

Online Similarity Prediction of Networked Data from Known and Unknown Graphs

Claudio Gentile, Mark Herbster, Stephen Pasteris

Universita' dell'Insubria, University College London

13 June 2013

What is Learning?

Supervised Learning

- Given data $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_l, y_l)\}$, infer a function f such that $f(\mathbf{x}_i) \approx y_i$ for all possible instances \mathbf{x}_i .

Unsupervised Learning

- Given data $\mathcal{S} = \{\mathbf{x}_1, \dots, \mathbf{x}_l\}$, model the data. e.g:
 - Fit a probability distribution to the space of all possible instances.
 - Map the instances to a low dimensional manifold in the instance space, such that the mapped instance is close to the original instance.

Semi-Supervised Learning

- Given data $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_l, y_l), \mathbf{x}_{l+1}, \dots, \mathbf{x}_n\}$, infer a function f such that $f(\mathbf{x}_i) \approx y_i$ for all possible instances \mathbf{x}_i . Utilises both supervised and unsupervised methods.

What is Learning?

Batch Learning

- Learner is given the data set \mathcal{S} and then performs the learning task

Online Learning

- We are given an initial data set \mathcal{S} .
- Learning proceeds in rounds. On each round:
 - 1 Learner is queried (with some instance).
 - 2 Correct answer is given, which updates the data set \mathcal{S} .

In this paper we focus on online, semi-supervised learning.

Classification

Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:					
Outcome:					
Mistakes:					

Classification


Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:					
Outcome:					
Mistakes:					

Classification


Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart				
Outcome:					
Mistakes:					

Classification


Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart				
Outcome:	Bart				
Mistakes:					

Classification


Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart				
Outcome:	Bart				
Mistakes:	0				

Classification



Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart				
Outcome:	Bart				
Mistakes:	0				

Classification



Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart	Bart			
Outcome:	Bart				
Mistakes:	0				

Classification



Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart	Bart			
Outcome:	Bart	Lisa			
Mistakes:	0				

Classification



Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart	Bart			
Outcome:	Bart	Lisa			
Mistakes:	0	1			

Classification




Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart	Bart			
Outcome:	Bart	Lisa			
Mistakes:	0	1			

Classification






Model

On each round:

- Nature presents an instance.
- Learner predicts the class.
- Nature reveals the class.

Aim: Minimize mistakes

Classification

Instance:					
Prediction:	Bart	Bart	Lisa	Maggie	Maggie
Outcome:	Bart	Lisa	Lisa	Maggie	Bart
Mistakes:	0	1	1	1	2

An Example Mistake Bound - The Halving Algorithm

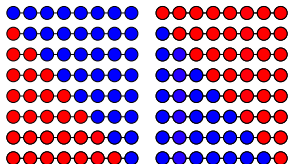
We have a labelled line graph with n vertices:



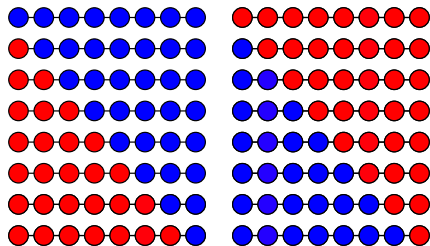
Suppose a priori:

- We know no labels
- We know that the cutsize is at most 1.

Let \mathcal{H} be the set of all $2n$ consistent classifiers:



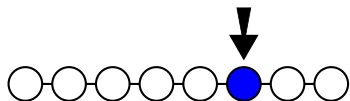
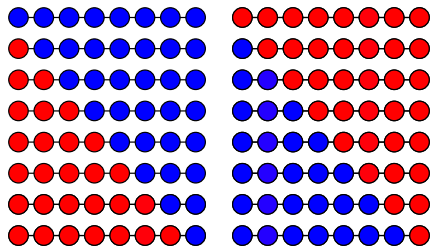
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red or Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

