $f(x)$ changes with x.

[A Tutorial on Bayesian Optimization, Frazier 2018]

Exception [*Berkenkamp, Schoellig, Krause JMLR 2019*] ***But very slow convergence!***

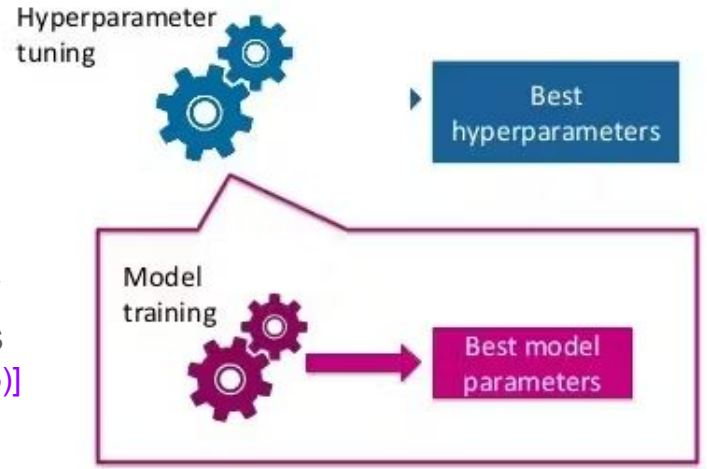
Gradient-based approaches

- **Gradient descent (and other gradient-based iterative optimization)**:
  - fundamental algorithm used across deep learning
  - typically used to train the model parameters e.g. neural network weights
  - gradient of loss w.r.t. parameters computed using chain-rule (aka back-propagation or Reverse-Mode Differentiation) [E.g. LeCun et al. (1989)]

- **Stochastic gradient descent**
  - computes gradient of "*one datapoint*" at a time

- "Stochastic **hypergradient** descent"
  - gradient of "*validation-loss*" w.r.t. hyperparameter
  - Usual too slow, but there are computational tricks

[Bengio (2000), Baydin & Pearlmutter (2014), Maclaurin et al. (2015)]

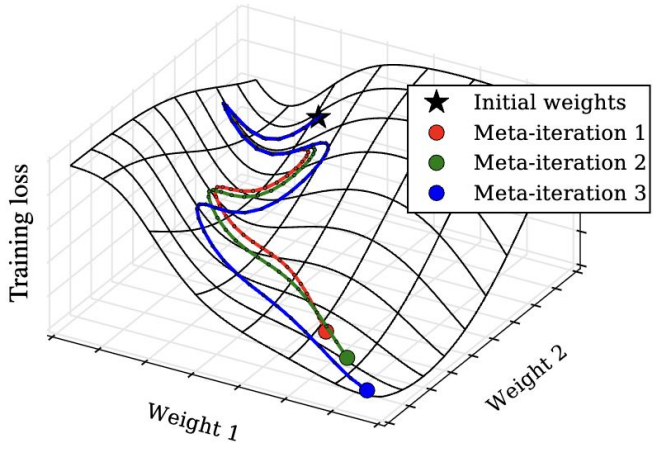


Gradient-based approaches

- "Stochastic **hypergradient** descent"
  - gradient of "*validation-loss*" w.r.t. hyperparameter
  - Usual too slow, but there are computational tricks

- Extension to multiple tasks (meta-learning)
  - MAML (Model-Agnostic Meta-Learning) [Finn et al. (2017)]
  - Online Meta-Learning [Khodak et al. (2019)]

- Unification [HPO + Meta-learning] [Franceschi et al. (2018)]

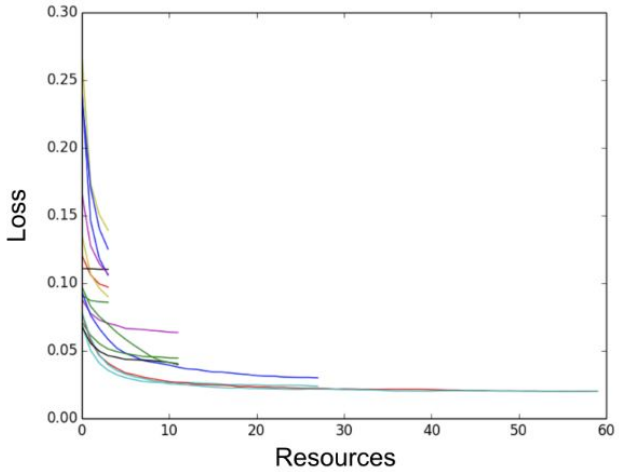


[Maclaurin et al. (2015)]

Bandit-based approaches

Essentially bandit problems with additional HP-specific assumptions

1. **Hyperband**: Each arm has a noisy non-stationary reward that eventually converges to a limiting value [Li, Jamieson, DeSalvo, Rostamizadeh, & Talwalkar (JMLR 2018)]

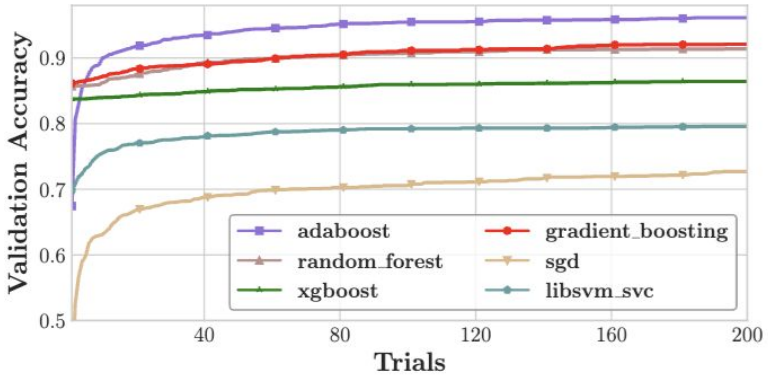


Bandit-based approaches

Essentially bandits problems with additional HP-specific assumptions

2. **Rising/improving bandits**: Arms have concave “learning curves”

[Heidari, Kearns, Roth (IJCAI 2016), Li et al. (AAAI 2020), Metelli et al. (ICML 2022), Mussi et al. (ICML 2024), Blum and Ravichandran (ALT 2025)]



Known guarantees (and lack thereof)

All approaches are black-box!!
(agnostic to structure)

Bayesian optimization

– Guarantees typically need strong prior assumptions

– Need design of kernels (with hyperparameters) and acquisition functions

Guarantees e.g. for GP-UCB assume you can magically do this!

[Srinivas, Krause, Kakade, Seegar (2010)]

Gradient-based methods

– Global optimality typically needs unrealistic convexity/smoothness assumptions

Bandit-based methods

– Guarantees typically only over a finite subset of hyperparameter values (arms)

Hyperparameter tuning and its challenges

- Deep Learning: is powerful, “automated”, and has revolutionized machine learning …

  but a major bottleneck for true automation: need for **extensive** hyperparameter tuning!

- Tedious engineering effort + expensive computational resources

- Typical approaches have **limited** theoretical guarantees and completely black-box!

$$$
manual tuning,
or
poor heuristics

Truly automatic,
provably good!

Roadmap

- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- **Machine learning for algorithm design**
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

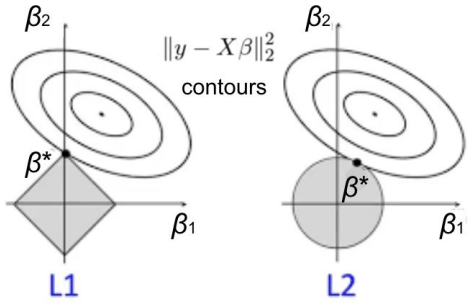
Data-driven algorithm design

[Gupta and Roughgarden, 2016] [Balcan, 2020] [Sharma, 2024]

★ **Algorithm families** occur frequently in machine learning

- Often as tunable “hyperparameters”
- One could smoothly “interpolate” good heuristics

Regularized linear regression



(sparse)

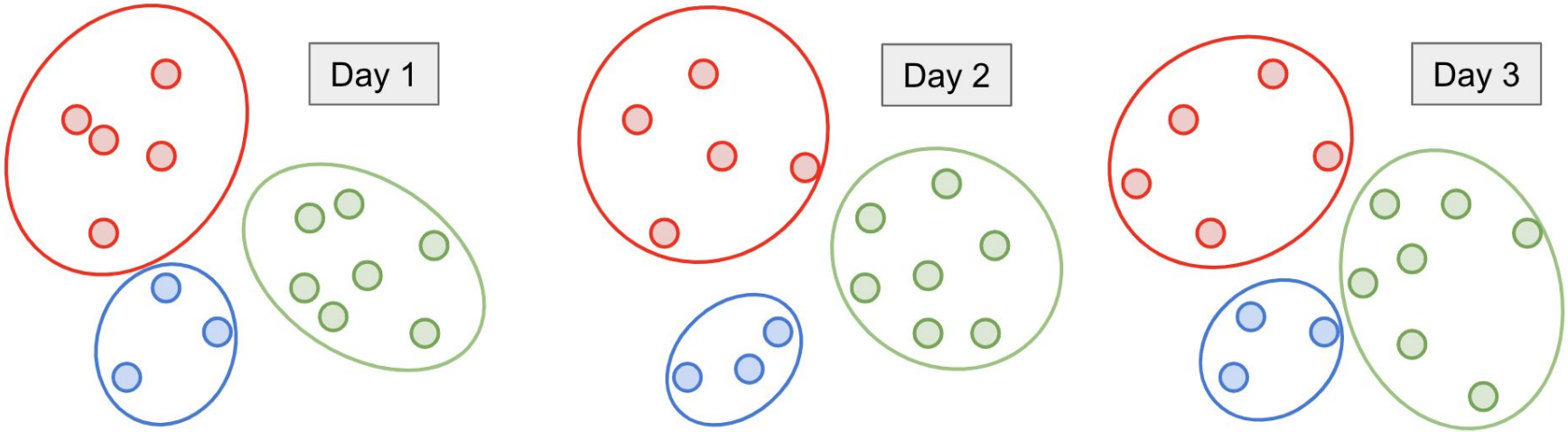
(handles overparameterization, multicollinearity well)

Interpolate: elastic net (best of both worlds!)

Data-driven algorithm design [GR16, Bal20, Sha24]

★ **Repeated problems** from the same problem domain

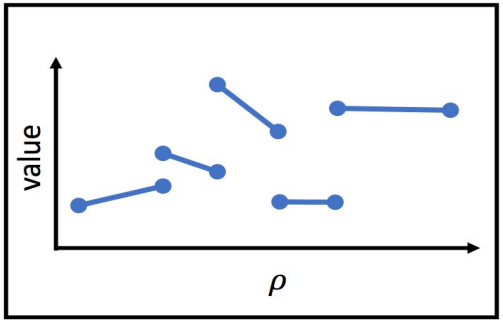
- Expected with regular use of ML
- May come *randomly* (optimistic) or in an *adversarial sequence* (pessimistic)

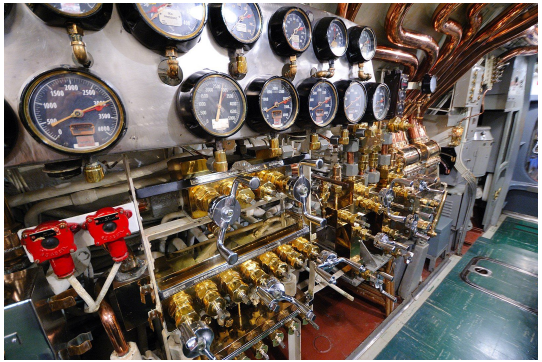


Data-driven algorithm design [GR16, Bal20, Sha24]

★ **Technical challenges**:

- Algorithms form an interesting “concept space”
- *Sharp transition boundaries* in optimization objective
- Particularly tricky to handle multiple “hyperparameters”





Data-driven algorithm design [GR16, Bal20, Sha24]

- Instead of tuning for one specific problem, we tune the **hyperparameter that generalizes** across **a collection of related problems**.
- Concretely:
  - x is a **problem instance** from a **problem set** $\mathcal{X}$, our (infinite) **algorithm family** A
  - D is a **problem distribution** over $\mathcal{X}$, representing the *application-specific domain*
  - We also study no-regret **online learning**, where instances arrive in a sequence
- E.g., academic email spam filter for Gmail, or electronic products sold on Amazon

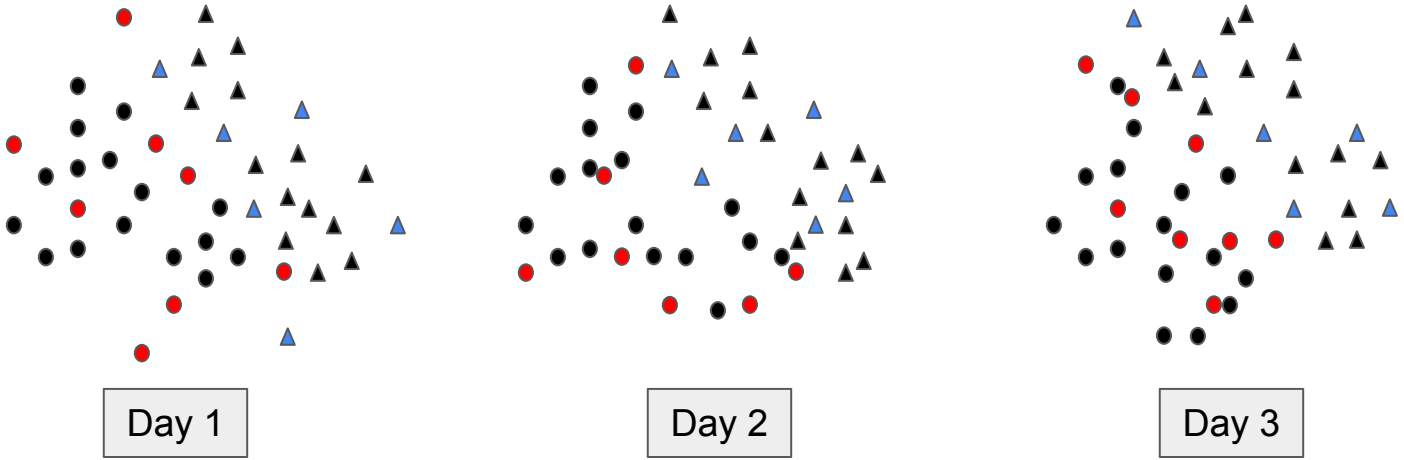
Example: Semi-Supervised Learning

[Balcan and Sharma (2021)];
Oral (55/9122, top 0.6%) at NeurIPS'2021

★ Repeated problems e.g. emails on an email server, spam vs. non-spam

Goal: learn how to connect points using a graph s.t. a (soft) min-cut yields accurate predictions

- **statistical learning**: tight upper+lower bounds on learning-theoretic complexity
- **online learning**: primal-dual style algorithms achieve no regret, under mild assumptions



Example: Decision Trees [Balcan and Sharma, UAI 2024 **Outstanding Student Paper Award**]

Tuning different aspects of decision tree learning

- **Splitting criterion** (which node to split when building the tree?)
  - A **novel algorithmic family** which unifies entropy, Gini impurity and Kearns-Mansour criterion
  - Sample complexity of selecting best splitting algorithm
- **Bayesian methods** (Parameters to select initial tree skeleton)
- **Pruning** (Deleting nodes to avoid overfitting)
- **Interpretability** (Adding tree size to cost with tunable parameter)

Roadmap

- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - **Learning-theoretic foundations**
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

Primal and dual utility functions

- Denote **input instance space** $\boldsymbol{\mathcal{X}}$ and **Hyperparameter space** $\boldsymbol{A}$

- Utility (performance) on any instance for any hyperparameter are given by a function:

$$u(\boldsymbol{x}, \alpha) : \boldsymbol{\mathcal{X}} \times \boldsymbol{A} \rightarrow [0, H]$$

- Primal utility function class:

$$U = \{u_{\alpha} : \boldsymbol{\mathcal{X}} \rightarrow [0, H] \mid \alpha \in A\}$$

- Dual utility function class:

$$U^* = \{u^*_{x} : A \rightarrow [0, H] \mid x \in \boldsymbol{\mathcal{X}}\}$$

Statistical learning theory: sample complexity and pseudo-dimension

Given $\varepsilon > 0$ and $0 < \delta < 1$, what is the sample complexity $m(\varepsilon, \delta)$?

- Standard PAC-Learning approach: bound the learning-theoretic complexity of U

$$U = \{u_\alpha : \boldsymbol{\mathcal{X}} \to [0, H] \mid \alpha \in A\}$$

- Complexity measure: pseudo-dimension, Pdim(U)
  - The maximum size n such that U can "shatter" $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\}$, using thresholds $\boldsymbol{t_1}, \ldots, \boldsymbol{t_n} \in \mathbb{R}$
  - by "shattering", we mean $|\{\text{sign}(u_\alpha(\boldsymbol{x_1}) - \boldsymbol{t_1}), \ldots, \text{sign}(u_\alpha(\boldsymbol{x_n}) - \boldsymbol{t_n}) \mid u_\alpha \in U\}| = 2^n$

- Classical learning theory: If Pdim(U) is finite, then $m(\varepsilon, \delta) = O(H/\varepsilon^2(\text{Pdim}(U) + \log 1/\delta))$

Analogue of VC dimension for real-valued functions

Statistical learning theory: sample complexity and pseudo-dimension

- Simple examples to illustrate pseudo-dimension

Straight lines in 2D, functions $f_{a,b,c}(x, y) = ax + by + c$ for real a, b, c.

$F = \{f_{a,b,c}\}$. $\mathrm{Pdim}(F) = ?$

Answer: 3

Primal and dual utility functions

- So we want to bound the pseudo-dimension of the primal function class U.

- But the structure of U is too complex!
- On the otherwise, it is often easier to establish the structure of the dual class U^*.

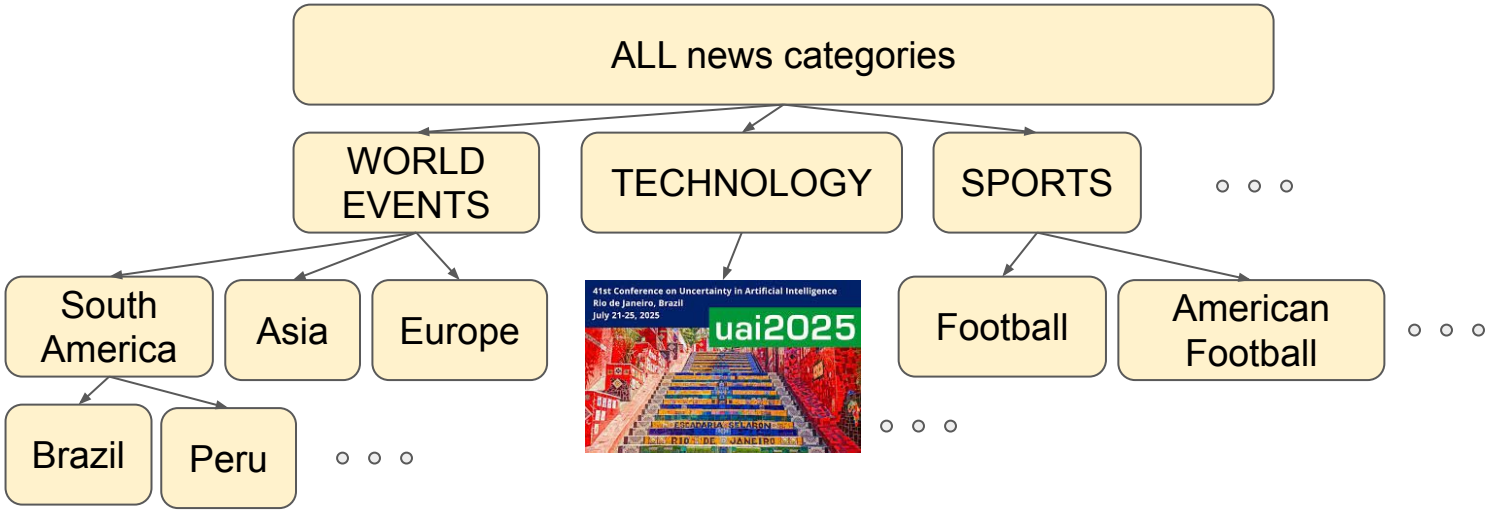
- A general tool (for bounding Pdim of primal using dual structure):

Theorem [BDDKSV STOC'21]: Suppose the dual function class has a piecewise-structure with k boundary functions coming from some function class F^*, and piece functions from class G^*. Then, $\text{Pdim}(U) = \tilde{O}((\text{VC}\tilde{\dim}(F^*) + \text{Pdim}(G^*))\log k)$.

Example: Linkage Clustering [BNVW COLT'17, BSS NeurIPS'24]

Example application: Linkage or hierarchical clustering.

