

An overview of Boosting

Yoav Freund

UCSD

Plan of talk

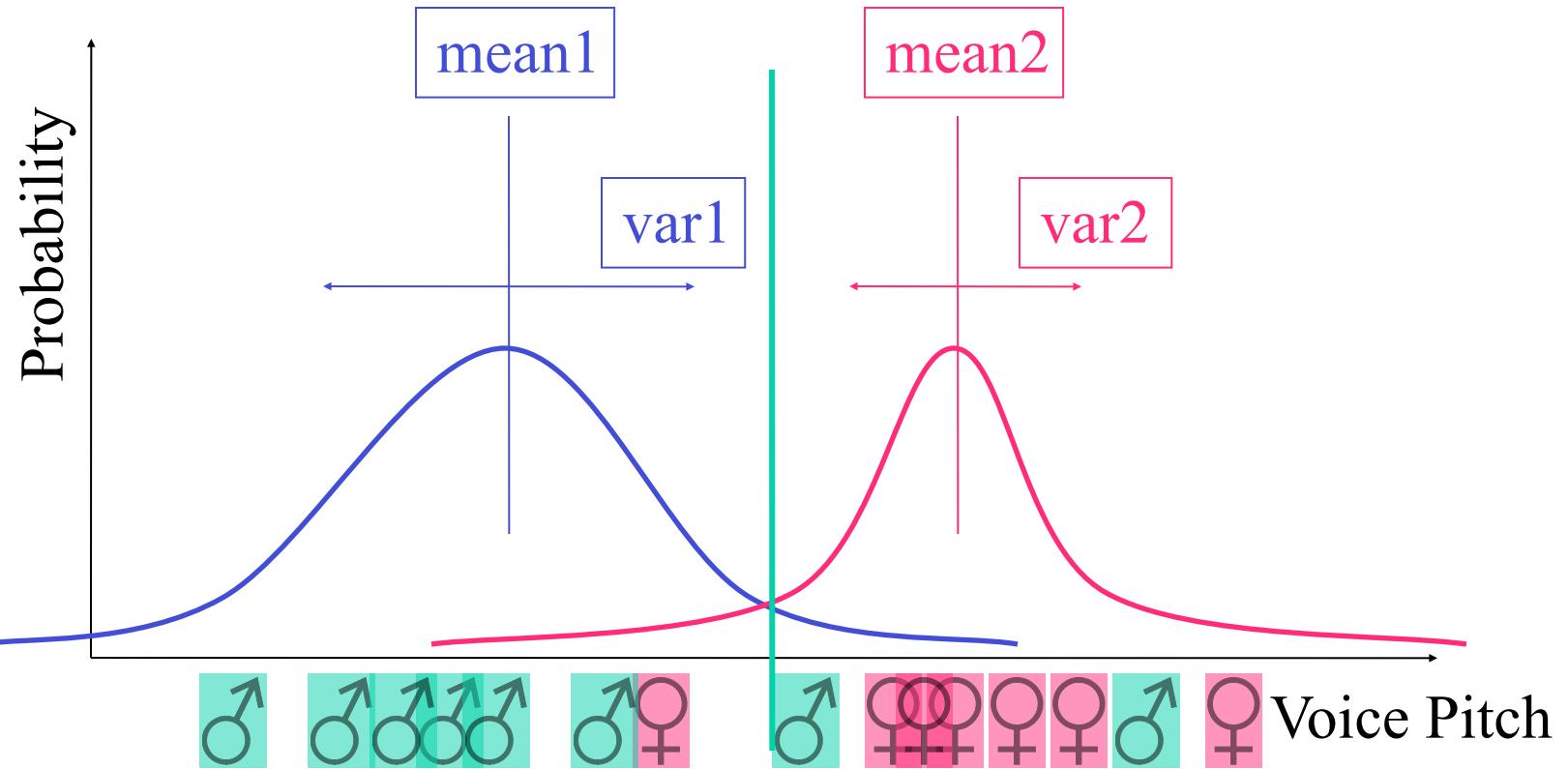
- Generative vs. non-generative modeling
- Boosting
- Alternating decision trees
- Boosting and over-fitting
- Applications

Toy Example

- Computer receives telephone call
- Measures Pitch of voice
- Decides gender of caller

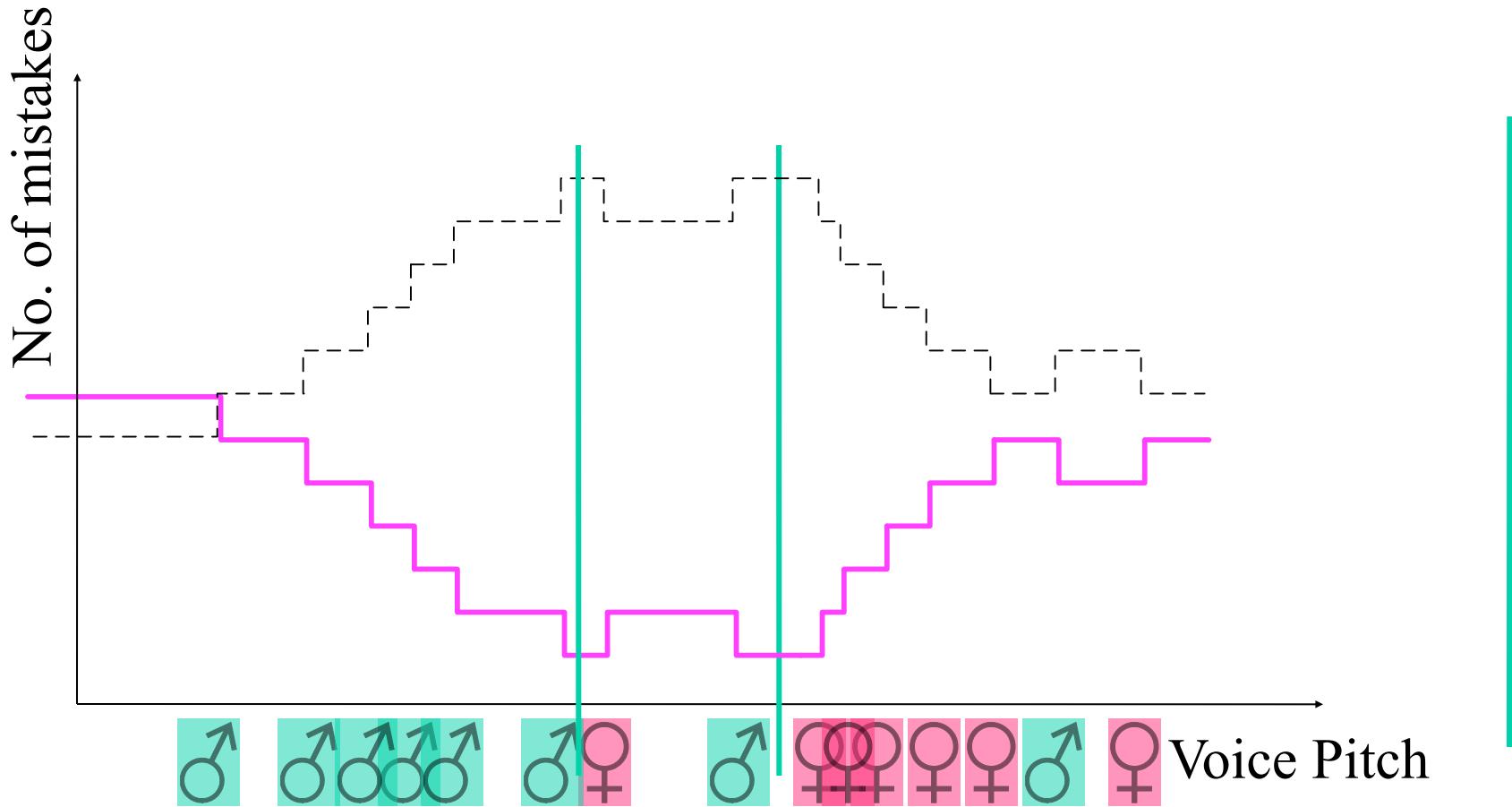


Generative modeling

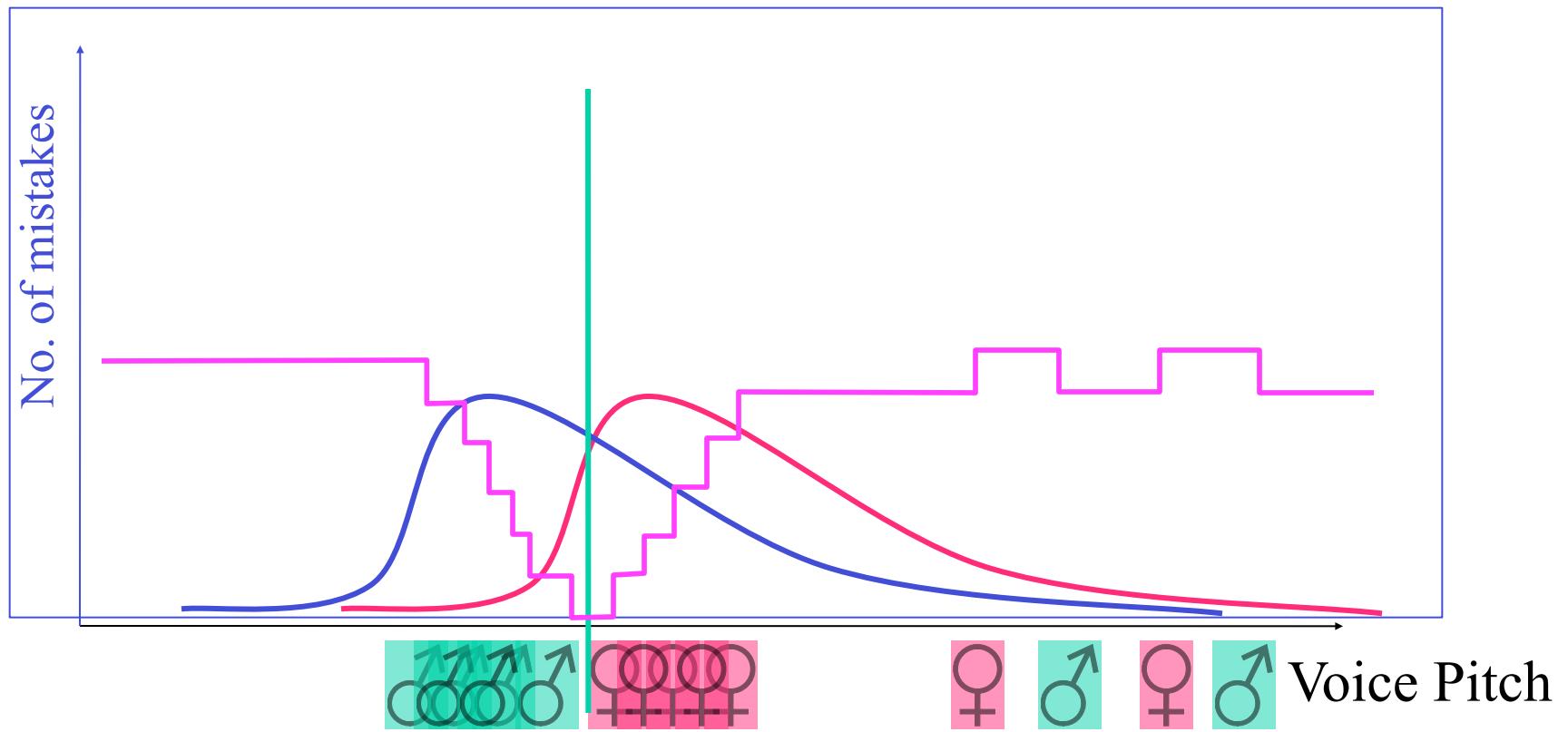


Discriminative approach

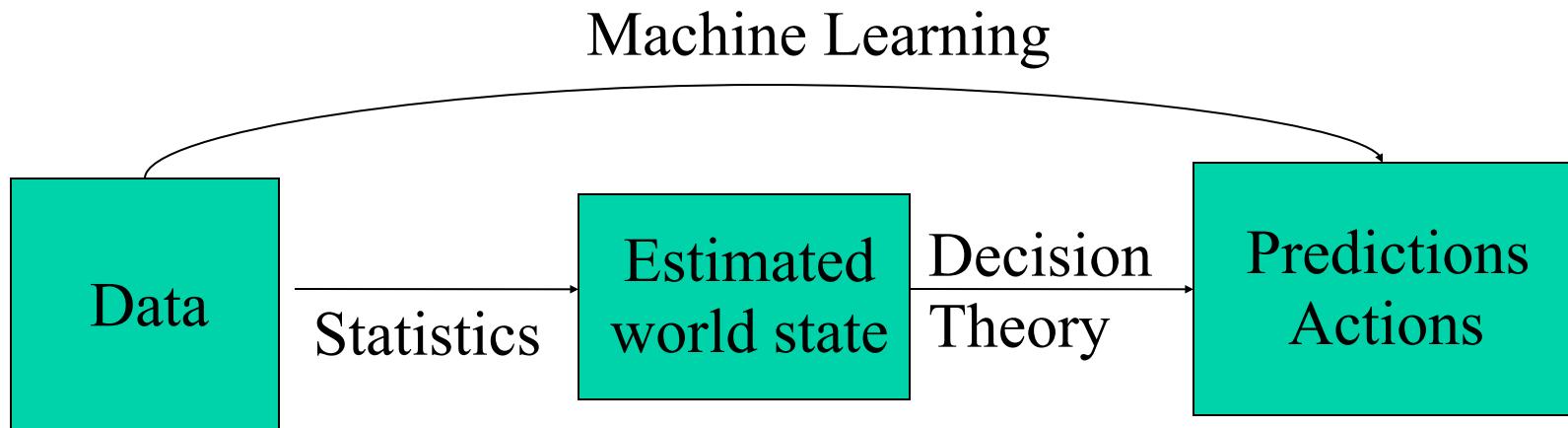
[Vapnik 85]



Ill-behaved data



Traditional Statistics vs. Machine Learning

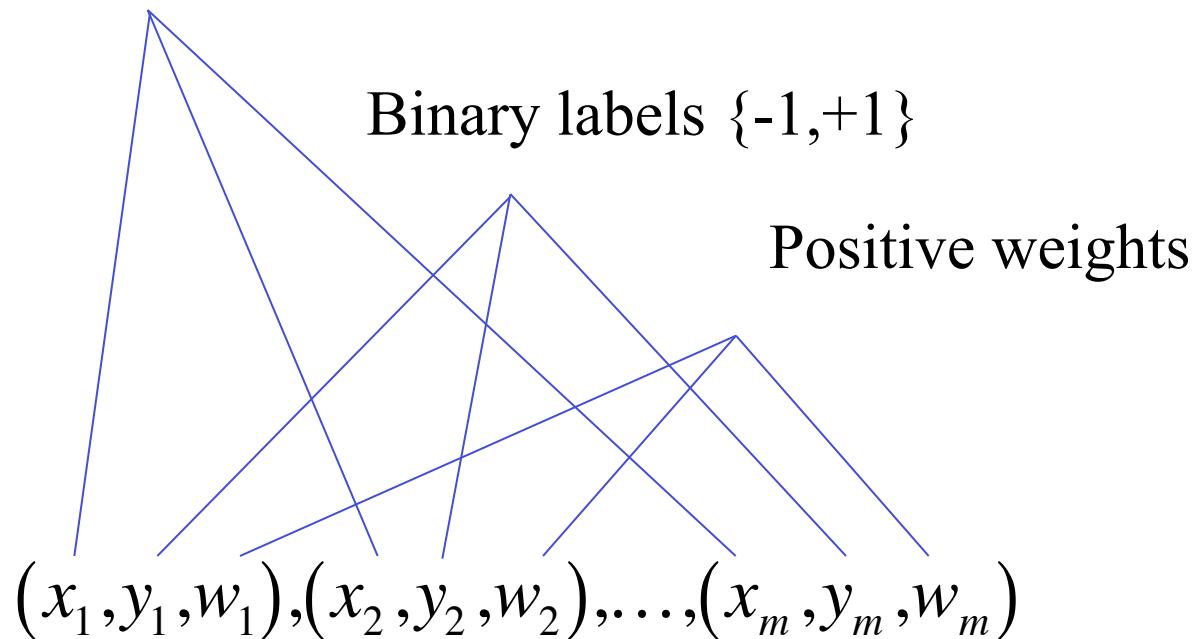


Comparison of methodologies

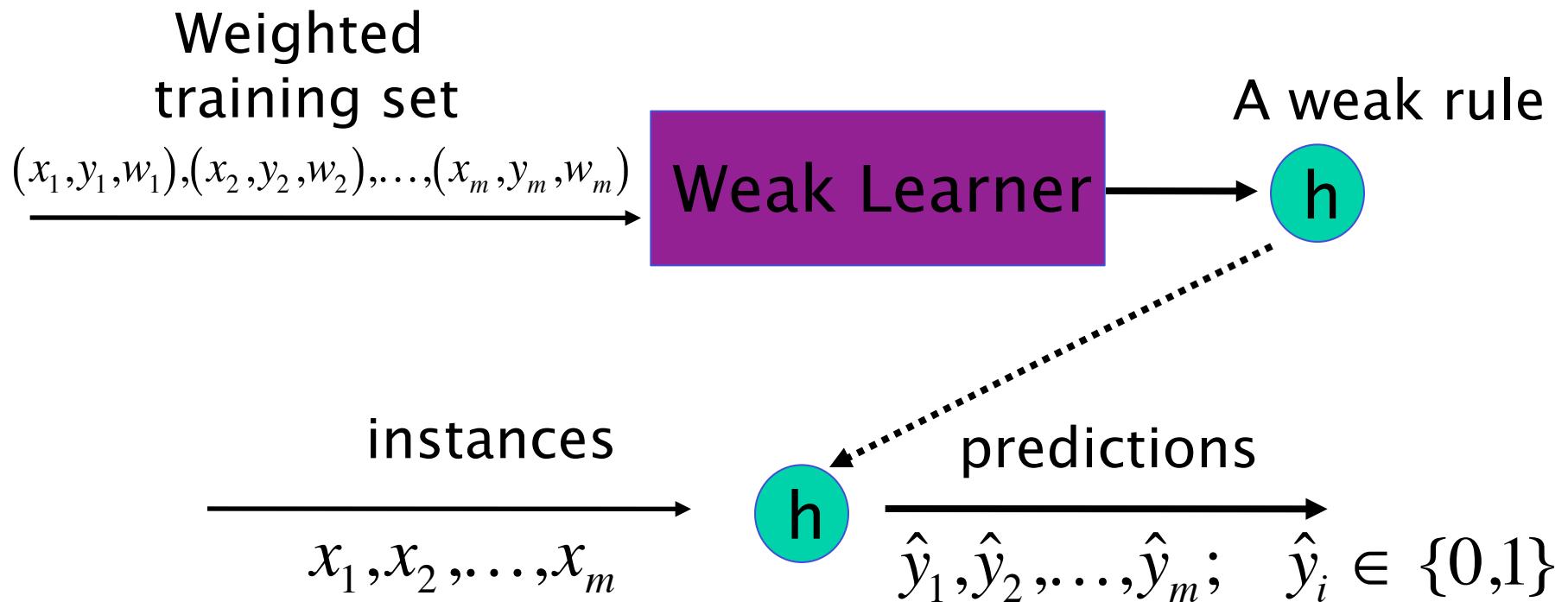
Model	Generative	Discriminative
Goal	Probability estimates	Classification rule
Performance measure	Likelihood	Misclassification rate
Mismatch problems	Outliers	Misclassifications

A weighted training set

Feature vectors



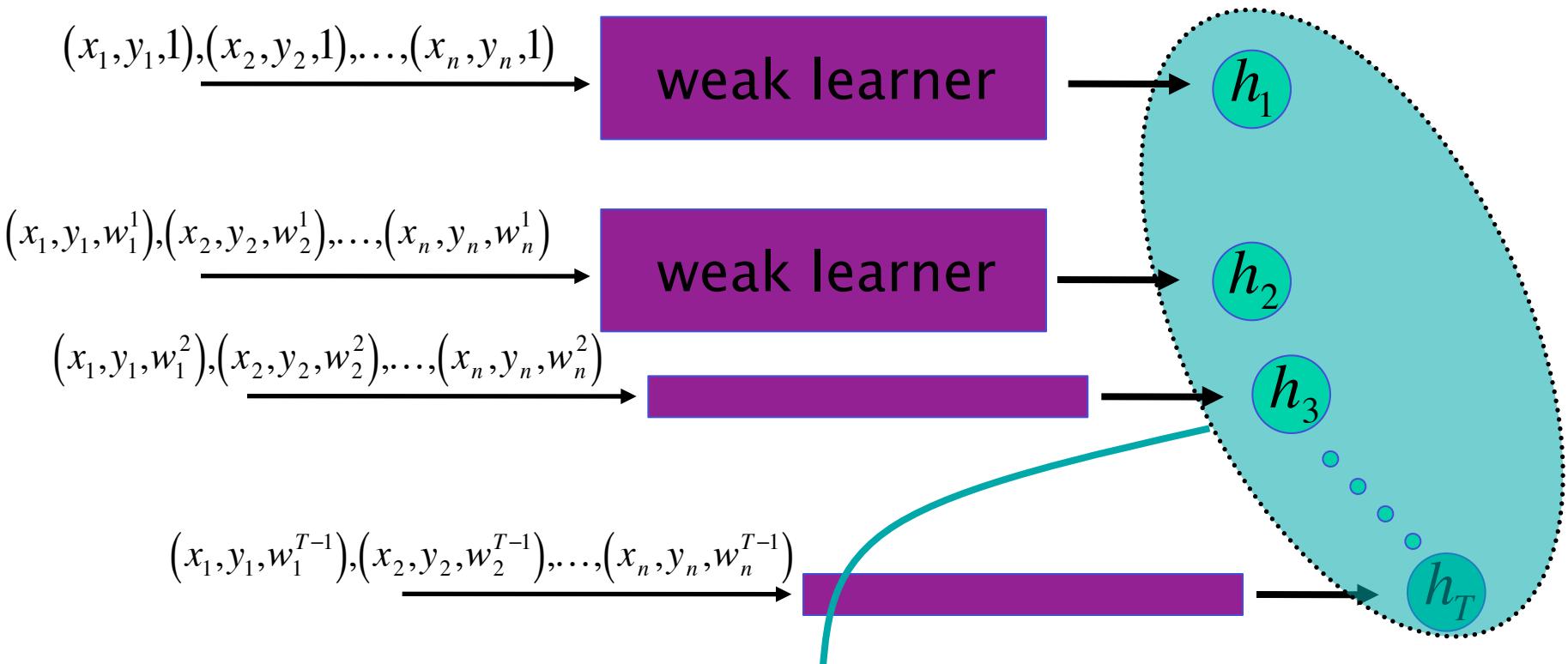
A weak learner



The weak requirement:

$$\left| \frac{\sum_{i=1}^m y_i \hat{y}_i w_i}{\sum_{i=1}^m w_i} \right| > \gamma > 0$$

The boosting process



$$F_T(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \dots + \alpha_T h_T(x)$$

Adaboost

Weak Rule: $h_t : X \rightarrow \{0,1\}$

Label: $y \in \{-1,+1\}$

Freund, Schapire 1997

Training set: $\{(x_1, y_1, 1), (x_2, y_2, 1), \dots, (x_n, y_n, 1)\}$

$$F_0(x) \equiv 0$$

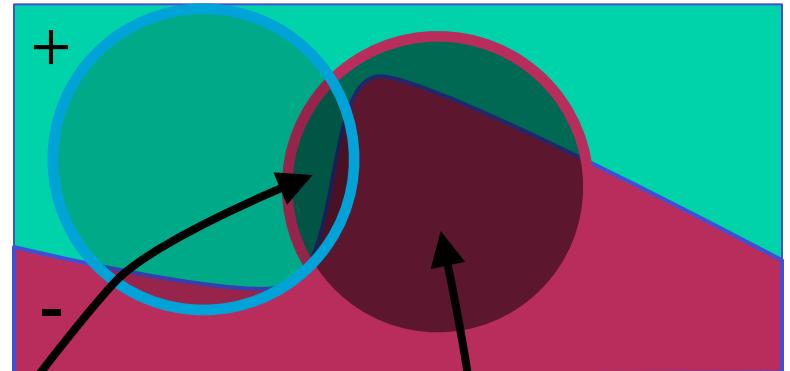
for $t = 1..T$

$$w_i^t = \exp(-y_i F_{t-1}(x_i))$$

Get h_t from *weak - learner*

$$\alpha_t = \frac{1}{2} \ln \left(\epsilon + \sum_{i:h_t(x_i)=1, y_i=1} w_i^t \right) / \left(\epsilon + \sum_{i:h_t(x_i)=1, y_i=-1} w_i^t \right)$$

$$F_t = F_{t-1} + \alpha_t h_t$$



A Formal Description of Boosting

[Slides from Rob Schapire]

- given **training set** $(x_1, y_1), \dots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \dots, T$:
 - construct distribution D_t on $\{1, \dots, m\}$
 - find **weak classifier** (“rule of thumb”)

$$h_t : X \rightarrow \{-1, +1\}$$

with small **error** ϵ_t on D_t :

$$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

- output **final classifier** H_{final}

AdaBoost

[Slides from Rob Schapire]

[with Freund]