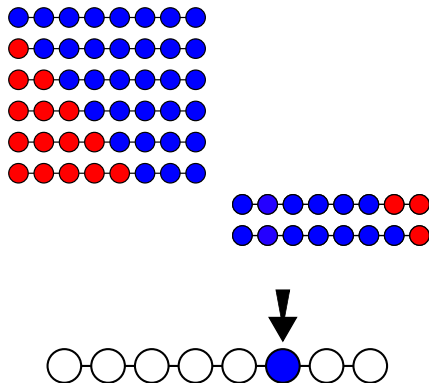
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red or Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

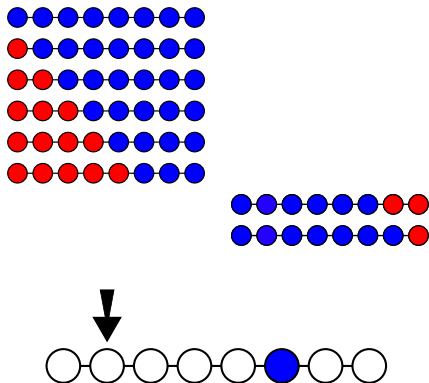
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red or Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

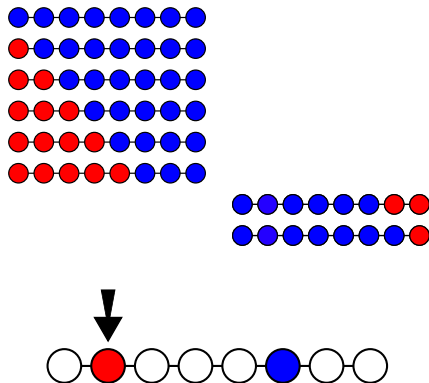
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red or Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

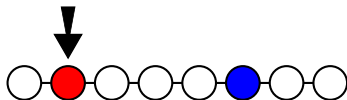
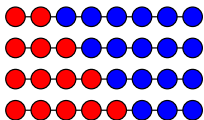
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red or Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

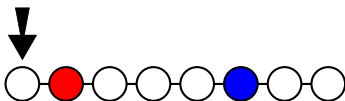
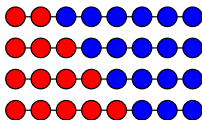
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red or Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

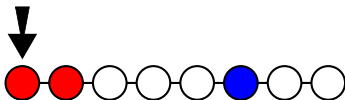
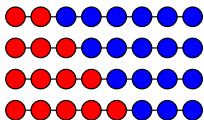
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red

Update: Remove inconsistent classifiers from \mathcal{H} .

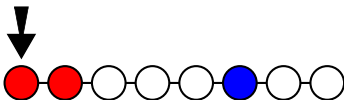
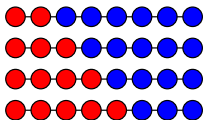
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red

Update: Remove inconsistent classifiers from \mathcal{H} .

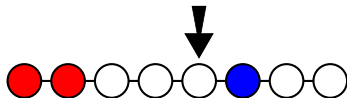
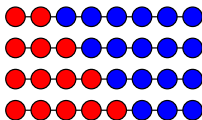
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red

Update: Remove inconsistent classifiers from \mathcal{H} .

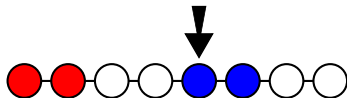
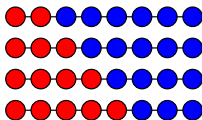
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

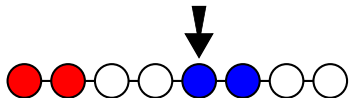
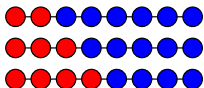
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

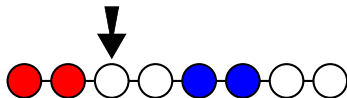
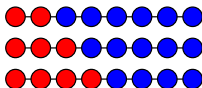
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Blue

Update: Remove inconsistent classifiers from \mathcal{H} .

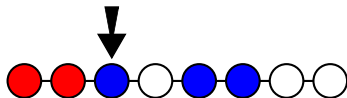
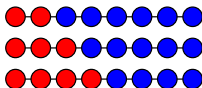
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red

Update: Remove inconsistent classifiers from \mathcal{H} .