Given a collection of n objects, organize them into hierarchy
e.g. "categories" of news articles

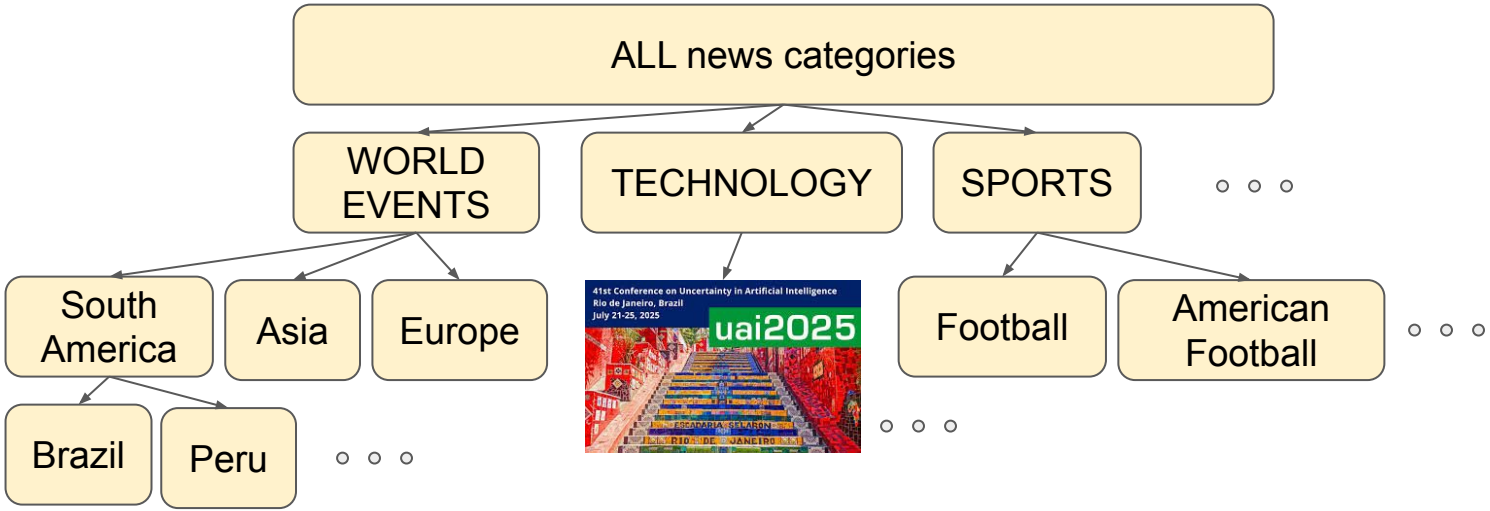


Example: Linkage Clustering [BNVW COLT'17, BS**S** NeurIPS'24]

Example application: Linkage or hierarchical clustering.

Algorithm:

1. Start with each object as its own cluster.
2. Repeatedly merge "most similar" clusters.



Example: Linkage Clustering [BNVW COLT'17, BSS NeurIPS'24]

Example application: Linkage or hierarchical clustering.
Algorithm:

1. Start with each object as its own cluster.
2. Repeatedly merge "most similar" clusters.

But what is "most similar"? Define a notion of distance between cluster pairs:

Single linkage: $D_{\min}(A, B) = \min_{a \in A, b \in B} d(a, b)$
Complete linkage: $D_{\max}(A, B) = \max_{a \in A, b \in B} d(a, b)$

How to tune α?

Interpolate linkage: $D_{\alpha}(A, B) = \alpha D_{\min}(A, B) + (1 - \alpha) D_{\max}(A, B)$

Piecewise constant structure with poly(n) pieces $\Rightarrow$ $\text{Pdim}(U) = O(\log n)$

Combined Algorithm and Hyperparameter Selection [A general tool]

What if we have multiple algorithms each with its own hyperparameters?

Algorithms: $A_1, A_2, \ldots, A_k$
Utility function classes (resp. Hyperparameters): $U_1, U_2, \ldots, U_k$

What is the sample complexity of algo+hyperparameter selection?

Theorem: Sample complexity of CASH is $O(H^2/\varepsilon^2(\log k + \max_i \text{Pdim}(U_i)))$.

[Balcan and Sharma, Arxiv'25]

Roadmap

- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - **GJ algorithm framework**
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

Goldberg-Jerrum (’95) Framework

Another general useful technique for bounding the pseudo-dimension of function classes based on algorithms with *real parameters* that perform *arithmetic operations*.

– Original results yield Pdim bounds in terms of the running time of the algorithm.

– The corresponding bounds are sub-optimal for data-driven algorithm design.

Recent works provide refined GJ frameworks for data-driven algorithm design.

[Bartlett, Indyk, Wagner, COLT’22], [Balcan, Nguyen, Sharma, TMLR’25]

Refined GJ Framework [Bartlett, Indyk, Wagner, COLT'22]

Takes in:

n real algorithm parameters

GJ (95) Algorithm

Two types of operations:

(1) Arithmetic (binary): +, –, ×, ÷

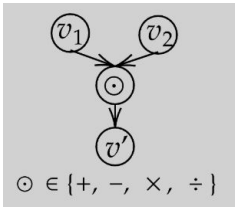
(1) Conditional: if .. then .. else ..

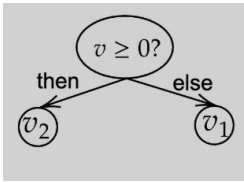
Output(s): E.g.

Cluster,

Matrix,

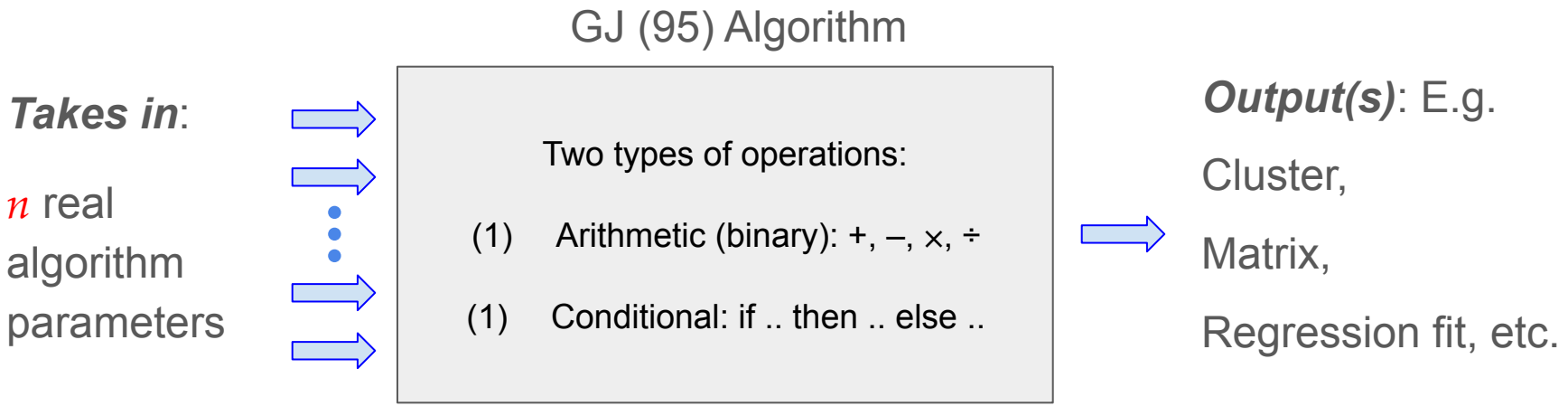
Regression fit, etc.





Note: All expressions computed by the GJ algorithm are rational functions (ratios of polynomials) of its inputs

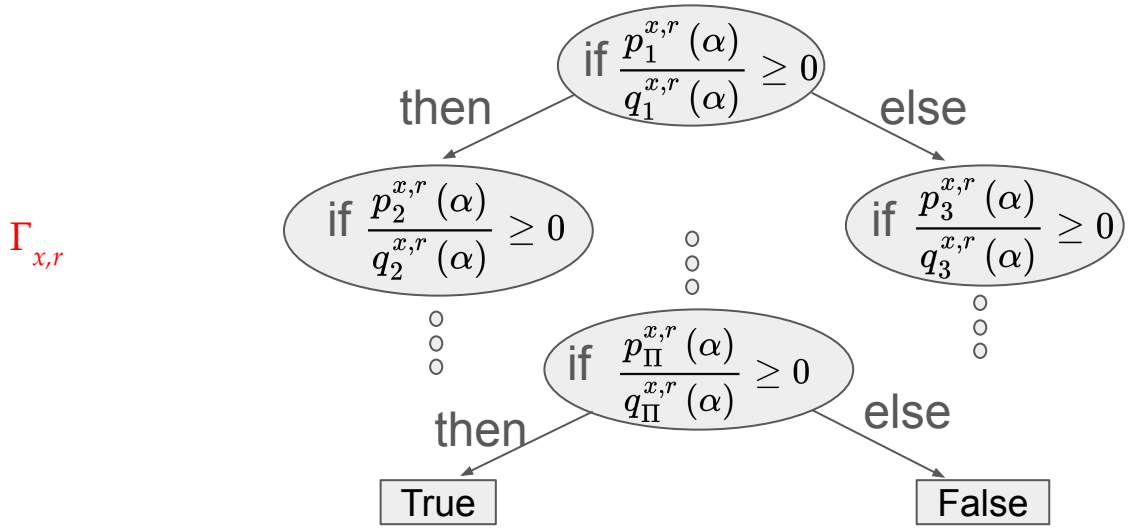
Refined GJ Framework [Bartlett, Indyk, Wagner, COLT'22]



Theorem: Suppose the algorithm family has n real parameters. Also, for any problem instance x and real threshold r, there is a GJ algorithm $\Gamma_{x,r}$ that determines whether $u_\alpha(x) \geq r$ by evaluating at most $\mathbf{\Pi}$ distinct predicates (rational expressions) with maximum degree Δ. Then,

$$\mathrm{Pdim}(U) = O(n \log(\Delta \mathbf{\Pi})).$$

Refined GJ Framework [Bartlett, Indyk, Wagner, COLT'22]



Π : # distinct expressions

Δ : max degree of all p's and q's

Theorem: Suppose the algorithm family has n real parameters. Also, for any problem instance x and real threshold r, there is a GJ algorithm $\Gamma_{x,r}$ that determines whether $u_\alpha(x) \geq r$ by evaluating at most Π distinct predicates (rational expressions) with maximum degree Δ. Then,

$$\text{Pdim}(U) = O(n \log(\Delta\Pi)).$$

Refined GJ Framework [Balcan, Goyal, Sharma, Arxiv'25]

Example application: Tuning the ridge penalty λ in linear regression.

$$\min_w ||Xw - y||^2 + \lambda||w||^2$$

Input: Training data X, y and validation data X', y'.
Goal: Tune λ to minimize validation loss.

Applying GJ framework: Note that the ridge solution is $w_\lambda = (X^TX + \lambda I)^{-1}X^Ty$.

Lemma: $w_\lambda = (X^TX + \lambda I)^{-1}X^Ty$ is a rational function of lambda with degree at most d (#features).

⇒ Validation loss is a rational function with degree at most $2d$.
⇒ GJ algorithm to check $u_\lambda(x) \geq r$ has degree $2d$ and predicate complexity 1.

Theorem: Sample complexity of tuning λ is $O(\log(d)/\varepsilon^2)$.

Refined GJ Framework [Bartlett, Indyk, Wagner, COLT'22]

Example application: Low-rank approximation.

Input: Given a sparse matrix $A \in \mathbb{R}^{n \times n}$ with $||A||_F = 1$, target rank $k < n$.
Goal: Sparse matrix $\tilde{A}$ with rank k that minimizes (approximates A well).

Exact algorithm based on SVD (singular value decomposition) is inefficient!

Faster algorithm IVY [Indyk, Vakilian, Yuan '19] is family of parameterized heuristics uses a m x n auxiliary matrix (runtime nearly linear in #non-zero entries!).

Theorem: Sample complexity of tuning IVY is $O(mn/\varepsilon^2)$.

Pfaffian functions

Pfaffian function chain: A sequence of multivariate functions $f_1, f_2, \ldots, f_q$ with arguments $a_1, \ldots, a_n$, if all partial derivatives can be expressed via polynomials of the arguments or previous functions in the chain, i.e.

$$\frac{\partial f_j}{\partial a_i} = P_{i,j}\left(a_1, \ldots, a_n, f_1, \ldots, f_j\right)$$

Pfaffian function: Polynomial fn of the Pfaffian chain $Q\left(a_1, \ldots, a_n, f_1, \ldots, f_q\right)$

Chain length, q: number of functions in the sequence

Pfaffian degree, M: Maximum degree of a derivative polynomials

Degree, Δ: Maximum degree of a polynomial of a chain of Pfaffian functions, Q

Pfaffian functions

Examples:

1. $e^{2a} + a^3$: Chain length ? Pfaffian degree ? degree ?

2. $\log \sqrt{a}$: Chain length ? Pfaffian degree ? degree ?

3. $a^{1/2} + a^{2/3}$: Chain length ? Pfaffian degree ? degree ?

Pfaffian functions

Pfaffian function chain: A sequence of multivariate functions $f_1, f_2, \ldots, f_q$ with arguments $a_1, \ldots, a_n$, if all partial derivatives can be expressed via polynomials of the arguments or previous functions in the chain, i.e. $\frac{\partial f_j}{\partial a_i} = P_{i,j}(a_1, \ldots, a_n, f_1, \ldots, f_j)$

Pfaffian function: Polynomial fn of the Pfaffian chain $Q(a_1, \ldots, a_n, f_1, \ldots, f_q)$

Chain length, q: number of functions in the sequence

Pfaffian degree, M: Maximum degree of a derivative polynomials

Degree, Δ: Maximum degree of a polynomial of a chain of Pfaffian functions, Q

1. $e^{2a} + a^3$: Chain length ? Pfaffian degree ? degree ?

Pfaffian functions

1. $e^{2a} + a^3$: Chain length ? Pfaffian degree ? degree ?

   $f_1(a) = e^{2a} + a^3$; $f_1'(a) = 2e^{2a} + 3a^2 = 2f_1(a) + 3a^2 = P(a, f_1(a))$; $Q(a, f_1(a)) = f_1(a)$

   Chain length = 1, Pfaffian degree = 2, degree = 1

   $f_1(a) = e^a$; $f_1'(a) = f_1(a) = P(a, f_1(a))$; $Q(a, f_1(a)) = (f_1(a))^2 + a^3$

   Chain length = 1, Pfaffian degree = 1, degree = 3

Pfaffian functions

Pfaffian function chain: A sequence of multivariate functions $f_1, f_2, \ldots, f_q$ with arguments $a_1, \ldots, a_n$, if all partial derivatives can be expressed via polynomials of the arguments or previous functions in the chain, i.e. $\frac{\partial f_j}{\partial a_i} = P_{i,j}(a_1, \ldots, a_n, f_1, \ldots, f_j)$

Pfaffian function: Polynomial fn of the Pfaffian chain $Q(a_1, \ldots, a_n, f_1, \ldots, f_q)$

Chain length, q: number of functions in the sequence

Pfaffian degree, M: Maximum degree of a derivative polynomials

Degree, Δ: Maximum degree of a polynomial of a chain of Pfaffian functions, Q

2. $\log \sqrt{a}$: Chain length ? Pfaffian degree ? degree ?

Pfaffian functions

2. $\log \sqrt{a}$: Chain length ? Pfaffian degree ? degree ?

$f_1(a) = \log a$; $f_1'(a) = 1/a$

Not a polynomial in $\log a$ and a!

$f_1(a) = 1/a$; $f_2(a) = \log a$;

$f_1'(a) = -a^{-2} = P(a, f_1(a))$; $f_1'(a) = 1/a = P(a, f_1(a), f_2(a))$; $Q(a, f_1(a)) = \frac{1}{2} f_2(a)$

Chain length = 2, Pfaffian degree = 2, degree = 1

Pfaffian functions

Pfaffian function chain: A sequence of multivariate functions $f_1, f_2, \ldots, f_q$ with arguments $a_1, \ldots, a_n$, if all partial derivatives can be expressed via polynomials of the arguments or previous functions in the chain, i.e. $\frac{\partial f_j}{\partial a_i} = P_{i,j}(a_1, \ldots, a_n, f_1, \ldots, f_j)$

Pfaffian function: Polynomial fn of the Pfaffian chain $Q(a_1, \ldots, a_n, f_1, \ldots, f_q)$

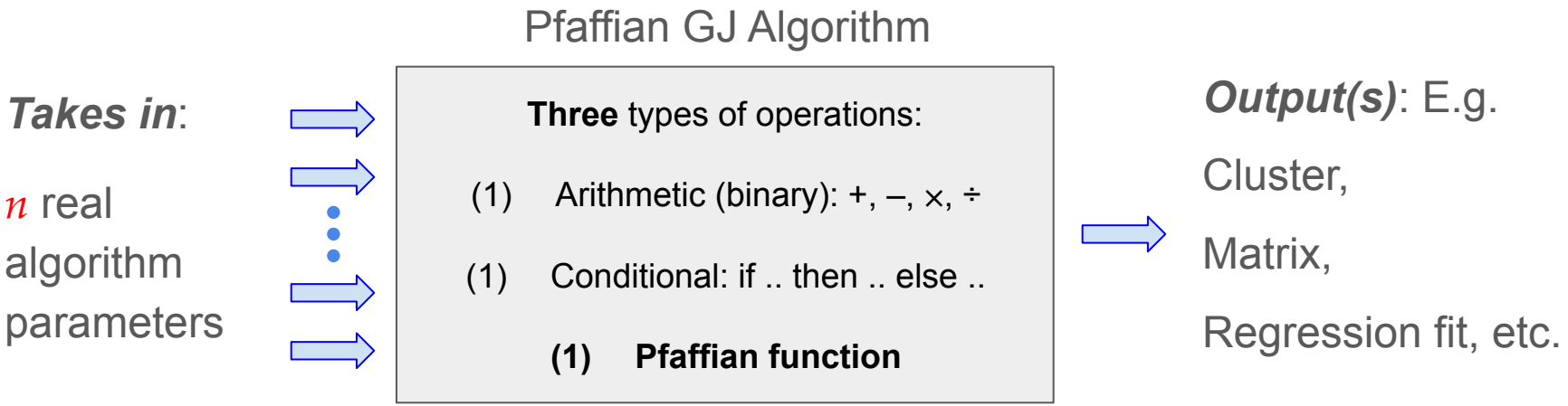
Chain length, q: number of functions in the sequence

Pfaffian degree, M: Maximum degree of a derivative polynomials

Degree, Δ: Maximum degree of a polynomial of a chain of Pfaffian functions, Q

3. $a^{1/2} + a^{2/3}$: Chain length ? Pfaffian degree ? degree ?

Pfaffian GJ Framework [Balcan, Nguyen, Sharma (TMLR 2025)]



Theorem: Suppose the algorithm family has n real parameters. Also, for any problem instance x and real threshold r, there is a **Pfaffian** GJ algorithm $\Gamma_{x,r}$ that determines whether $u_\alpha(x) \geq r$ by evaluating Π distinct predicates with Pfaffian chain length q, degree Δ, and Pfaffian degree M. Then,

$$\mathrm{Pdim}(\mathcal{U}) = O\left(n^2q^2 + nq\ln(\Delta + M) + n\ln\Pi\right)$$

Pfaffian GJ Framework Example: Linkage Clustering [BNS TMLR'25]

Algorithm:

1. Start with each object as its own cluster.
2. Repeatedly merge "most similar" clusters.

But what is "most similar"? Define a notion of distance between cluster pairs:

Single linkage: $D_{\min}(A, B) = \min_{a \in A, b \in B} d(a, b)$

Complete linkage: $D_{\max}(A, B) = \max_{a \in A, b \in B} d(a, b)$

Also, what if we have multiple distances $d_1, d_2, \ldots, d_L$?

How to tune α, $\boldsymbol{\beta}$?

1. Interpolate distances: $d_{\boldsymbol{\beta}} = \beta_1 d_1 + \beta_2 d_2 + \ldots + \beta_L d_L$
2. Interpolate linkage: $D_{\alpha,\boldsymbol{\beta}}(A, B) = (\min_{a \in A, b \in B} (d_{\boldsymbol{\beta}}(a, b))^{\alpha} + \min_{a \in A, b \in B} (d_{\boldsymbol{\beta}}(a, b))^{\alpha})^{1/\alpha}$

Pfaffian GJ Framework Example: Linkage Clustering [BN**S** TMLR'25]

Algorithm:

1. Start with each object as its own cluster.
2. Repeatedly merge "most similar" clusters.

$$D_{\alpha,\boldsymbol{\beta}}(A, B) = (\min_{a \in A,\, b \in B} (d_{\boldsymbol{\beta}}(a, b))^{\alpha} + \min_{a \in A,\, b \in B} (d_{\boldsymbol{\beta}}(a, b))^{\alpha})^{1/\alpha}$$

The algorithm uses exponents:
so arithmetic operations not enough to compute the clusters!
But Pfaffian GJ framework applies!

Theorem: Sample complexity of tuning α, $\boldsymbol{\beta}$ is $O(n^4 L^2/\varepsilon^2)$.

Pfaffian GJ Framework Example: Linkage Clustering [BNS TMLR'25]

Algorithm:

1. Start with each object as its own cluster.
2. Repeatedly merge "most similar" clusters.

What are the Pfaffian chains?

$$\mathrm{D}_{\alpha,\boldsymbol{\beta}}(A, B) = (\min_{a \in A,\, b \in B} (\mathrm{d}_{\boldsymbol{\beta}}(a, b))^{\alpha} + \min_{a \in A,\, b \in B} (\mathrm{d}_{\boldsymbol{\beta}}(a, b))^{\alpha})^{1/\alpha}$$

Merge decisions are governed by boundaries given by following inequation in α, $\boldsymbol{\beta}$

$$\mathrm{D}_{\alpha,\boldsymbol{\beta}}(A, B) \gtrless \mathrm{D}_{\alpha,\boldsymbol{\beta}}(A', B')$$

for some clusters A, B, A', B'

Equivalently, the boundaries are given by (at most n^8 equations)

$$(\mathrm{d}_{\boldsymbol{\beta}}(a_1, b_1))^{\alpha} + (\mathrm{d}_{\boldsymbol{\beta}}(a_2, b_2))^{\alpha} - (\mathrm{d}_{\boldsymbol{\beta}}(a_3, b_3))^{\alpha} - (\mathrm{d}_{\boldsymbol{\beta}}(a_4, b_4))^{\alpha} \gtrless 0$$

for some points $a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4$

Pfaffian GJ Framework Example: Linkage Clustering [BNS TMLR'25]

$$(\mathrm{d}_{\boldsymbol{\beta}}(a_1, b_1))^{\alpha} + (\mathrm{d}_{\boldsymbol{\beta}}(a_2, b_2))^{\alpha} - (\mathrm{d}_{\boldsymbol{\beta}}(a_3, b_3))^{\alpha} - (\mathrm{d}_{\boldsymbol{\beta}}(a_4, b_4))^{\alpha} \gtrless 0$$

What are the Pfaffian chains?