- constructing D_t :

- $D_1(i) = 1/m$
- given D_t and h_t :

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i)) \end{aligned}$$

where Z_t = normalization factor

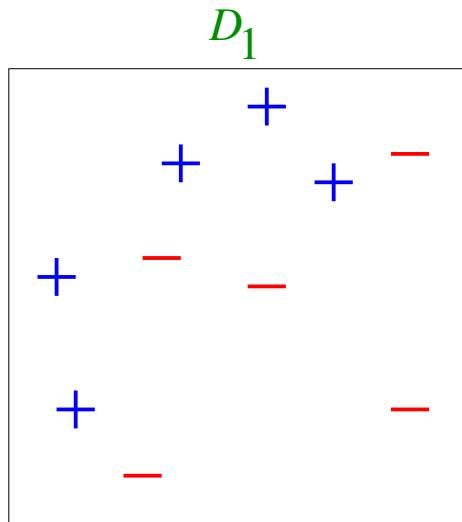
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

- final classifier:

- $H_{\text{final}}(x) = \text{sign} \left(\sum_t \alpha_t h_t(x) \right)$

Toy Example

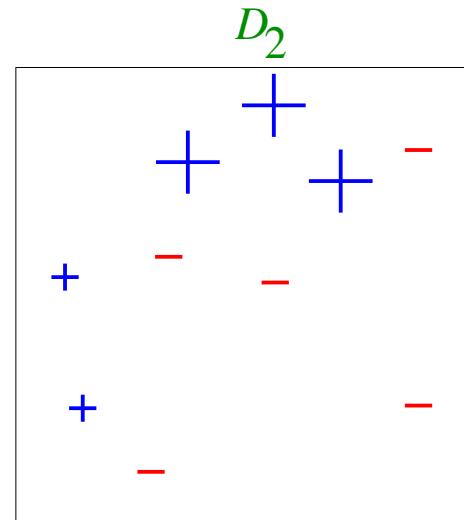
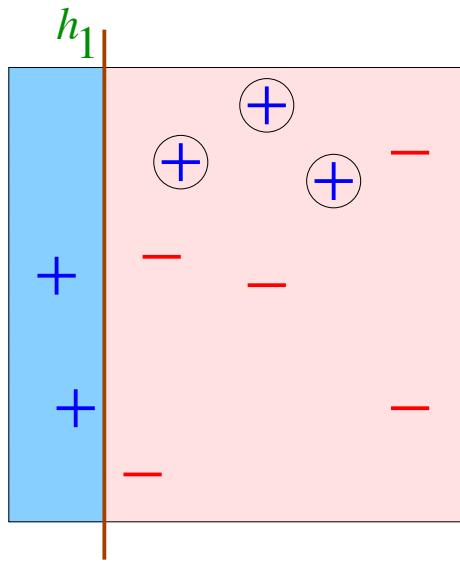
[Slides from Rob Schapire]



weak classifiers = vertical or horizontal half-planes

Round 1

[Slides from Rob Schapire]

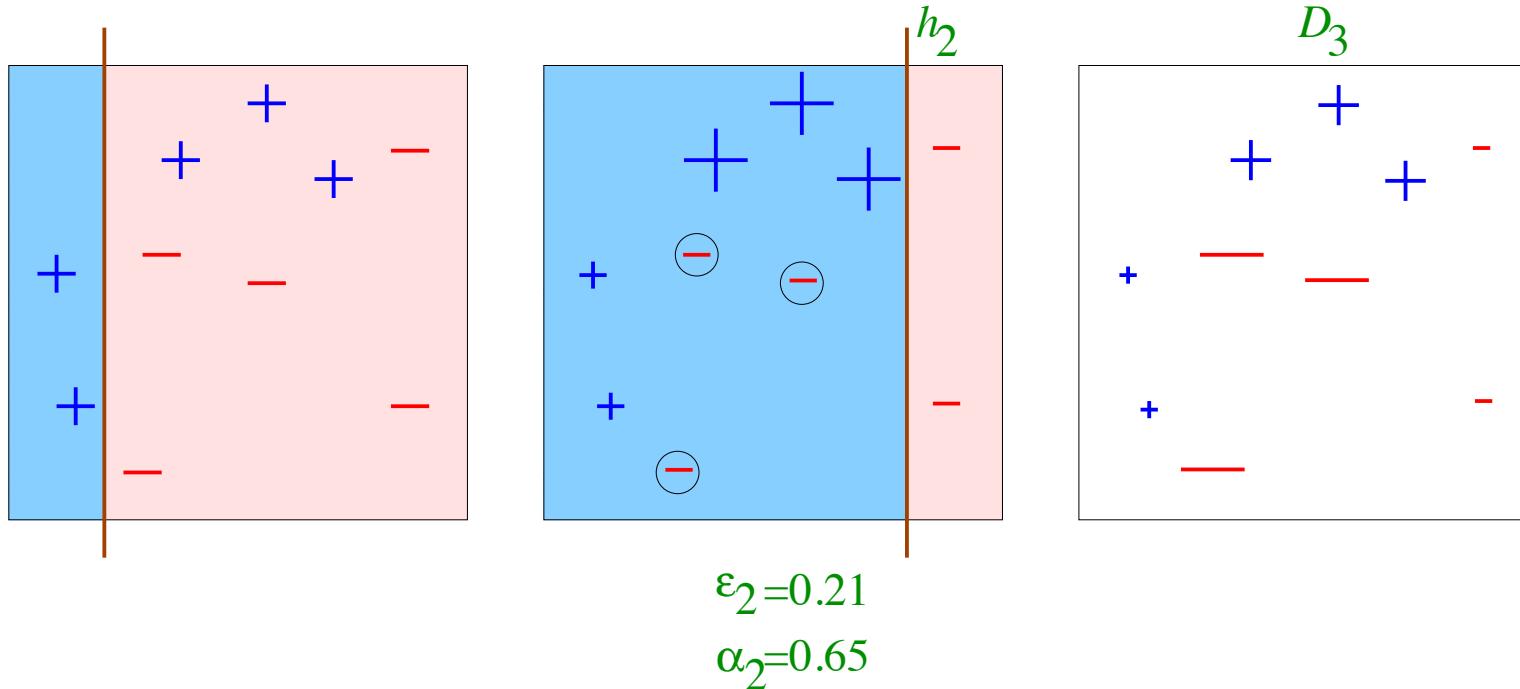


$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$

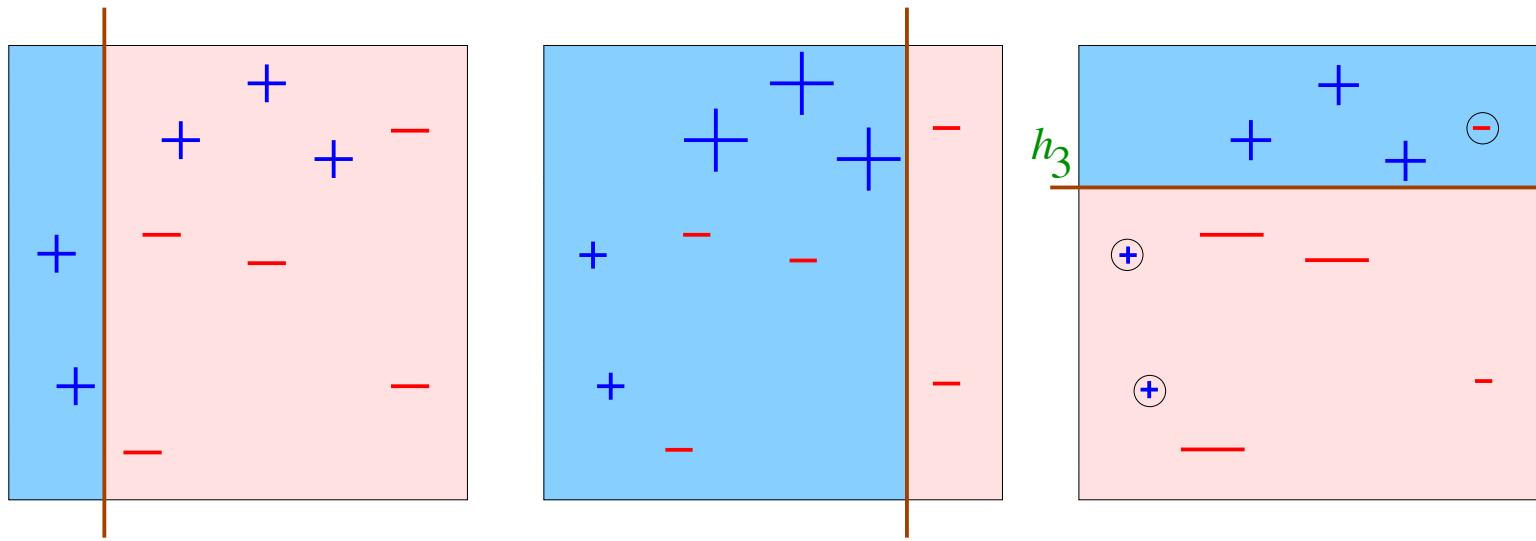
Round 2

[Slides from Rob Schapire]



Round 3

[Slides from Rob Schapire]

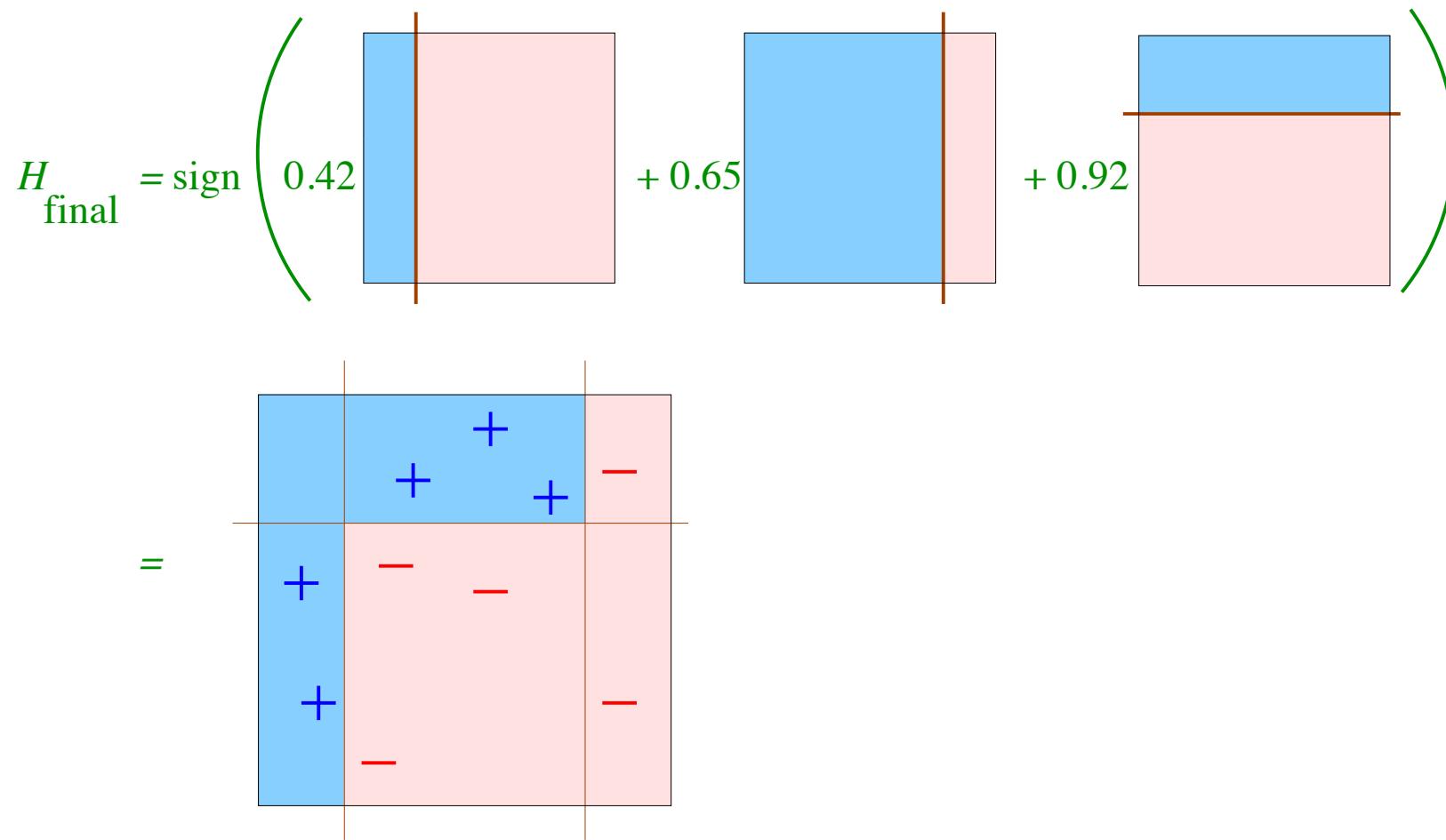


$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Final Classifier

[Slides from Rob Schapire]



Analyzing the Training Error

[Slides from Rob Schapire]

[with Freund]

- Theorem:

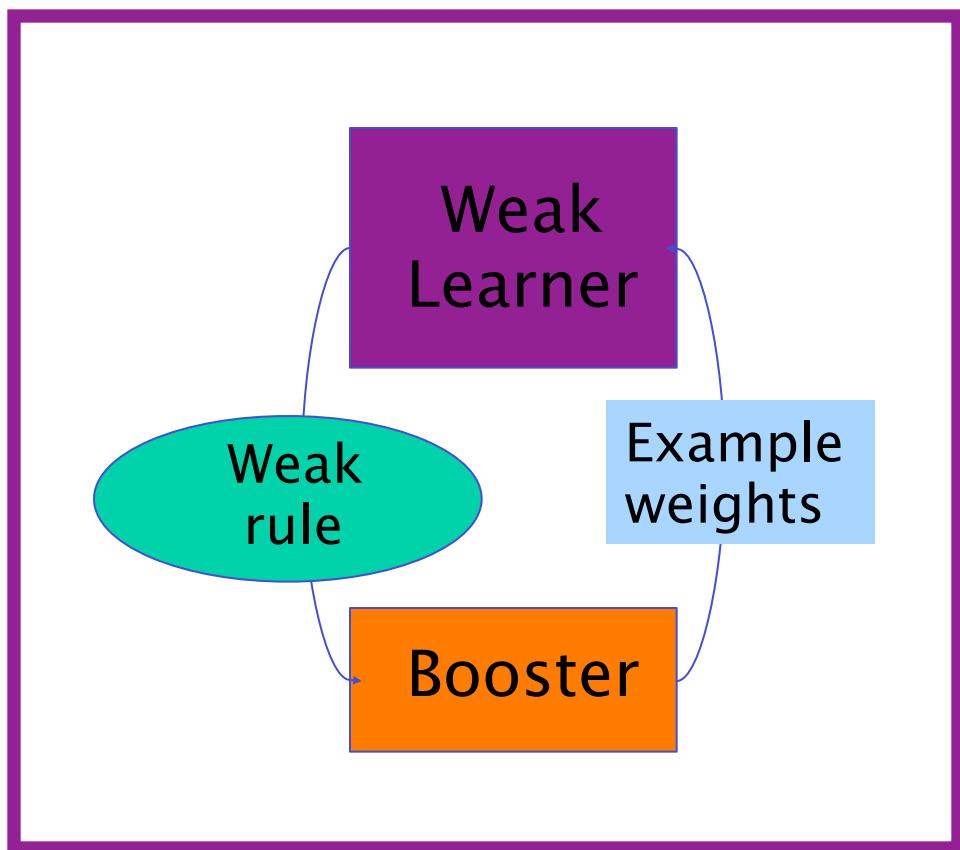
- write ϵ_t as $\frac{1}{2} - \gamma_t$ [γ_t = “edge”]
- then

$$\begin{aligned}\text{training error}(H_{\text{final}}) &\leq \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)} \right] \\ &= \prod_t \sqrt{1 - 4\gamma_t^2} \\ &\leq \exp \left(-2 \sum_t \gamma_t^2 \right)\end{aligned}$$

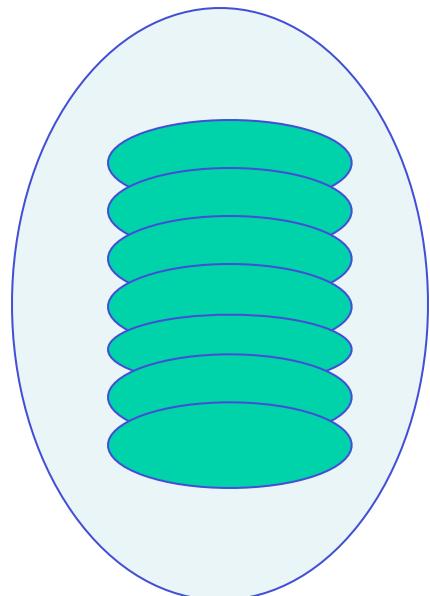
- so: if $\forall t : \gamma_t \geq \gamma > 0$
then $\text{training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$
- AdaBoost is adaptive:
 - does **not** need to know γ or T a priori
 - can exploit $\gamma_t \gg \gamma$

Boosting block diagram

Strong Learner

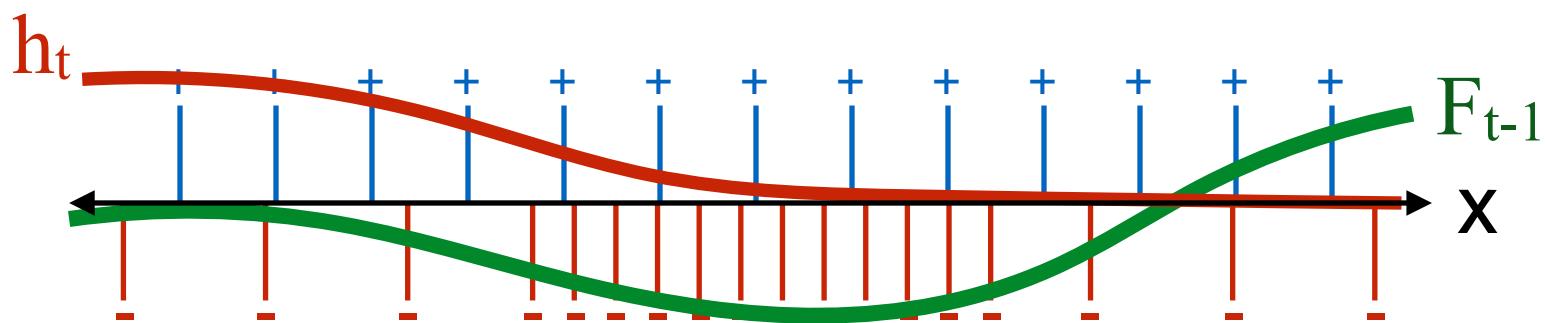


Accurate
Rule



AdaBoost as variational optimization.

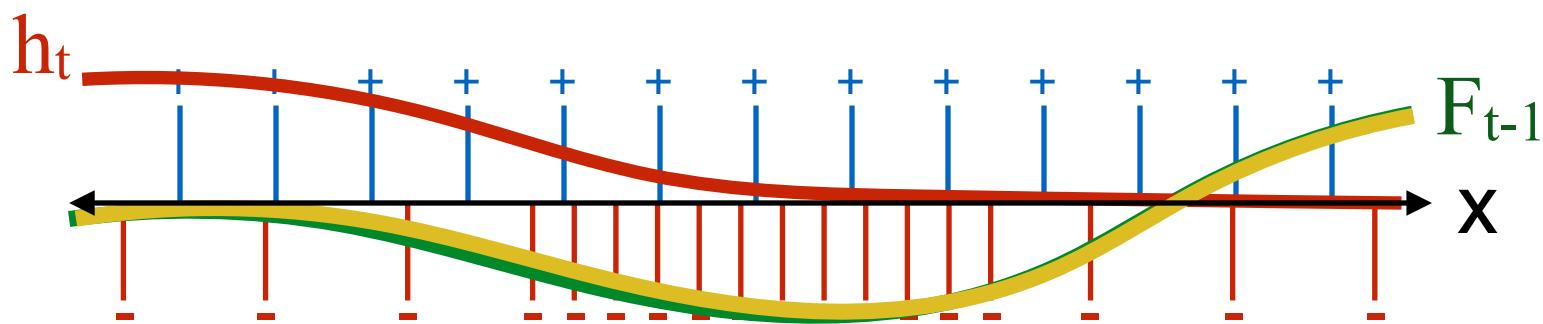
- x = input, a scalar,
- y = output, -1 or +1
- $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ = training set
- F_{t-1} = Strong rule after $t-1$ boosting iterations.
- h_t = Weak rule produced at iteration t



AdaBoost as variational optimization.

- F_{t-1} = Strong rule after t-1 boosting iterations.
- h_t = Weak rule produced at iteration t

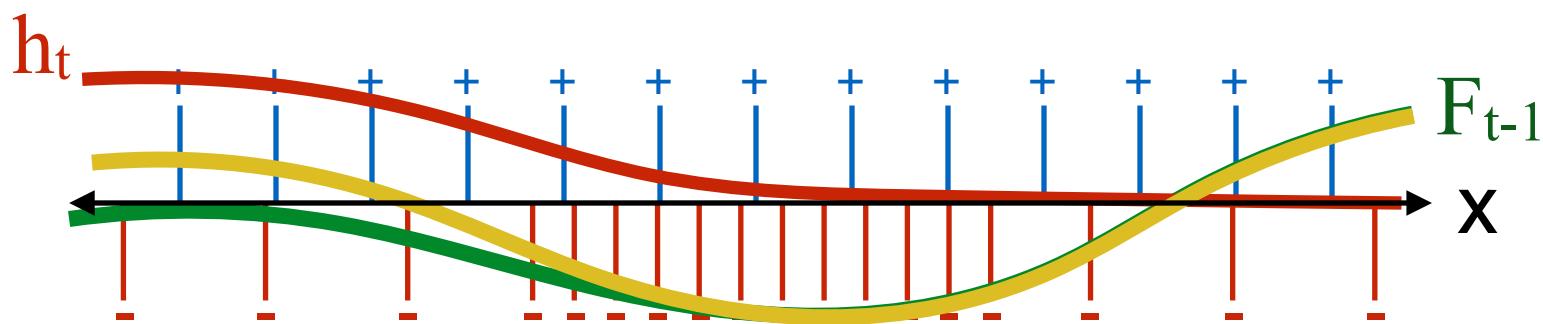
$$F_{t-1}(x) + \alpha h_t(x)$$



AdaBoost as variational optimization.

- F_{t-1} = Strong rule after t-1 boosting iterations.
- h_t = Weak rule produced at iteration t

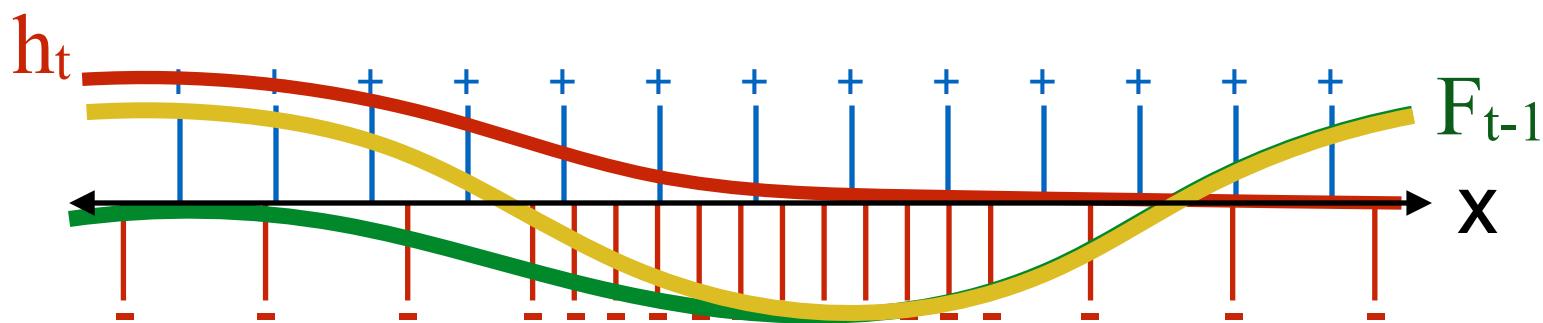
$$F_{t-1}(x) + 0.4h_t(x)$$



AdaBoost as variational optimization.