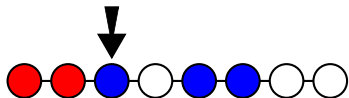
An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red

Update: Remove inconsistent classifiers from \mathcal{H} .

An Example Mistake Bound - The Halving Algorithm



Prediction: Majority vote amongst classifiers in \mathcal{H} : Red

Update: Remove inconsistent classifiers from \mathcal{H} .

An Example Mistake Bound - The Halving Algorithm

- $|\mathcal{H}|$ is initially $2n$.
- When a mistake is made at least half the classifiers are removed from \mathcal{H} .
- The correct classifier is never removed from \mathcal{H} so we always have $|\mathcal{H}| \geq 1$

Hence:

No more than $\log_2(2n)$ mistakes made.

Similarity

Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:					
Outcome:					
Mistakes:					

Similarity


Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:					
Outcome:					
Mistakes:					

Similarity


Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:	Similar				
Outcome:					
Mistakes:					

Similarity


Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:	Similar				
Outcome:	Similar				
Mistakes:					

Similarity


Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:	Similar				
Outcome:	Similar				
Mistakes:	0				

Similarity



Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:	Similar				
Outcome:	Similar				
Mistakes:	0				

Similarity






Model

On each round:

- Nature presents an instance pair.
- Learner predicts similarity of pair.
- Nature reveals similarity.

Aim: Minimize mistakes

Similarity

Instance:					
Prediction:	similar	disim.	disim.	similar	similar
Outcome:	similar	disim.	similar	similar	disim.
Mistakes:	0	0	1	1	2

Connection Between Classification and Similarity

Notation

- A **concept** y is a mapping from instances into K -classes.
- $\mathbb{B}_A(y)$ the maximal mistakes by algorithm A wrt concept y .

Theorem

Given classification algorithm C there exists similarity algorithm S such that for any concept y :

$$\mathbb{B}_S(y) \leq 5 \mathbb{B}_C(y) \log_2 K$$

Given similarity algorithm S there exists classification algorithm C such that for any concept y :

$$\mathbb{B}_C(y) \leq \mathbb{B}_S(y) + K$$

Construction requires exponential-time!

Recipe for a solution

Ingredients (basic)

- 1 Linear classifiers via “metric”-learning kernel [XNJR02,SSN04]
- 2 Online algs: Matrix Perceptron and Matrix Winnow [W07]

Ingredients (fancy) :

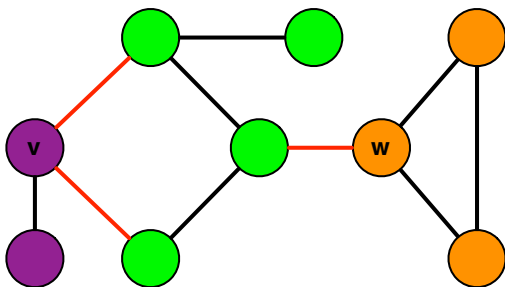
Aim: optimal mistake bounds or poly-log-time predictions

- 1 Prediction on a graph framework [CGVZ10,HLP09]
- 2 Expected mistake bound with **random spanning trees**
- 3 Linearization with **path graph** embedding
- 4 Reduced diameter and fast prediction with **binary support tree**

Results: Matrix winnow (**optimality**)
Matrix perceptron (**speed**)

Similarity prediction on a graph

- The graph is labeled by $y : \text{vertices} \rightarrow \{\text{purple}, \text{green}, \text{orange}\}$
- Instances are pairs of vertices, for example (v, w)