For each pair of points (a, b), define 3 functions

$$f_{a,b}(\boldsymbol{\beta}) = 1/\mathrm{d}_{\boldsymbol{\beta}}(a, b)\ ;$$
$$g_{a,b}(\boldsymbol{\beta}) = \ln \mathrm{d}_{\boldsymbol{\beta}}(a, b)\ ;$$
$$h_{a,b}(\boldsymbol{\beta}) = (\mathrm{d}_{\boldsymbol{\beta}}(a, b))^{\alpha}$$

Chain length < $3\mathrm{n}^2$, degree 1, Pfaffian degree 2
Number of parameter = $L + 1$
Number of distinct predicates < n^8
Our result implies $\mathrm{Pdim}(U) = \mathrm{O}(n^4 L^2)$

Roadmap

- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- **Tuning core ML algorithms**
  - Decision Trees
  - Neural networks
- Other aspects, ongoing and future research

Applications [ML, stats, optimization]

Low-rank approximation [Bartlett, Indyk, Wagner, COLT 2022]

Regularizing linear (Elastic Net) and logistic regression [BK**S**T NeurIPS 2022, BN**S** NeurIPS 2023, BG**S** 2025]

Simulated Annealing [Blum, Dan, Seddighin, AISTATS 2021]

Learning to branch and cut [Balcan, Dick, Sandholm, Vitercik, ICML 2018, JACM 2024]

Clustering (both k-center and hierarchical) [BNVW COLT 2017, BDW NeurIPS 2018, BDL ICLR 2020]

Gradient descent [Gupta and Roughgarden, ITCS 2016]

Integer and Linear Programming [Balcan et al., Khodak et al., Cheng and Basu, Sakaue and Oki (2024)]

More applications [CS theory, Comp bio, Mechanism design …]

Knapsack, Maximum Weighted Independent Set [Gupta and Roughgarden, ITCS 2016, Balcan et al., FOCS 2018]

Max cut, Max 2-SAT [Balcan et al., COLT 2017]

Dynamic Programming, Sequence Alignment [Balcan et al., COLT 2017, STOC 2021, NeurIPS 2024]

Mechanism and game design [Balcan et al., EC18, NeurIPS 2024, Jin et al. NeurIPS 2024]

Open questions and research directions

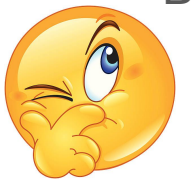
- Provable tuning of hyperparameters in other fundamental algorithms and areas, E.g.
  - Causal inference algorithms
  - Constraint Satisfaction e.g. algorithms for SAT
  - Graph Algorithms
  - Bayesian Optimization itself! (e.g. [Sharma and Suggala (AAAI 25)] tune GP bandits)
  - …
- Computational efficiency and complexity of hyperparameter tuning
- Lower bounds on sample complexity
  - Tight bounds known only in some cases

Roadmap

- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - **Decision Trees**
  - Neural networks
- Other aspects, ongoing and future research

ML needs to be interpretable!

Trustworthy?
Biased?



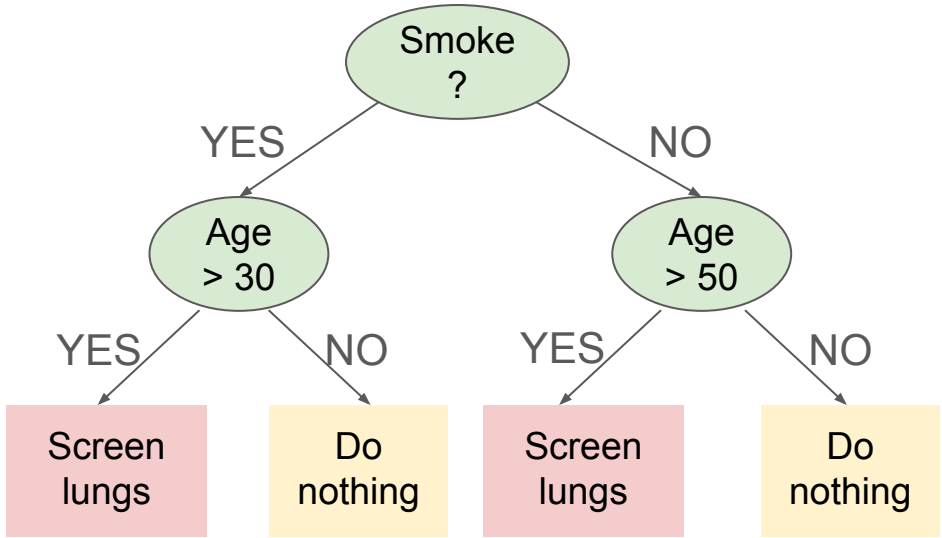
Decision Trees

Trees for classification:

- Each internal node ⇔ Splitting rule
- Each leaf node ⇔ Single Class

Interpretable ML models

- axis-parallel decision boundaries
- Neural nets are hard to interpret



Hard to learn optimal trees, but several useful heuristics!

Learning optimal decision trees is hard!

Hardness of DT learning

- NP-complete. [Laurent and Rivest (1976)]
- *Superconstant Inapproximability of Decision Tree Learning*.

  [Koch et al. COLT 2024] [Koch and Strassle FOCS 2023, FOCS 2024]

Faster optimal decision trees (speed up the exp time branch-and-bound algorithm)

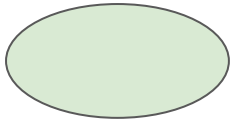
- [Hu et al. NeurIPS 2019]
- [McTavish et al. AAAI 2022]
- [Babbar et al. ICML 2025] (combines greedy with branch-and-bound)

Splitting criterion (a greedy approach)

Top-down decision tree learning

Inputs: Node function class $\mathcal{F}$, tree size **t**,
splitting criterion **G**

Splitting criterion

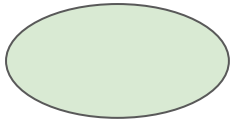


Top-down decision tree learning

Inputs: Node function class $\mathcal{F}$, tree size **t**,
splitting criterion **G**

- Start with leaf node

Splitting criterion



Top-down decision tree learning

Inputs: Node function class $\mathcal{F}$, tree size **t**,
splitting criterion **G**

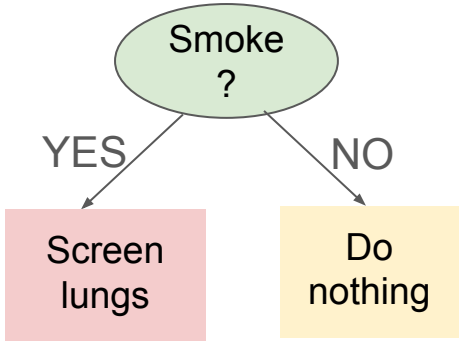
- Start with leaf node
- While at most **t** leaf nodes
  - Split leaf node **l** using node function **f** which maximizes “splitting criterion”

Splitting criterion

Top-down decision tree learning

Inputs: Node function class $\mathcal{F}$, tree size **t**, splitting criterion **G**

- Start with leaf node
- While at most **t** leaf nodes
  - Split leaf node **l** using node function **f** which maximizes “splitting criterion”



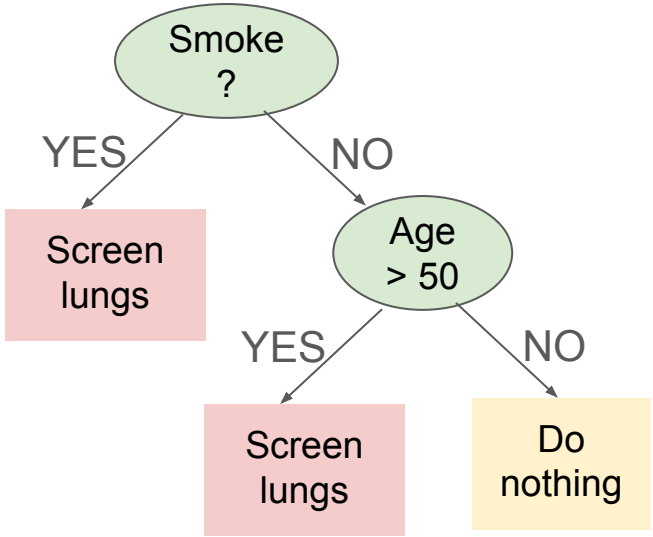
$\mathcal{F}$ = {Smoke, Age >30, Age >50}

Splitting criterion

Top-down decision tree learning

Inputs: Node function class $\mathcal{F}$, tree size **t**, splitting criterion **G**

- Start with leaf node
- While at most **t** leaf nodes
  - Split leaf node **l** using node function **f** which maximizes “splitting criterion”



Key decision: Which node to split next and how?