- F_{t-1} = Strong rule after t-1 boosting iterations.
- h_t = Weak rule produced at iteration t

$$F_t(x) = F_{t-1}(x) + 0.8h_t(x)$$



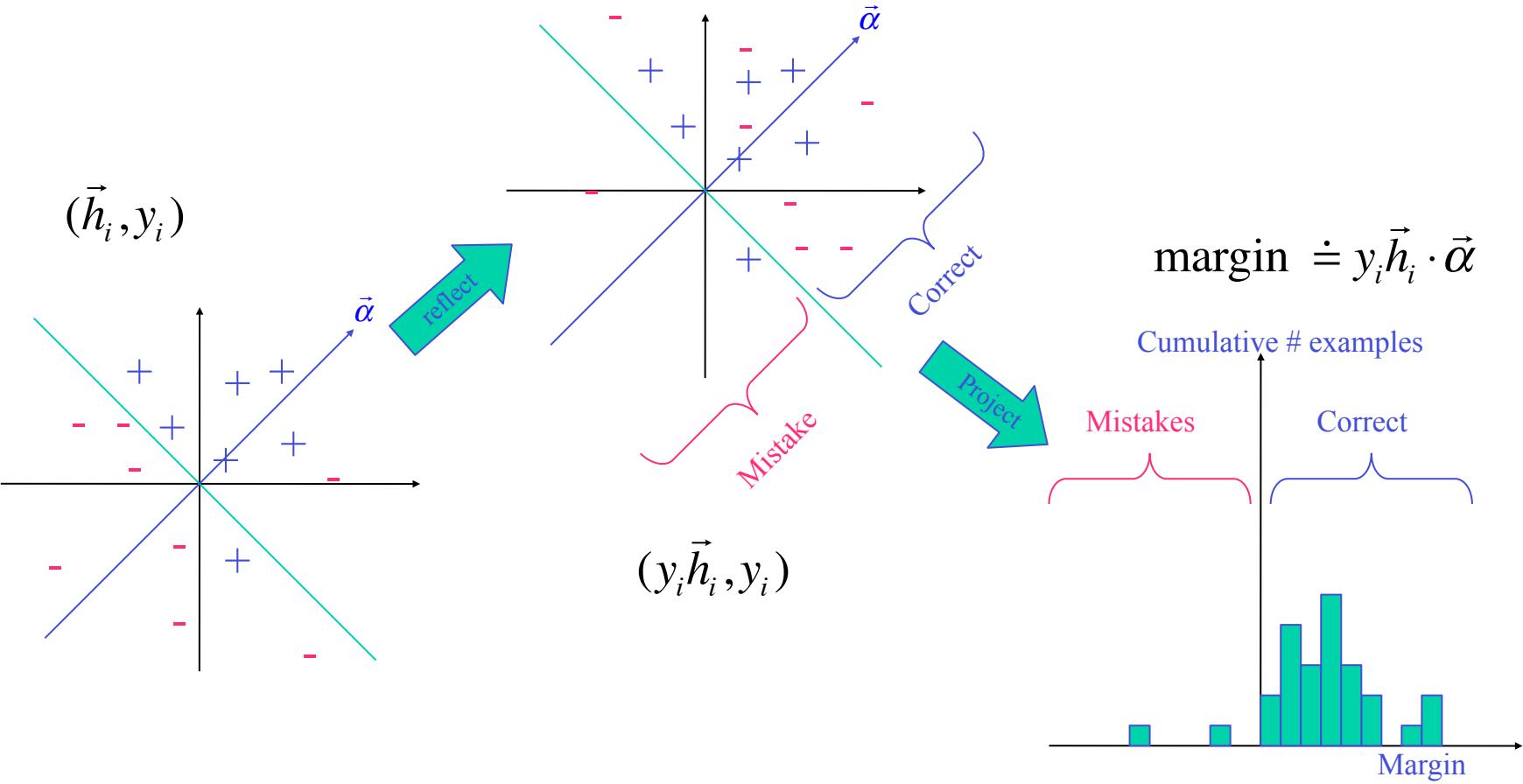
Margins

Fix a set of weak rules: $\vec{h}(x) = (h_1(x), h_2(x), \dots, h_T(x))$

Represent the i'th example x_i with outputs of the weak rules: \vec{h}_i

Labels: $y \in \{-1, +1\}$ Training set: $(\vec{h}_1, y_1), (\vec{h}_2, y_2), \dots, (\vec{h}_n, y_n)$

Goal : Find a weight vector $\vec{\alpha} = (\alpha_1, \dots, \alpha_T)$ that minimizes number of training mistakes

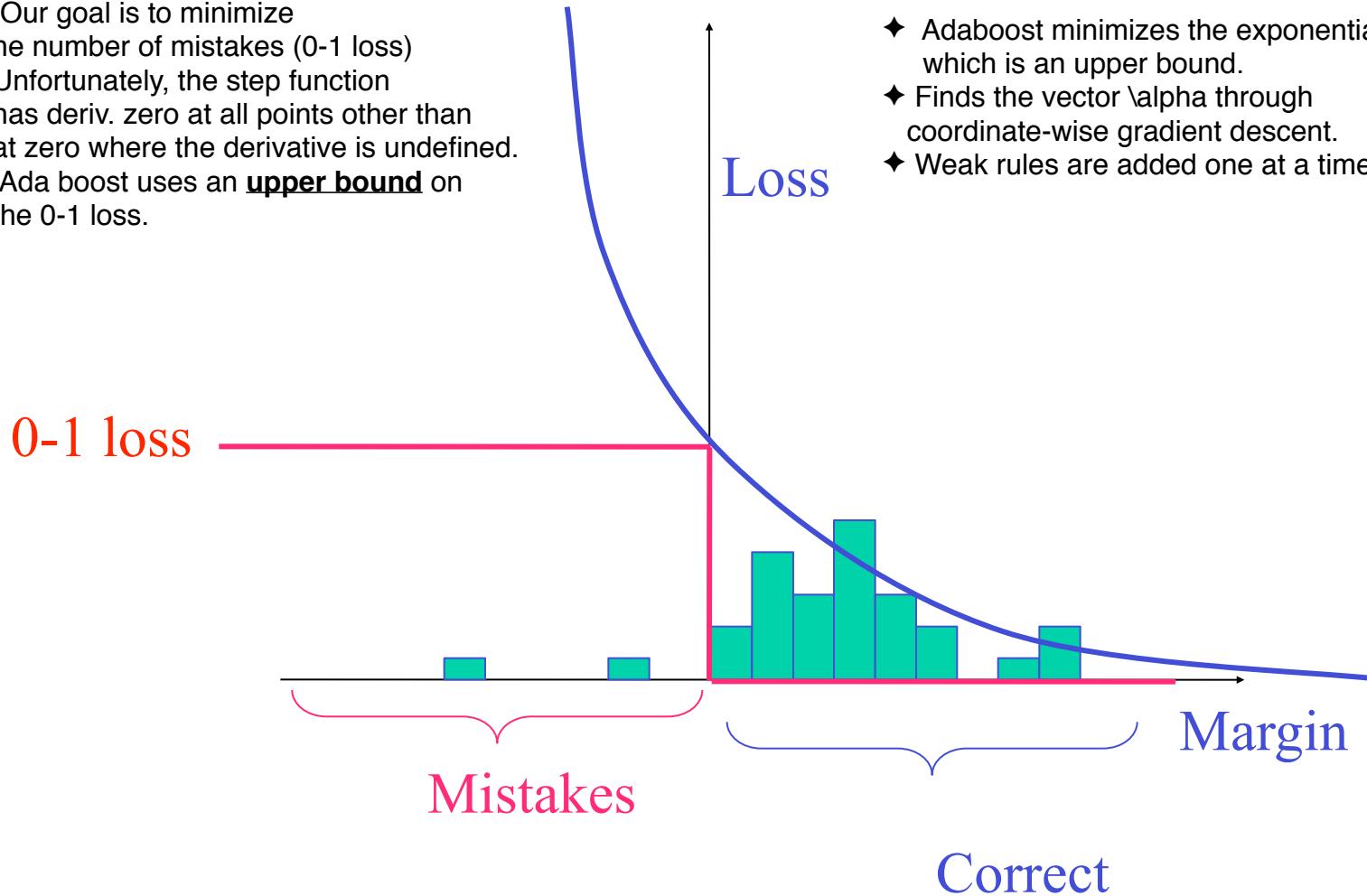


Boosting as gradient descent

- ◆ Our goal is to minimize the number of mistakes (0-1 loss)
- ◆ Unfortunately, the step function has deriv. zero at all points other than at zero where the derivative is undefined.
- ◆ Ada boost uses an upper bound on the 0-1 loss.

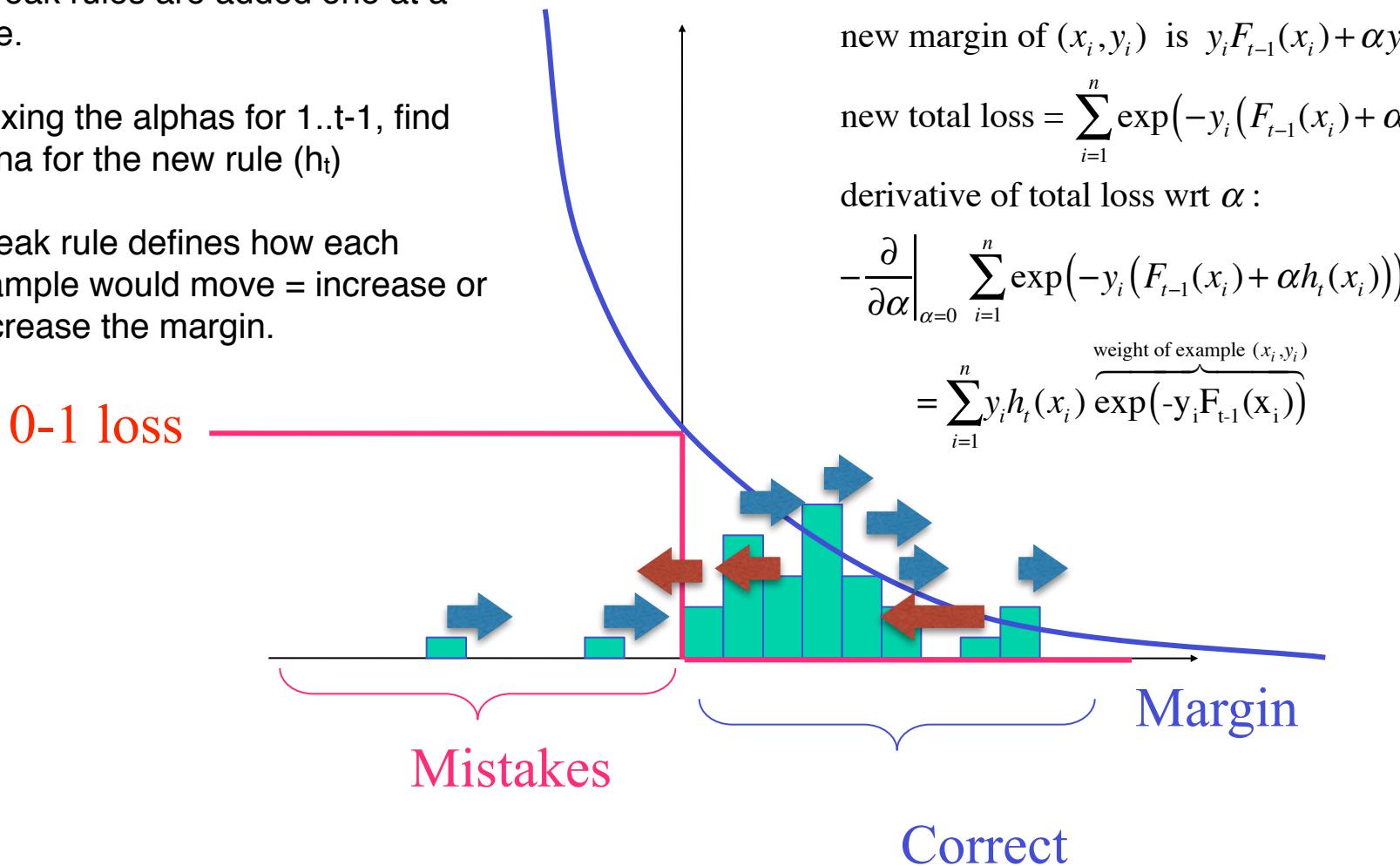
Adaboost : $e^{-margin}$

- ◆ Adaboost minimizes the exponential loss which is an upper bound.
- ◆ Finds the vector α through coordinate-wise gradient descent.
- ◆ Weak rules are added one at a time.



Adaboost as gradient descent

- ◆ Weak rules are added one at a time.
- ◆ Fixing the alphas for 1..t-1, find alpha for the new rule (h_t)
- ◆ Weak rule defines how each example would move = increase or decrease the margin.



Logitboost as gradient descent

Also called Gentle-Boost and Logit Boost, Hastie, Freedman & Tibshirani

Logitboost loss

Adaboost (loss and weight)

margin= m

Instead of the loss e^{-m}

define loss to be $\log(1+e^{-m})$

When margin $\gg 0$: $\log(1+e^{-m}) \approx e^{-m}$

When margin $\ll 0$: $\log(1+e^{-m}) \approx -m$

Logitboost weight

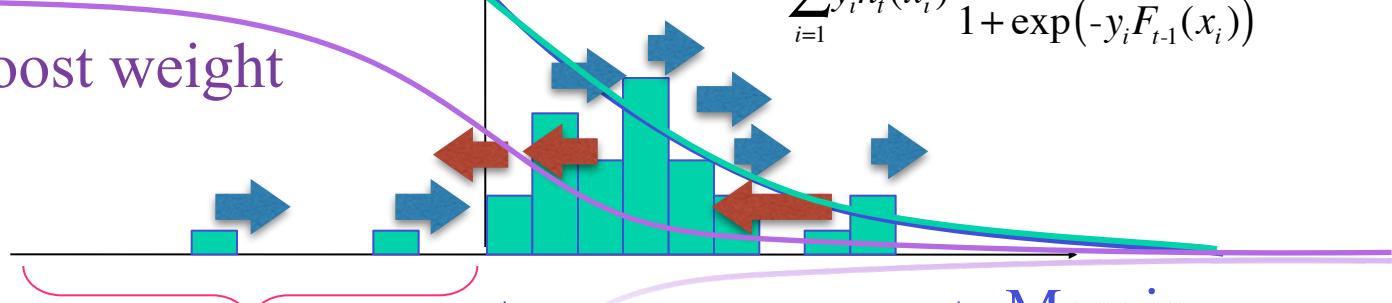
new margin of (x_i, y_i) is $y_i F_{t-1}(x_i) + \alpha y_i h_t(x_i)$

new total loss = $\sum_{i=1}^n \log [1 + \exp(-y_i(F_{t-1}(x_i) + \alpha h_t(x_i)))]$

derivative of total loss wrt α :

$$-\frac{\partial}{\partial \alpha} \Big|_{\alpha=0} \sum_{i=1}^n \log [1 + \exp(-y_i(F_{t-1}(x_i) + \alpha h_t(x_i)))]$$

$$= \sum_{i=1}^n y_i h_t(x_i) \frac{\text{weight of example } (x_i, y_i)}{\frac{\exp(-y_i F_{t-1}(x_i))}{1 + \exp(-y_i F_{t-1}(x_i))}}$$



Mistakes

Correct

Noise resistance

- Adaboost:
 - perform well when a achievable error rate is close to zero (almost consistent case).
 - Errors = examples with negative margins, get very large weights, can overfit.
- Logitboost:
 - Inferior to adaboost when achievable error rate is close to zero.
 - Often better than Adaboost when achievable error rate is high.
 - Weight on any example never larger than 1.

Gradient-Boost / AnyBoost

- A general recipe for learning by incremental optimization
- Applies to any (differentiable) loss function.

labeled example is (x, y) (can be anything).

prediction is of the form: $F_t(x) = \sum_{i=1}^t \alpha_i h_i(x)$

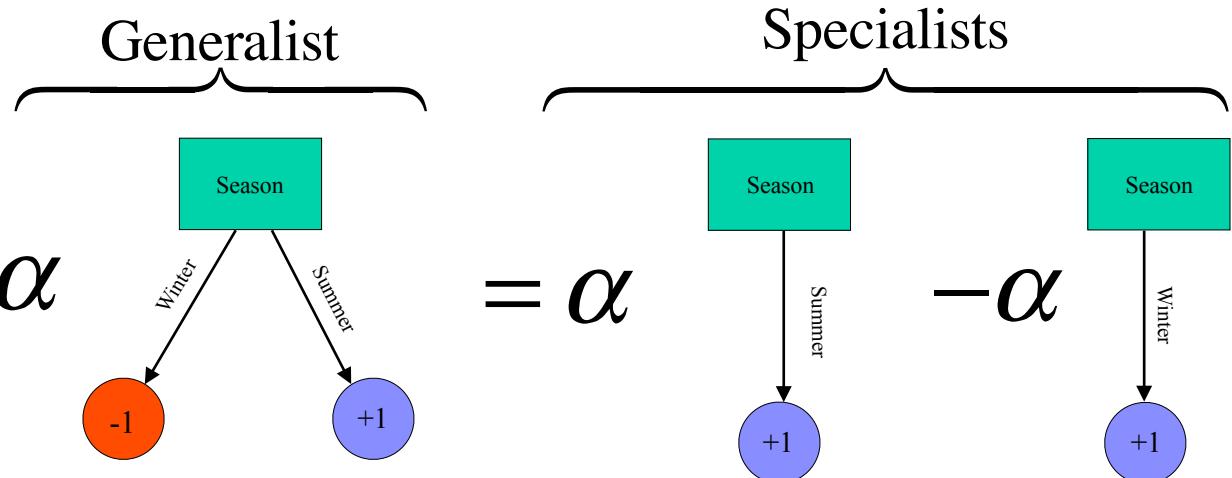
Loss: $L(F_t(x), y)$ Total Loss so far is: $\sum_{j=1}^n L(F_{t-1}(x_j), y_j)$

Weight of example (x, y) : $\frac{\partial}{\partial z} L(F_{t-1}(x) + z, y)$

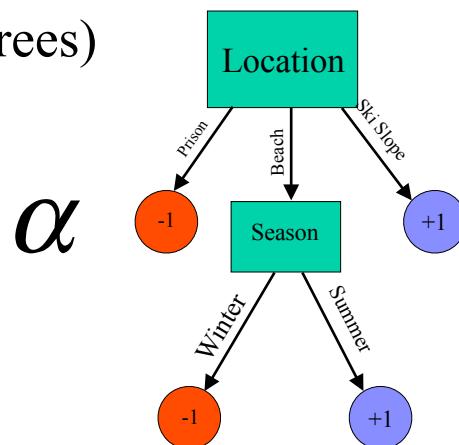
- At each iteration:
 - calc example weights
 - call weak learner with weighted example
 - Add generated weak rule to create new rule: $F_t = F_{t-1} + \alpha_t h_t$

Styles of weak learners

- Simple (stumps)



- Complex (fully grown trees)

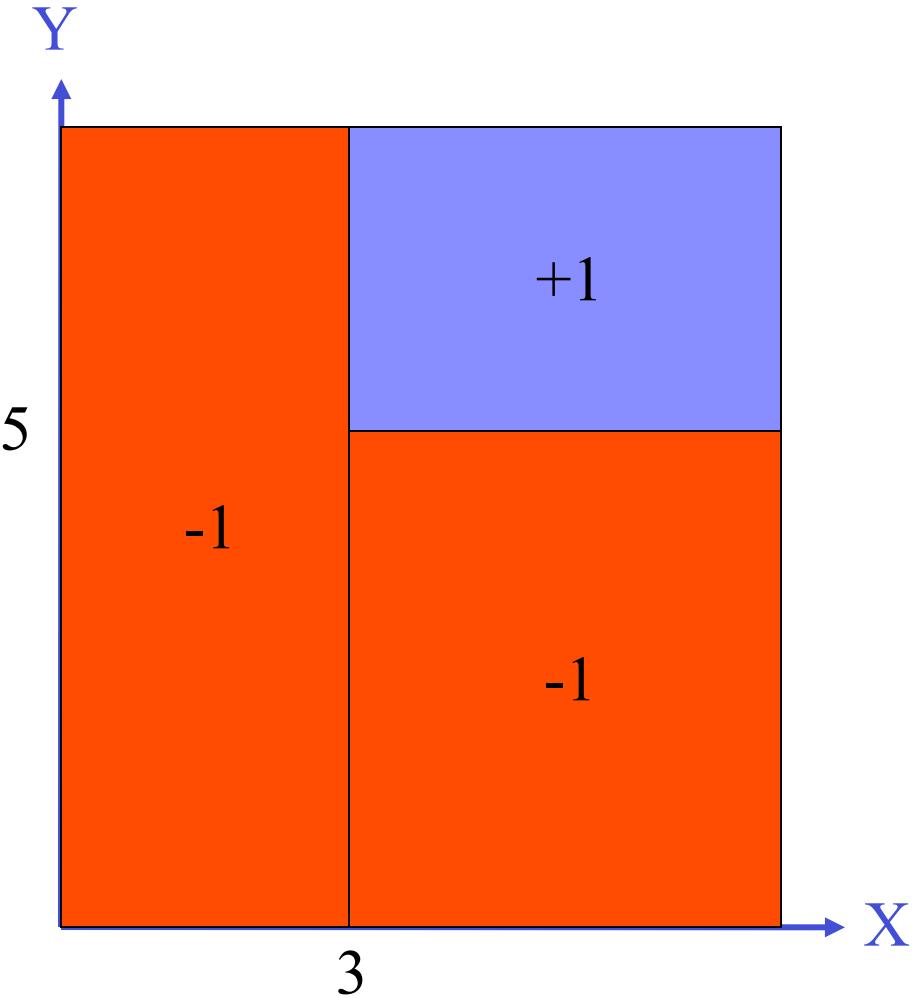
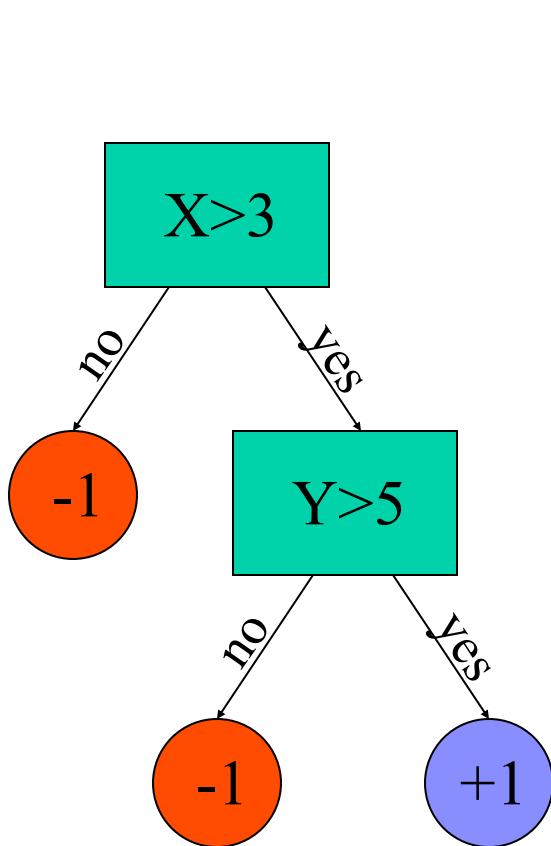


- Everything in between (neural networks, Nearest neighbors, Naive Bayes,....)