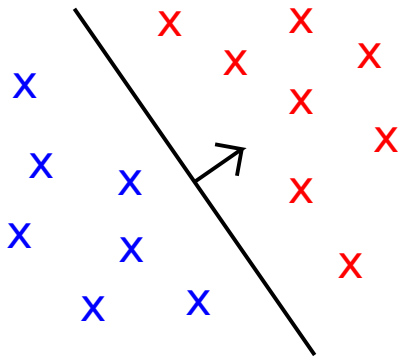


$$\Phi(y) = 3 \quad (\text{cut})$$

$$R(v, w) = 2 \quad (\text{eff. resistance})$$

$$\varphi_r(y) = 2\frac{1}{2} \quad (\text{eff. resistance-weighted cut})$$

Linear classification: Perceptron and Winnow



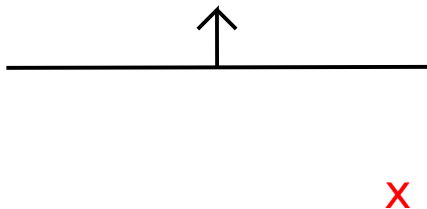
- We have a finite dimensional inner product space \mathcal{V} .
- Instances are vectors in \mathcal{V} .
- There exists a hyperplane \mathcal{H} which classifies instances.
- Goal: Learn \mathcal{H} .

Linear classification: Perceptron and Winnow



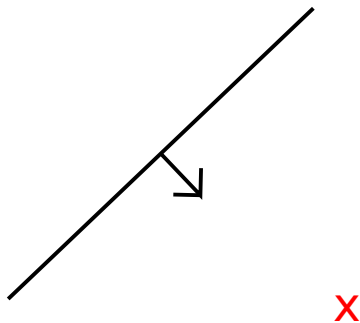
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



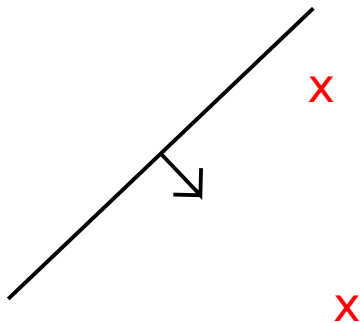
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



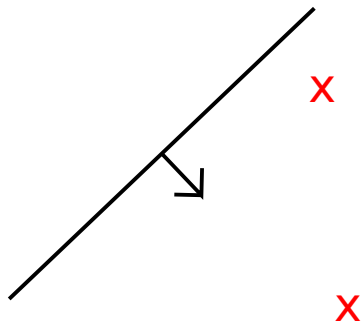
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



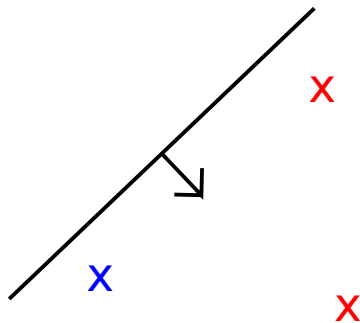
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



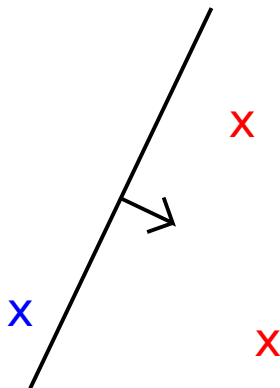
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



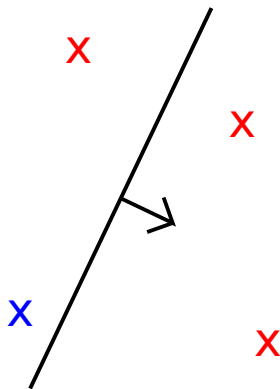
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



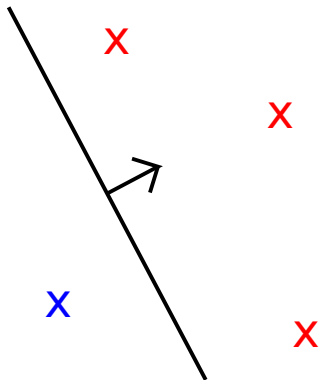
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