$\mathcal{F}$ = {Smoke, Age >30, Age >50}

Splitting criterion

DecisionTreeClassifier

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini',
splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None,
ccp_alpha=0.0, monotonic_cst=None)                                    [source]
```

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy", "log_loss"}, default="gini"*

Splitting criterion

Empirical research suggests different criteria work best on different data [Mingers 1989]

- Entropy criterion
- Gini impurity
- Kearns Mansour 96

Algorithm selection via hyperparameter tuning

(α, β)-Tsallis entropy

A single criterion which interpolates all three!

$$g_{\alpha,\beta}^{\text{TSALLIS}}(P) := \frac{C}{\alpha - 1}\left(1 - \left(\sum_{i=1}^{c} p_i^{\alpha}\right)^{\beta}\right)$$

Splitting criterion

$g_{2,1}^{\text{TSALLIS}}(P)$ Gini impurity

$g_{\frac{1}{2},2}^{\text{TSALLIS}}(P)$ KM96

$\lim_{\alpha \to 1} g_{\alpha,1}^{\text{TSALLIS}}(P)$ Entropy

Splitting criterion

$g_{2,1}^{\text{TSALLIS}}(P)$

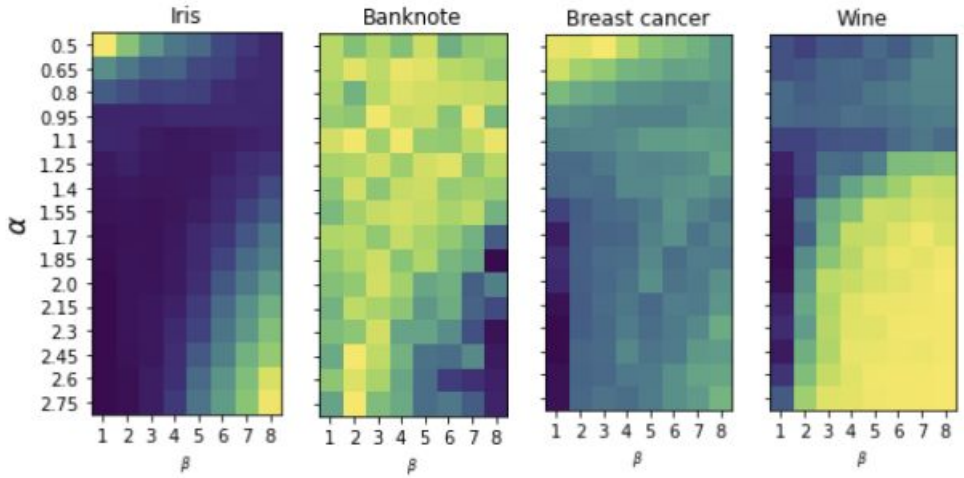
$g_{\frac{1}{2},2}^{\text{TSALLIS}}(P)$

$\lim_{\alpha \to 1} g_{\alpha,1}^{\text{TSALLIS}}(P)$

Gini impurity

KM96

Entropy

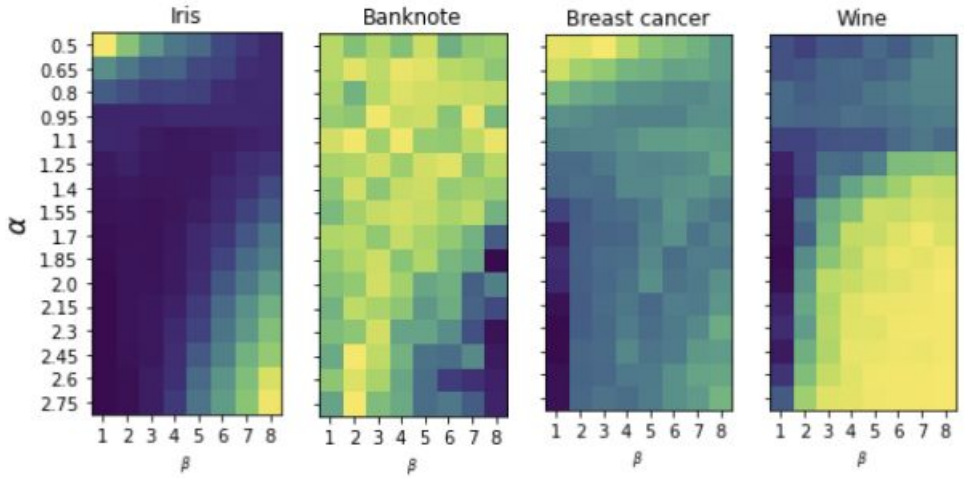


Splitting criterion

$g_{2,1}^{\text{TSALLIS}}(P)$ Gini impurity

$g_{\frac{1}{2},2}^{\text{TSALLIS}}(P)$ KM96

$\lim_{\alpha \to 1} g_{\alpha,1}^{\text{TSALLIS}}(P)$ Entropy



Theorem: We can learn to tune (α, β) using $O\left(\frac{t \log |\mathcal{F}| t}{\epsilon^2}\right)$ problem samples.

Splitting criterion

Theorem: We can learn to tune (α, β) using $O\left(\frac{t \log |\mathcal{F}| t}{\epsilon^2}\right)$ problem samples.

Proof insights:

- Analyse accuracy as a function of (α, β) on a fixed instance (X, y)
- Induction over top-down rounds, bounding the number of distinct behaviors (which node is split and how) in each round
- Over t rounds, $\tilde{O}(|\mathcal{F}|^{2t} t^{2t})$ distinct behaviors, which implies pseudo-dimension is $O(t \log |\mathcal{F}| t)$.

Bayesian trees

The algorithm [Chipman, George, McCulloch 1998]

1. **Prior**:
   a. Start with a single root node
   b. For each node, split it with probability $p_{\text{SPLIT}} = \sigma(1 + d)^{-\varphi}$
   c. Select uniformly random splitting rule at each node if split
   d. Repeat step b for each new node

Bayesian trees

The algorithm [Chipman, George, McCulloch 1998]

1. **Prior**:
   a. Start with a single root node
   b. For each node, split it with probability $p_{\text{SPLIT}} = \sigma(1 + d)^{-\varphi}$
   c. Select uniformly random splitting rule at each node if split
   d. Repeat step b for each new node
2. **Stochastic search**:
   a. T^0 = initial skeleton with random rules according to Prior
   b. $T^* \leftarrow$ obtained by small modification to T^i
   c. $T^{i+1} = T^*$ with probability $q(T^i, T^*)$ based on Dirichlet posterior, $T^{i+1} = T^i$ otherwise

Bayesian trees

The algorithm [Chipman, George, McCulloch 1998]

1. **Prior**:
   a. Start with a single root node
   b. For each node, split it with probability $p_{\text{SPLIT}} = \sigma(1 + d)^{-\varphi}$
   c. Select uniformly random splitting rule at each node if split
   d. Repeat step b for each new node
2. **Stochastic search**:
   a. T^0 = initial skeleton with random rules according to Prior
   b. T^* ← obtained by small modification to T^i
   c. $T^{i+1} = T^*$ with probability $q(T^i, T^*)$ based on Dirichlet posterior, $T^{i+1} = T^i$ otherwise

σ, φ are tunable hyperparameters

Bayesian trees

Goal: Tune σ, φ to maximize expected accuracy of the learned decision tree for datasets sampled according to some distribution D.

Insight: Analyze the structure of the loss as a function of hyperparameters for fixed random bits

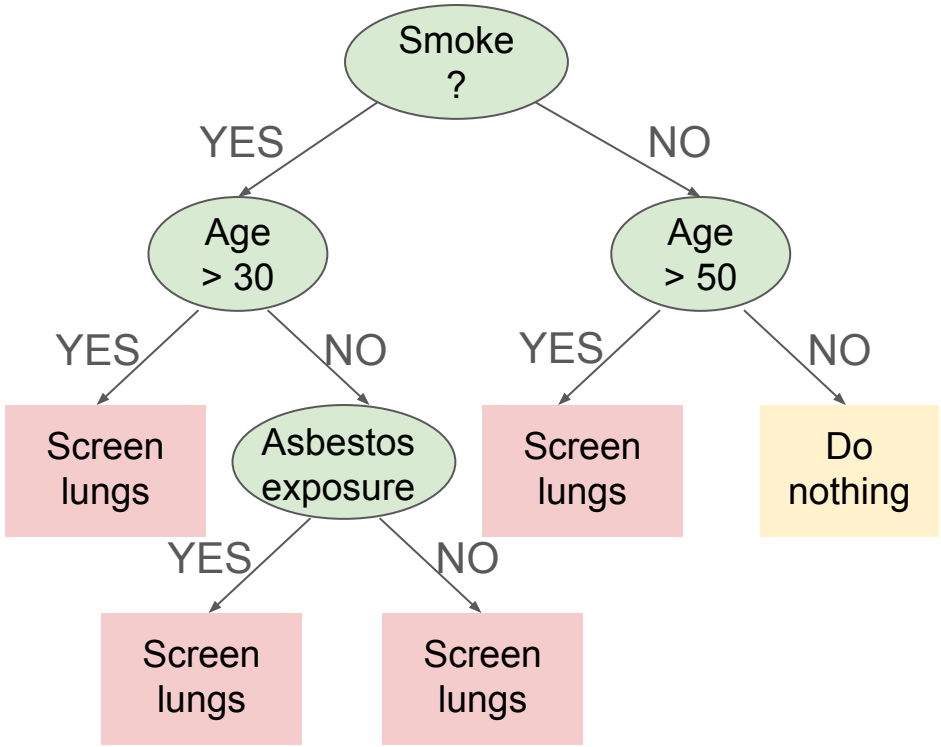
piecewise constant with exponential boundaries and at most t^2N^2 pieces over N problem samples.

Result: $\mathrm{O}(\log t \,/\, \varepsilon^2)$ datasets sampled from D are sufficient to learn near-optimal parameters σ, φ.

Pruning

Post-processing step to simplify tree:

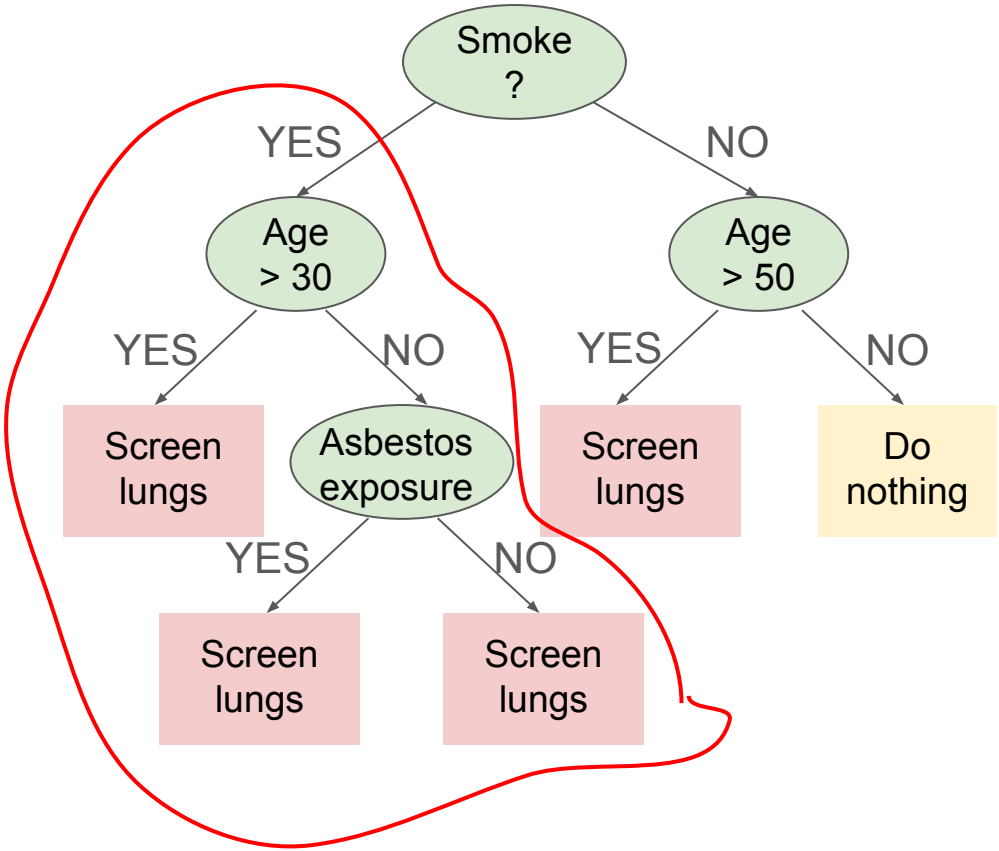
- Reduces overfitting
- Increases interpretability



Pruning

Post-processing step to simplify tree:

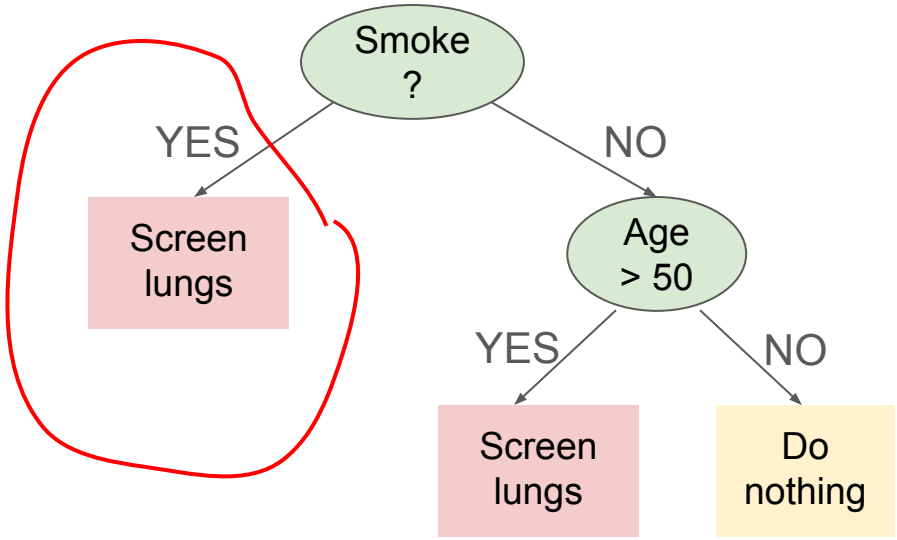
- Reduces overfitting
- Increases interpretability



Pruning

Post-processing step to simplify tree:

- Reduces overfitting
- Increases interpretability



Pruning

Min cost-complexity pruning

- Maximizing accuracy on training set typically leads to large trees
- Add tree size as a penalty term in **training loss**

$$\text{Cost-complexity, } R(T, D) = L(T, D) + \alpha \, |\text{leaves}(T)|$$

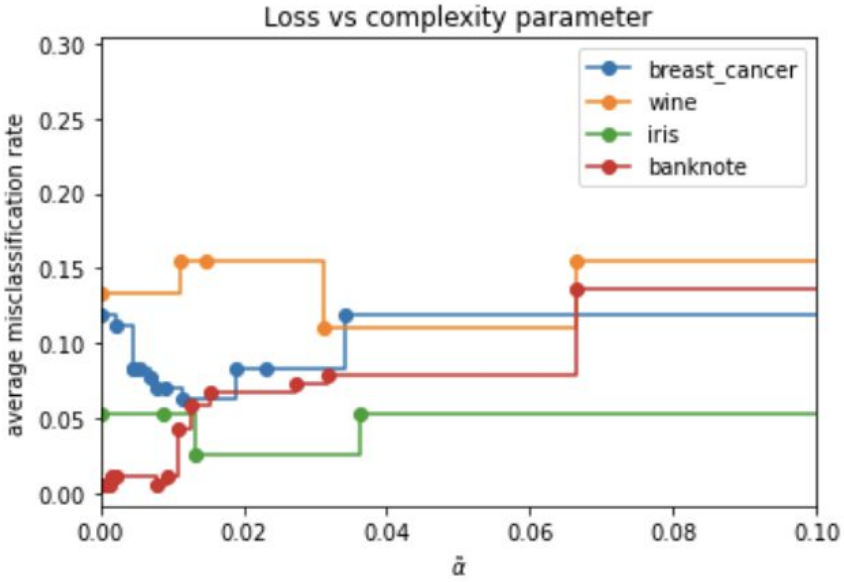
Tunable hyperparameter

Pruning

Best HP is data-specific,

so need to learn!!

Result: $O(\log t / \varepsilon^2)$ datasets sampled from D are sufficient to learn near-optimal α



Interpretability vs accuracy

$$\text{Modified objective, } R(T, D) = L(T, D) + \eta \, |\text{leaves}(T)|$$

Similar to cost-complexity pruning, but also modify **test loss**

- η controls the accuracy-interpretability trade-off
- we tune splitting/pruning hyperparameters simultaneously to maximize the modified objective

Gradient-boosted decision trees

Regularized objective over a collection of K trees (size at most t),

$L(\{T_i\}, D) = l(\{T_i\}, D) + \frac{1}{2} \lambda \sum_k ||\text{weights of leaves in } T_k||^2$

Splitting-criterion in XGBOOST [Chen and Guestrin (2016)]:

- Across all nodes of all trees in the ensemble, split the one that maximizes a score based on first and second order gradients $\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}$

State-of-the-art approach for tabular datasets!
[McElfresh et al. (NeurIPS 2023), Jayawardhana et al. (2025)]

We use a GJ framework based analysis.

Gradient-boosted decision trees

Regularized objective over a collection of K trees (size at most t),

$L(\{T_i\}, D) = l(\{T_i\}, D) + \frac{1}{2} \lambda \sum_k ||\text{weights of leaves in } T_k||^2$

Splitting-criterion in XGBOOST [Chen and Guestrin (2016)]:

- Across all nodes of all trees in the ensemble, split the one that maximizes a score based on first and second order gradients $\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}$

There are at most $tK|\mathcal{F}|$ different candidate splits, or at most $t^2K^2|\mathcal{F}|^2$ pairs
Also over the course of XGBOOST, we have at most tK splits.
⇒ Computable using a GJ algorithm with at most $(t^2K^2|\mathcal{F}|^2)^{tK}$ predicates (degree 6)
⇒ $\text{Pdim}(U) = O(tK \log(tK|\mathcal{F}|))$

Open questions and research directions

- Efficient implementations of learning algorithms
- Extensions to other interpretable techniques
- Lower bounds on sample efficiency
- Online learning
- Combining with other guarantees e.g. robustness

Roadmap

- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - **Neural networks**
- Other aspects, ongoing and future research

Tuning deep networks: parameters and hyperparameters

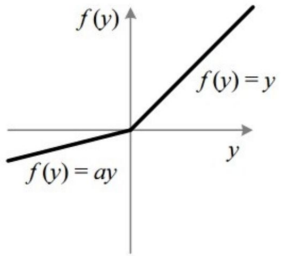
- **Hyperparameter space** $A = [\alpha_{min}, \alpha_{max}] \subset \mathbb{R}$ (**hyperparameter** α)
- **Model parameter space** $W \subset \mathbb{R}$ (**parameters/weights** w)

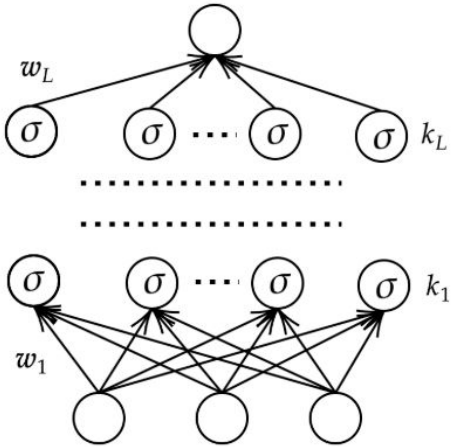
fixed during training

updated during training

- Example (learning activation functions):
  - Consider a DNN $\tau_{\alpha,w}$ with model weights $w = (w_1, \ldots, w_L)$
  - Parametric ReLU activation function

$$\text{PReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$$