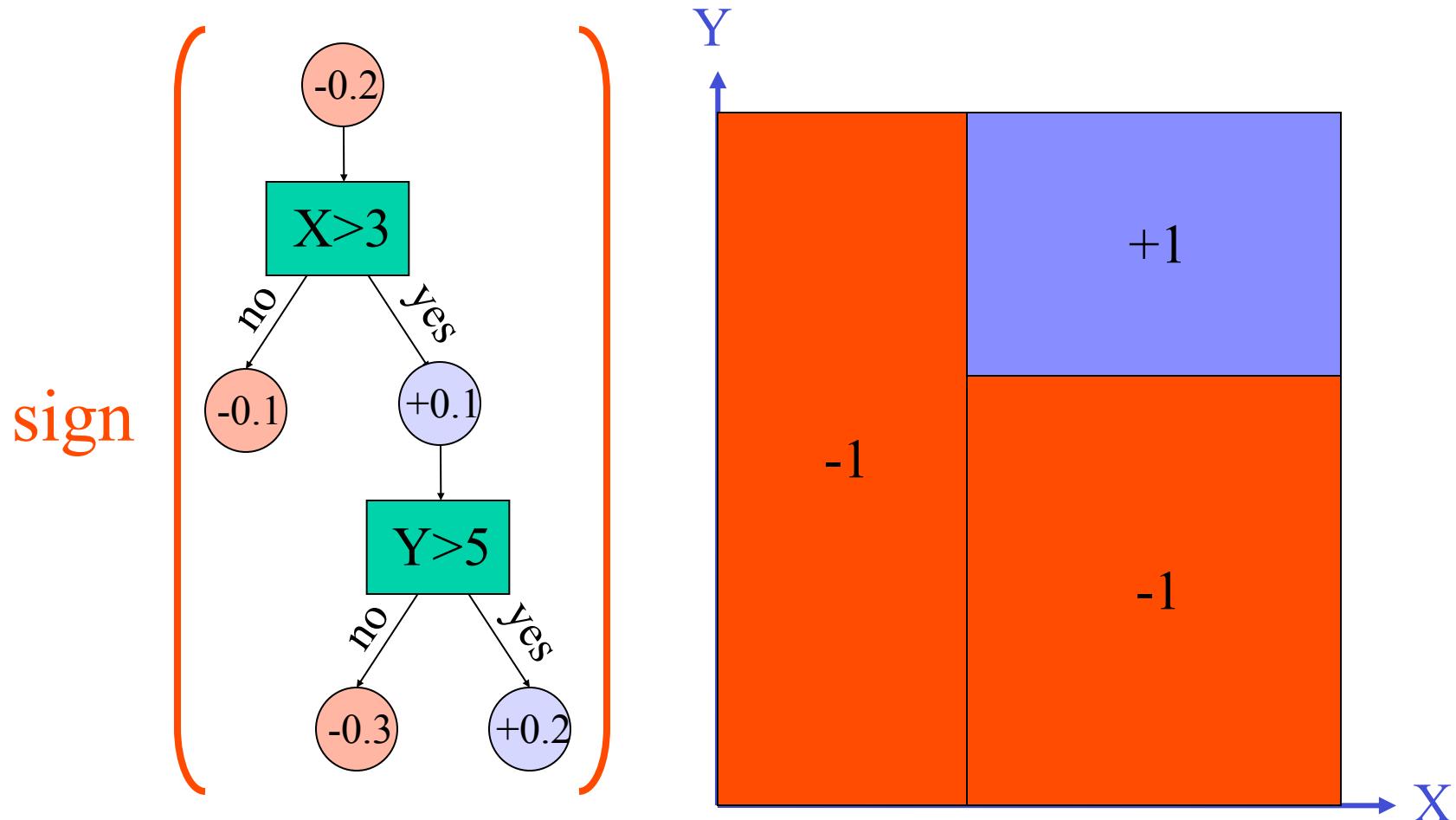
Alternating Decision Trees

Joint work with Llew Mason

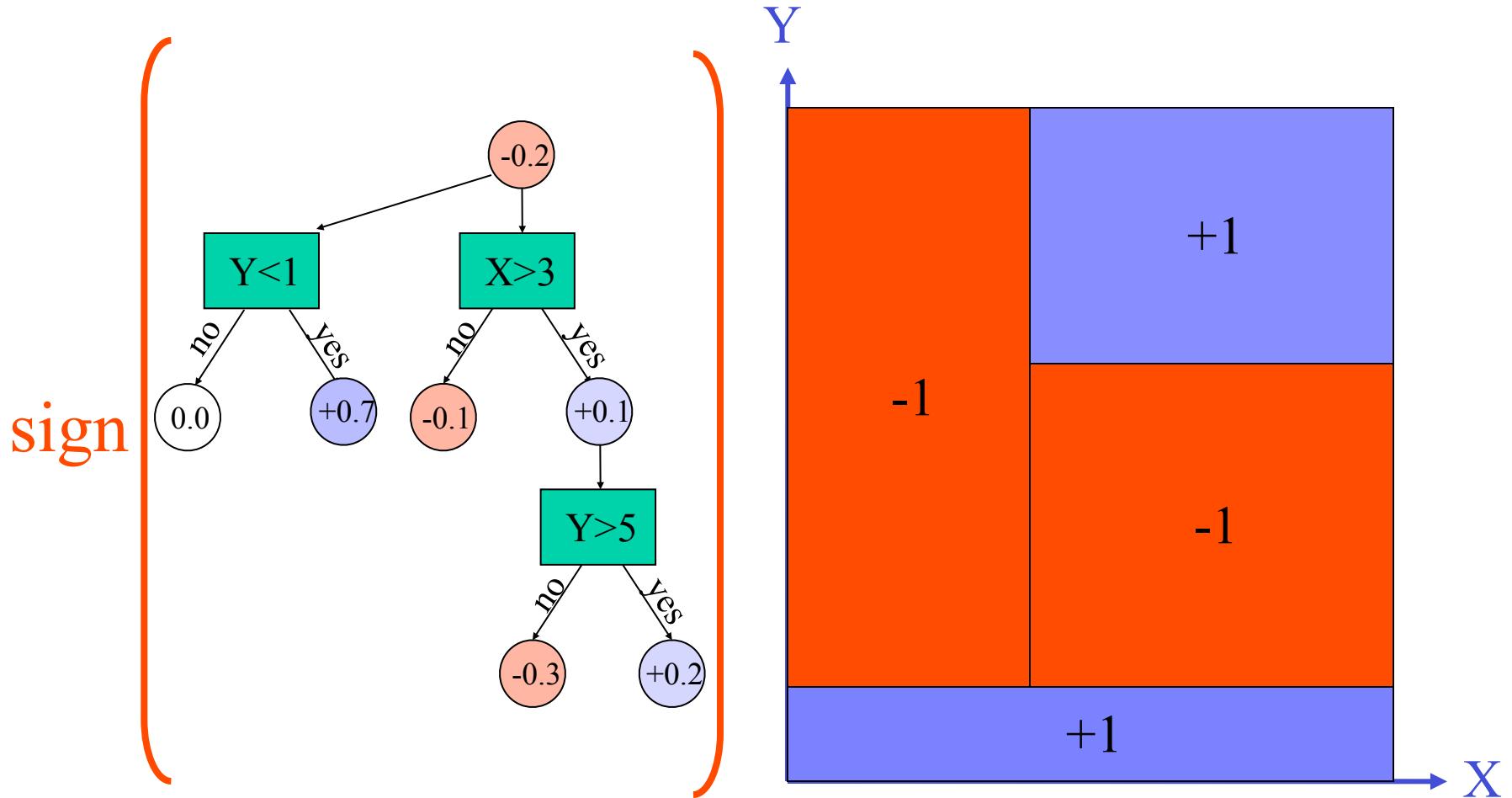
Decision Trees



Decision tree as a sum



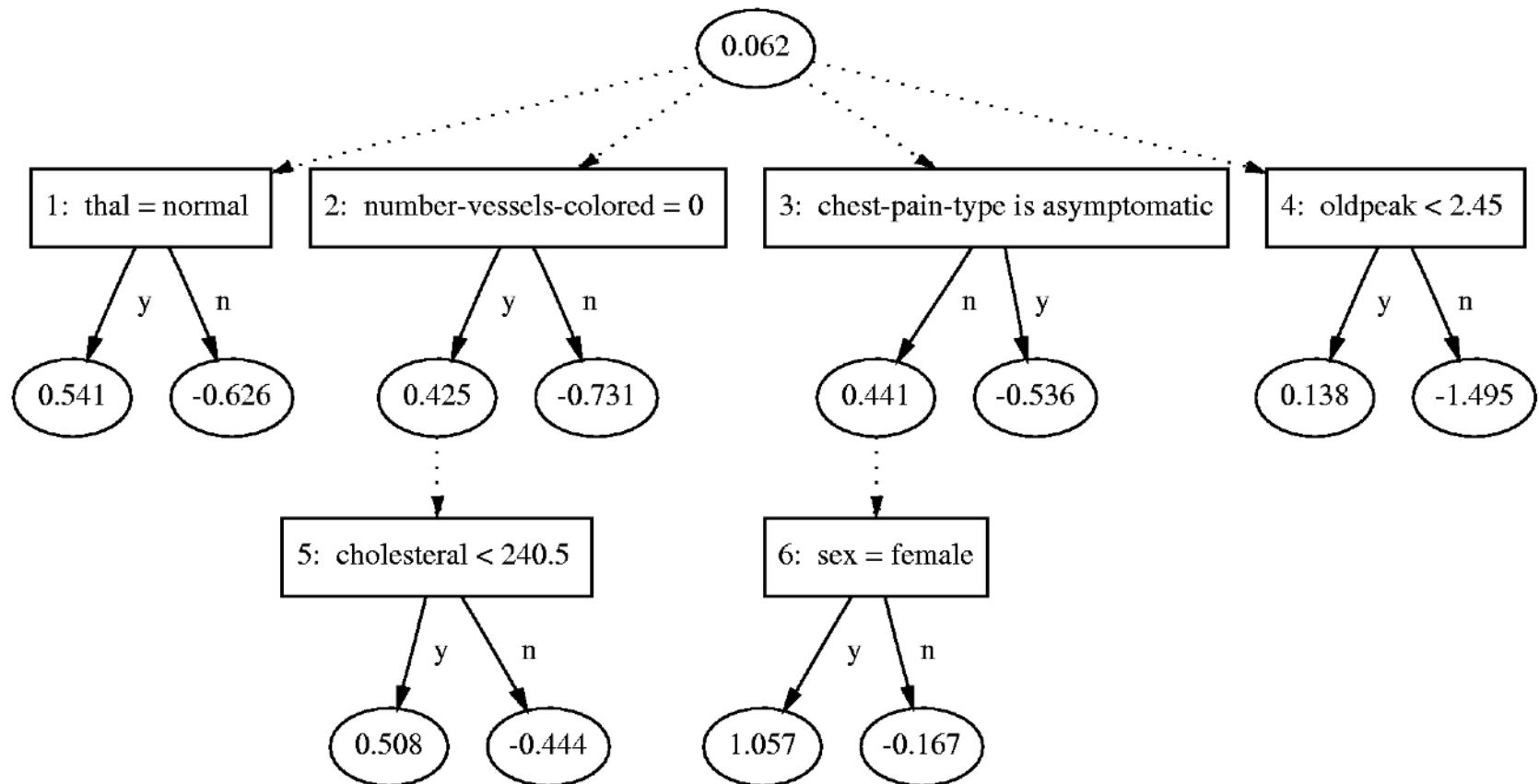
An alternating decision tree



Example: Medical Diagnostics

- **Cleve** dataset from UC Irvine database.
- Heart disease diagnostics (+1=healthy,-1=sick)
- 13 features from tests (real valued and discrete).
- 303 instances.

Adtree for Cleveland heart-disease diagnostics problem

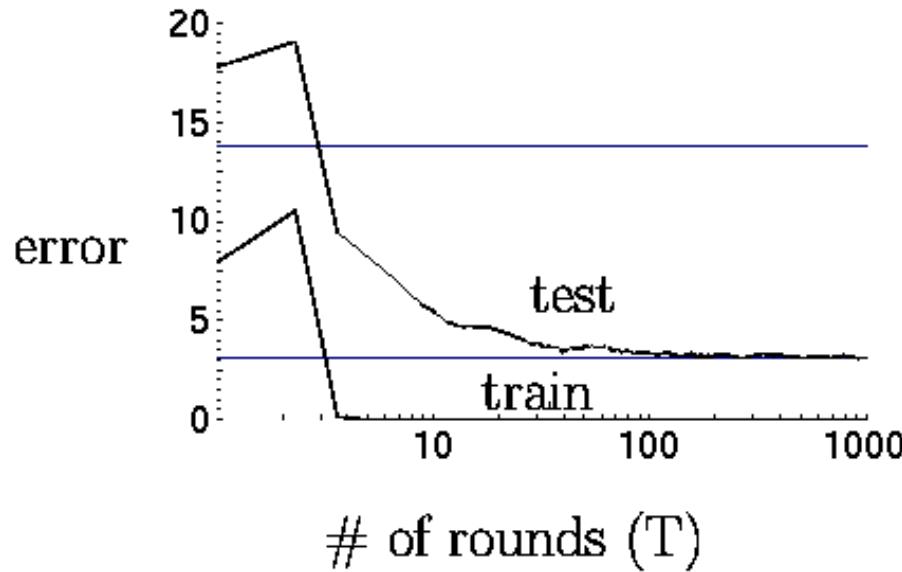


Cross-validated accuracy

Learning algorithm	Number of splits	Average test error	Test error variance
ADtree	6	17.0%	0.6%
C5.0	27	27.2%	0.5%
C5.0 + boosting	446	20.2%	0.5%
Boost Stumps	16	16.5%	0.8%

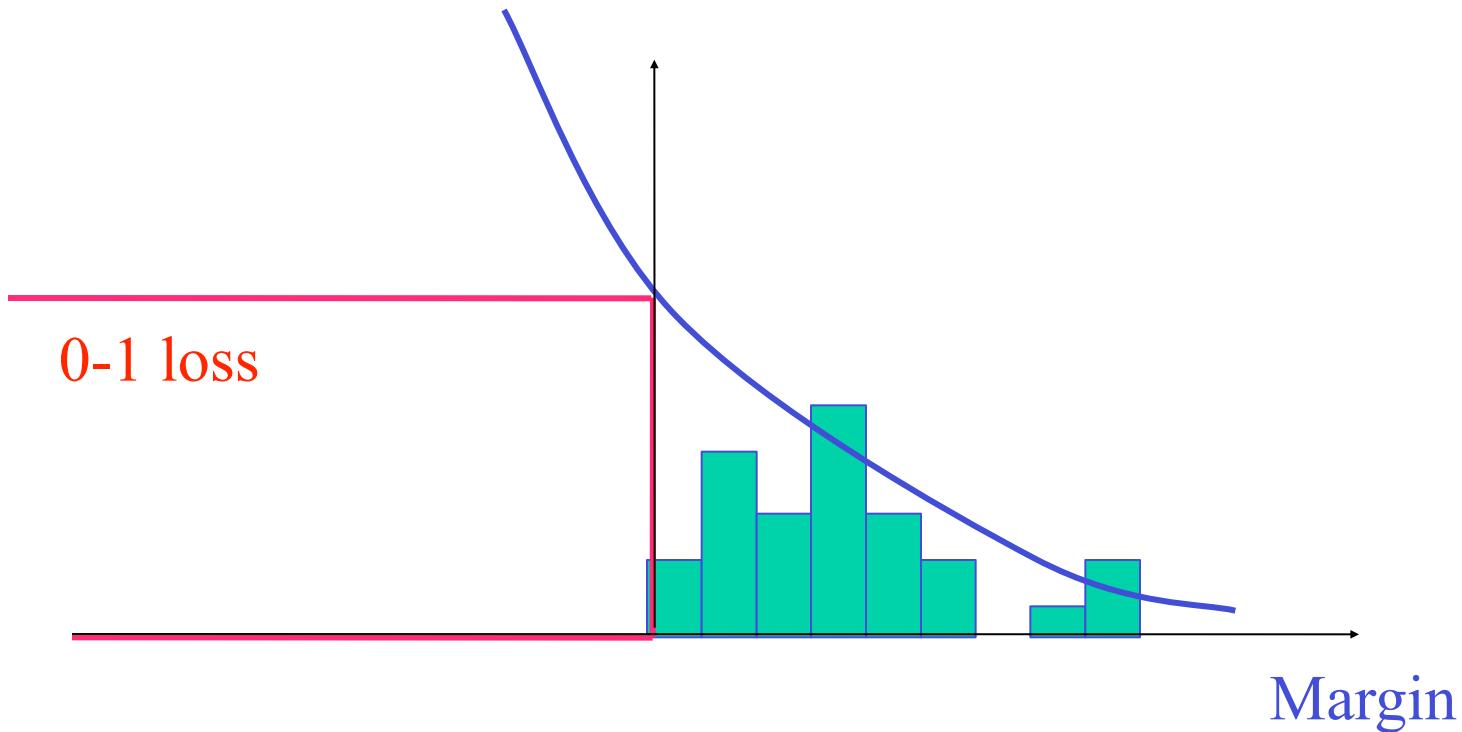
Curious phenomenon

Boosting decision trees

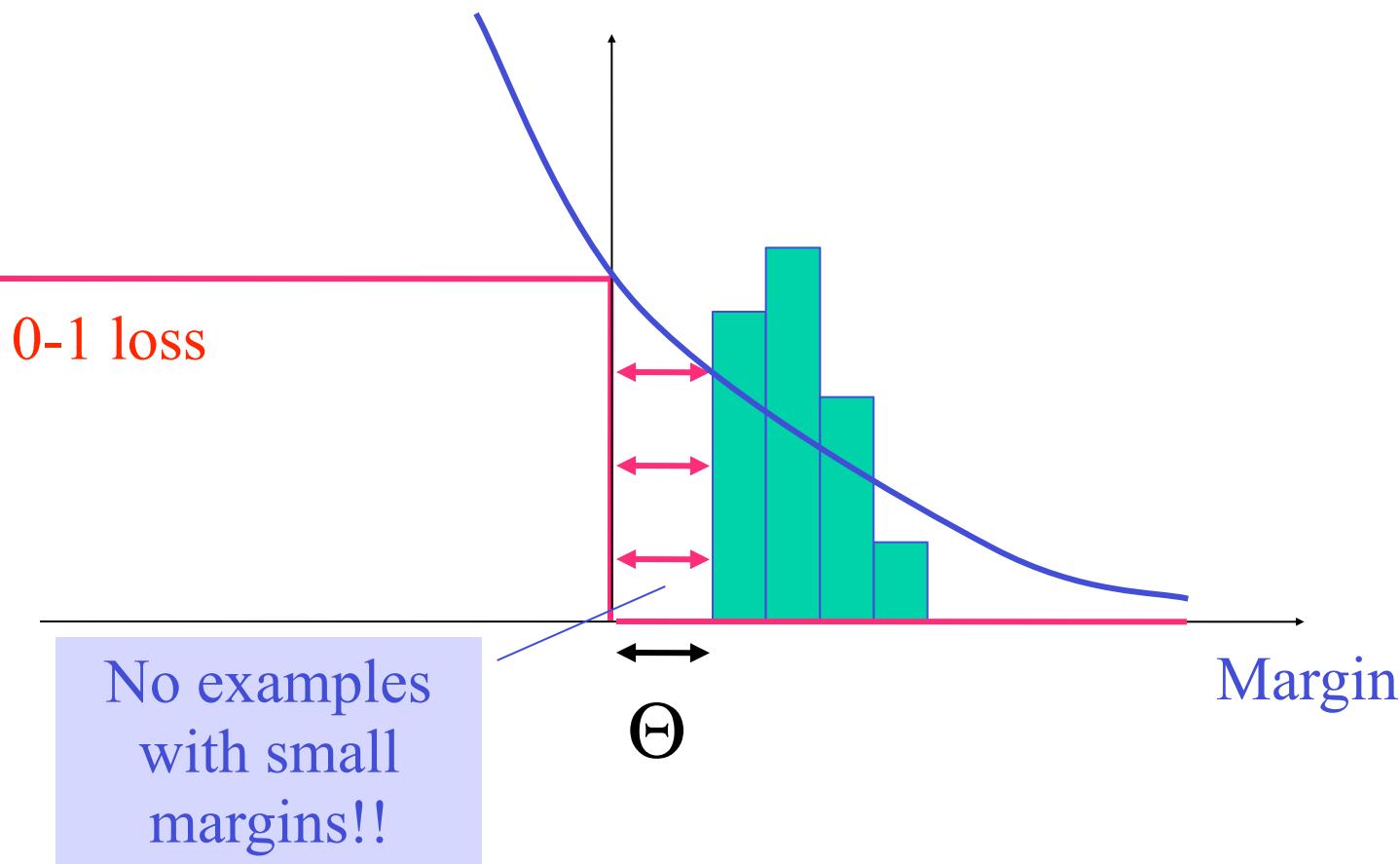


Using <10,000 training examples we fit >2,000,000 parameters

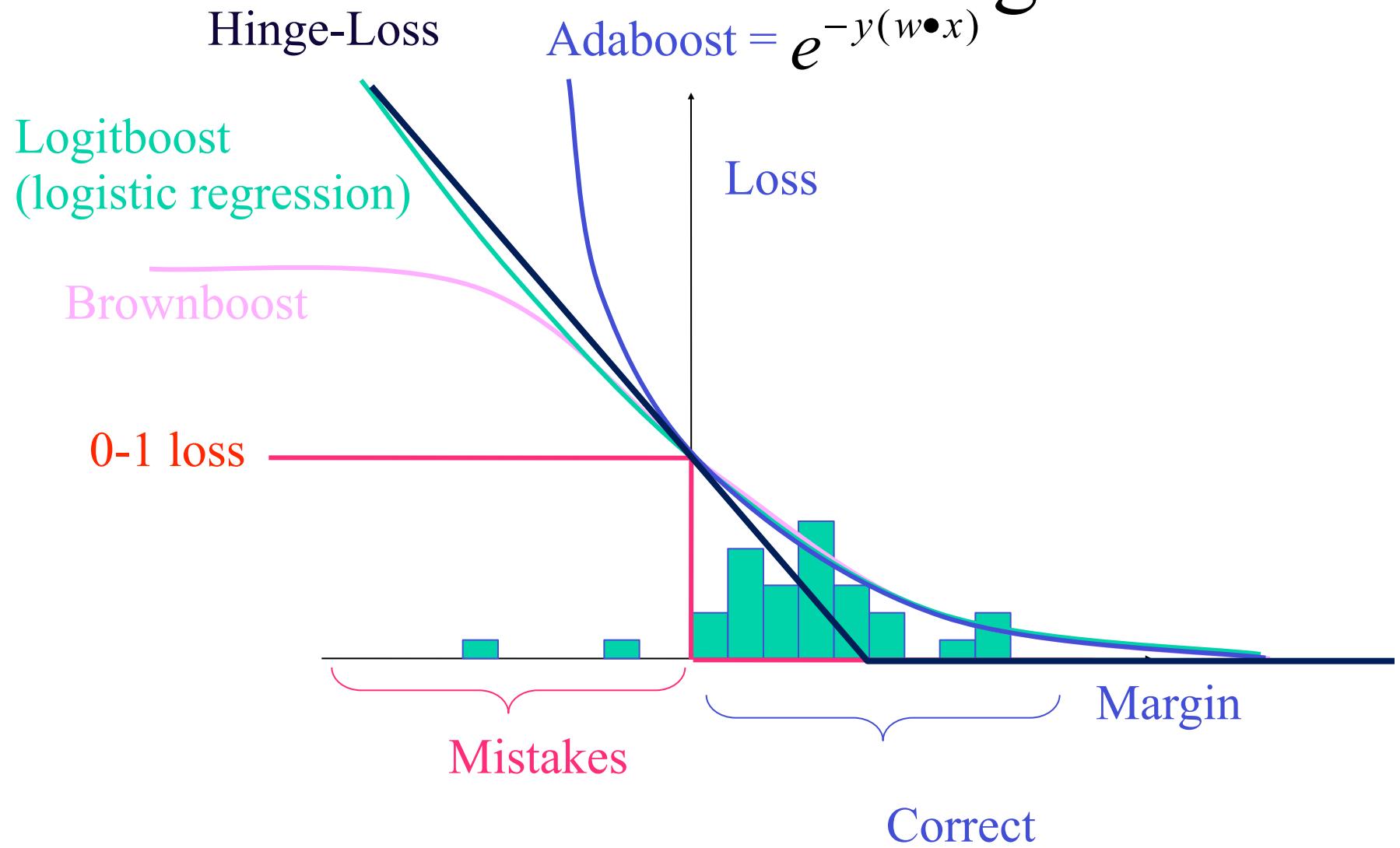
Explanation using margins



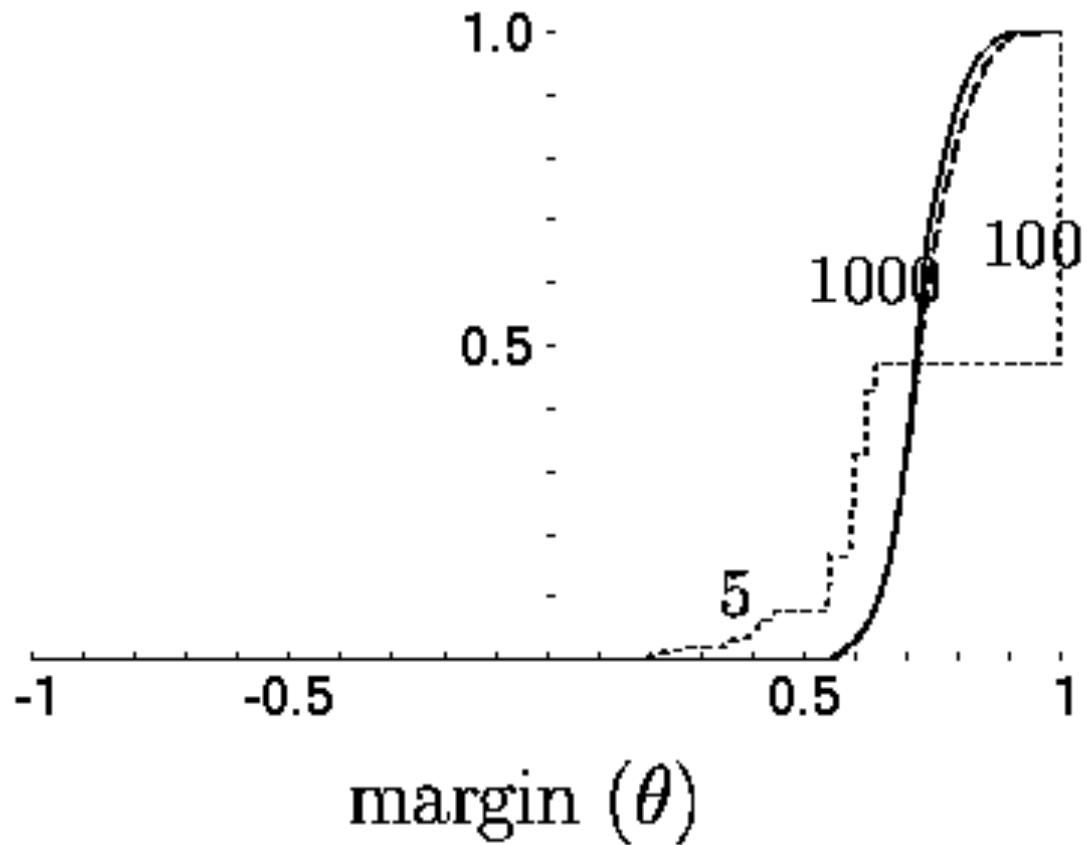
Explanation using margins



Minimizing error using loss functions on margin



Experimental Evidence



Theorem

Schapire, Freund, Bartlett & Lee
Annals of stat. 98

For any convex combination and any threshold $\forall f \in \mathcal{C}, \forall \theta > 0:$