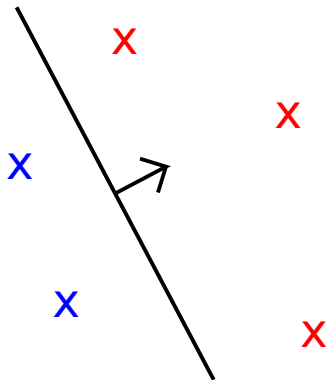
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



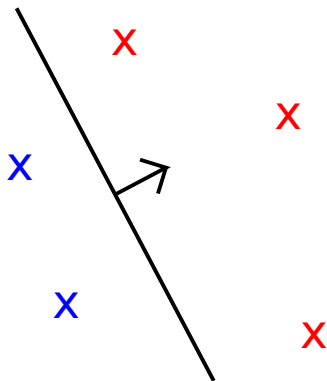
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



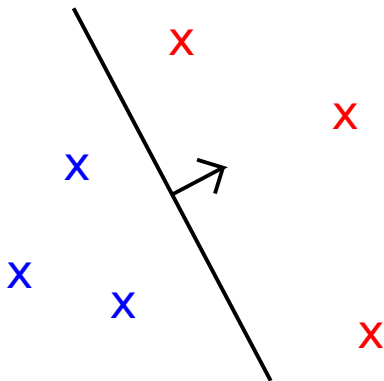
- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Linear classification: Perceptron and Winnow



- Predict according to \mathcal{H}
- If mistake is made then \mathcal{H} is updated according to the new instance.

Similarity prediction via linear classification

Inner-Product Space

\mathcal{V} is the space of $n \times n$ matrices with:

$$\langle A, B \rangle := \text{Trace}(A^T B) \quad (1)$$

Encoding

A pair of vertices (v, w) is encoded as the matrix:

$$\sqrt{L^+}(e_v - e_w)(e_v - e_w)^T \sqrt{L^+}$$

Graph Laplacian: L ; Basis vector: e_v

Similarity prediction via linear classification

Mistake bounds

$$M_W \leq \mathcal{O}([\phi(y)R^G] \log(n)) \quad (\text{Winnow})$$

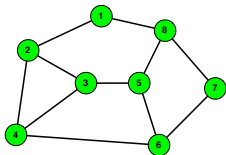
$$M_P \leq \mathcal{O}([\phi(y)R^G]^2) \quad (\text{Perceptron})$$

Resistance diameter: R^G ; Number of vertices: n

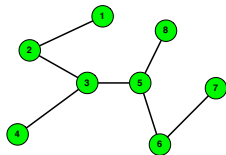
Graph approximation via a random BST

Construct: Random BST

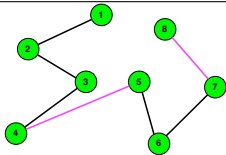
- 1 Bounds its resistance-weighted cut-size [$\mathbb{E}[\Phi^{G'}(y)] = \varphi_r^G(y)$]
- 2 Intermediate step [$\Phi^{G''}(y) \leq 2\Phi^{G'}(y)$]
- 3 Enables polylog time [$\Phi^{G'''}(y) \leq (\log n)\Phi^{G''}(y)$; $R^{G'''} = \log n$]
- 4 Hence $\mathbb{E}[\Phi^{G'''}(y)] \leq 2\varphi_r^G(y) \log n$



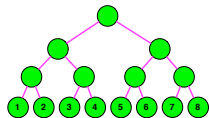
Original graph G (Step 0)



Sample random tree G' (Step 1)



Embed in path graph G'' (Step 2)

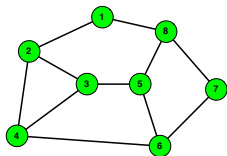


Build binary sup. tree G''' (Step 3)