  - More generally, one can interpolate* any activation functions
    $\sigma(z) = \alpha\, o_1(z) + (1 - \alpha)\, o_2(z)$
    where o_1, o_2 are common activation functions, α is interpolation hyperparameter

*inspired by DARTS approach for Neural Architecture Search [Liu et al. ICLR'19]

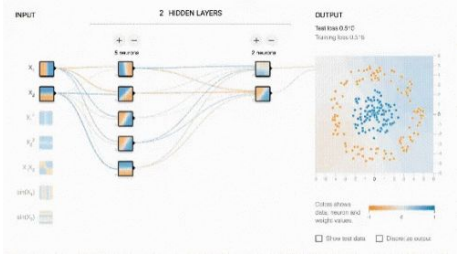
Model vs optimization hyperparameters

Our focus here is on tuning “model” or “architectural” hyperparameters:

- Are directly a part of the learned deep network $\tau_{\alpha,w}$
- Impact training, but stay fixed as we learn the weights w

e.g. activation function parameters,

kernel parameter in graph neural networks



Contrast this with “optimization” hyperparameters in the training procedure of the deep network

- They impact training too, but their effect on the learned network is fully captured by w

e.g. learning rate

Formalism: the utility function

- **Parameter-dependent utility function** $f(\boldsymbol{x}, \alpha, \boldsymbol{w})$
  the performance when using hyperparameter α and parameter $\boldsymbol{w}$, operating on problem instance $\boldsymbol{x}$

- **Utility function** $u_{\alpha}(\boldsymbol{x}) = \sup_{\boldsymbol{w}} f(\boldsymbol{x}, \alpha, \boldsymbol{w})$

  the performance of trained network using hyperparameter α, operating on problem instance $\boldsymbol{x}$

- Example
  - $f(\boldsymbol{x}, \alpha, \boldsymbol{w}) = H - ||y - \tau_{\alpha,\boldsymbol{w}}(X)||_2^2$ is the parameter-dependent **utility** function

    (the loss is $||y - \tau_{\alpha,\boldsymbol{w}}(X)||_2^2$)
  - $u_{\alpha}(\boldsymbol{x}) = \sup_{\boldsymbol{w}} f(\boldsymbol{x}, \alpha, \boldsymbol{w})$ is the utility function

Formalism: data-driven hyperparameter tuning

- Tuned hyperparameter $\hat{a}$ that has performance close to the optimal $\alpha^* = \max_\alpha \mathrm{E}_{x \sim D}[u_\alpha(x)]$

$$|\mathrm{E}_{x \sim D}[u_{\hat{a}}(x)] - \mathrm{E}_{x \sim D}[u_{\alpha^*}(x)]| < \varepsilon$$

  with probability at least $1 - \delta$, using problem instances $x_1, \ldots, x_m \sim D^m$

- **Question**: How many problem instances $m(\varepsilon, \delta)$ are enough?

Statistical learning theory: sample complexity and pseudo-dimension

Given $\varepsilon > 0$ and $0 < \delta < 1$, what is the sample complexity $m(\varepsilon, \delta)$?

- Standard PAC-Learning approach: bound the learning-theoretic complexity of U

$$U = \{u_\alpha : \boldsymbol{\mathcal{X}} \to [0, \mathrm{H}] \mid \alpha \in A\}$$

- Complexity measure: pseudo-dimension, Pdim(U)
  - The maximum size n such that U can "shatter" $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, using thresholds $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n \in \mathbb{R}$
  - by "shattering", we mean $|\{\mathrm{sign}(u_\alpha(\boldsymbol{x}_1) - \boldsymbol{t}_1), \ldots, \mathrm{sign}(u_\alpha(\boldsymbol{x}_n) - \boldsymbol{t}_n) \mid u_\alpha \in U\}| = 2^n$

- Classical learning theory: If Pdim(U) is finite, then $m(\varepsilon, \delta) = \mathrm{O}(\mathrm{H}^2/\varepsilon^2(\mathrm{Pdim}(U) + \log 1/\delta))$

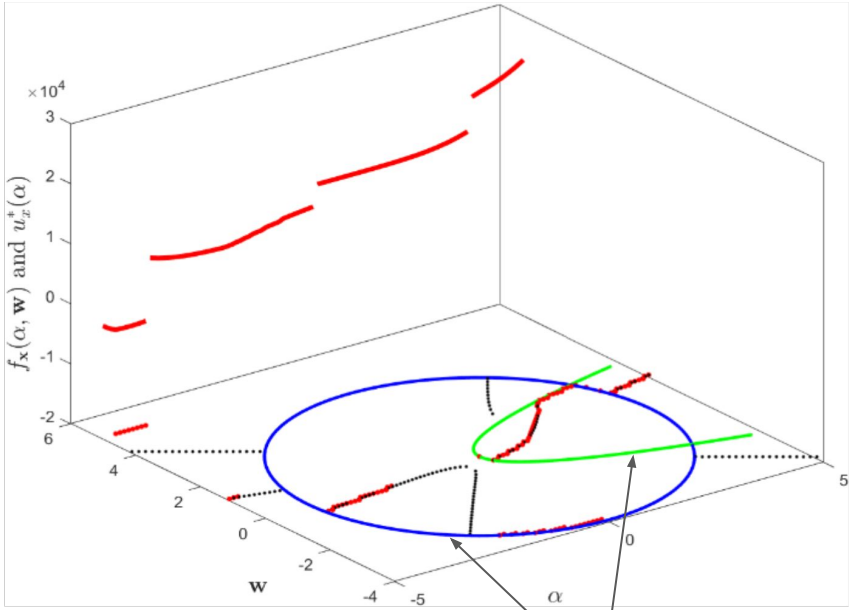
Piecewise polynomial parameter-dependent utility function

- Recall utility function: $u_\alpha(\boldsymbol{x}) = \sup_w f(\boldsymbol{x}, \alpha, w)$, where parameter-dependent utility: $f(\boldsymbol{x}, \alpha, w)$

- Motivated by classical work on NNs*, we assume: for any fixed problem instance $\boldsymbol{x}$, the **parameter-dependent dual** $f_{\boldsymbol{x}}(\alpha, w) := f(\boldsymbol{x}, \alpha, w)$ admits a **piecewise polynomial structure**:
  - There are polynomial **boundary functions** $h_{\boldsymbol{x},1}(\alpha, w), \ldots, h_{\boldsymbol{x},\mathrm{M}}(\alpha, w)$ …
  - that partition the domain $A \times W$ of $f_{\boldsymbol{x}}(\alpha, w)$ into connected components "pieces" $R_{\boldsymbol{x},1}, \ldots, R_{\boldsymbol{x},N}$
  - $f_{\boldsymbol{x}}(\alpha, w)$ restricted on $R_{\boldsymbol{x},i}$ is polynomial $f_{\boldsymbol{x},i}(\alpha, w)$ (**piece function**)

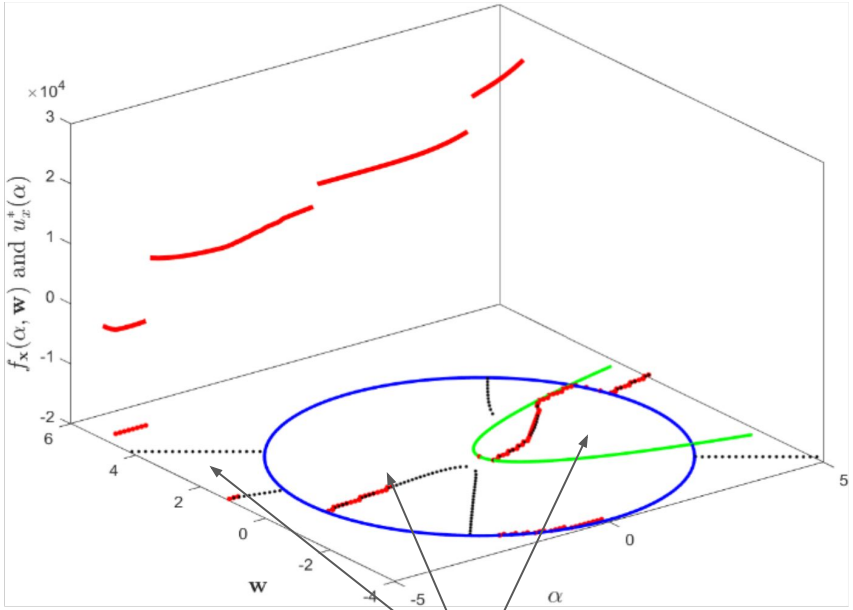
*[Bartlett et al. 1998, Bartlett et al. 2019]

Piecewise polynomial structure: an example



- Boundary functions $h_{x,1}$ and $h_{x,2}$

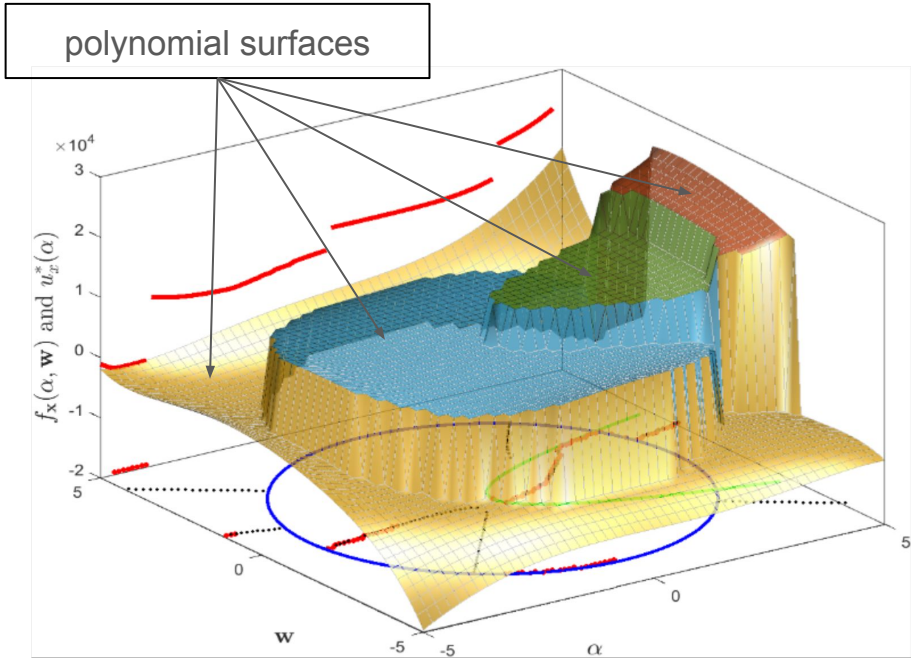
Piecewise polynomial structure: an example



Connected components

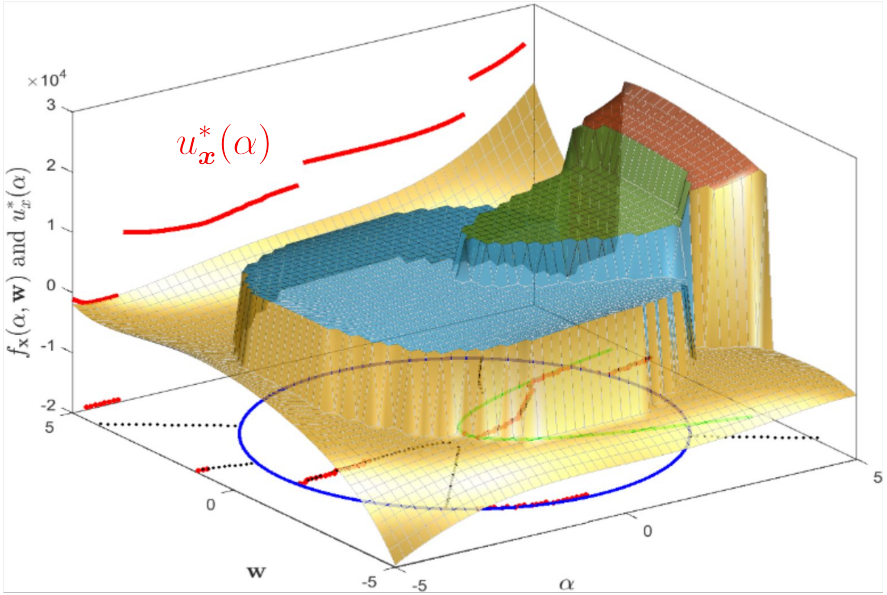
- Boundary functions $h_{x,1}$ and $h_{x,2}$
- partition domain into connected components $R_{x,1}$, …, $R_{x,N}$

Piecewise polynomial structure: an example



- Boundary functions $h_{x,1}$ and $h_{x,2}$
- partition domain into connected components $R_{x,1}$, …, $R_{x,N}$
- $f_x(\alpha, w)$ restricted on $R_{x,i}$ is poly. $f_{x,i}(\alpha, w)$

Piecewise polynomial structure: an example

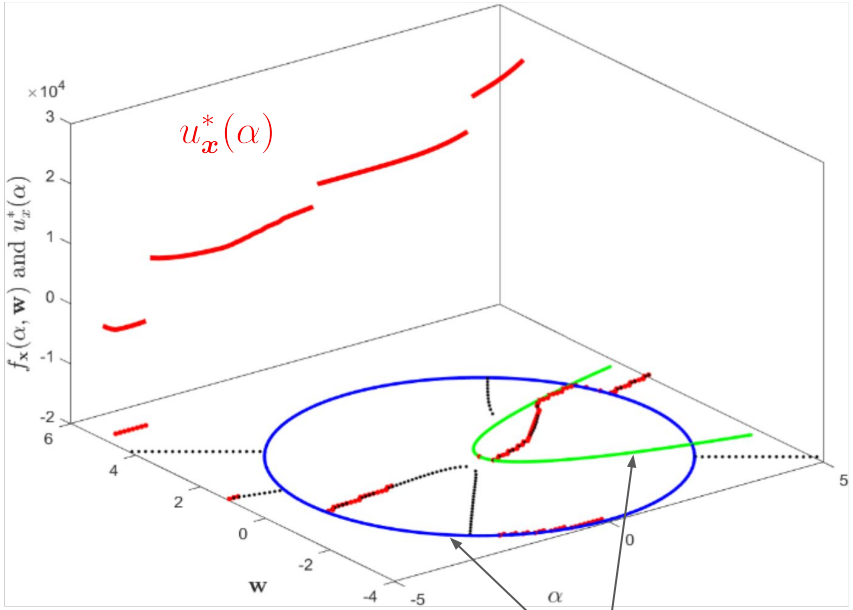


- Boundary functions $h_{x,1}$ and $h_{x,2}$
- partition domain into connected components $R_{x,1}$, …, $R_{x,N}$
- $f_x(\alpha, w)$ restricted on $R_{x,i}$ is poly. $f_{x,i}(\alpha, w)$

To bound Pdim(U), we're interested in:

$$u^*_x(\alpha) := u_\alpha(x) = \sup_w f_x(\alpha, w)$$

Key mathematical question



- If $f_x(\alpha, w)$ is piecewise-polynomial, can we give a bound on the piecewise structure of

$$u^*_x(\alpha) := u_\alpha(x) = \sup_w f_x(\alpha, w)$$

- To bound Pdim(U), it is sufficient to bound the number of discontinuities and number of local maxima of $u^*_x(\alpha)$

Main result

Theorem (informal): $\mathrm{Pdim}(U) = \mathrm{O}(\log N + d \log(\Delta M))$, where

N is the number of connected components

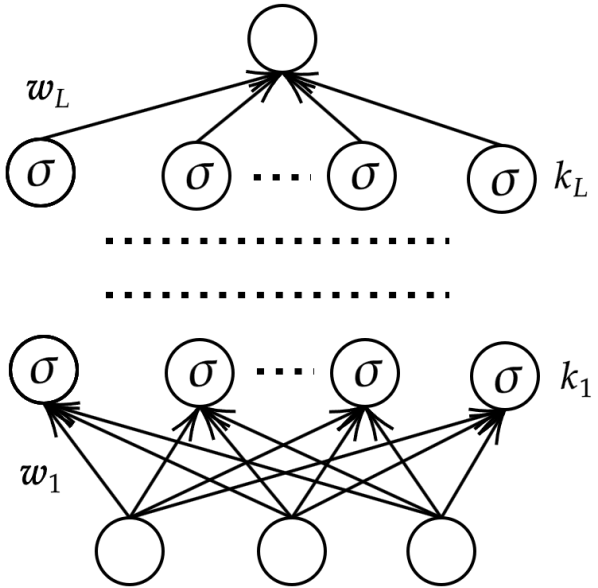
M is the number of boundaries

d is the dimension of $\boldsymbol{w}$

Δ is the maximum polynomial degree

Learning the interpolated activation function

- DNN $\tau_{\alpha,w}$ with L layers
- Layer i: W_i params (total W), k_i nodes (total k)
- $\sigma(z) = \alpha\, o_1(z) + (1-\alpha)\, o_2(z)$, where o_1, o_2 piecewise poly. with max degree Δ, p breakpoints
- T samples (not assumed iid) in each problem instance



Learning the interpolated activation function

Theorem (informal): $\mathrm{Pdim}(U) = \mathrm{O}(\log N + d \log(\Delta M))$, where

N is the number of connected components

M is the number of boundaries

d is the dimension of $\boldsymbol{w}$

Δ is the maximum polynomial degree

Application: For the activation function interpolation:
$\mathrm{Pdim}(U) = \mathrm{O}(L^2 W \log \Delta + LW \log(Tpk))$

New Techniques

Overall approach to bound Pdim(U)

Analyze the piecewise structure of dual utility function: $u^*_{\boldsymbol{x}}(\alpha) := u_{\alpha}(\boldsymbol{x}) = sup_{w} f_{\boldsymbol{x}}(\alpha, w)$

- bound number of discontinuities
- bound number of local maxima

Key steps in our analysis:

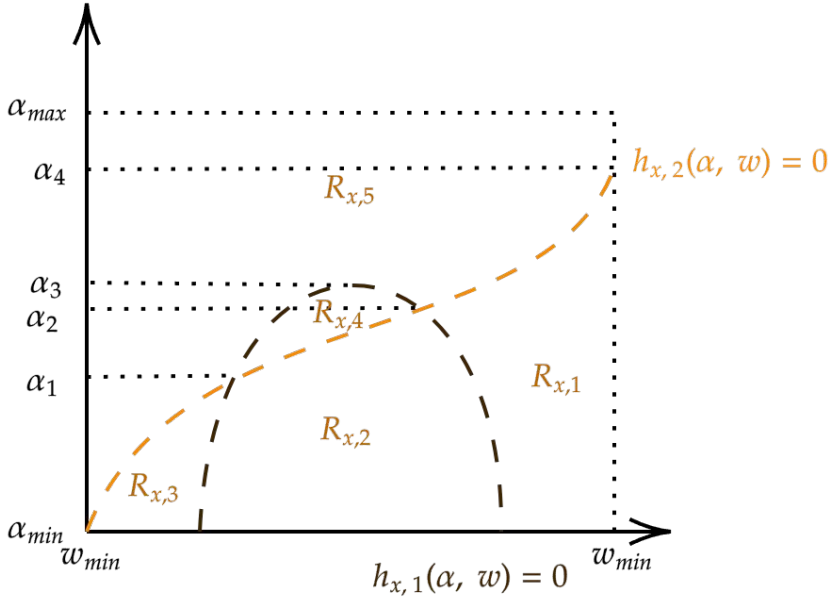
1. Effectively **reduce the problem to a single piece** with polynomial boundaries
2. Identify **all possible locations of "best weights"** w as the hyperparameter α is varied.
   a. Roughly speaking, these are smooth 1-dimensional manifolds corresponding to (appropriate intersections of) derivative curves or boundaries
3. Decompose these locations into "**monotonic curves**"
4. Bound the **number of local extrema** of $f_{\boldsymbol{x}}(\alpha, w)$ of along any monotonic curve

Step 1

Reduce the problem to a single piece with polynomial boundaries

- Partition hyperparameter space A into intervals based on α-end points of the pieces

- Given a fixed finite set of pieces, it is sufficient to analyze a single piece



Step 2