Probability of mistake

Fraction of training example
with small margin

$$P_{(x,y) \sim D} [\text{sign}(f(x)) \neq y] \leq P_{(x,y) \sim S} [\text{margin}_f(x, y) \leq \theta]$$

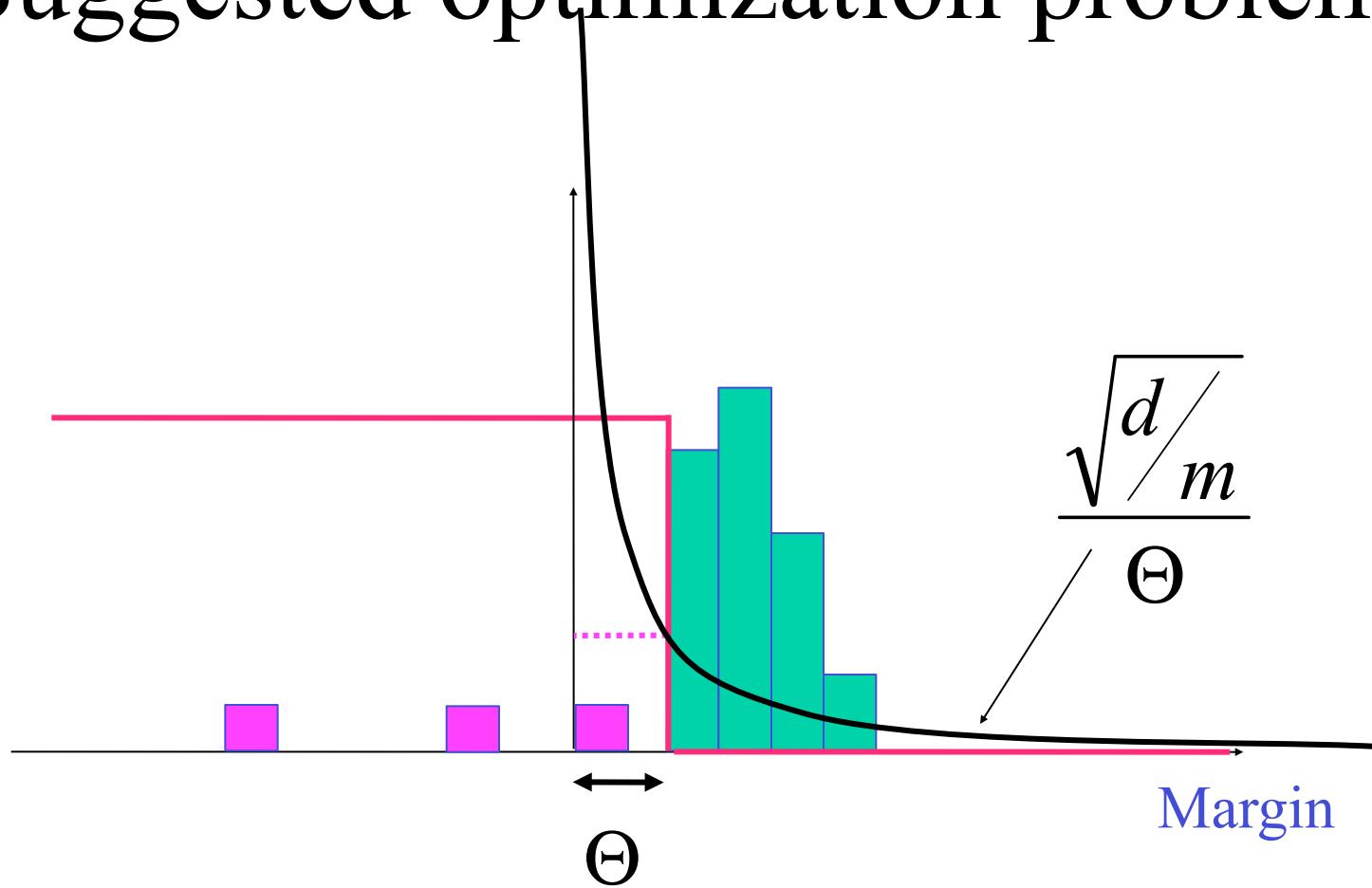
Size of training sample

$$+ \tilde{O} \left(\frac{\sqrt{d/m}}{\theta} \right)$$

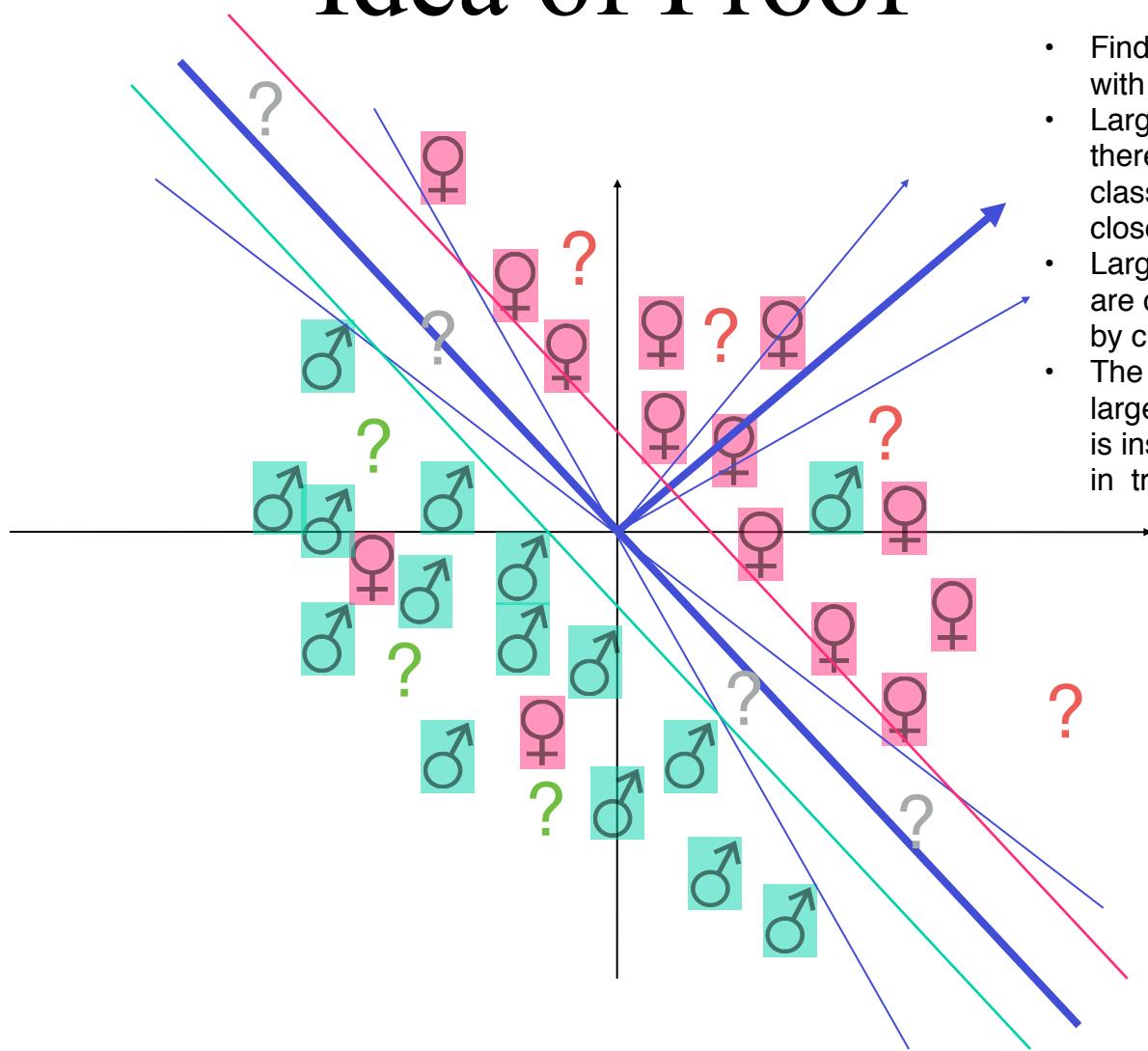
VC dimension of weak rules

No dependence on number
of weak rules
that are combined!!!

Suggested optimization problem



Idea of Proof



- Find linear classifier with large margin
- Large margin implies there is a **cone** of linear classifiers with close-to-minimal error.
- Large margin test examples are classified identically by classifiers in cone.
- The prediction on large margin test examples is insensitive to small changes in training set.

Why are large margins good?

- On examples with large margin, the prediction is stable.
- In other words, small changes in the training set will not cause the prediction to flip.
- We can predict with confidence.
- The confidence is different from when $P(y|x)$ is close to 0 or to 1.
- It has to do with the stability of the classifier w.r.t. small changes in the training data.

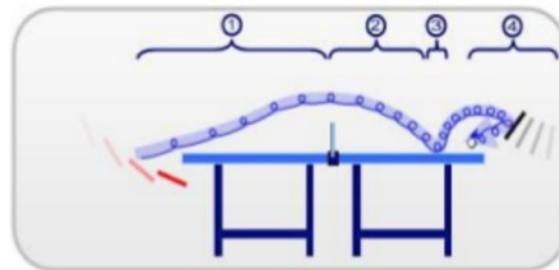
Prediction in sports



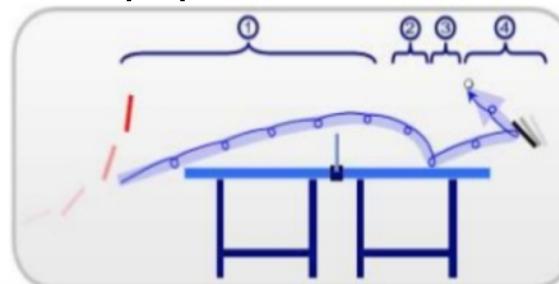
Prediction of spin in table tennis

- To respond correctly, we need to predict whether the ball has backspin or topspin.
- By the time the ball is hit, it is too late.
- We predict from body posture, from experience with individual player,...
- Definite action according to prediction.
- Prediction has to be correct more often than incorrect

Backspin



Topspin



The three sources of classification error

- **Bayes Error:** the minimal error rate achieved when the true distribution is known (exactly).
- **Model Bias:** error resulting from the inability of the model class to represent the true distribution.
- **Data Variation:** error resulting from the difference between the training set and the true distribution.

Tradeoffs

- Weight decay, reducing the number of parameters decrease data variation, increase model bias.
- Bagging decreases data variation without significantly increasing model bias.
- Boosting reduces both variation and model bias.

The Bias/Variance tradeoff for squared error

- The squared error can always be decomposed into a bias term and a variance term.

$$B(x) = E_{(x,y)}[y|x] \text{ The Bayes optimal regression function}$$

$$E_{(x,y)}[(f(x)-y)^2] = \overbrace{E_x\left[\left(f(x) - B(x)\right)^2\right]}^{\text{Bias}} + \overbrace{E_{(x,y)}\left[\left(y - B(x)\right)^2\right]}^{\text{Variance}}$$

- For regression:
 - Increasing the number of features decreases bias and increases variance.
 - Reducing the size of the weights (Ridge regression, Lasso) Increases bias and decreases variance.

Applications of Boosting

Applications of Boosting

- Academic research
- Applied research
- Commercial deployment

Academic research

% test error rates

Database	Other	Boosting	Error reduction
Cleveland	27.2 (DT)	16.5	39%
Promoters	22.0 (DT)	11.8	46%
Letter	13.8 (DT)	3.5	74%
Reuters 4	5.8, 6.0, 9.8	2.95	~60%
Reuters 8	11.3, 12.1, 13.4	7.4	~40%

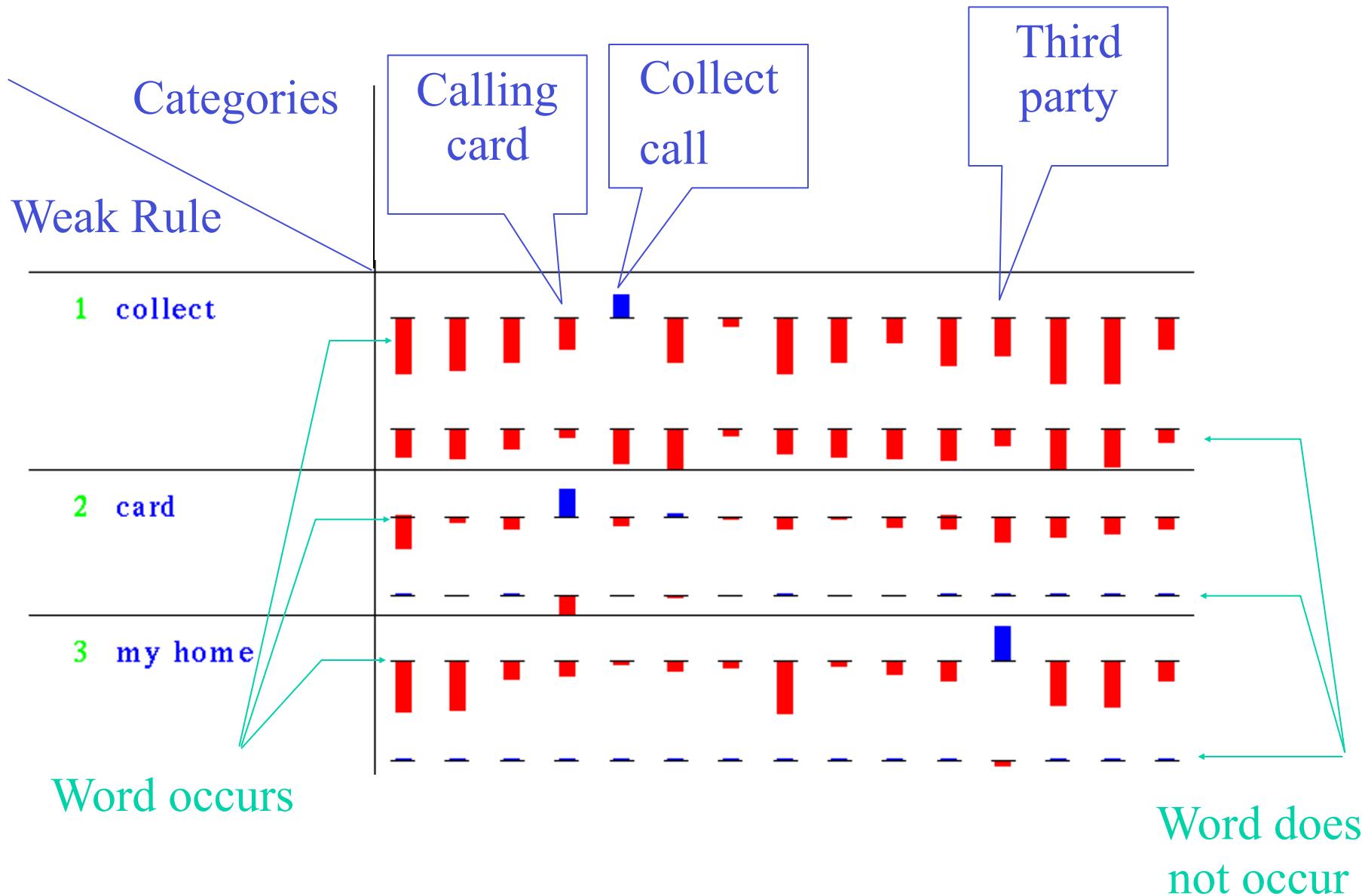
Applied research

- “*AT&T, How may I help you?*”
 - Classify voice requests
 - Voice -> text -> category
 - Fourteen categories
 - Area code,
 - AT&T service,
 - billing credit,
 - calling card,
 - collect,
 - competitor
 - competitor,
 - dial assistance,
 - directory,
 - how to dial,
 - person to person,
 - rate,
 - third party,
 - time charge

Example transcribed sentences

- Yes I'd like to place a collect call long distance please ➤ **collect**
- Operator I need to make a call but I need to bill it to my office ➤ **third party**
- Yes I'd like to place a call on my master card please ➤ **calling card**
- I just called a number in Sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off my bill ➤ **billing credit**

Weak rules generated by “boostexter”



Results

- 7844 training examples
 - hand transcribed
- 1000 test examples
 - hand / machine transcribed
- Accuracy with 20% rejected
 - Machine transcribed: 75%
 - Hand transcribed: 90%

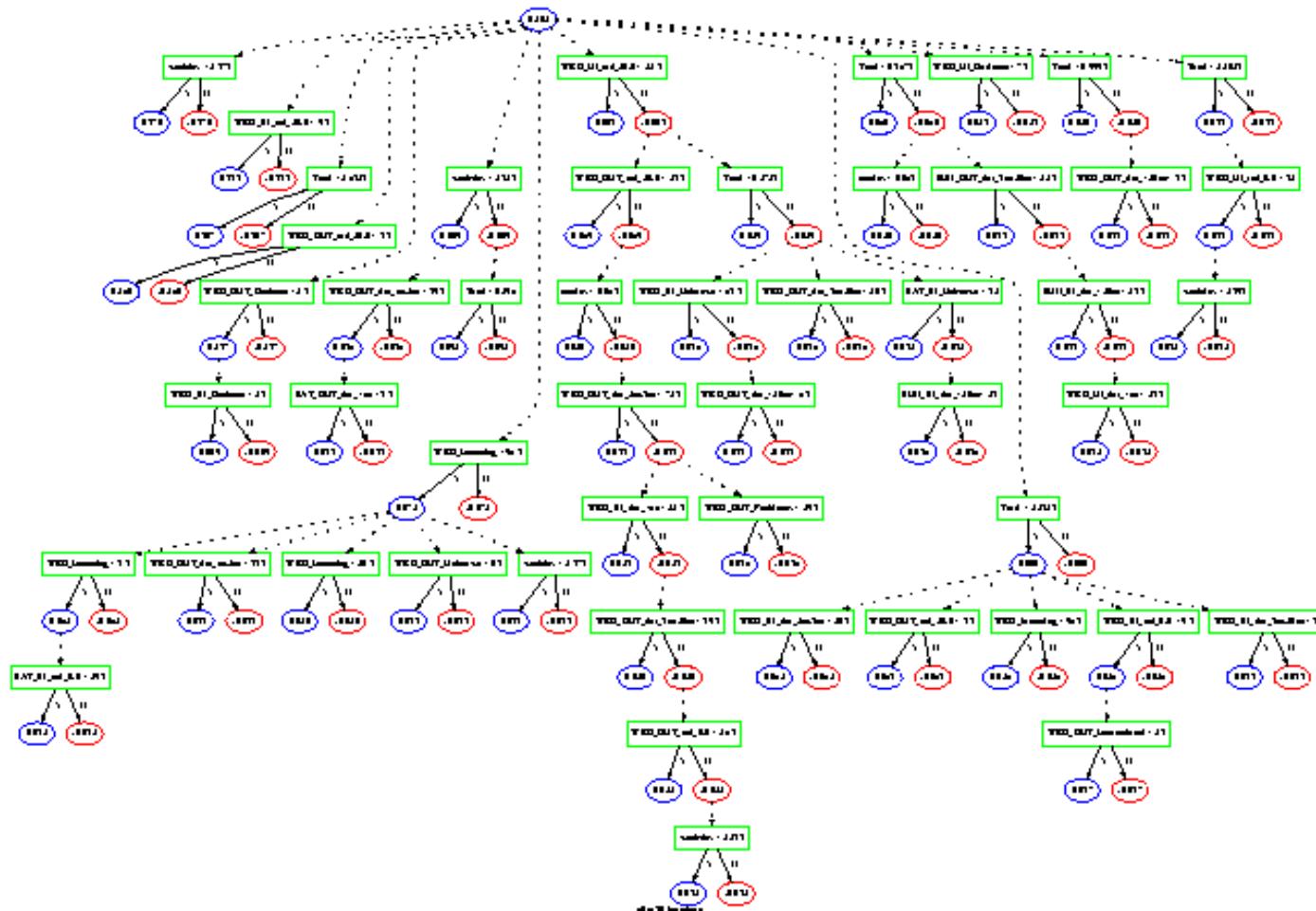
Commercial deployment

- Distinguish business/residence customers
- Using statistics from call-detail records
- Alternating decision trees
 - Combines very simple rules
 - Can over-fit, cross validation used to stop

Massive datasets (for 1997)

- 260M calls / day
- 230M telephone numbers
 - Label unknown for ~30%
- Hancock: software for computing statistical signatures.
- 100K randomly selected training examples,
- ~10K is enough
 - Training takes about 2 hours.
 - Generated classifier has to be both accurate and efficient

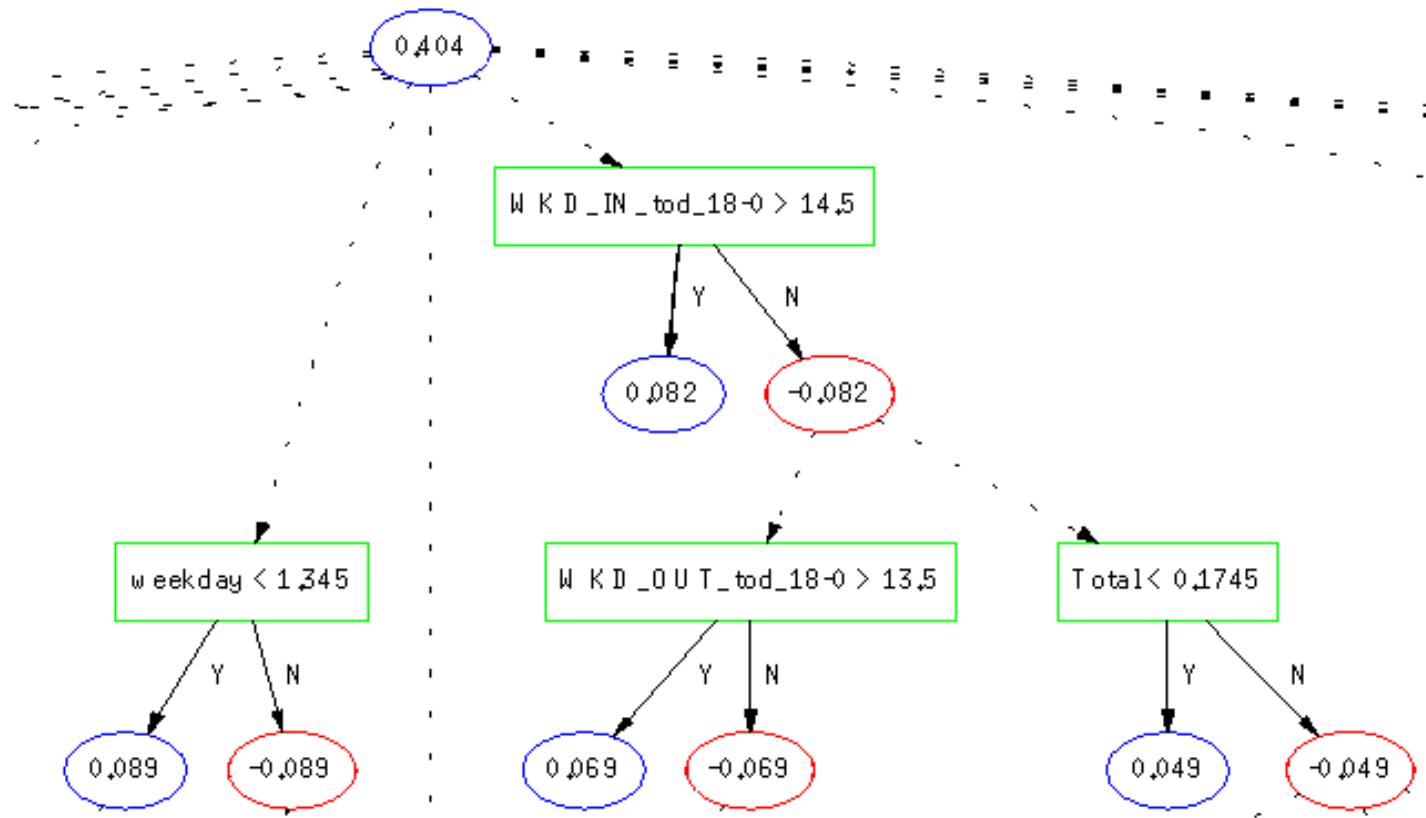
Alternating tree for “buizocity”