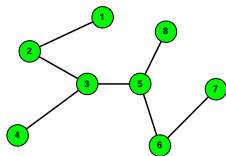
Graph approximation via a random BST

Construct: Random BST

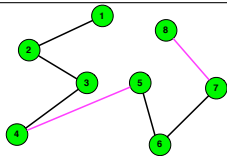
- 1 Bounds its resistance-weighted cut-size [$\mathbb{E}[\Phi^{G'}(y)] = \varphi_r^G(y)$]
- 2 Intermediate step [$\Phi^{G''}(y) \leq 2\Phi^{G'}(y)$]
- 3 Enables polylog time [$\Phi^{G'''}(y) \leq (\log n)\Phi^{G''}(y)$; $R^{G'''} = \log n$]
- 4 Hence $\mathbb{E}[\Phi^{G'''}(y)] \leq 2\varphi_r^G(y) \log n$



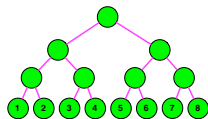
Original graph G (Step 0)



Sample random tree G' (Step 1)



Embed in path graph G'' (Step 2)

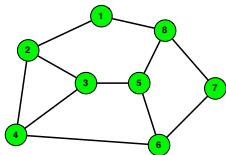


Build binary sup. tree G''' (Step 3)

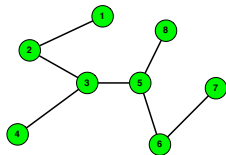
Graph approximation via a random BST

Construct: Random BST

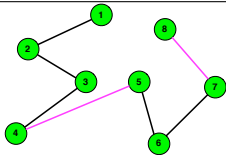
- 1 Bounds its resistance-weighted cut-size [$\mathbb{E}[\Phi^{G'}(y)] = \varphi_r^G(y)$]
- 2 Intermediate step [$\Phi^{G''}(y) \leq 2\Phi^{G'}(y)$]
- 3 Enables polylog time [$\Phi^{G'''}(y) \leq (\log n)\Phi^{G''}(y)$; $R^{G'''} = \log n$]
- 4 Hence $\mathbb{E}[\Phi^{G'''}(y)] \leq 2\varphi_r^G(y) \log n$



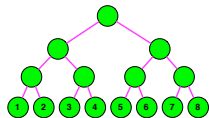
Original graph G (Step 0)



Sample random tree G' (Step 1)



Embed in path graph G'' (Step 2)

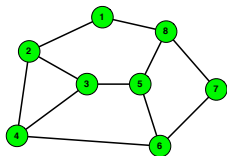


Build binary sup. tree G''' (Step 3)

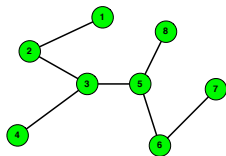
Graph approximation via a random BST

Construct: Random BST

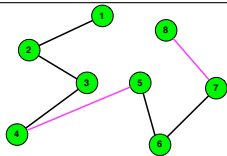
- 1 Bounds its resistance-weighted cut-size [$\mathbb{E}[\Phi^{G'}(y)] = \varphi_r^G(y)$]
- 2 Intermediate step [$\Phi^{G''}(y) \leq 2\Phi^{G'}(y)$]
- 3 Enables polylog time [$\Phi^{G'''}(y) \leq (\log n)\Phi^{G''}(y)$; $R^{G'''} = \log n$]
- 4 Hence $\mathbb{E}[\Phi^{G'''}(y)] \leq 2\varphi_r^G(y) \log n$



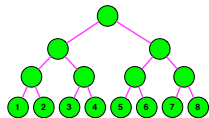
Original graph G (Step 0)



Sample random tree G' (Step 1)



Embed in path graph G'' (Step 2)



Build binary sup. tree G''' (Step 3)

An optimal algorithm (Matrix Winnow + random BST)

Theorem

The mistakes of Winnow + random BST is bounded above $\forall y$:

$$\mathbb{E}[M] \leq \mathcal{O}(\varphi_r(y) \log^3(n))$$

Direct implementation requires $\mathcal{O}(n^3)$ time per round.