Where can the “best weights” possibly be located?

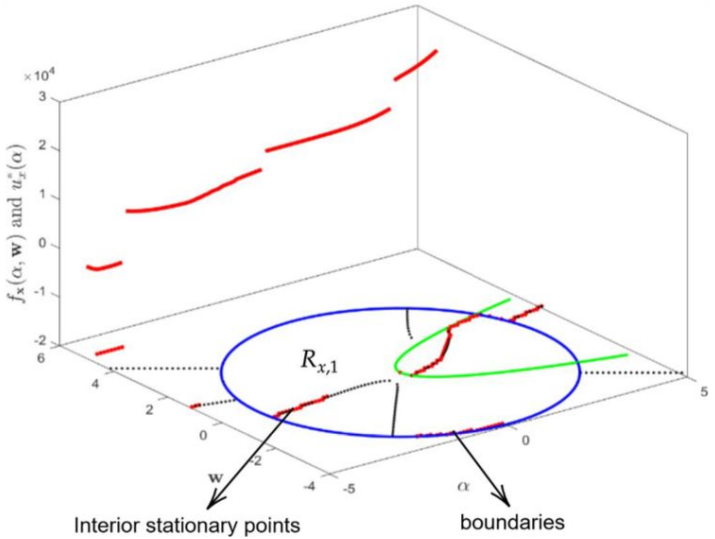
$$u^*_x(\alpha) := u_\alpha(\boldsymbol{x}) = sup_w f_x(\alpha, w)$$

Fermat’s interior extremum theorem

⇒ either boundaries, or

“derivative curves”

i.e. $\partial f_x(\alpha, w) / \partial w_i = 0$ for i = 1, …, d



Step 2

Where can the “best weights” possibly be located?

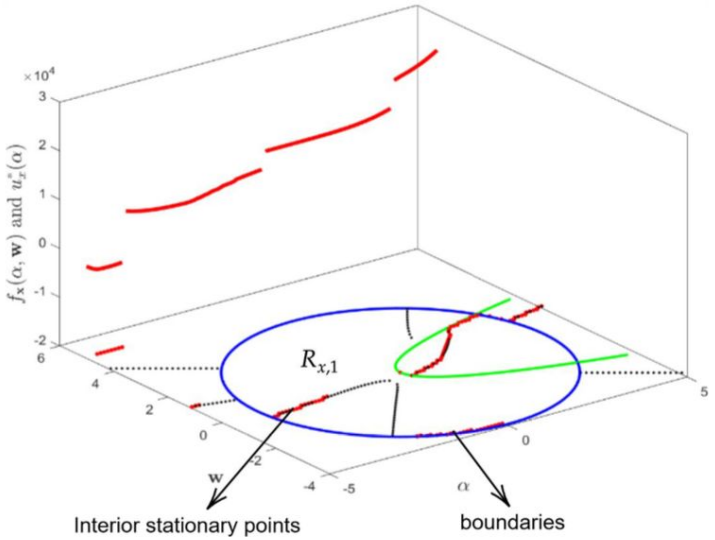
– boundaries or derivative curves

(roughly) these are both smooth 1-dimensional manifolds under mild regularity assumptions

Derivative curves:

$\partial f_x(\alpha, w) / \partial w_i = 0$ for $i = 1, \ldots, d$

d (d-dimensional) hypersurfaces in $\mathbb{R}^{d+1}$



Step 2

Where can the “best weights” possibly be located?

– boundaries or derivative curves

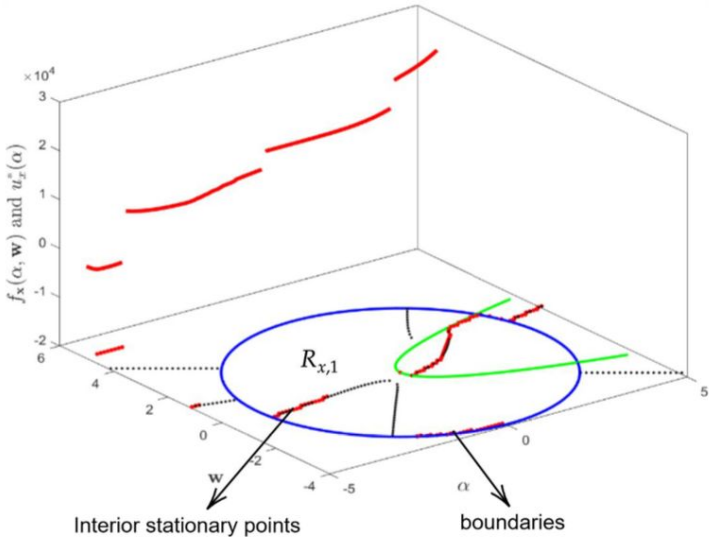
(roughly) these are both smooth 1-dimensional manifolds under mild regularity assumptions

Intersections of $S \leq d$ boundaries:

$h_{x,j}(\alpha, w) = 0$ for $j \in S$

Lagrangian: $f_x(\alpha, w) + \sum_j \lambda_j h_{x,j}(\alpha, w)$

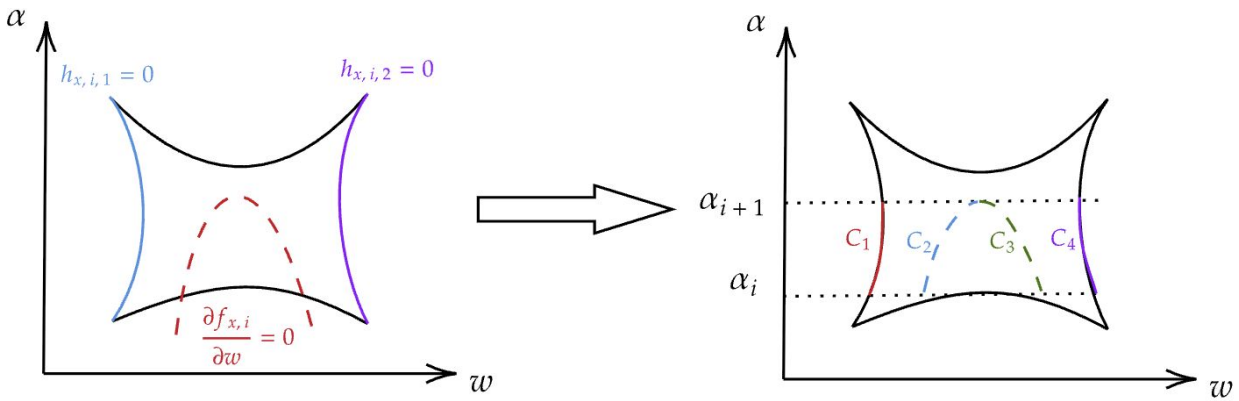
$\Rightarrow$ $d + S$ hypersurfaces in $\mathbb{R}^{d+S+1}$



Step 3

Decompose these (almost everywhere) one-dimensional manifolds into monotonic curves

Key property: if the hyperplane $\alpha = \alpha_0$ intersects monotonic curve C, it does so in a unique point



Step 4

Use the Lagrange Multiplier theorem and Bezout's theorem* to bound the number of local extrema of $f_x(\alpha, w)$ along any monotonic curve C.

C → intersection of polynomial equations (in α, w and possibly some Lagrange multipliers)

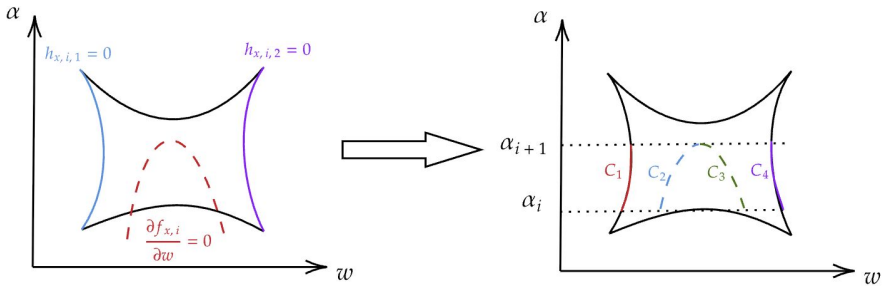
$f_x(\alpha, w)$ → a polynomial objective

⇒ All the Lagrangian derivatives are polynomial equations

* (from algebraic geometry) roughly, gives a bound on the number of simultaneous solutions of polynomial equations

Putting it all together

(a) The **discontinuities** of $u^*_x(\alpha)$ are upper bounded by our partition into intervals with a fixed set of monotonic curves



(b) **Lemma**: The local maxima of $g(x) = \max g_i(x)$ are contained in the set of local maxima of $g_i(x)$

Lemma: If there are at most B_1 discontinuities and at most B_2 local maxima in any u^*_x, then $\text{Pdim}(U) = O(\log(B_1 + B_2))$.

Beyond model parameters: gradient descent

Gradient descent algorithm

Inputs: initial point x, iterations H, threshold θ. Hyperparameter: η

1: **Initialize** $x_1 \leftarrow x$
2: **for** $i = 1, \ldots, H$ **do**
3: **if** $||\nabla f(x_i)|| < \theta$ **then**
4: **Return** x_i
5: $x_{i+1} = x_i - \eta \nabla f(x_i)$

Output: x_i

Prior work by Gupta and Roughgarden (2016):

Assumes: f is convex and smooth

Sample complexity of tuning learning rate is $O(H^3/\varepsilon^2)$

We get $O(H^3/\varepsilon^2)$ sample complexity even for non-convex non-smooth functions!

Roadmap

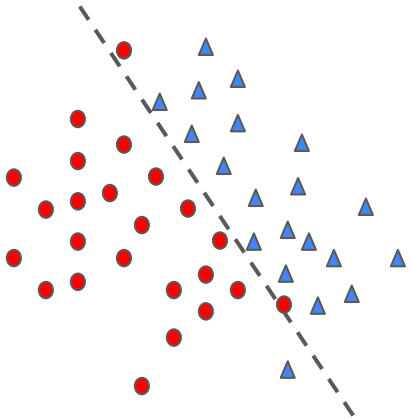
- Algorithm design for machine learning (aka HP tuning)
- Current approaches in practice
  - Bayesian Optimization, Gradient-based and Bandit-based methods
- Machine learning for algorithm design
  - Learning-theoretic foundations
  - GJ algorithm framework
- Tuning core ML algorithms
  - Decision Trees
  - Neural networks
- **Other aspects, ongoing and future research**

Semi-supervised learning

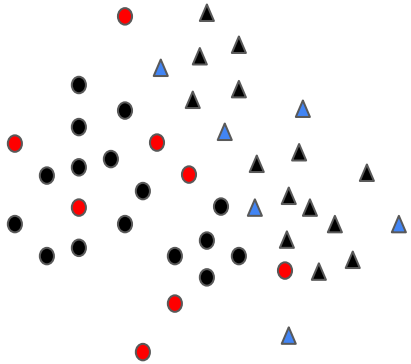
[Chapelle, Scholkopf and Zien, 2006]
[Zhu and Goldberg, 2009]

Three main ways to learn from data:

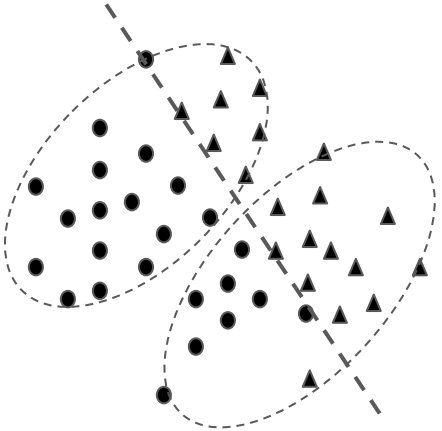
Supervised



Semi-supervised



Unsupervised



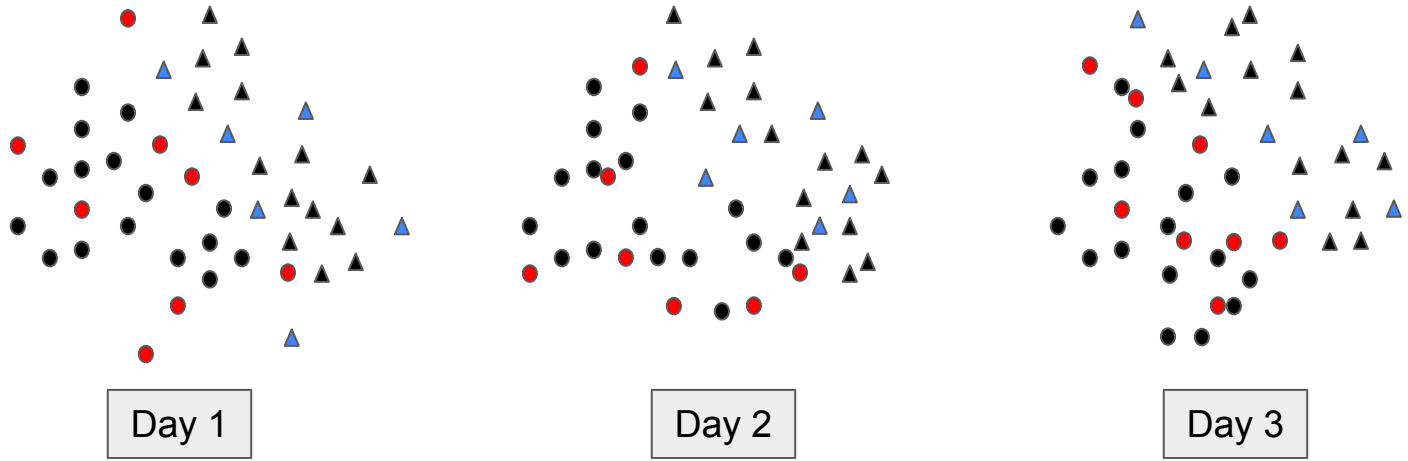
▲ positive (true label)
● negative (true label)
red/blue: observed label

cheaper, but can be less accurate and/or less trustable

Data-driven semi-supervised learning

[Oral (55/9122, top 0.6%) at NeurIPS'2021; joint work with Nina Balcan]

★ Repeated problems e.g. emails on an email server, spam vs. non-spam
- **data**: multiple partially labeled data sets from the same domain
- **desiderata**: efficiency (labels, samples, computational)

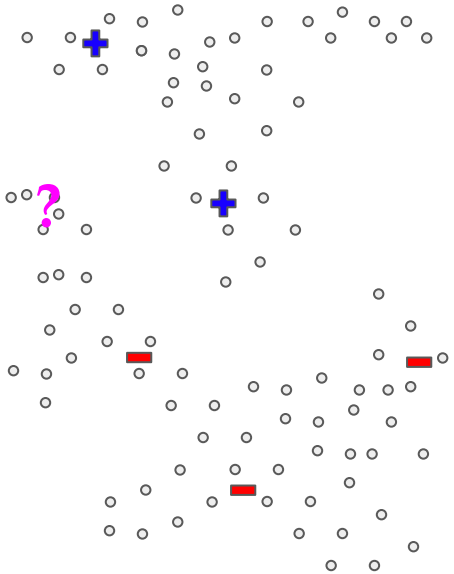


Graph-based algorithm families

Use feature similarity of unlabeled examples

Use a graph to account for global and local patterns in similarity

1. Add (stronger) edges between similar examples
2. Run a graph partitioning algorithm, assign appropriate labels to the pieces

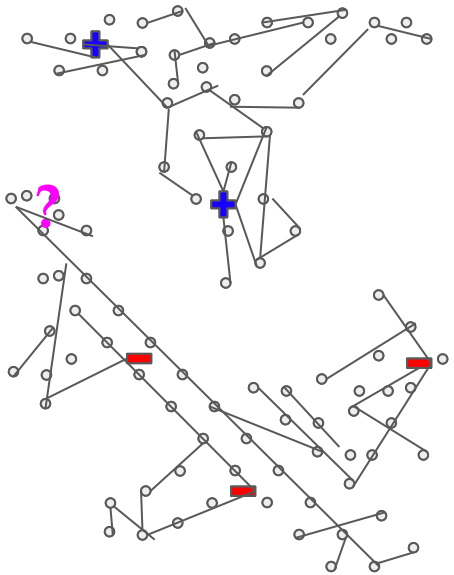


Graph-based algorithm families

Use feature similarity of unlabeled examples

Use a graph to account for global and local patterns in similarity

1. Add (stronger) edges between similar examples
2. Run a graph partitioning algorithm, assign appropriate labels to the pieces



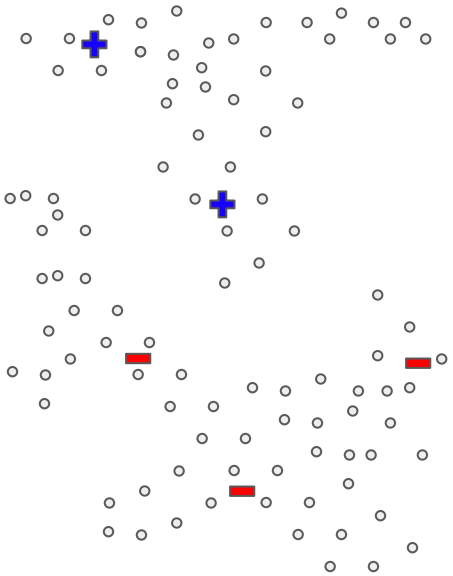
Graph-based algorithm families

Add (stronger) edges between similar examples

Typical setup: Given a distance metric $d(u,v)$,

Set graph edges based on $d(u,v)$:

- Threshold (unweighted): add an edge if $d(u,v) < \mathbf{r}$
- Exponential kernel: $w(u,v) = \exp(-d(u,v)^2/\boldsymbol{\sigma}^2)$



r, $\boldsymbol{\sigma}$ **are hyperparameters**

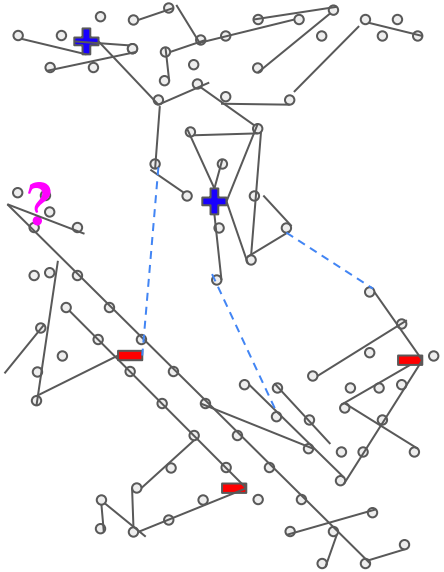
Graph-based algorithm families

Run a graph partitioning algorithm, assign appropriate labels to the pieces

Graph-based algorithm families

2 Run a graph partitioning algorithm, assign appropriate labels to the pieces

s,t min-cut [Blum and Chawla, 2001]

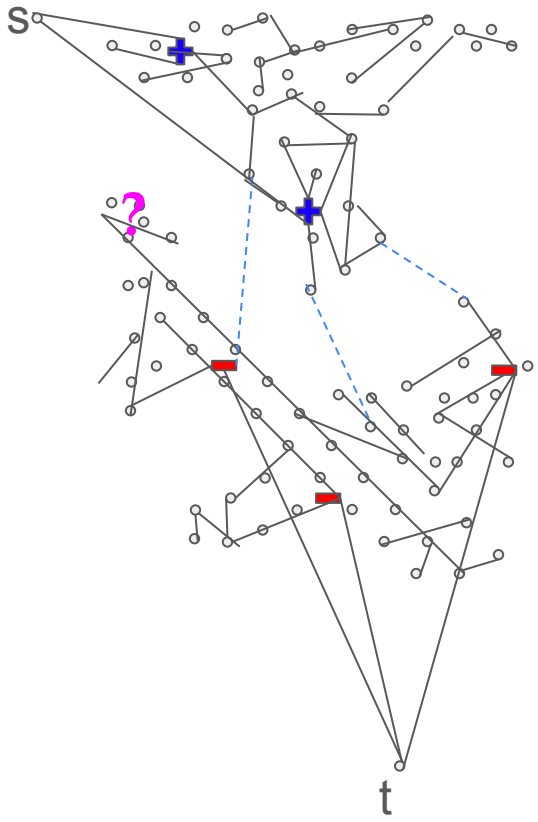


Graph-based algorithm families

Run a graph partitioning algorithm, assign appropriate labels to the pieces

s,t min-cut [Blum and Chawla, 2001]

- Connect a single node (with infinite weight) to all nodes with the same label

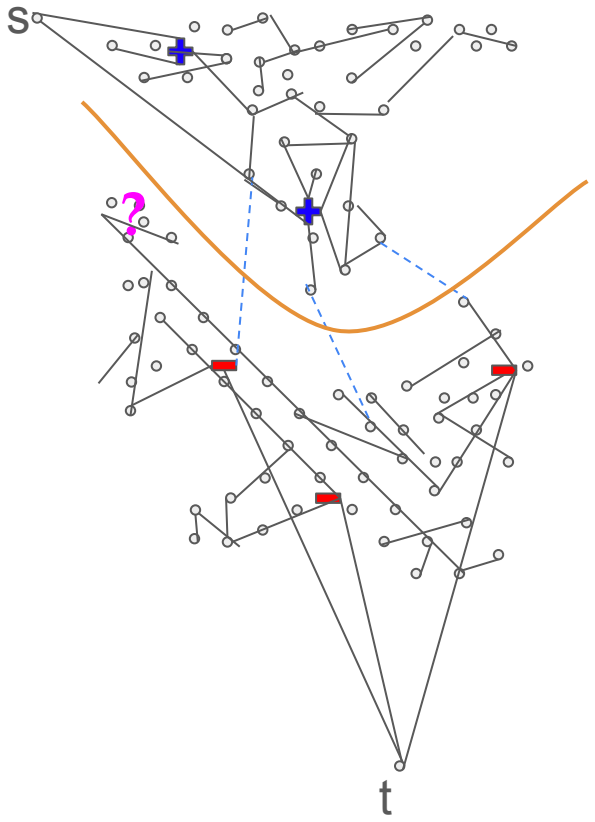


Graph-based algorithm families

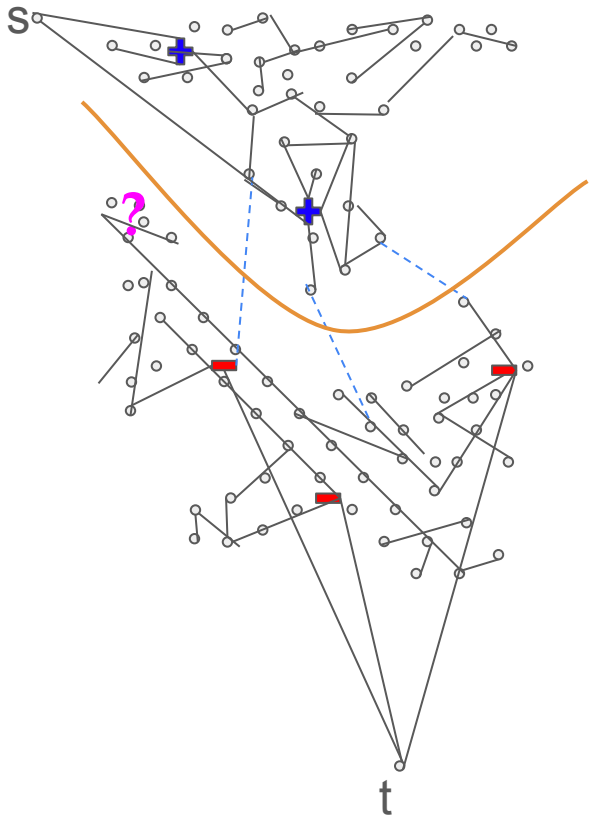
Run a graph partitioning algorithm, assign appropriate labels to the pieces

s,t min-cut [Blum and Chawla, 2001]

- Connect a single node (with infinite weight) to all nodes with the same label
- Compute the graph min-cut separating these new nodes



Graph-based algorithm families



2 Run a graph partitioning algorithm, assign appropriate labels to the pieces

soft Mincut [Zhu, Lafferty and Ghahramani, 2003]

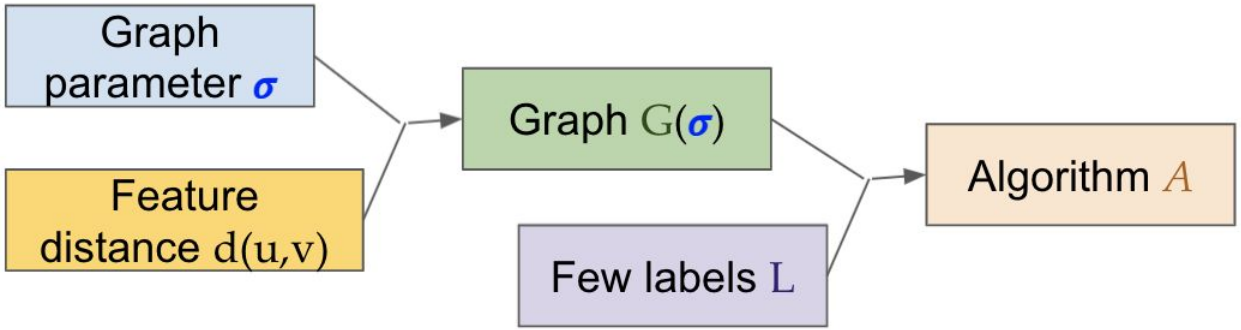
$$L(f, G) = \Sigma_{u,v} w(u,v)(f(u) - f(v))^2$$

$$\text{argmin}_f \, l(f) = L(f, G)$$

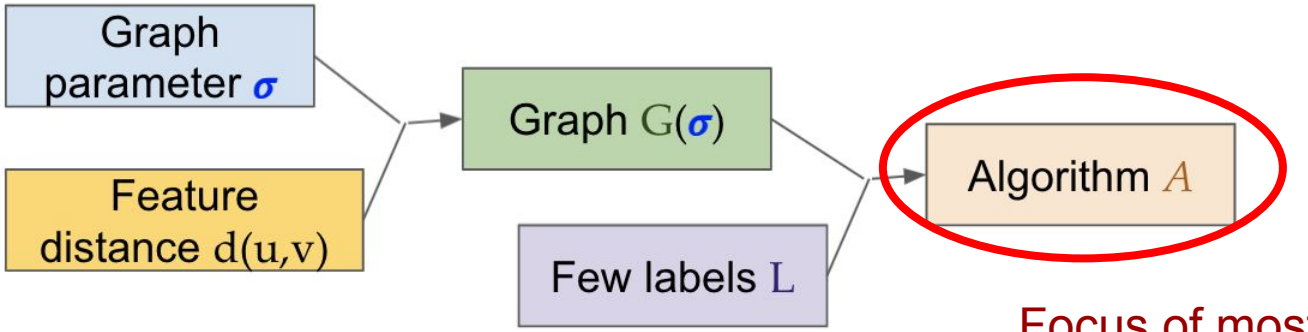
$f(u) \in \{0,1\}$ (hard labels, min-cut)

or $f(u) \in [0,1]$ (soft labels, harmonic objective)

Model


Graph parameter σ
Feature distance $d(u,v)$
Graph $G(\sigma)$
Few labels L
Algorithm A

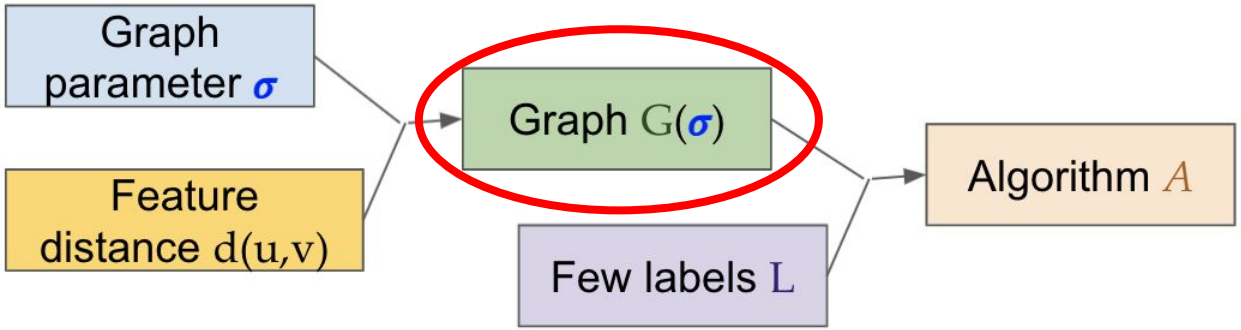
Model – the labeling algorithm

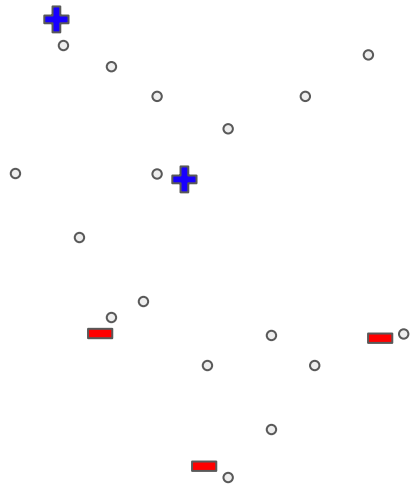


Focus of most research!

[Blum&Chawla 2001]
[Zhu et al. 2003]
[Zhou et al. 2004]
…
…
[Wang et al. 2016]
[Avrachenkov et al. 2017]
[Liao et al. 2018]

Model – selecting the graph





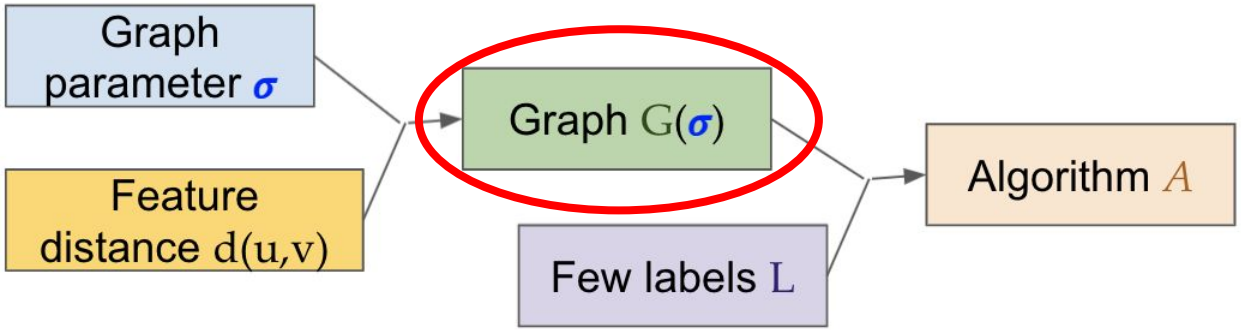
Applied papers manually select r,σ. [Balcan et al. ICML 2005]

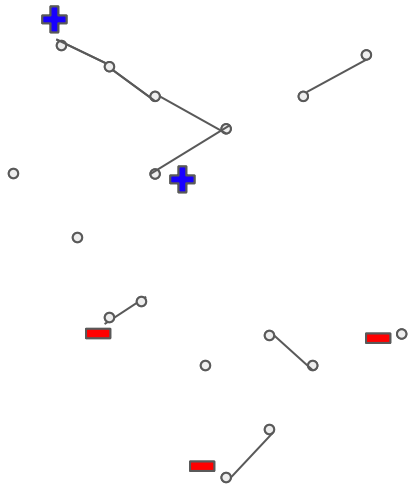
Heuristics

- Select r^* = smallest r that connects the graph
- Select $\sigma = r^*/3$ [Zhu 2005]

Recall:
Threshold graph $G(r)$
$w(u,v) = I[d(u,v) < r]$
Gaussian $G(\sigma)$
$w(u,v) = \exp(-d(u,v)^2/\sigma^2)$

Model – selecting the graph





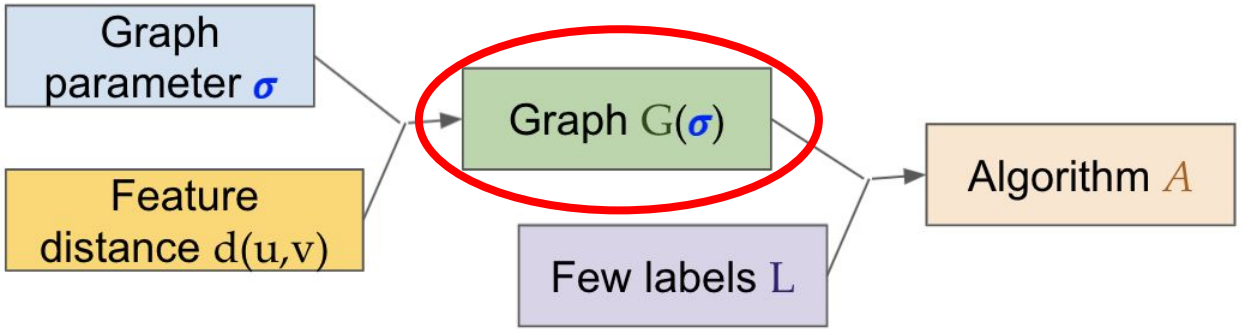
Applied papers manually select r,σ. [Balcan et al. ICML 2005]

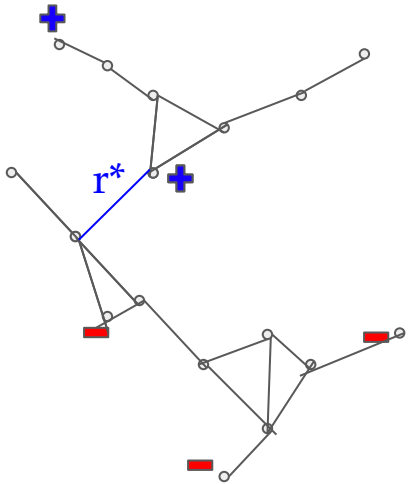
Heuristics

- Select r^* = smallest r that connects the graph
- Select $\sigma = r^*/3$ [Zhu 2005]

Recall:
Threshold graph $G(r)$
$w(u,v) = I[d(u,v) < r]$
Gaussian $G(\sigma)$