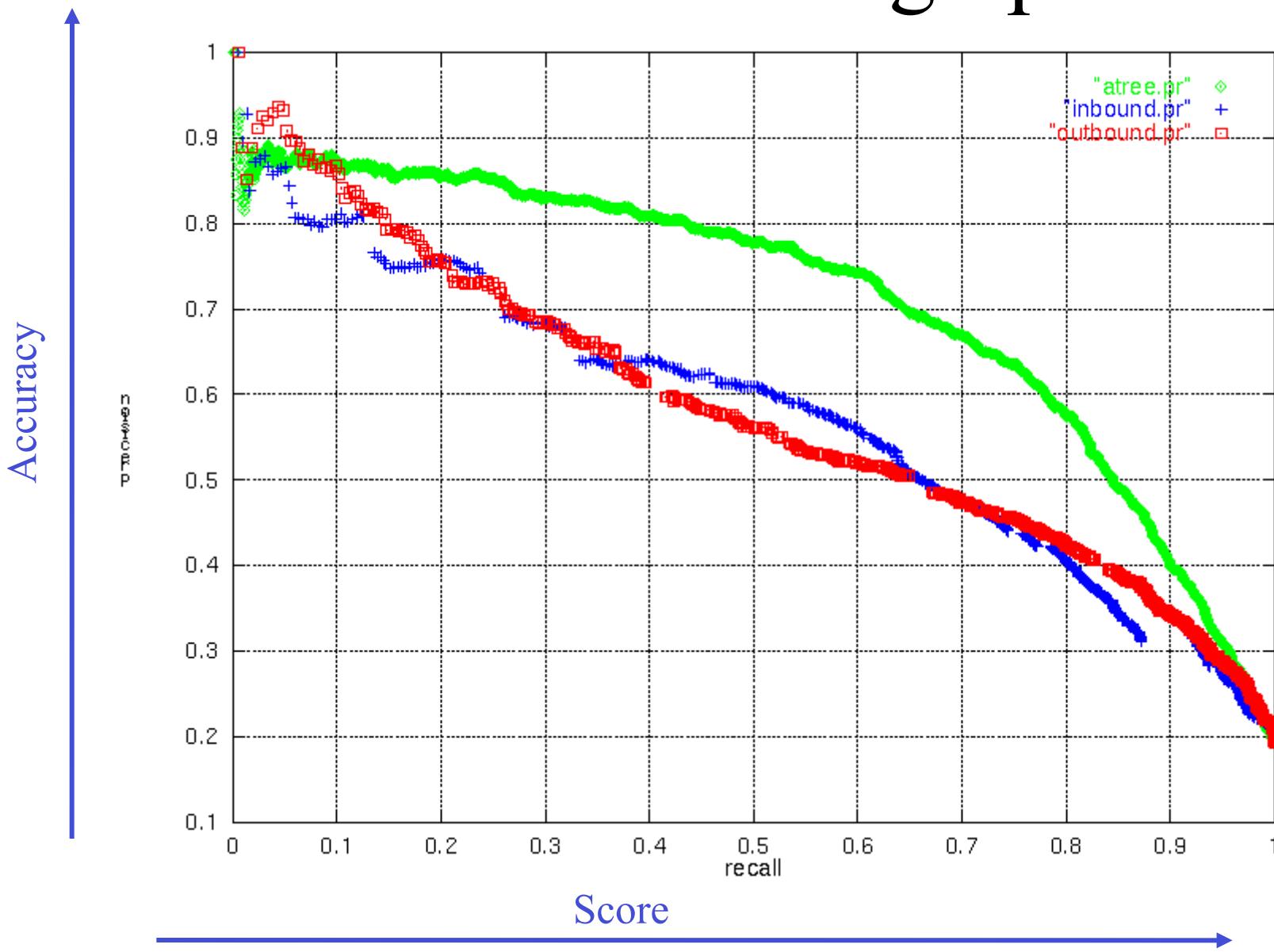
Alternating Tree (Detail)

Positive predictions \Leftrightarrow Residences

Negative predictions \Leftrightarrow Businesses



Precision/recall graphs



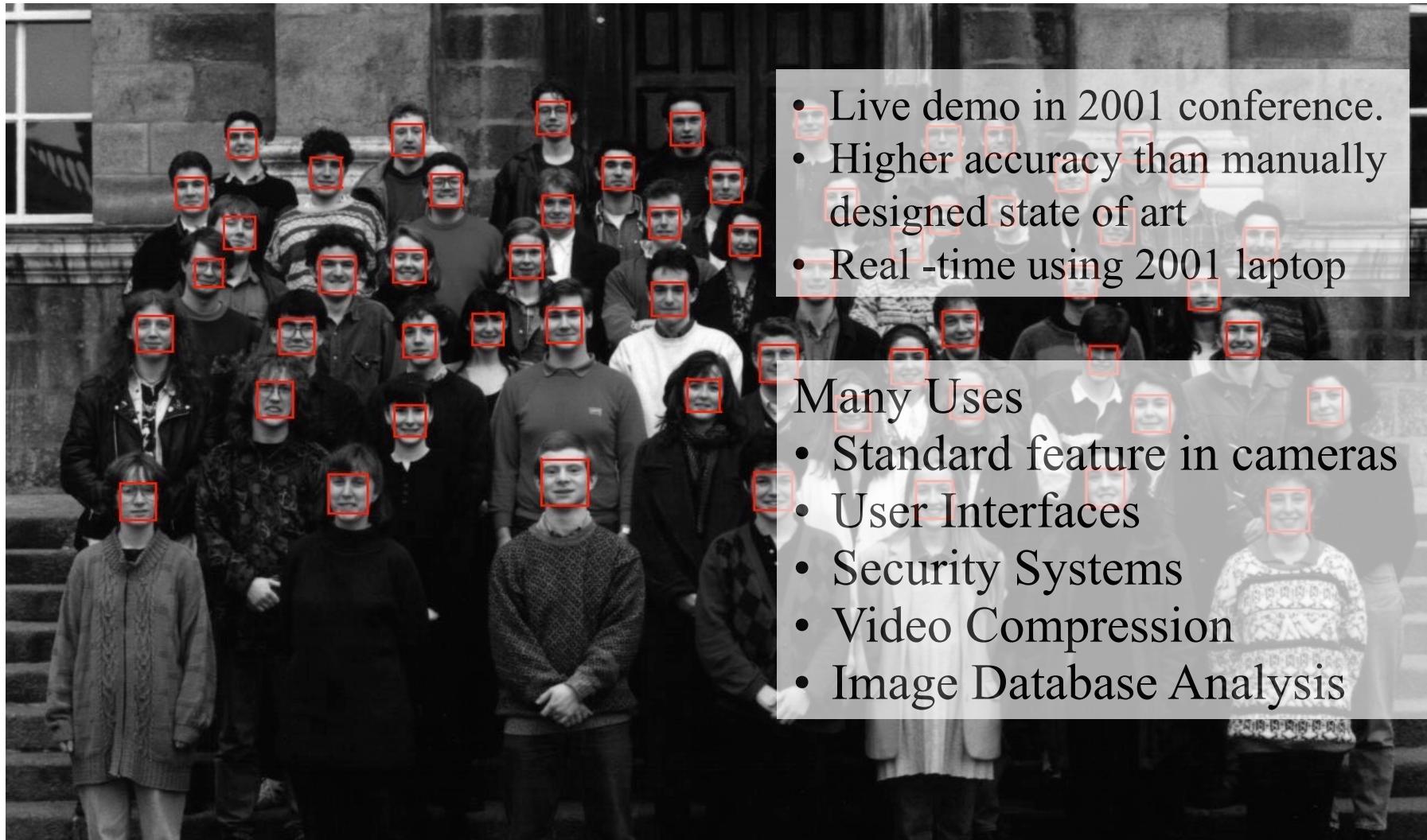
Business impact

- Increased coverage from 44% to 56%
for Accuracy ~94%
- Saved AT&T **15M\$** in the year 2000 in operations costs and missed opportunities.

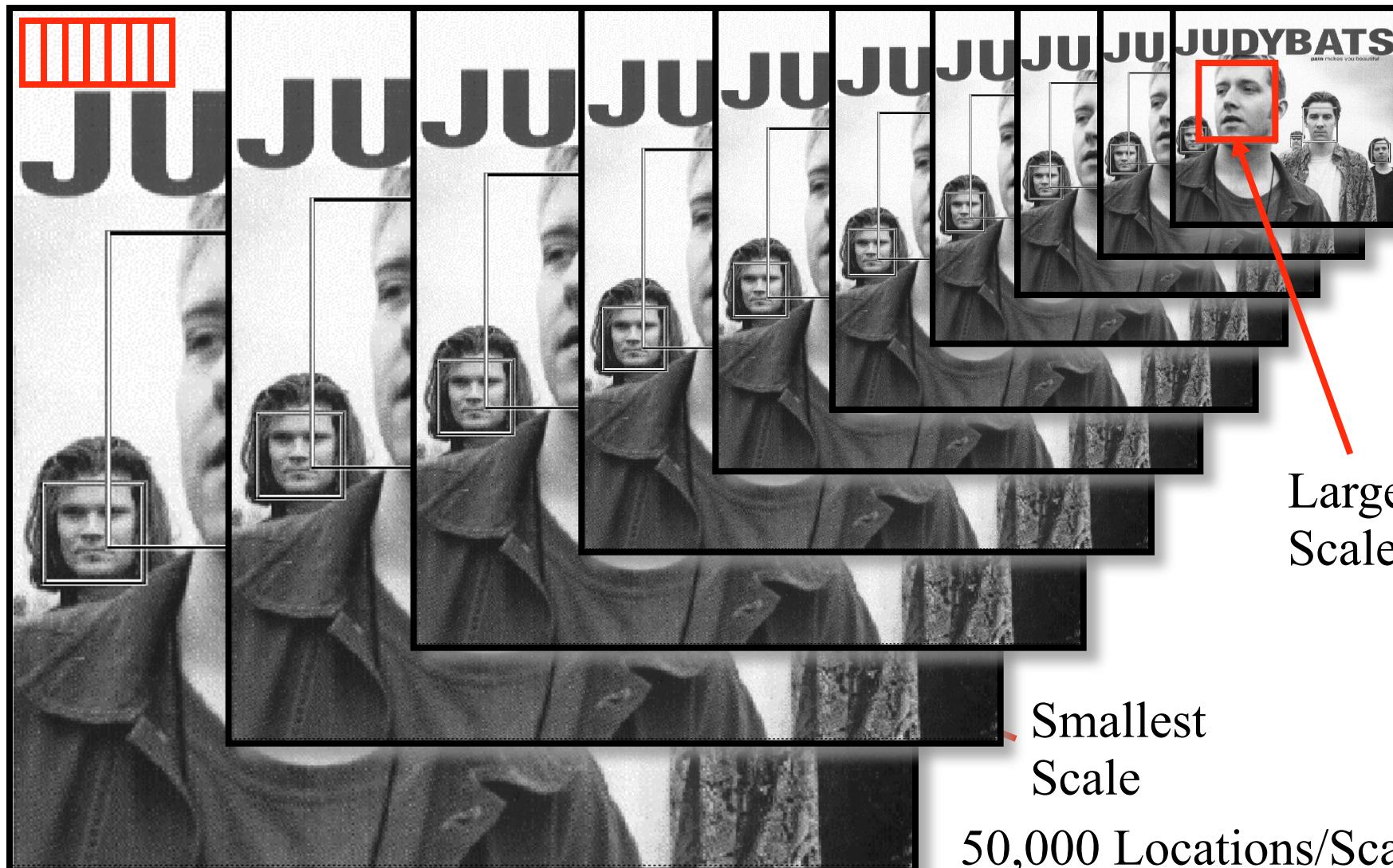
Face Detection

Viola and Jones / 2001

Face Detection / Viola and Jones



Face Detection as a Filtering process



50,000 Locations/Scales

Most Negative

Classifier is Learned from Labeled Data

- Example: 28x28 image patch
- Label:
Face / Non face
- 5000 faces, 10^8 non faces
- Faces are normalized
 - Scale, translation
 - Rotation remains...



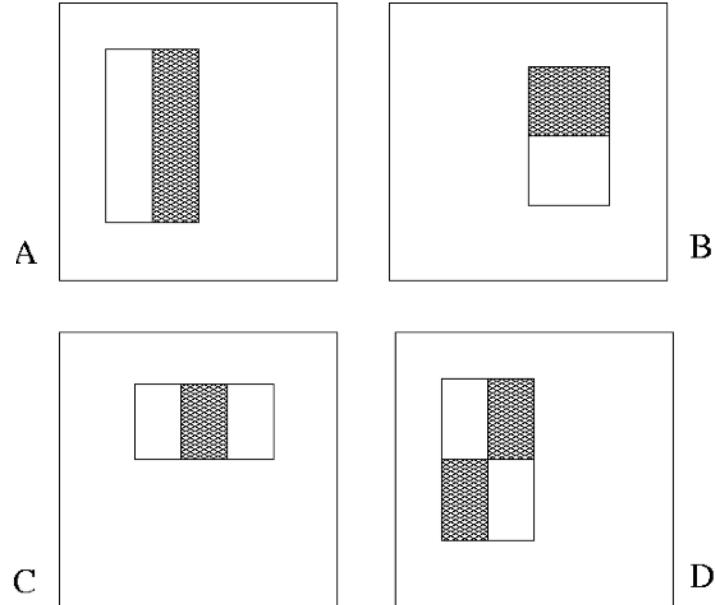
Image Features

“Rectangle filters”

Similar to Haar wavelets

Papageorgiou, et al.

$$h_t(x_i) = \begin{cases} 1 & \text{if } f_t(x_i) > \theta_t \\ 0 & \text{otherwise} \end{cases}$$



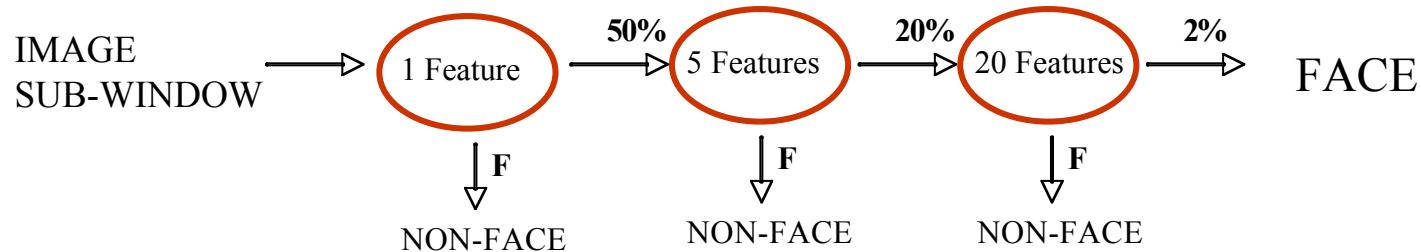
Very fast to compute using
“integral image”.

$60,000 \times 100 = 6,000,000$
Unique Binary Features

Combined using adaboost

Cascaded boosting

- Features combined using Adaboost
 - Find best weak rule by exhaustive search.
 - Ran for 2 days on a 250 node cluster
- For detection, features combined in a cascade:



- Runs in real time, 15 FPS, on laptop

Paul Viola and Mike Jones



Summary

- Boosting is a computational method for learning accurate classifiers
 - Resistance to over-fit explained by margins
 - Boosting has been applied successfully to a variety of classification problems

**Boosting
under high
noise.**

Adaboost is sensitive to label noise

- Letter / Irvine Database
- Focus on a binary problem: {F,I,J} vs. other

Label Noise	Adaboost	Logitboost
0%	0.8% \pm 0.2%	0.8% \pm 0.1%
20%	33.3% \pm 0.7%	31.6% \pm 0.6%

- Boosting puts too much weight on outliers.
- Need to give up on outliers.

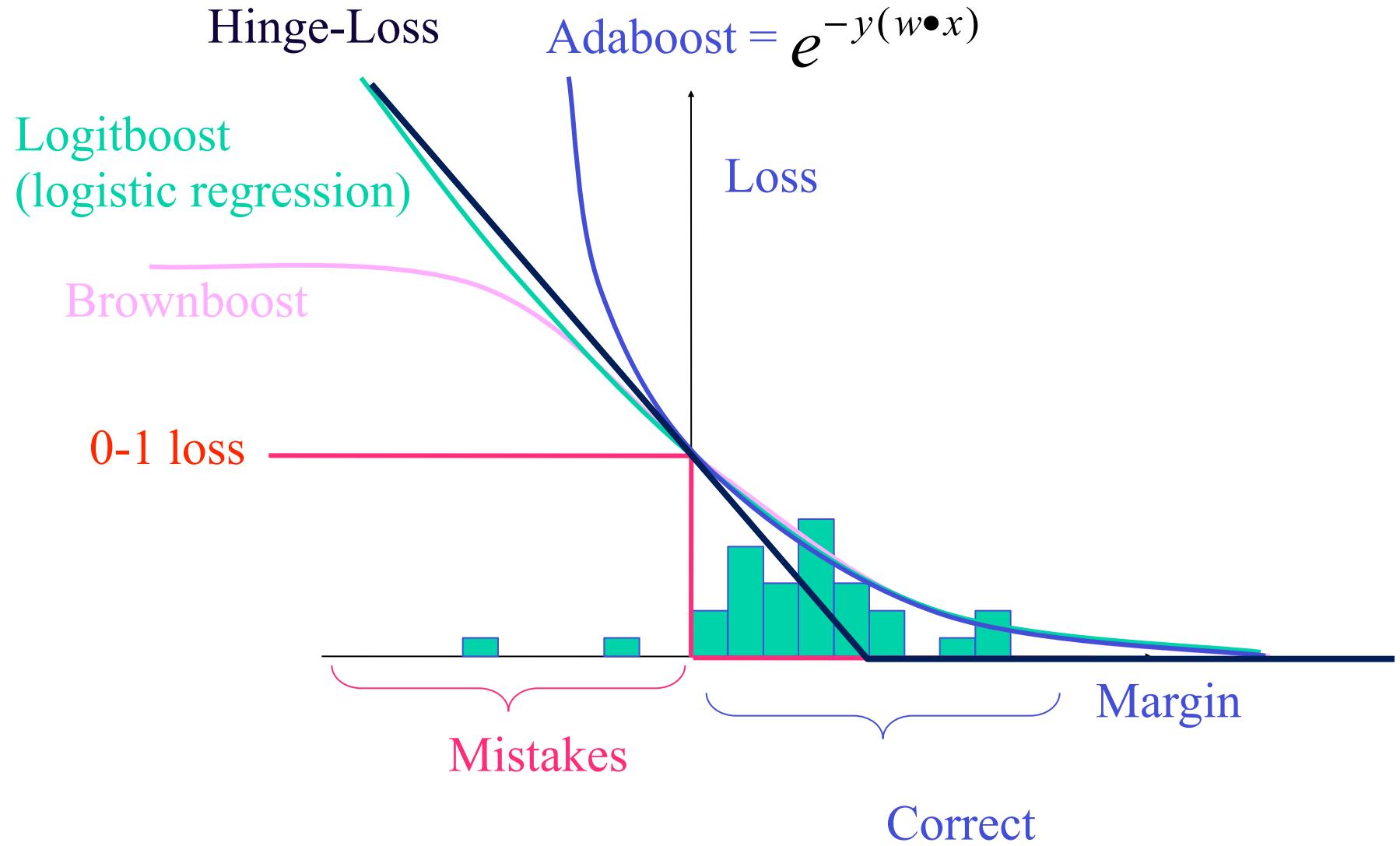
Robustboost - A new boosting algorithm

Label Noise	Adaboost	Logitboost	Robustboost
0%	0.8% \pm 0.2%	0.8% \pm 0.1%	2.9% \pm 0.2%
20%	33.3% \pm 0.7%	31.6% \pm 0.6%	22.2 \pm 0.8%

error with respect to original (noiseless) labels

20%	22.1% \pm 1.2%	19.4% \pm 1.3%	3.7% \pm 0.4%
-----	------------------	------------------	-----------------

Approximating mistake loss with convex functions

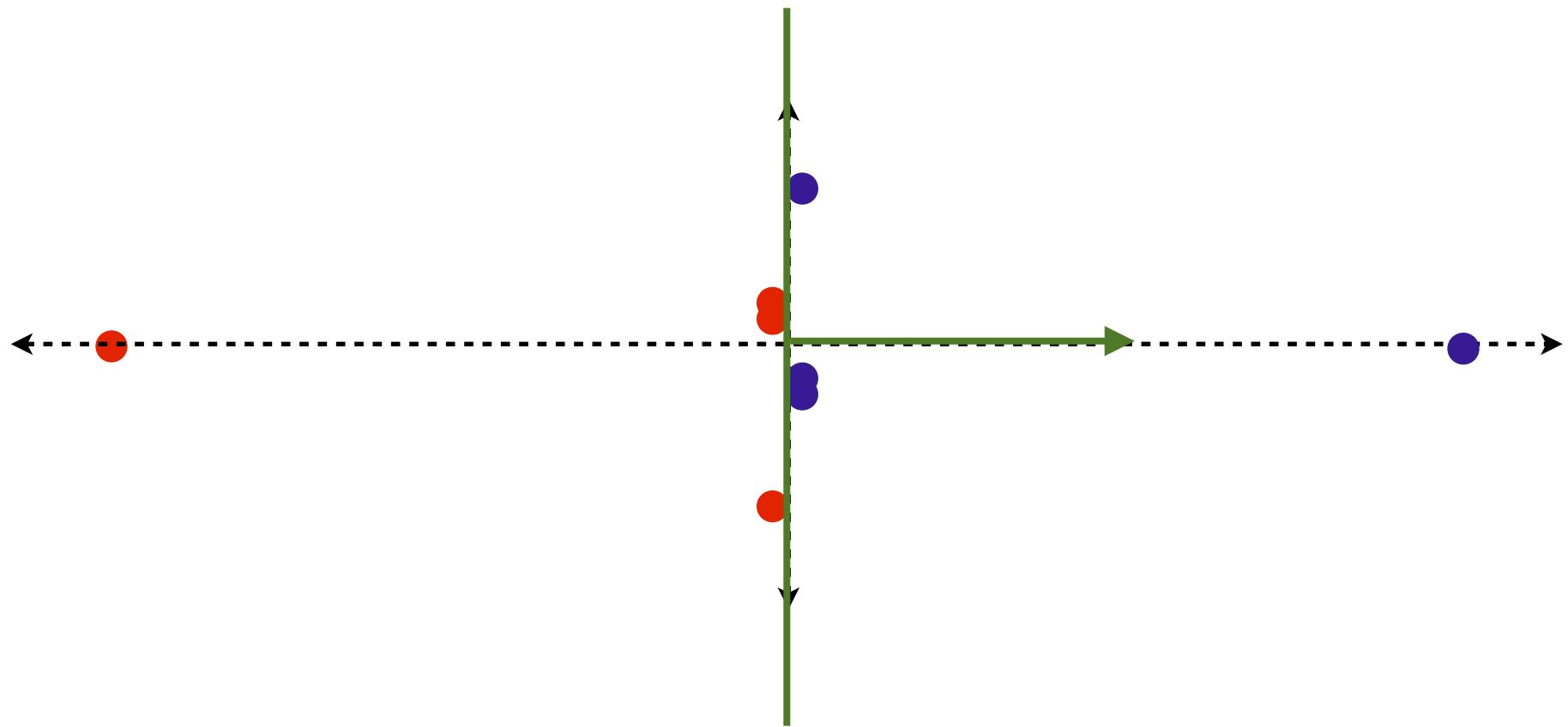


Label noise and convex loss functions

- Algorithms for learning a classifier based on minimizing a convex loss function: perceptron, Adaboost, Logitboost, Logistic regression, soft margins SVM.
- Work well when data is linearly separable.
- Can get into trouble when not linearly separable.
- **Problem:** Convex loss functions are a poor approximation for classification error.
- **But:** No known efficient algorithms for

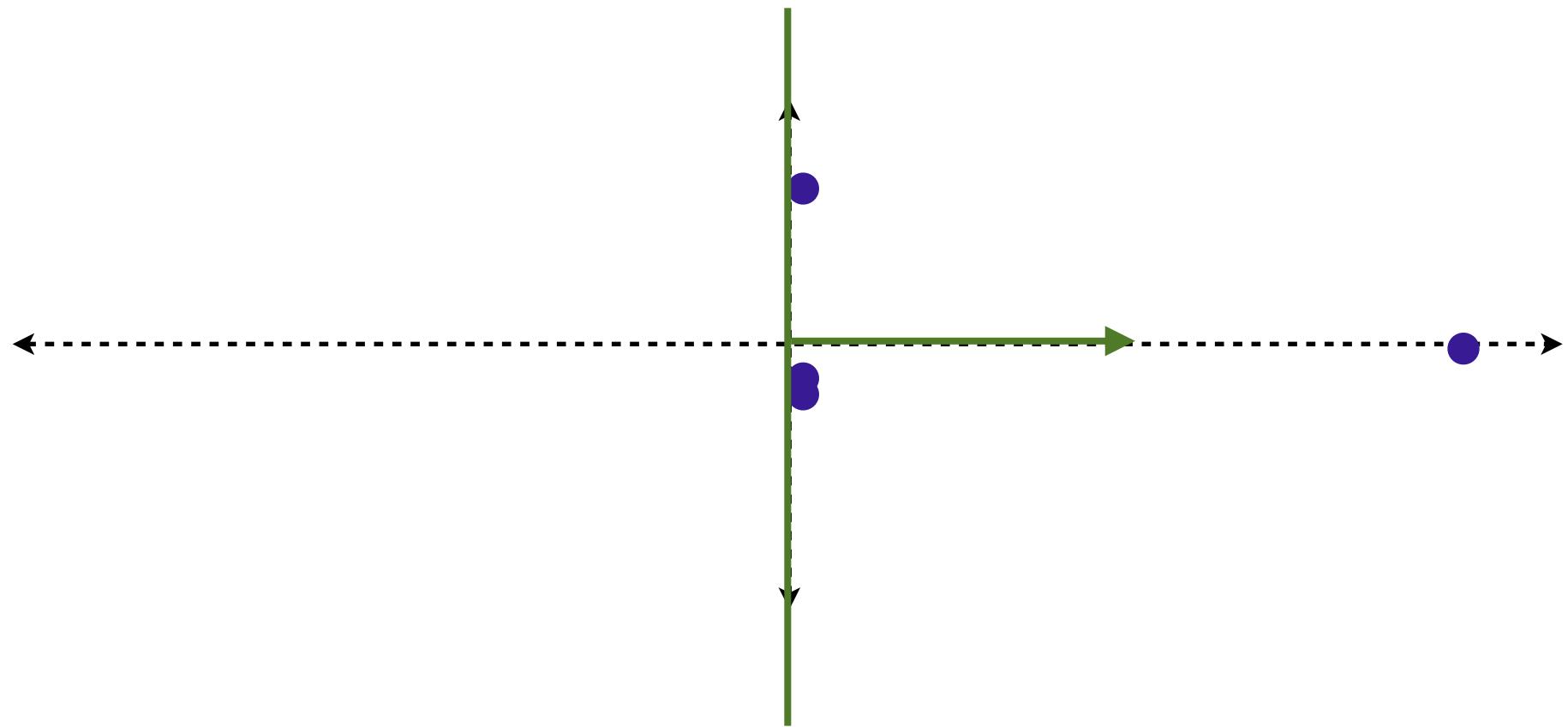
Random label noise defeats any convex loss function