A fast algorithm (Matrix Perceptron + random BST)

Theorem

The mistakes of Perceptron + random BST is bounded $\forall y$ by

$$\mathbb{E}[M] \leq \mathcal{O}(\varphi_r(y)^2 \log^4(n))$$

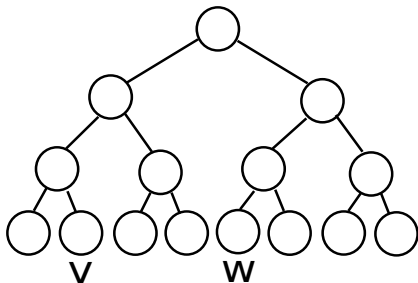
There exists an $\mathcal{O}(\log^2 n)$ time per round implementation

- An **exponentially** faster per-round prediction •

Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

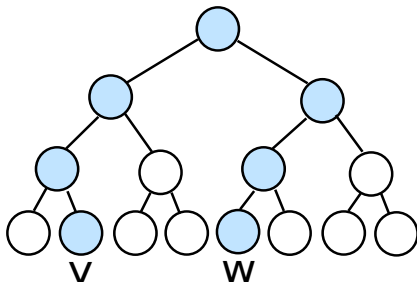
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

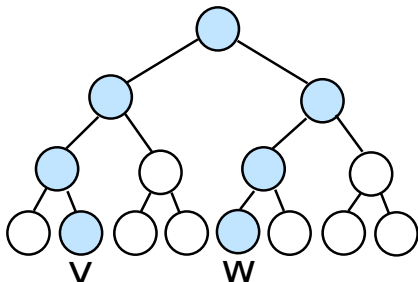
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

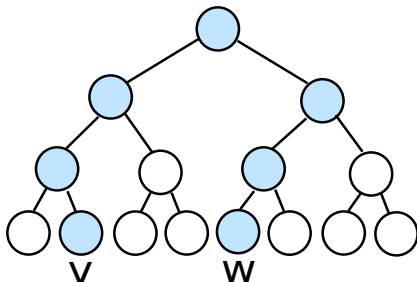
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

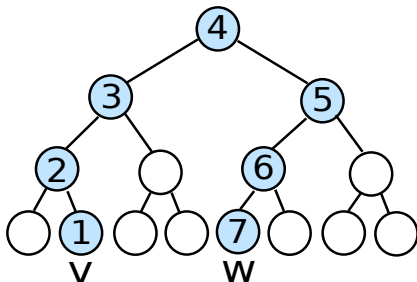
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

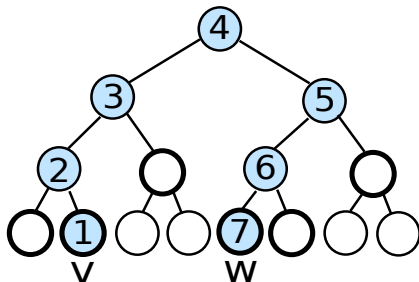
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

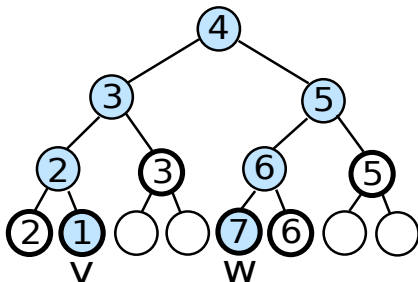
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

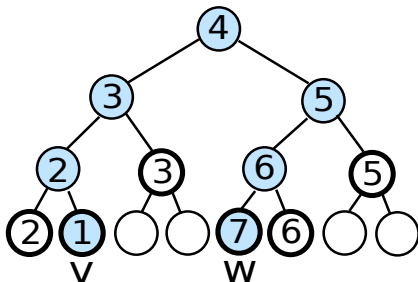
- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



Fast perceptron prediction – sketch

Fast perceptron ($\mathcal{O}(\log^2 n)$) per round

- **Receive:** vertex instance pair (v, w)
- Compute \mathcal{P} path from v to w (blue fill)
- **Predict:** $\hat{y} = \mathcal{I}[\sum_{i,j \in \mathcal{P}} F_{ij} > 4 \log^2 n]$
- **Receive:** “similarity” label y_t ('0' is similar/'1' is dissimilar)
- Compute f (circled numbers)
- Determine \mathcal{S} (bolded circles)
- Extend f to \mathcal{S}
- **Update:** $\forall i, j \in \mathcal{S}, F_{ij} \leftarrow F_{ij} + (2y_t - 1)(f_i - f_j)^2$