$w(u,v) = \exp(-d(u,v)^2/\sigma^2)$

Model – selecting the graph





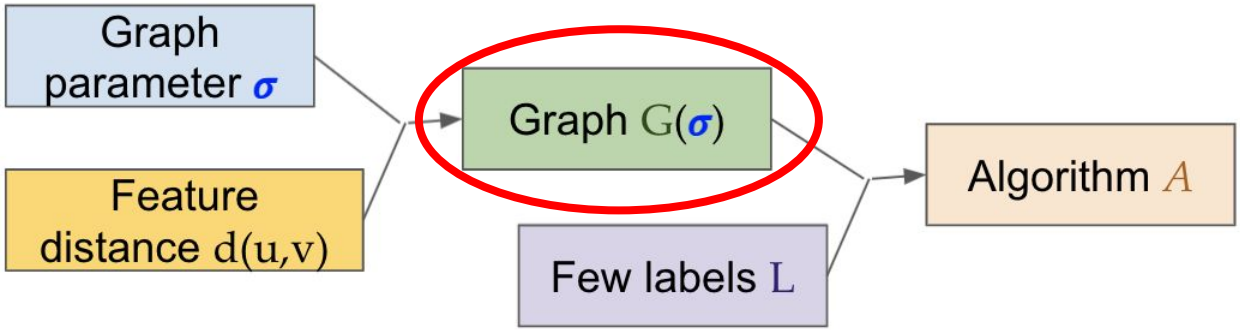
Applied papers manually select r,σ. [Balcan et al. ICML 2005]

Heuristics

- Select r^* = smallest r that connects the graph
- Select $\sigma = r^*/3$ [Zhu 2005]

Recall:
Threshold graph $G(r)$
$w(u,v) = I[d(u,v) < r]$
Gaussian $G(\sigma)$
$w(u,v) = \exp(-d(u,v)^2/\sigma^2)$

Model – the labeling algorithm

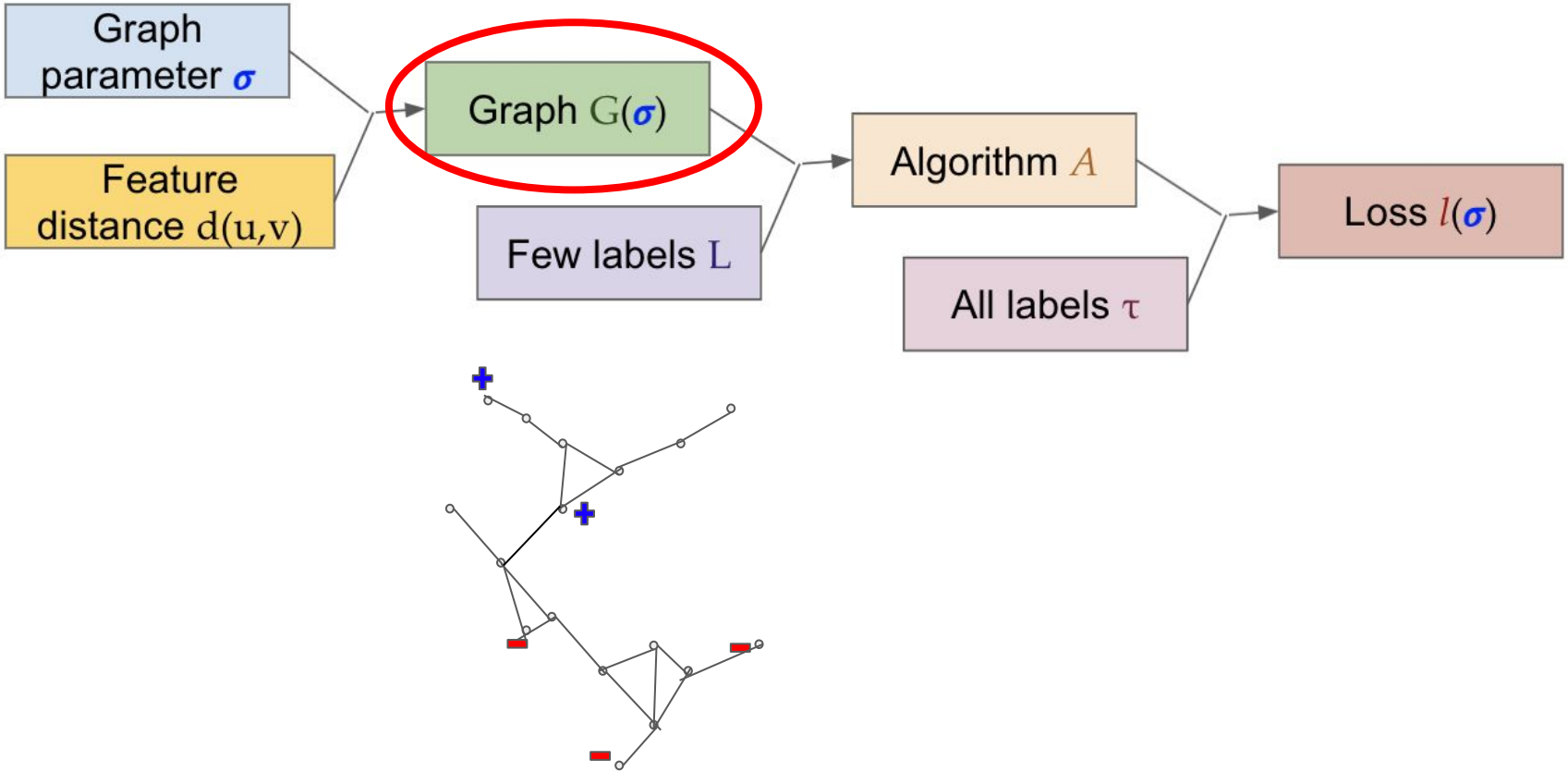


Graph construction is "**more of an art, than science**" [Zhu 2005]

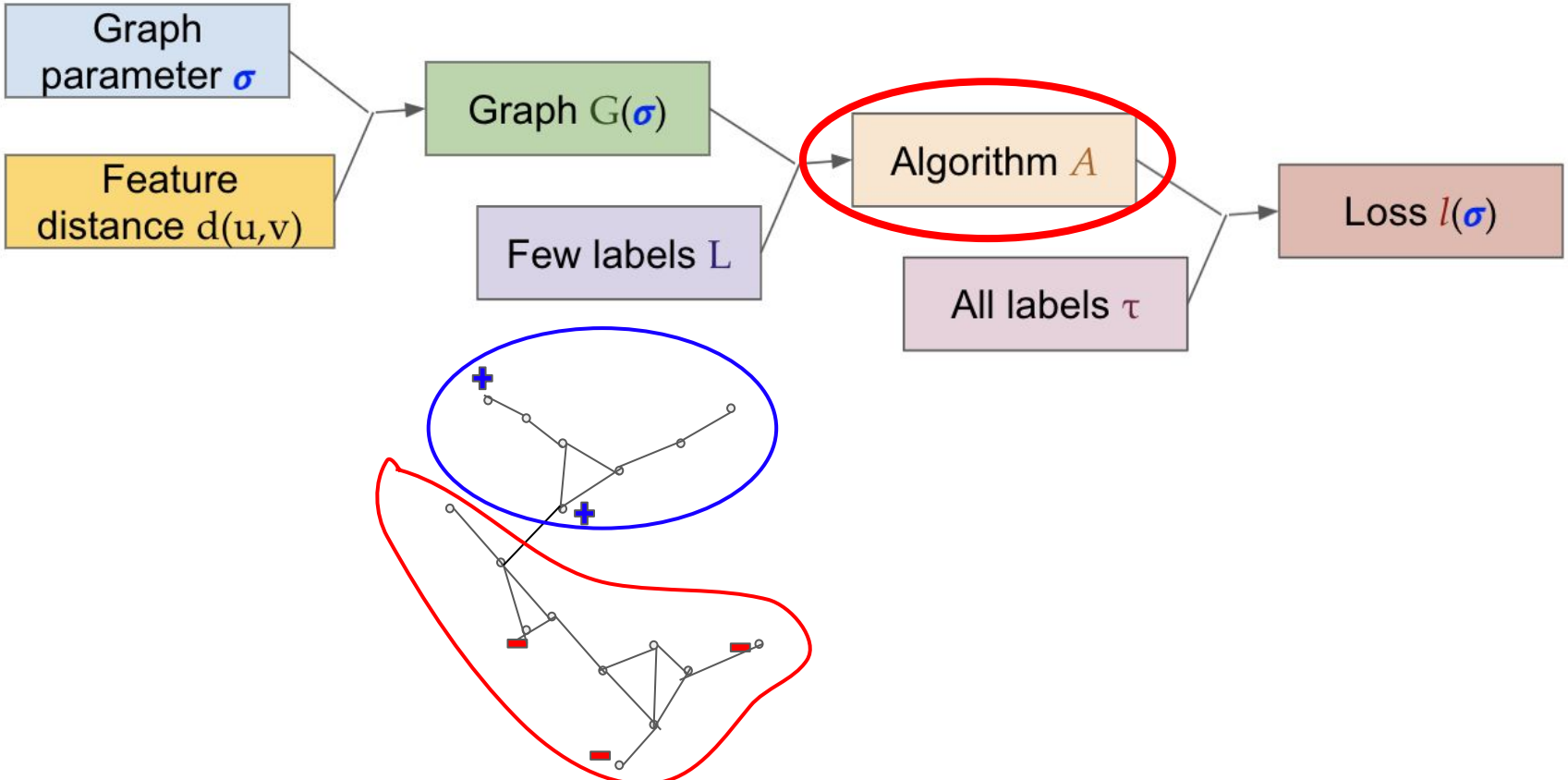
Currently, graph construction "is more of an art than science" [Alexandrescu, Kirchhoff 2007]

Remains "more of an art, than science" [Subramanya, Bilmes 2009] [Ozaki et al. 2011] [Eriguchi, Kobayashi 2014] … [Domingue et al. 2019]

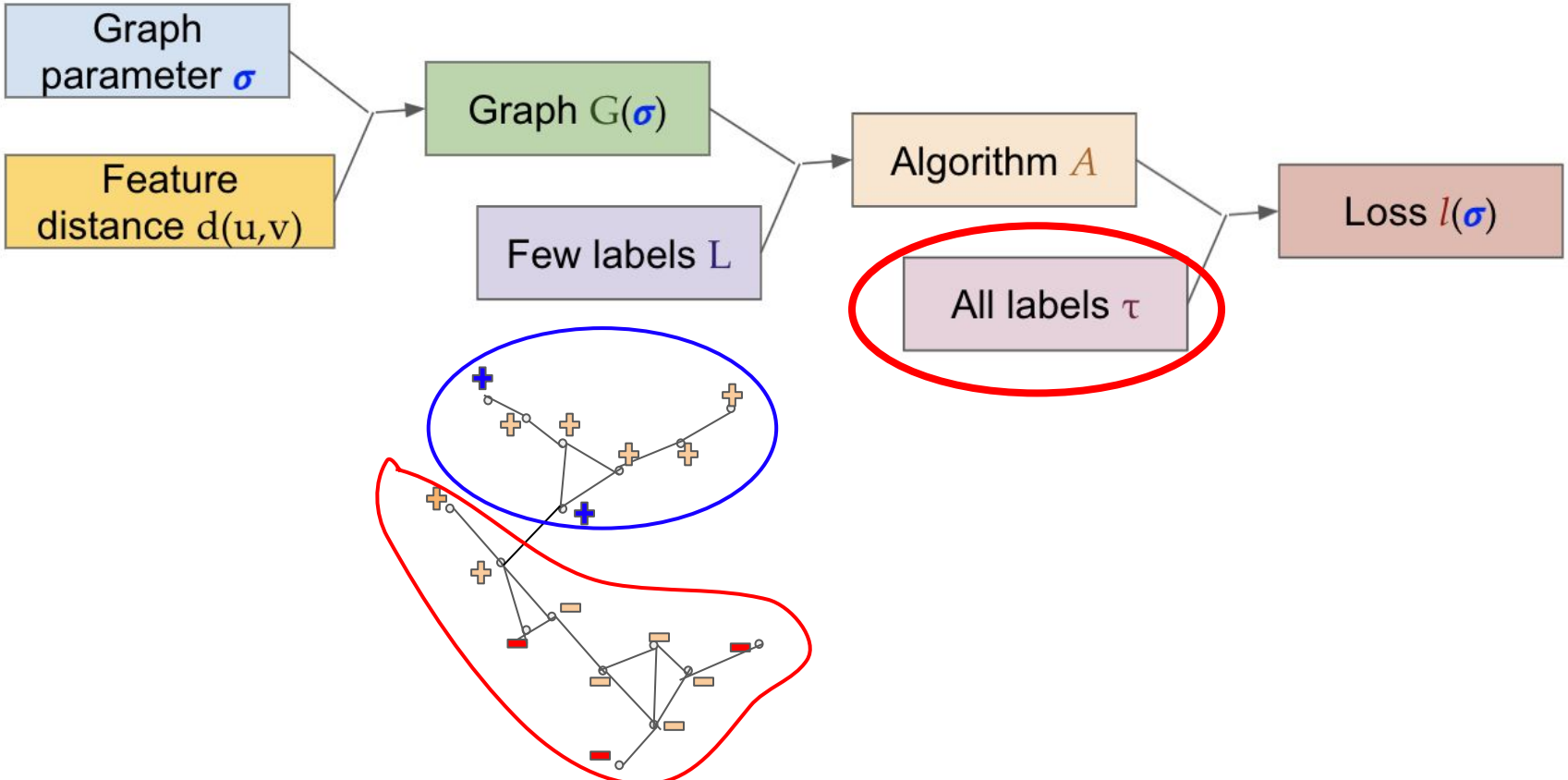
Model – the loss function



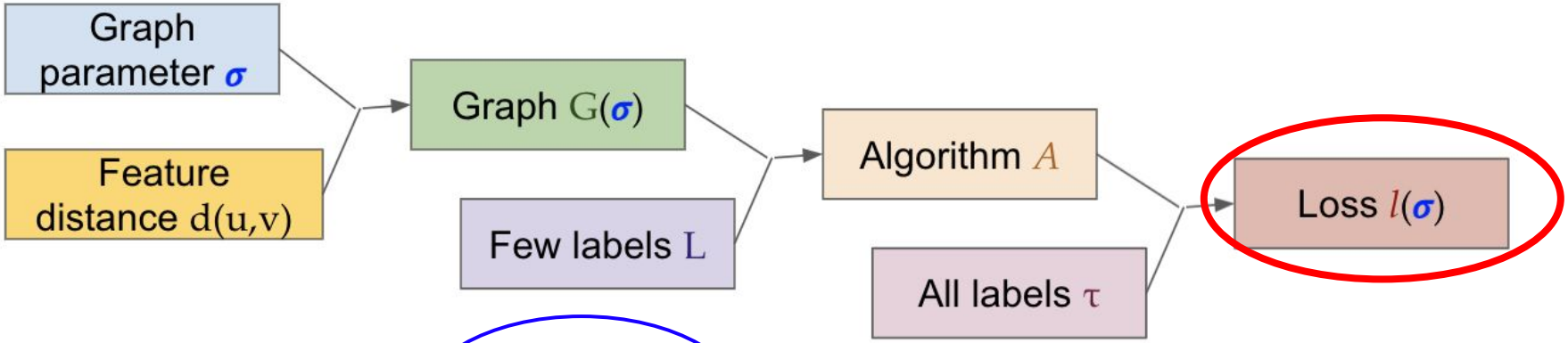
Model – the loss function

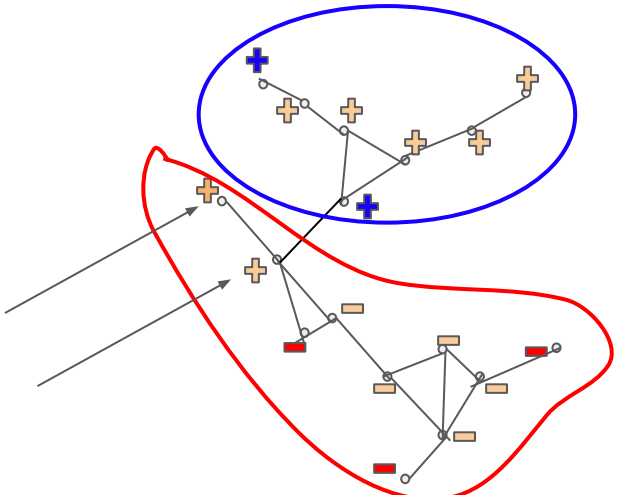


Model – the loss function



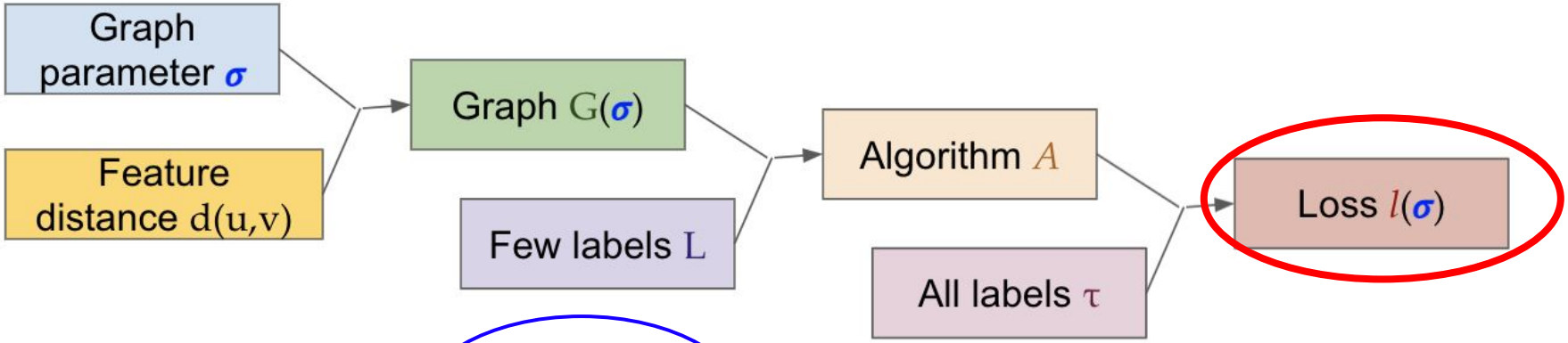
Model – the loss function

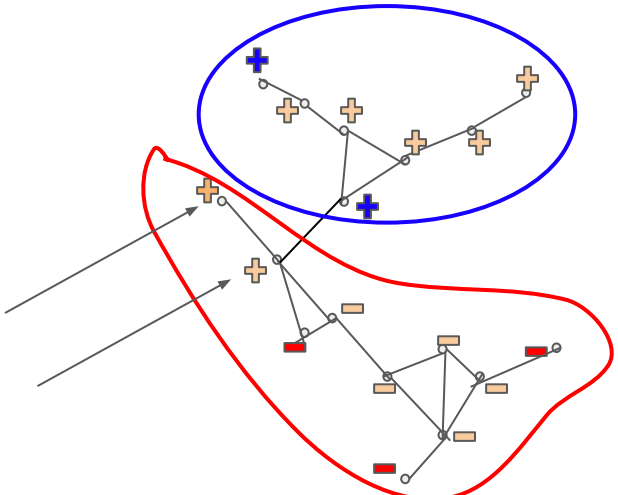




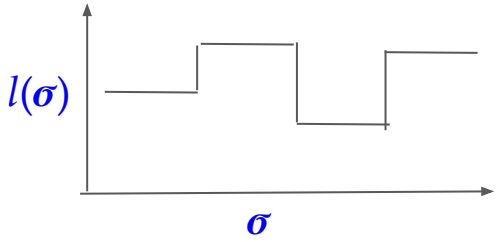
Loss = 2/12 = 0.17

Model – the loss function





As you vary σ …



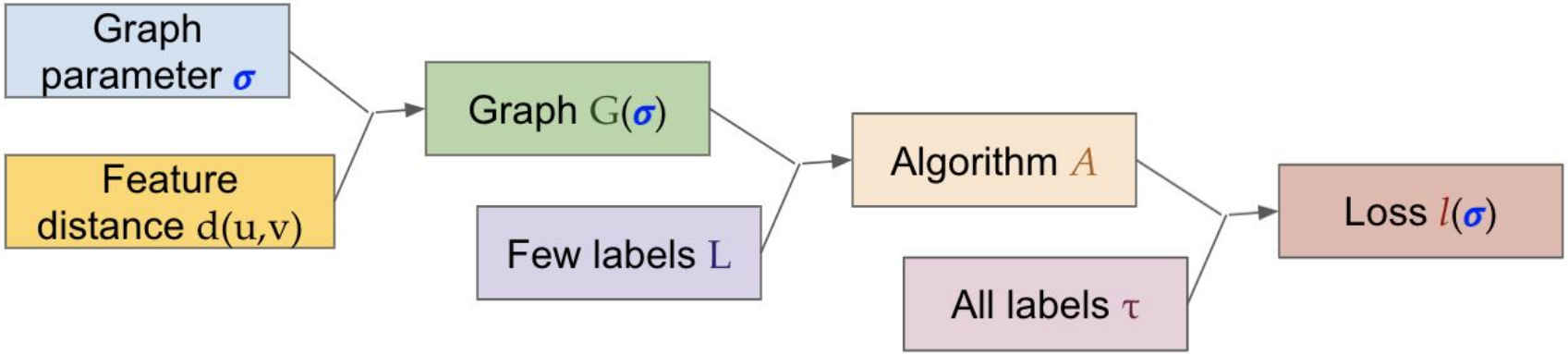
Pseudo-dimension bounds and generalization

Tight upper and lower bounds!

Result: Pseudo-dimension of threshold-based family $G(r)$ is $\Theta(\log n)$.

Result: Pseudo-dimension of Gaussian kernel family $G(\sigma)$ is $\Theta(n)$.

Model – online learning



Online learning: Given a sequence of problems, for each problem

- Select a graph $G(\sigma)$ *(by choosing* σ*)*
- Label a partially labeled problem instance using labeling algorithm A
- All true labels are revealed, we suffer loss $l(\sigma)$ for mislabeled examples

A key challenge

$l(\sigma)$ is piecewise constant, can we still optimize?

Worst case: NO!

But real world data is usually not worst case ...

A key challenge

$l(\boldsymbol{\sigma})$ is piecewise constant, can we still optimize?

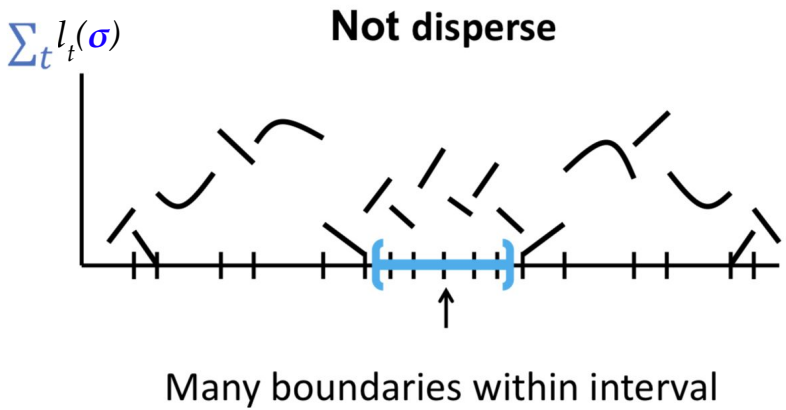
Yes, provided discontinuities (over time) do *not concentrate in any interval*.

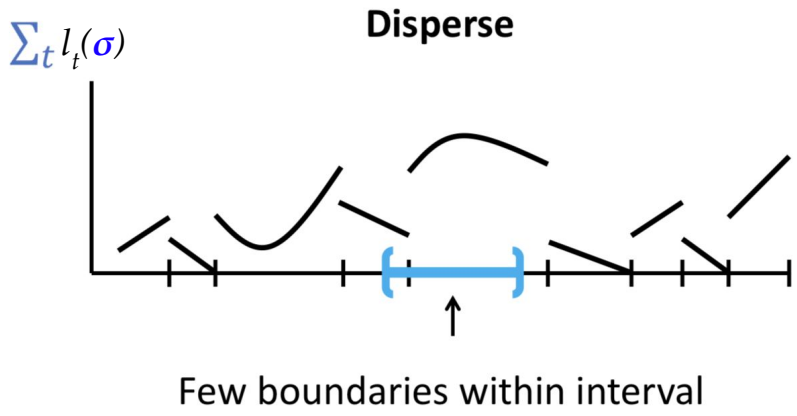
When can we learn the graph?

$l(\boldsymbol{\sigma})$ is piecewise constant, can we still optimize?
Yes, provided discontinuities (over time) do *not concentrate in any interval*.

Dispersion: If in any interval I of width $\varepsilon \geq 1/\sqrt{\mathrm{T}}$, few discontinuities (in expectation).

$\mathrm{E}[\max_{\mathrm{I}}(\#\text{ discontinuities in } \mathrm{I})] = \mathrm{O}(\varepsilon \mathrm{T})$ [Balcan, Dick, Vitercik, FOCS 2018]





Online learning

Theorem: There exists an algorithm with $O(1/\sqrt{T})$ **expected average regret** provided distance metric $d(u,v)$ is "**smooth**".

i.e. $d(u,v)$ is distributed with κ-bounded density.

Regret(Alg. **E**) =

$$\left[\begin{array}{l}\text{Loss suffered by}\\ \textbf{your graphs}\end{array}\right] - \left[\begin{array}{l}\text{Loss of best } G\\ \text{(in hindsight)}\end{array}\right]$$

Online learning

Theorem: Algorithm E enjoys $O(1/\sqrt{T})$ **expected average regret** provided distance metric $d(u,v)$ is "**smooth**".

i.e. $d(u,v)$ is distributed with κ-bounded density.

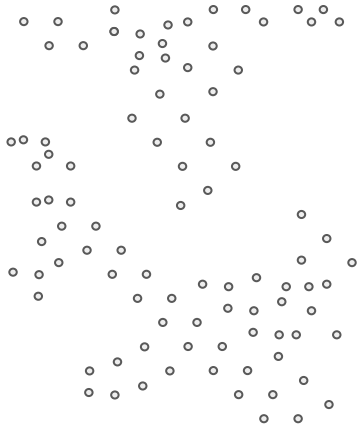
Avg. Regret(Alg. **E**) =

$$1/T \left[\begin{array}{l} \text{Loss suffered by} \\ \textbf{E} \text{ in T rounds} \end{array} \right] - \begin{array}{l} \text{Loss of best } G \\ \text{(in hindsight)} \end{array}$$

Online learning

Theorem: Algorithm E enjoys $O(1/\sqrt{T})$ **expected average regret** provided distance metric $d(u,v)$ is "**smooth**".

i.e. $d(u,v)$ is distributed with κ-bounded density.



Online learning

Theorem: Algorithm E enjoys $O(1/\sqrt{T})$ **expected average regret** provided distance metric $d(u,v)$ is "**smooth**".

i.e. $d(u,v)$ is distributed with κ-bounded density.

κ-bounded

e.g. $N(\mu,\sigma^2)$

$\Rightarrow$

dispersion!

Online learning

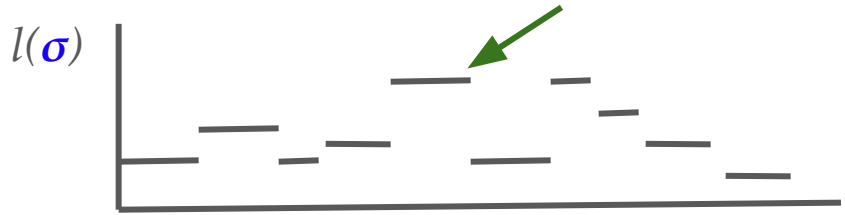
Theorem: Algorithm E enjoys $O(1/\sqrt{T})$ **expected average regret** provided distance metric $d(u,v)$ is "**smooth**".

i.e. $d(u,v)$ is distributed with κ-bounded density.

⇒ We (almost) learn the best possible graph; gap decreases with T

Speeding up the algorithm…

Challenge: Large number of pieces!!



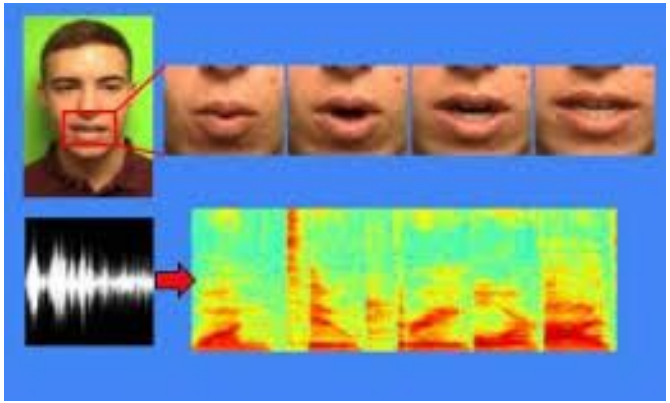
Solution: Just compute one piece – "**feedback set**" – in each round!

[Balcan, Dick, Pegden 2020]

1. Can be implemented in poly(n) time.
2. Can still achieve $O(1/\sqrt{T})$ **expected average regret**!!

Learning multi-parameter graphs

Multi-modal data – even more challenging to annotate!



Audio-visual Speech Recognition

d_1 metric: audio signals

d_2 metric: lip movement

$$d(u,v) = \Sigma_i \, \alpha_i \, d_i(u,v)$$

Lots of applications: Image captioning; video description; AVSR

Learning multi-parameter graphs (algorithms)

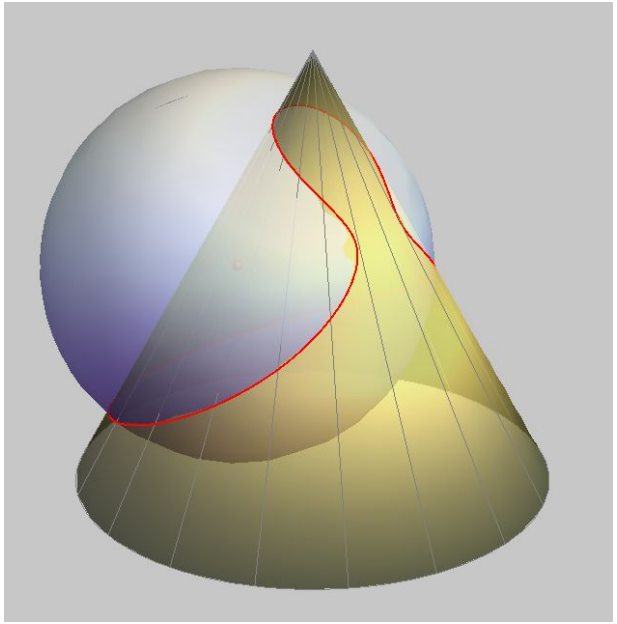
Challenge: Discontinuities lie along complex hypersurfaces (in parameter space)

$\mathbf{P(x,y,z) = 0}$, $\mathbf{P}$ polynomial in x,y and z
e.g. $x^2+y^2+yz=0$

Our results:

- $O(1/\sqrt{T})$ **expected average regret!**
- **general** results beyond semi-supervised learning
- tools from algebraic geometry

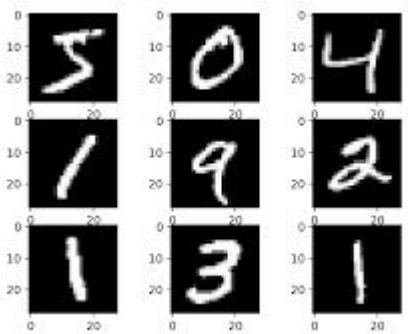
  [Tarski–Seidenberg theorem]



Real-world datasets

- Classifying handwritten digits, handwritten letters, pictures…

d(u,v) = Euclidean distance b/w pixel vectors



MNIST

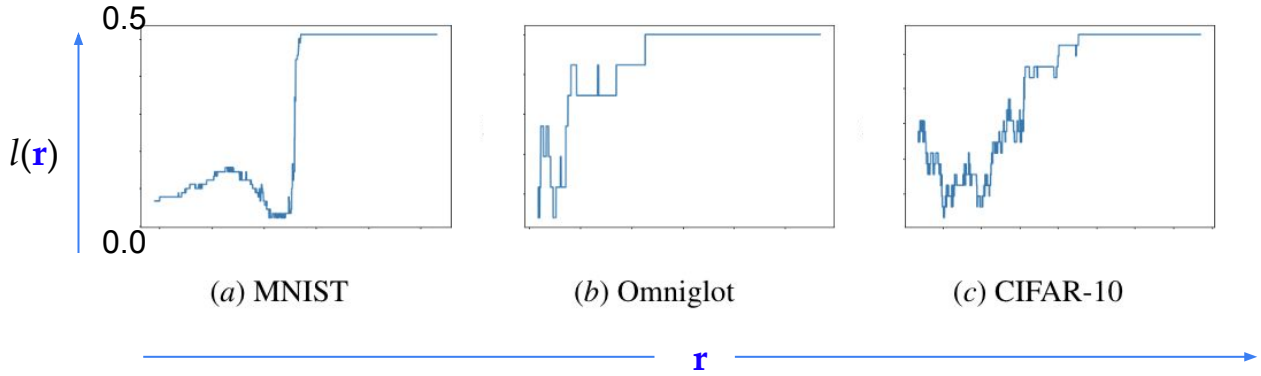
Omniglot

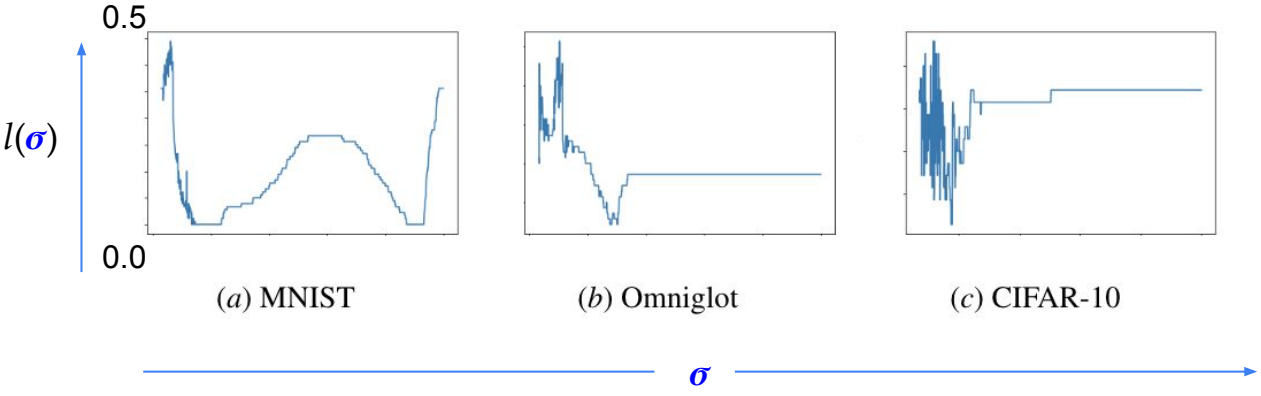
CIFAR-10

Real-world datasets (single problem instance)

Performance of $G(\sigma)$ strongly depends on σ

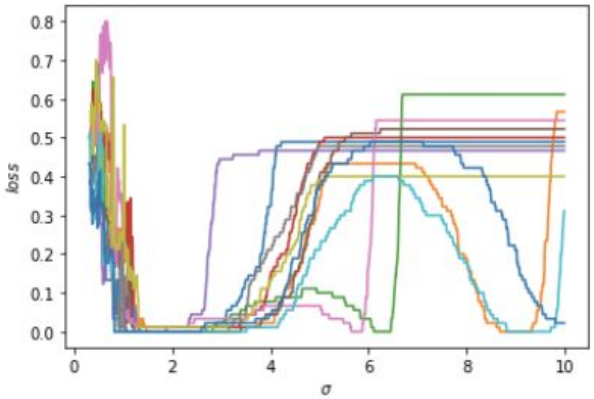
It is a piecewise constant dependence



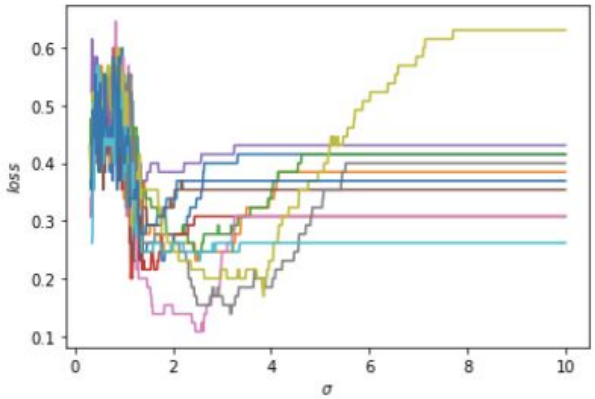


Recall:
Threshold graph $G(\mathbf{r})$
$w(u,v) = I[d(u,v) < \mathbf{r}]$
Gaussian $G(\sigma)$
$w(u,v) = \exp(-d(u,v)^2/\sigma^2)$

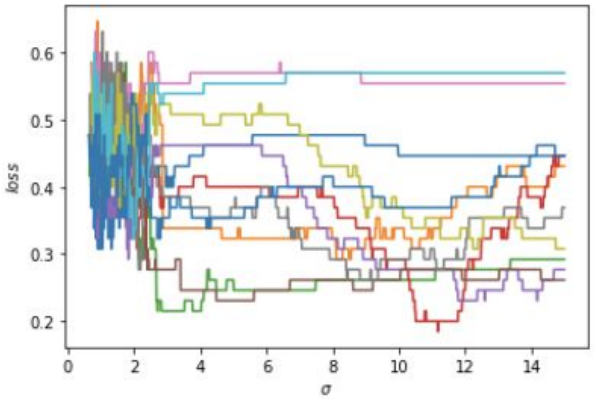
Randomly drawn problem instances



(a) MNIST



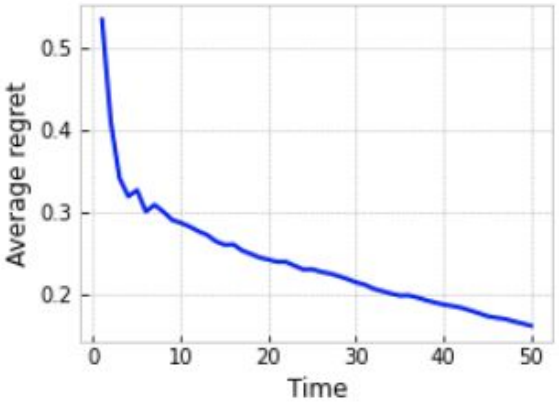
(b) Omniglot



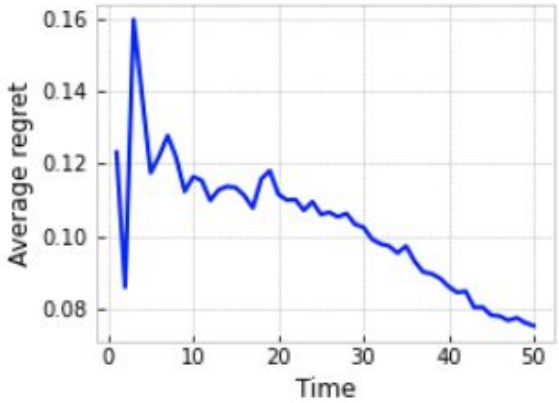
(c) CIFAR-10

Variation in optimal parameters!

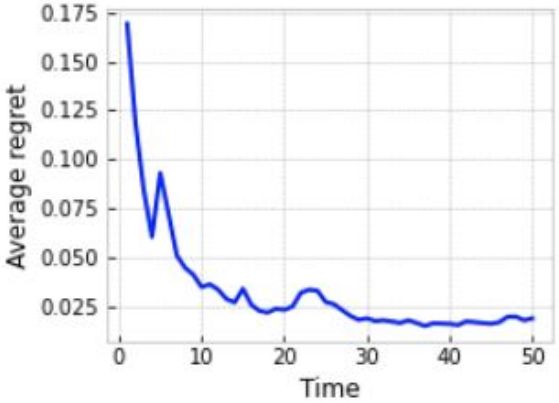
Randomly drawn problem instances



(a) MNIST



(b) Omniglot



(c) CIFAR-10

Our algorithm obtains low regret!

(as good as optimal graph)

Other aspects of online learning

- Handling distribution shifts via shifting regret [Balcan, Dick, Sharma (AISTATS 2020)]

Usual "static" regret

Avg. Regret(Alg. **E**) =

$$1/T \left[\text{Loss suffered by } \mathbf{E} \text{ in T rounds} \; - \; \text{Loss of best } G \text{ (in hindsight)} \right]$$

"shifting" regret

Avg. Regret(Alg. **E**) =

$$1/T \left[\text{Loss suffered by } \mathbf{E} \text{ in T rounds} \; - \; \text{Loss of best sequence } G_1, G_2, \ldots, G_k \text{ (up to } k \text{ shifts)} \right]$$

Other aspects of online learning

- Handling multiple tasks [Balcan, Khodak, Sharma, Talwalkar (NeurIPS 2021)]

Usual "single-task" regret

Avg. Regret(Alg. **E**) =

$$1/T \left[\text{Loss suffered by } \mathbf{E} \text{ in T rounds} \; - \; \text{Loss of best } G \text{ (in hindsight)} \right]$$

Average "multi-task" regret

Task Avg. Regret(Alg. **E**) =

$$1/m \left[1/T \left[\text{Loss suffered by } \mathbf{E} \text{ in T rounds} \; - \; \text{Loss of best } G_i \text{ (for task } i\text{)} \right] \right]$$

Open questions and research directions

- Other applications to tuning important hyperparameters and algorithms