[Servedio, Long 2010]



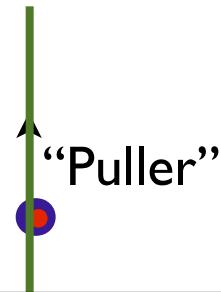
Considering one symmetric half

[Servedio, Long 2010]



Adding random label noise

[Servedio, Long 2010]



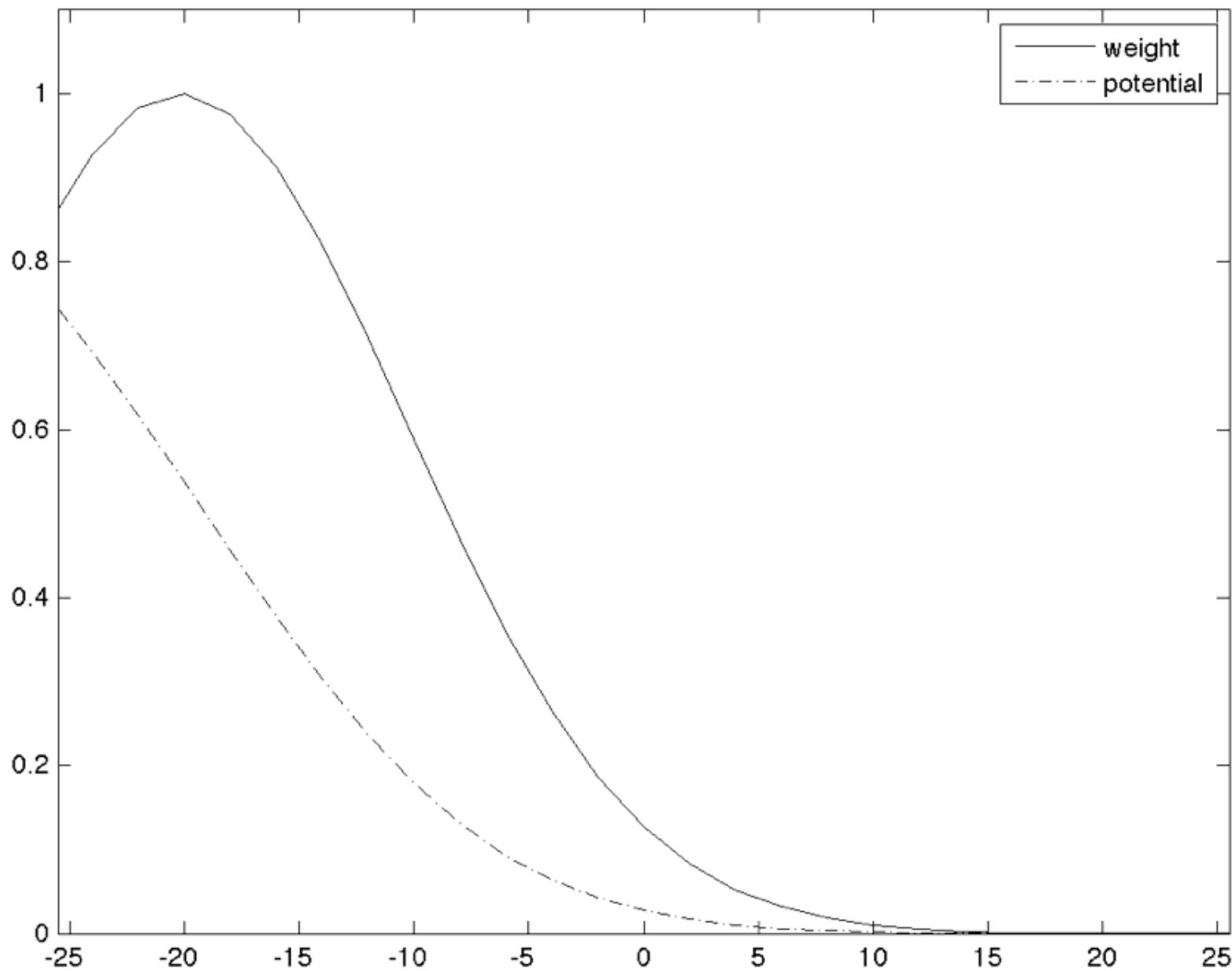
Theorem: for any convex loss function there exists a linearly separable distribution such that when independent label noise is added, the linear classifier that minimizes the loss function has very high classification error.

←----- “Margin” -----→

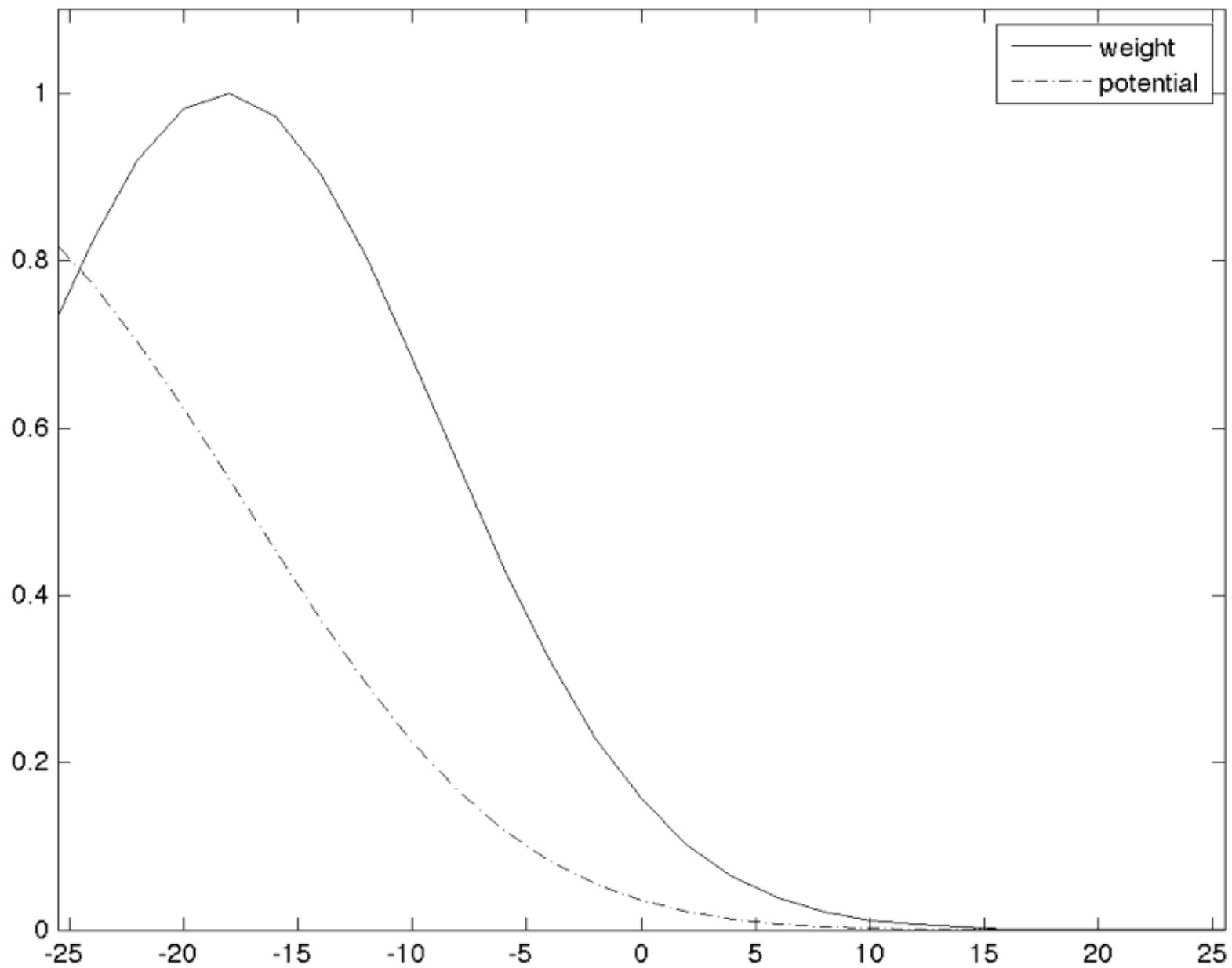
Boost by majority, Brownboost,

- Target error set at start.
- Defines how many boosting iterations are needed
- The loss function depends on the time-to-finish.
- Close to end - give up on examples with large negative margins.