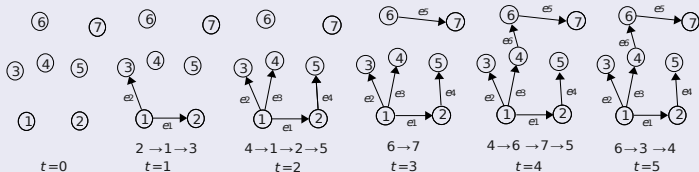
What if the graph is unknown?

Unknown graph

Model: progressive graph disclosure

- Nature presents a vertex pair & a path connecting the vertices
- Learner predicts similarity of pair.
- Nature reveals similarity.

Algorithm sketch



Theorem

There exists a p -norm perceptron-based algorithm such that

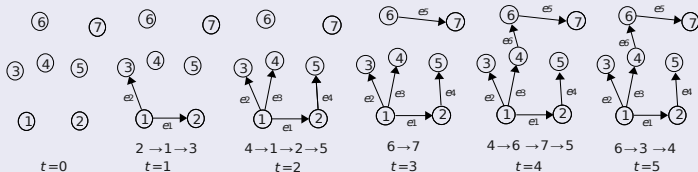
$$M \leq \mathcal{O}(\Phi(y)^4 \log(n))$$

Unknown graph

Model: progressive graph disclosure

- Nature presents a vertex pair & a path connecting the vertices
- Learner predicts similarity of pair.
- Nature reveals similarity.

Algorithm sketch



Theorem

There exists a p -norm perceptron-based algorithm such that

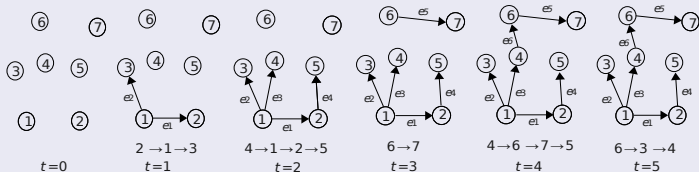
$$M \leq \mathcal{O}(\Phi(y)^4 \log(n))$$

Unknown graph

Model: progressive graph disclosure

- Nature presents a vertex pair & a path connecting the vertices
- Learner predicts similarity of pair.
- Nature reveals similarity.

Algorithm sketch



Theorem

There exists a p -norm perceptron-based algorithm such that

$$M \leq O(\Phi(y)^4 \log(n))$$

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

Summary

- Established an equivalence between classification and similarity
- Modeled as a graph labeling problem
- Designed a randomized BST graph-approximation
- Optimal to log-factors prediction with Matrix Winnow
- Fast poly-log-time prediction with the Matrix Perceptron
- Introduced a novel “unknown” graph framework

Future directions

- Weaken assumption: $\text{sim}(a, b) \ \& \ \text{sim}(b, c) \implies \text{sim}(a, c)$
- Structurally richer graph approximations with fast algorithms
- Tight lower and upper bounds in the unknown graph setting

References

- [CGVZ10] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. Random spanning trees and the prediction of weighted graphs. In *ICML 2010*.
- [HKS12] E. Hazan, S. Kale, and S. Shalev-Shwartz. Near-Optimal Algorithms for Online Matrix Prediction In *COLT*, 2002.
- [HLP09] M. Herbster, G. Lever, and M. Pontil. Online prediction on large diameter graphs. In *NIPS 2009*.
- [SSN04] S. Shalev-Shwartz, Y. Singer, and A. Ng. Online and batch learning of pseudo-metrics. In *ICML 2004*.
- [XNJR02] E. P. Xing, A. Y. Ng, M. I. Jordan, and J. Russell S. Distance metric learning with application to clustering with side-information. In *NIPS 2002*.
- [W07] M. K. Warmuth. Winoing subspaces. In *ICML 2007*.