- Focus on statistical complexity $\longrightarrow$ computationally efficient methods?
- Making currently used approaches in practice more structure-aware
- Beyond the worst-case complexity: distribution-dependent bounds

  e.g. [Balcan, Goyal, Sharma (2025)]
- More challenging high-dimensional and distributed settings
  - E.g. extend our model hyperparameter tuning result to multiple hyperparameters

References

[1] Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization." *Advances in neural information processing systems* 24 (2011).

[2] Feurer, Matthias, and Frank Hutter. *Hyperparameter optimization*. Springer International Publishing, 2019.

[3] Bischl, Bernd, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas et al. "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 13, no. 2 (2023): e1484.

[4] Franceschi, Luca, Michele Donini, Valerio Perrone, Aaron Klein, Cédric Archambeau, Matthias Seeger, Massimiliano Pontil, and Paolo Frasconi. "Hyperparameter Optimization in Machine Learning." *arXiv preprint arXiv:2410.22854* (2024).

[5] Mockus, Jonas. "The Bayesian approach to local optimization." In *Bayesian approach to global optimization: Theory and applications*, pp. 125-156. Dordrecht: Springer Netherlands, 1989.

[6] Srinivas, Niranjan, Andreas Krause, Sham Kakade, and Matthias Seeger. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design." In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1015-1022. Omnipress, 2010.

[7] Frazier, Peter I. "A tutorial on Bayesian optimization." *arXiv preprint arXiv:1807.02811* (2018).

[8] Maclaurin, Dougal, David Duvenaud, and Ryan Adams. "Gradient-based hyperparameter optimization through reversible learning." In *International conference on machine learning*, pp. 2113-2122. PMLR, 2015.

[9] Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." In *International conference on machine learning*, pp. 1126-1135. PMLR, 2017.

References

[10] Franceschi, Luca, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. "Bilevel programming for hyperparameter optimization and meta-learning." In International conference on machine learning, pp. 1568-1577. PMLR, 2018.

[11] Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. "Hyperband: A novel bandit-based approach to hyperparameter optimization." *Journal of Machine Learning Research* 18, no. 185 (2018): 1-52.

[12] Falkner, Stefan, Aaron Klein, and Frank Hutter. "BOHB: Robust and efficient hyperparameter optimization at scale." In *International conference on machine learning*, pp. 1437-1446. PMLR, 2018.

[13] Parker-Holder, Jack, Vu Nguyen, and Stephen J. Roberts. "Provably efficient online hyperparameter optimization with population-based bandits." *Advances in neural information processing systems* 33 (2020): 17200-17211.

[14] Gupta, Rishi, and Tim Roughgarden. "A PAC approach to application-specific algorithm selection." In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pp. 123-134. 2016.

[15] Maria-Florina Balcan. Data-Driven Algorithm Design. In Tim Roughgarden, editor, Beyond the Worst-Case Analysis of Algorithms. Cambridge University Press, 2020.

[16] Linial, Nathan, Yishay Mansour, and Noam Nisan. "Constant depth circuits, Fourier transform, and learnability." Journal of the ACM (JACM) (1993).

[17] Balcan, Maria-Florina, and Dravyansh Sharma. "Learning Accurate and Interpretable Decision Trees." In *Uncertainty in Artificial Intelligence*, pp. 288-307. PMLR, 2024. Extended version “Learning Accurate and Interpretable Tree-based Models” arXiv preprint arXiv:2405.15911.

[18] Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "DARTS: Differentiable Architecture Search." In *International Conference on Learning Representations*, 2019.

References

[19] Balcan, Maria-Florina, Anh Tuan Nguyen, and Dravyansh Sharma. "Sample complexity of data-driven tuning of model hyperparameters in neural networks with structured parameter-dependent dual function." *arXiv preprint arXiv:2501.13734* (2025).

[20] Balcan, Maria-Florina, Travis Dick, and Manuel Lang. "Learning to Link." In *International Conference on Learning Representation*. 2020.

[21] Balcan, Maria-Florina, Misha Khodak, Dravyansh Sharma, and Ameet Talwalkar. "Provably tuning the ElasticNet across instances." *Advances in Neural Information Processing Systems* 35 (2022): 27769-27782.

[22] Balcan, Maria-Florina, and Dravyansh Sharma. "Data driven semi-supervised learning." *NeurIPS* (2021): 14782-14794.

[23] Balcan, Maria-Florina, Travis Dick, and Ellen Vitercik. "Dispersion for data-driven algorithm design, online learning, and private optimization." In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 603-614. IEEE, 2018.

[24] Balcan, Maria-Florina, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. "How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design." *Symposium on Theory of Computing (STOC)*, 2021.

[25] Balcan, Maria-Florina, Anh Tuan Nguyen, and Dravyansh Sharma. "Algorithm Configuration for Structured Pfaffian Settings." *TMLR* (2025).

[26] Koch, Caleb, Carmen Strassle, and Li-Yang Tan. "Properly learning decision trees with queries is NP-hard." In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 2383-2407. IEEE, 2023.

[27] Lin, Jimmy, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. "Generalized and scalable optimal sparse decision trees." In *International conference on machine learning*, pp. 6150-6160. PMLR, 2020.

[28] Balcan, Maria-Florina, Saumya Goyal, and Dravyansh Sharma. "Distribution-dependent Generalization Bounds for Tuning Linear Regression Across Tasks." *arXiv preprint arXiv:2507.05084* (2025).

References

[29] Balcan, Maria-Florina, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. "Learning-to-learn non-convex piecewise-Lipschitz functions." *Advances in Neural Information Processing Systems* 34 (2021): 15056-15069.

[30] Balcan, Maria-Florina, Anh Nguyen, and Dravyansh Sharma. "New bounds for hyperparameter tuning of regression problems across instances." *Advances in Neural Information Processing Systems* 36 (2023): 80066-80078.

[31] Balcan, Maria-Florina, Christopher Seiler, and Dravyansh Sharma. "Accelerating ERM for data-driven algorithm design using output-sensitive techniques." *Advances in Neural Information Processing Systems* 37 (2024): 72648-72687.

[32] Sharma, Dravyansh, and Maxwell Jones. "Efficiently learning the graph for semi-supervised learning." In *Uncertainty in Artificial Intelligence*, 2023.

[33] Sharma, Dravyansh. "Data-driven algorithm design and principled hyperparameter tuning in machine learning." PhD dissertation, CMU (2024).

[34] Sharma, Dravyansh, and Arun Suggala. "Offline-to-online hyperparameter transfer for stochastic bandits." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 19, pp. 20362-20370. 2025.

[35] Balcan, Maria-Florina, Avrim Blum, Dravyansh Sharma, and Hongyang Zhang. "An analysis of robustness of non-lipschitz networks." *Journal of Machine Learning Research* 24, no. 98 (2023): 1-43.

[36] Sharma, Dravyansh, Maria-Florina Balcan, and Travis Dick. "Learning piecewise Lipschitz functions in changing environments." In *International Conference on Artificial Intelligence and Statistics*, pp. 3567-3577. PMLR, 2020.

[37] Hazan, Elad, Adam Klivans, and Yang Yuan. "Hyperparameter Optimization: A Spectral Approach." *ICLR* (2018).

[38] Bartlett, Peter, Piotr Indyk, and Tal Wagner. "Generalization bounds for data-driven numerical linear algebra." In *Conference on Learning Theory*, pp. 2013-2040. PMLR, 2022.