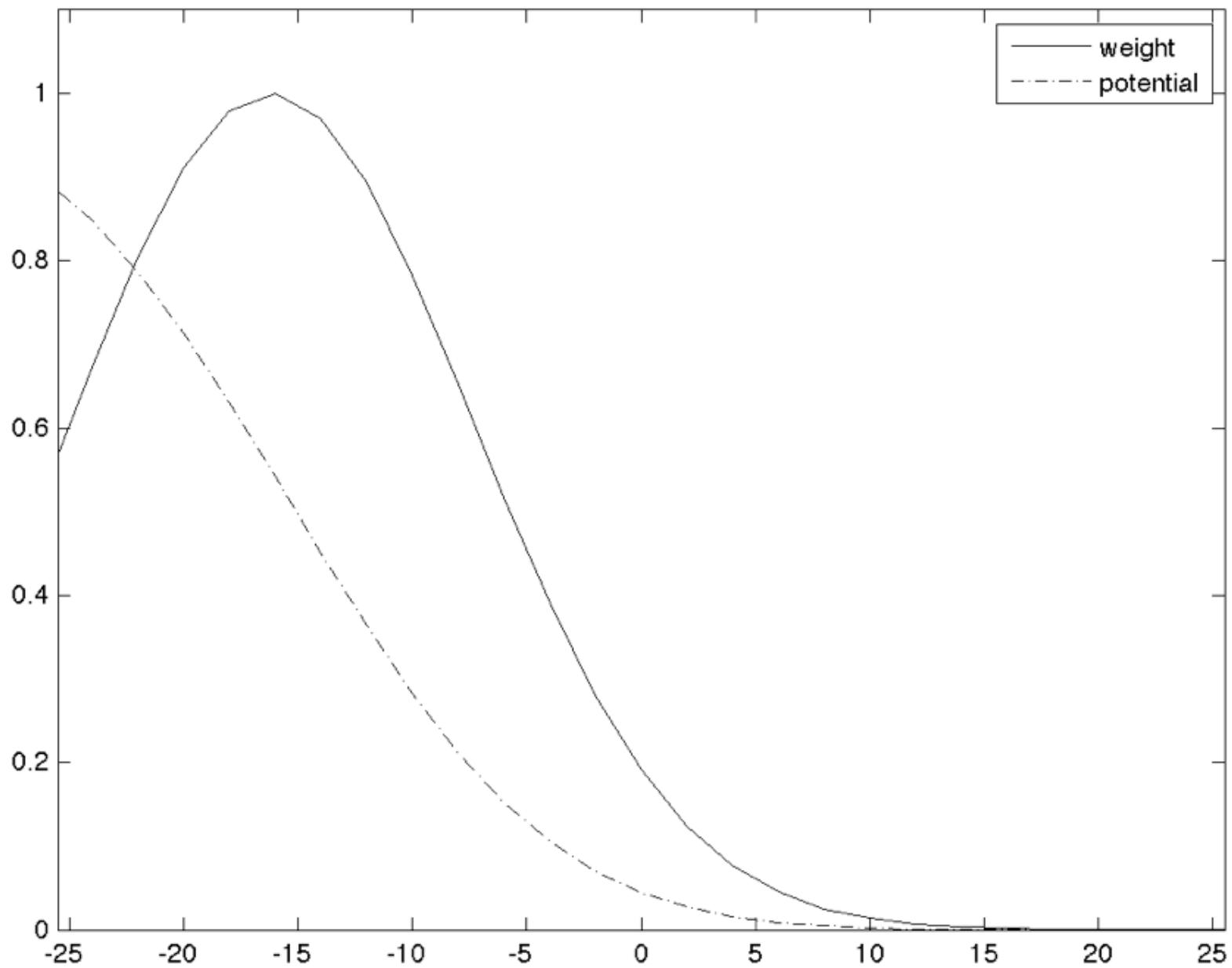
$t=1$



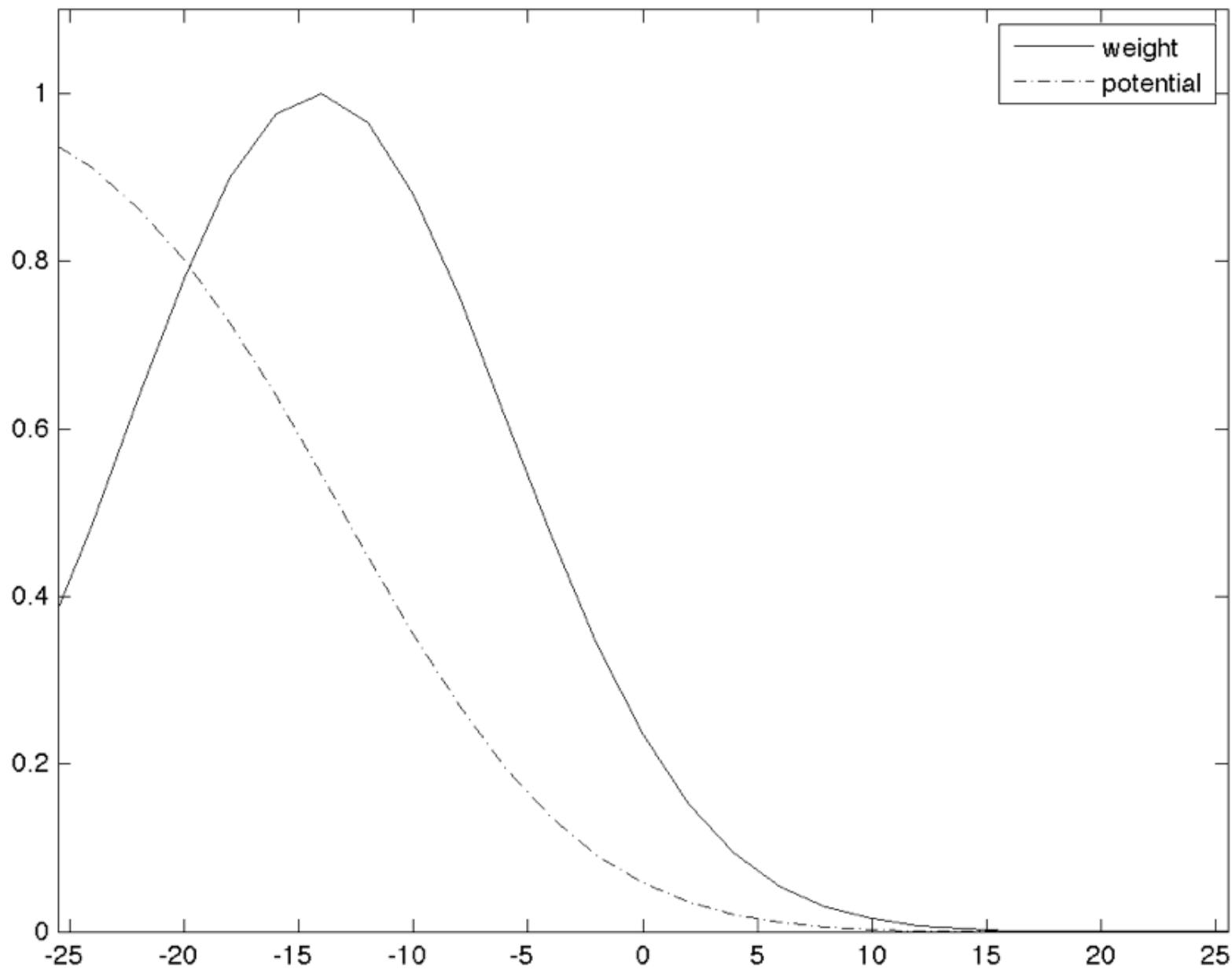
t=11



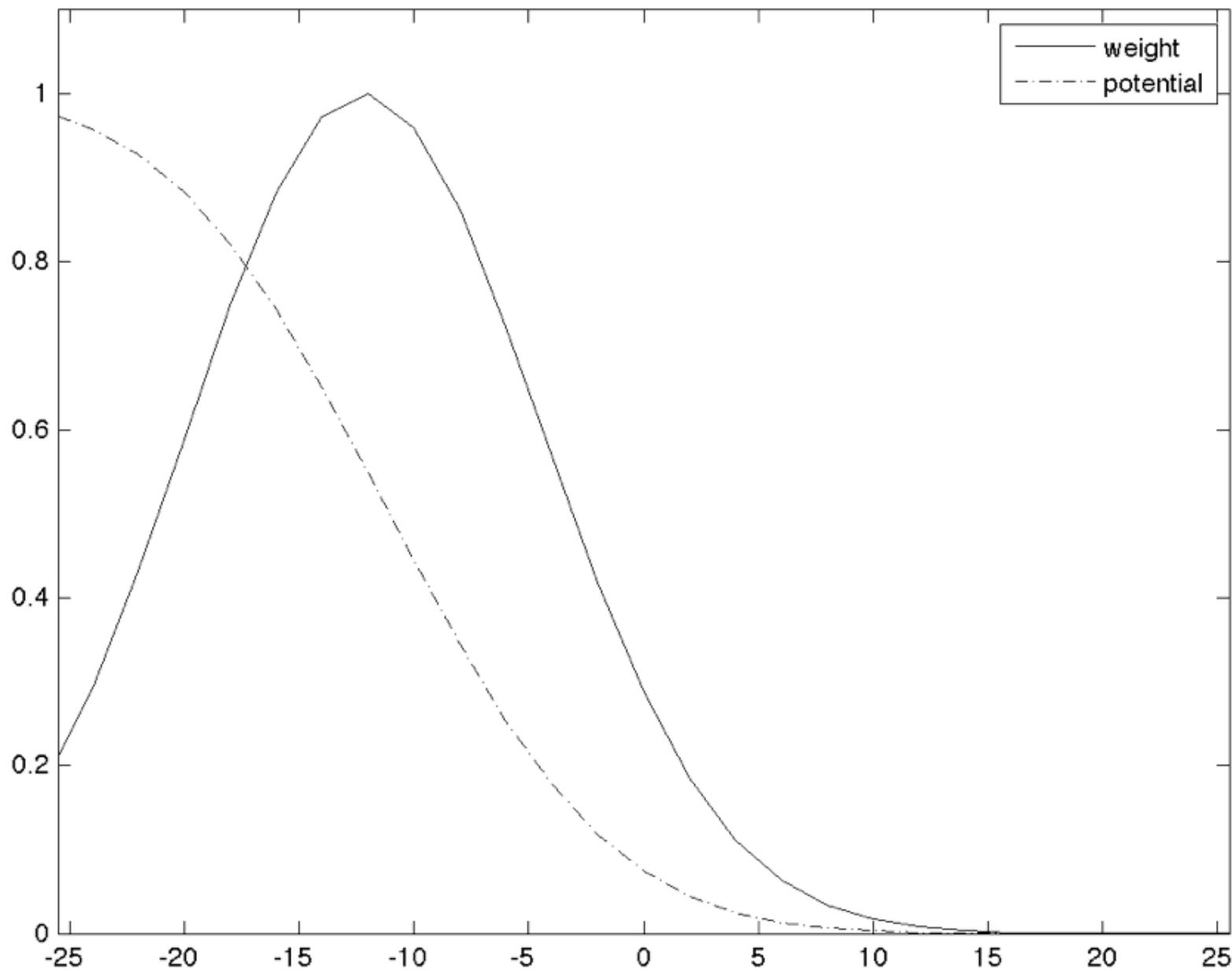
$t=21$



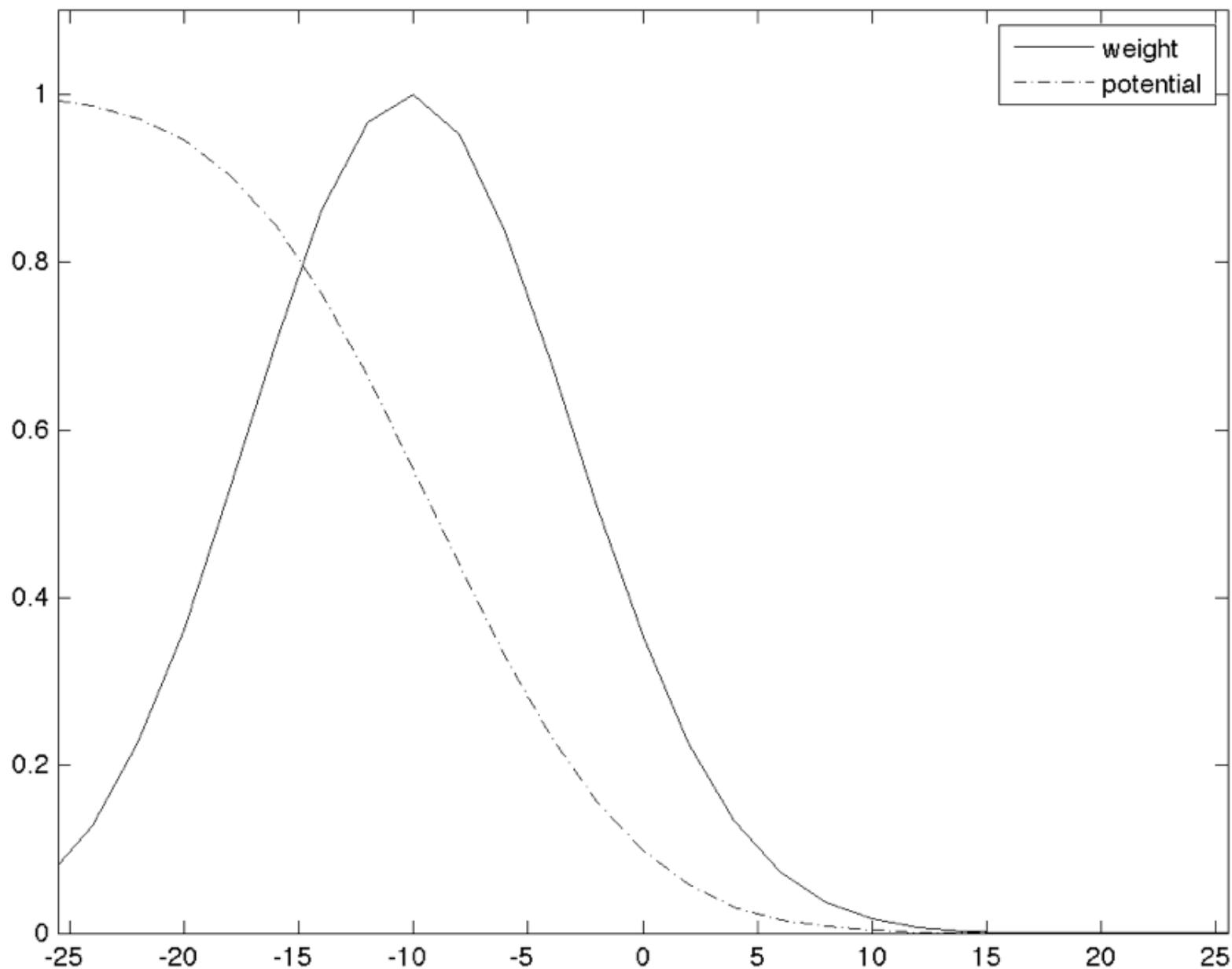
$t=31$



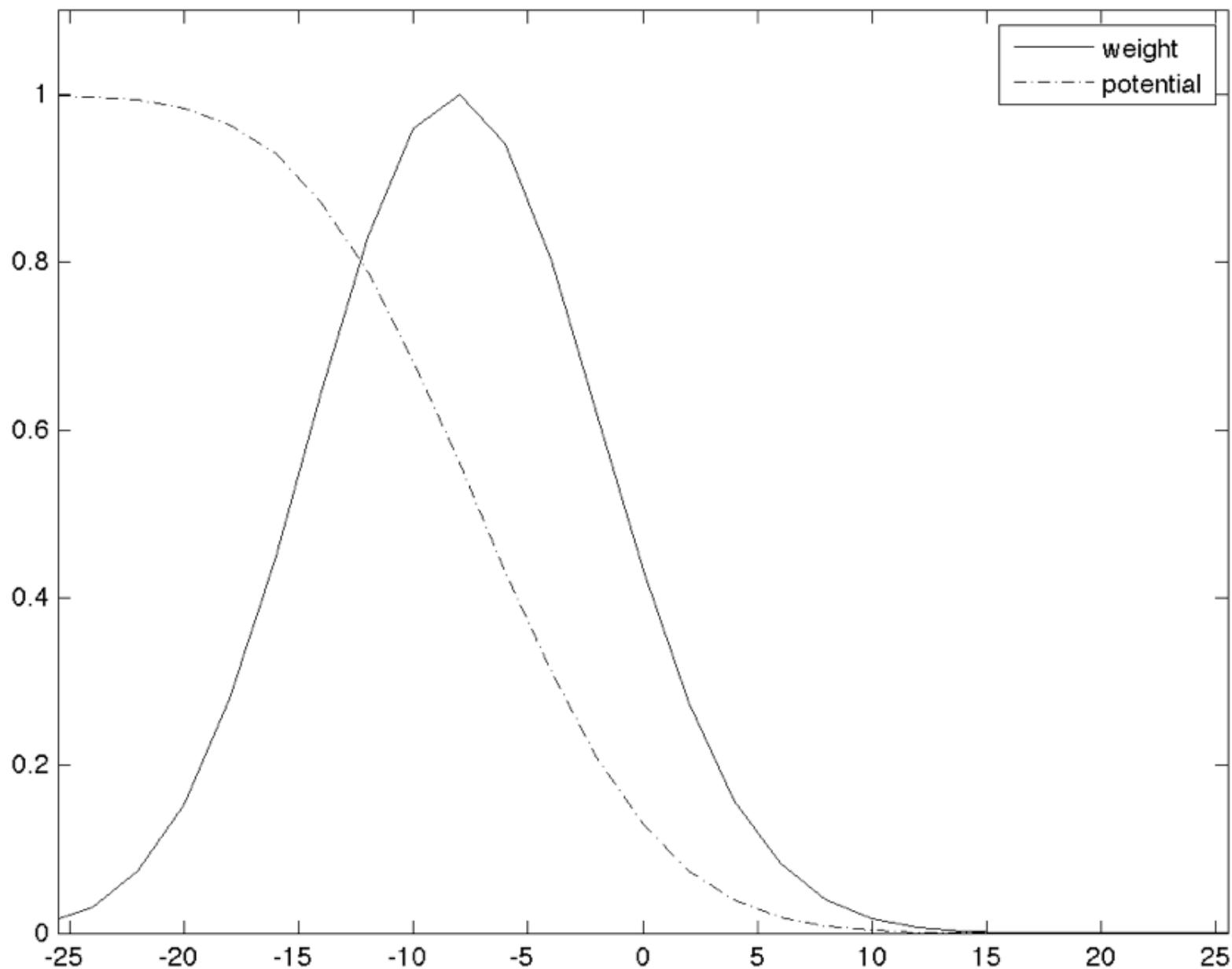
t=41



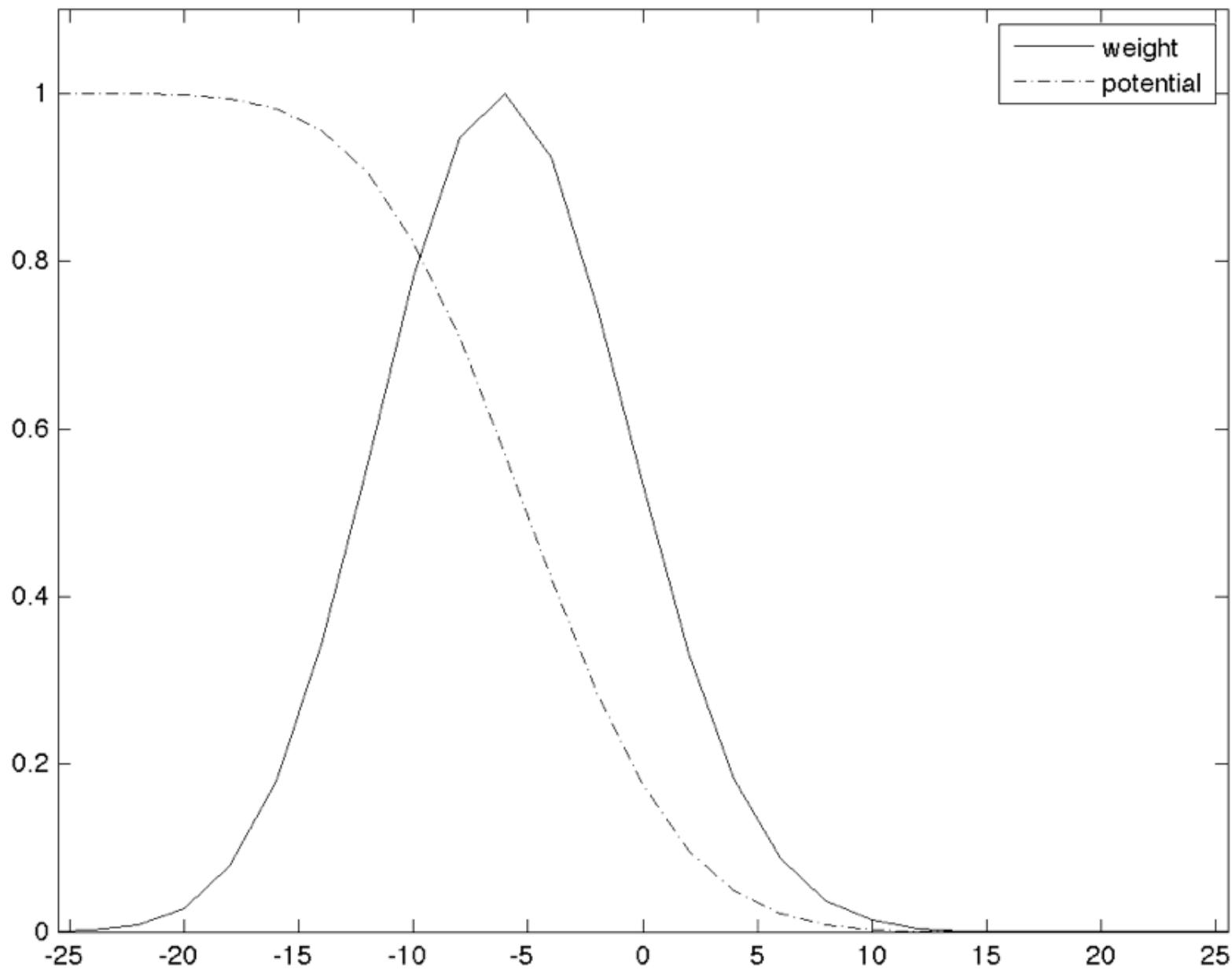
t=51



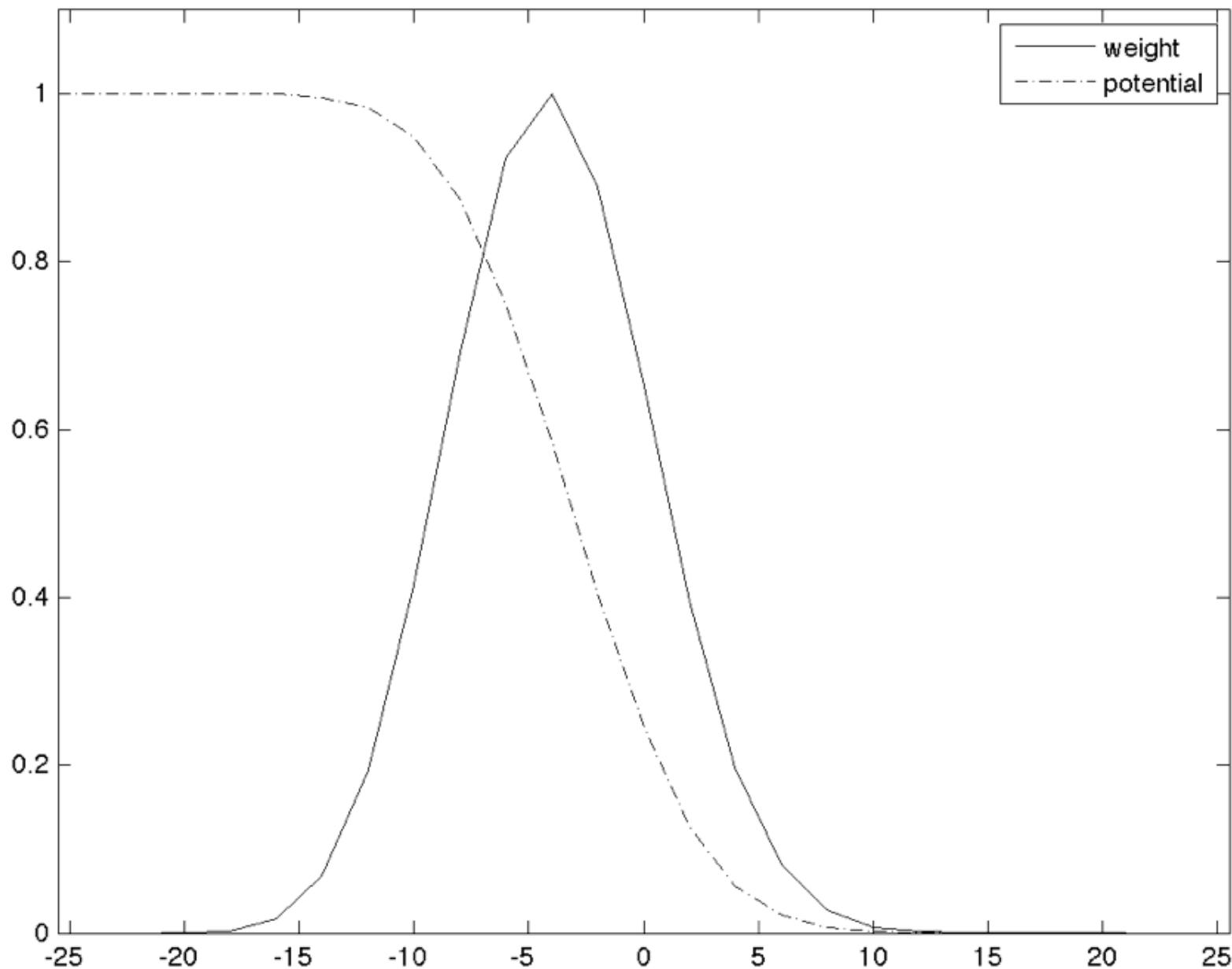
t=61



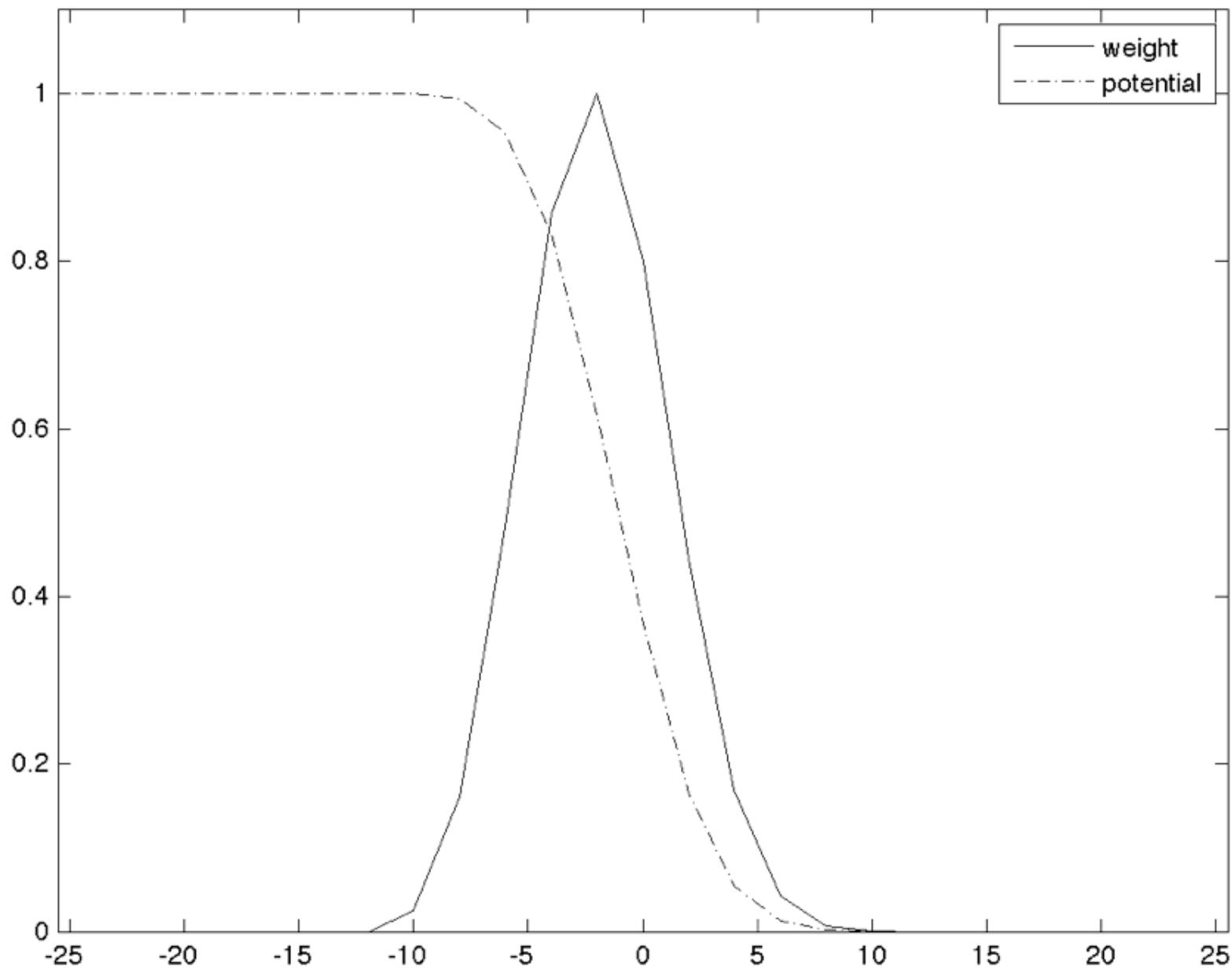
t=71



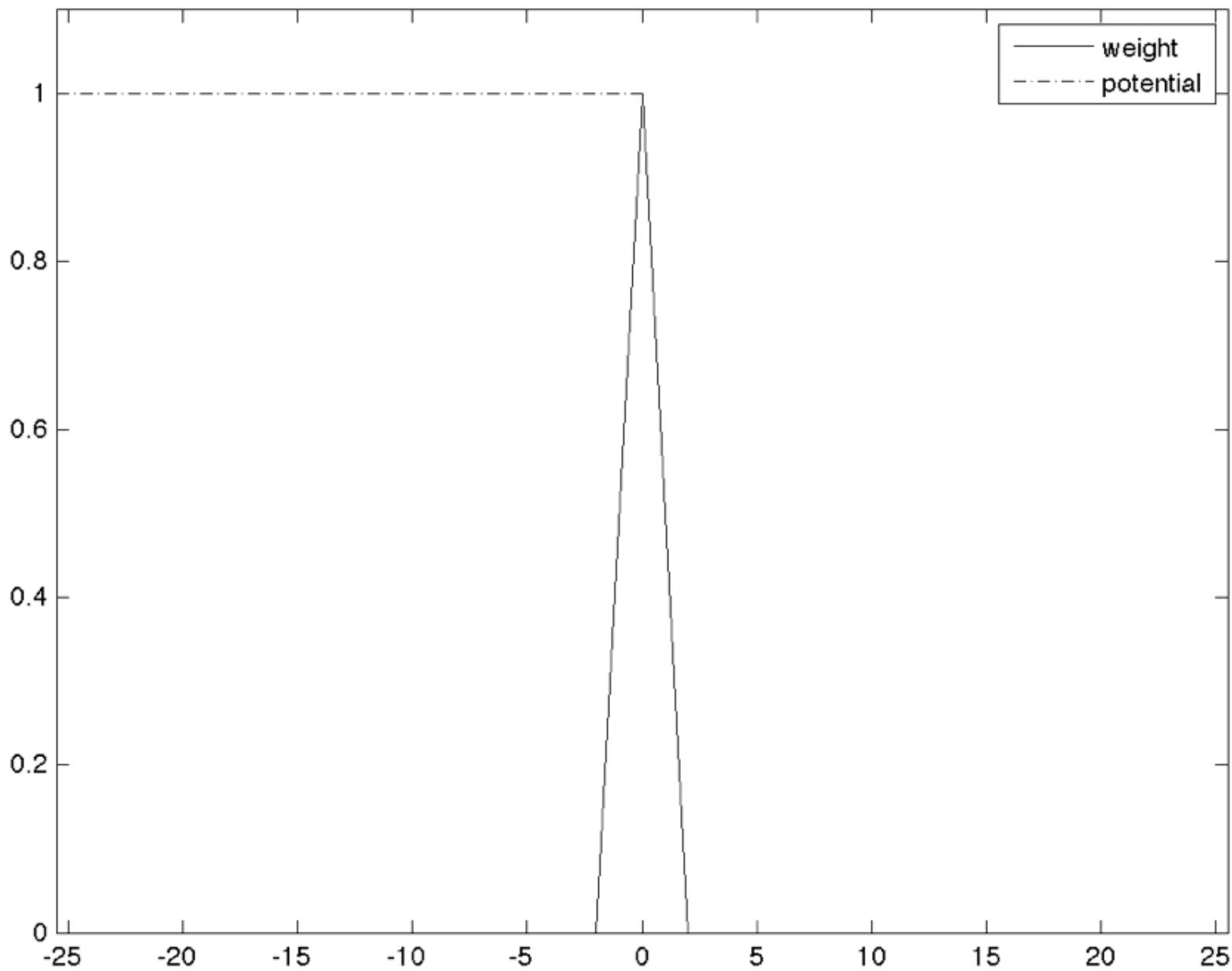
t=81



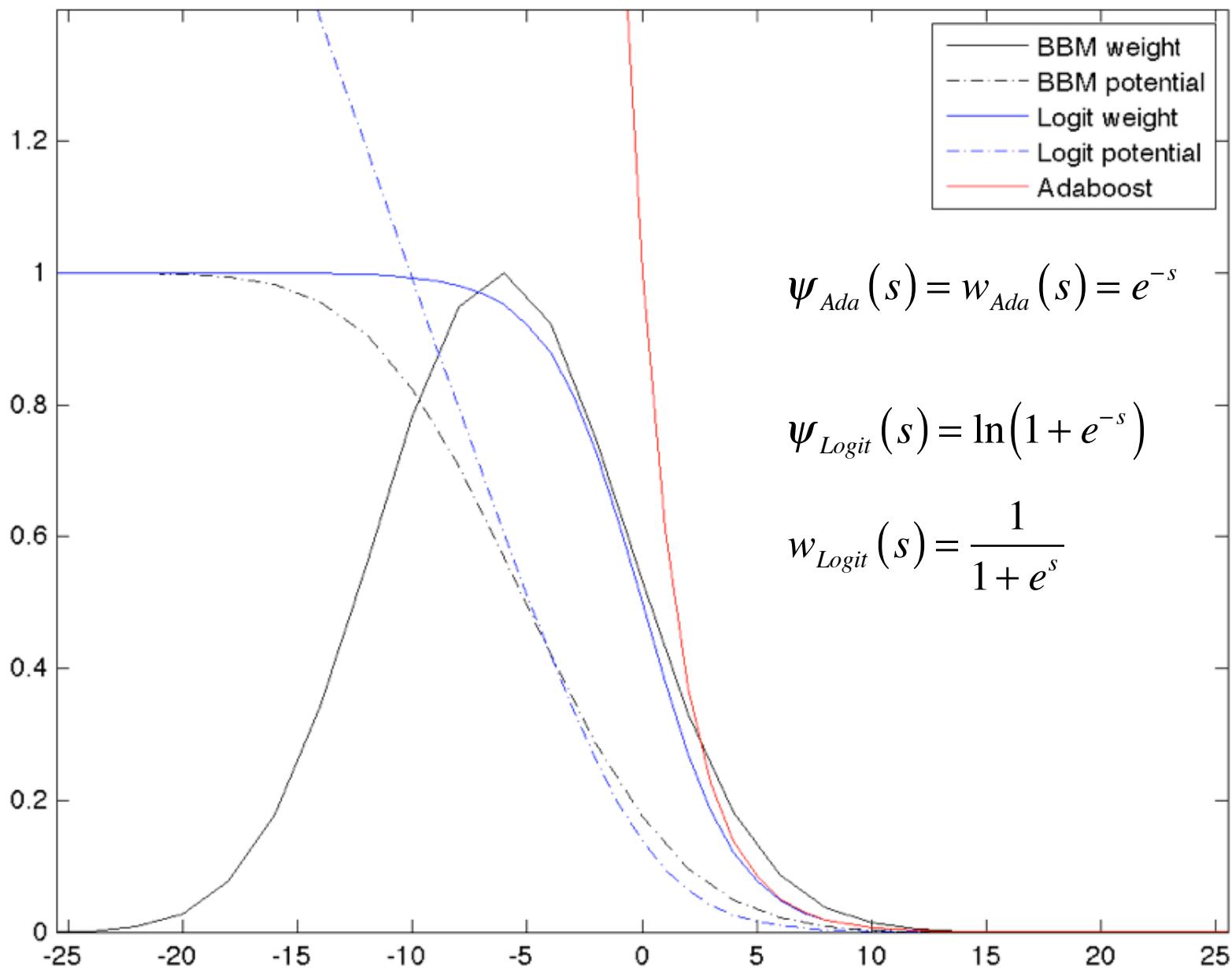
t=91



$t=101$

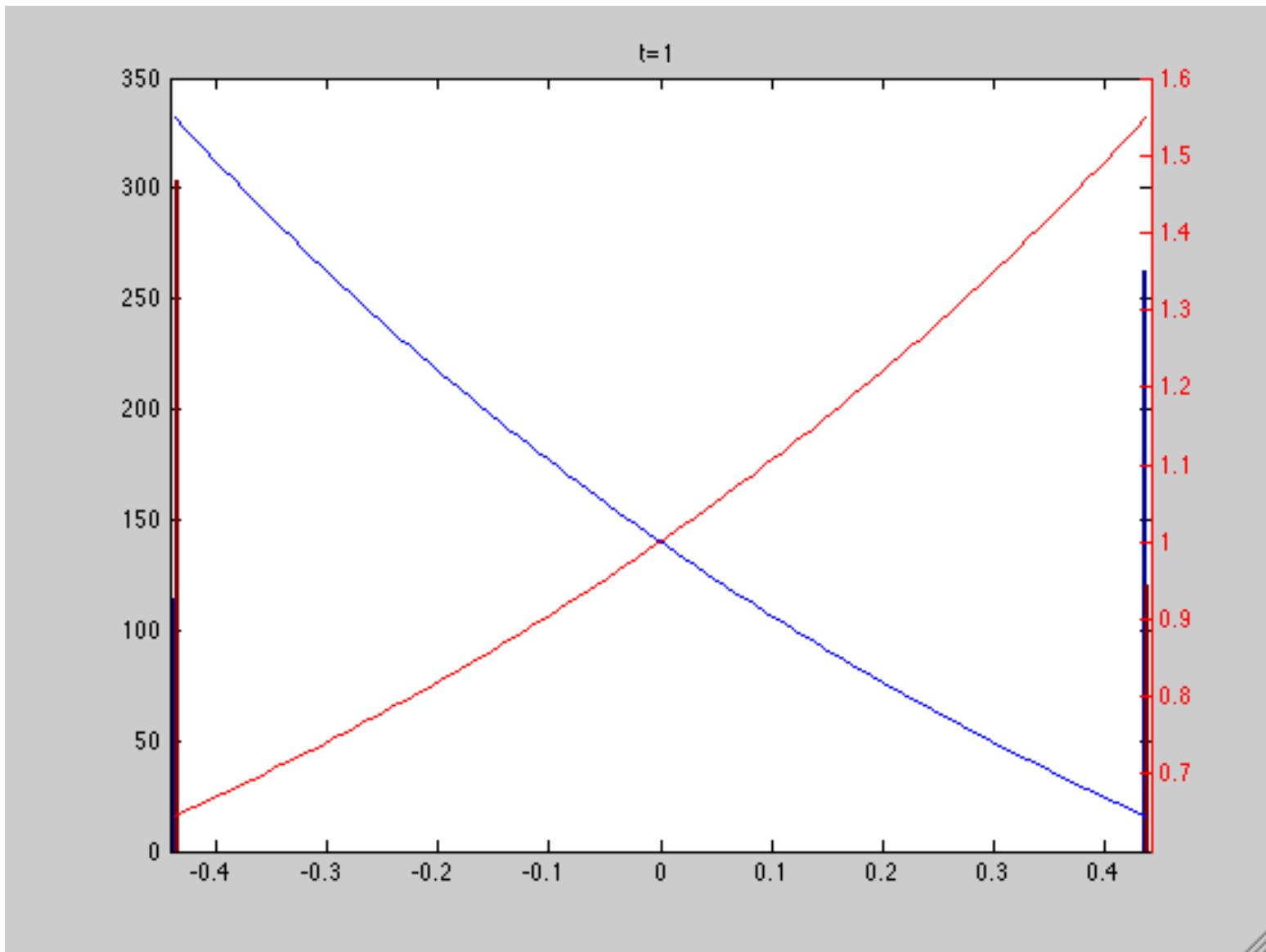


BBM/Logitboost/Adaboost

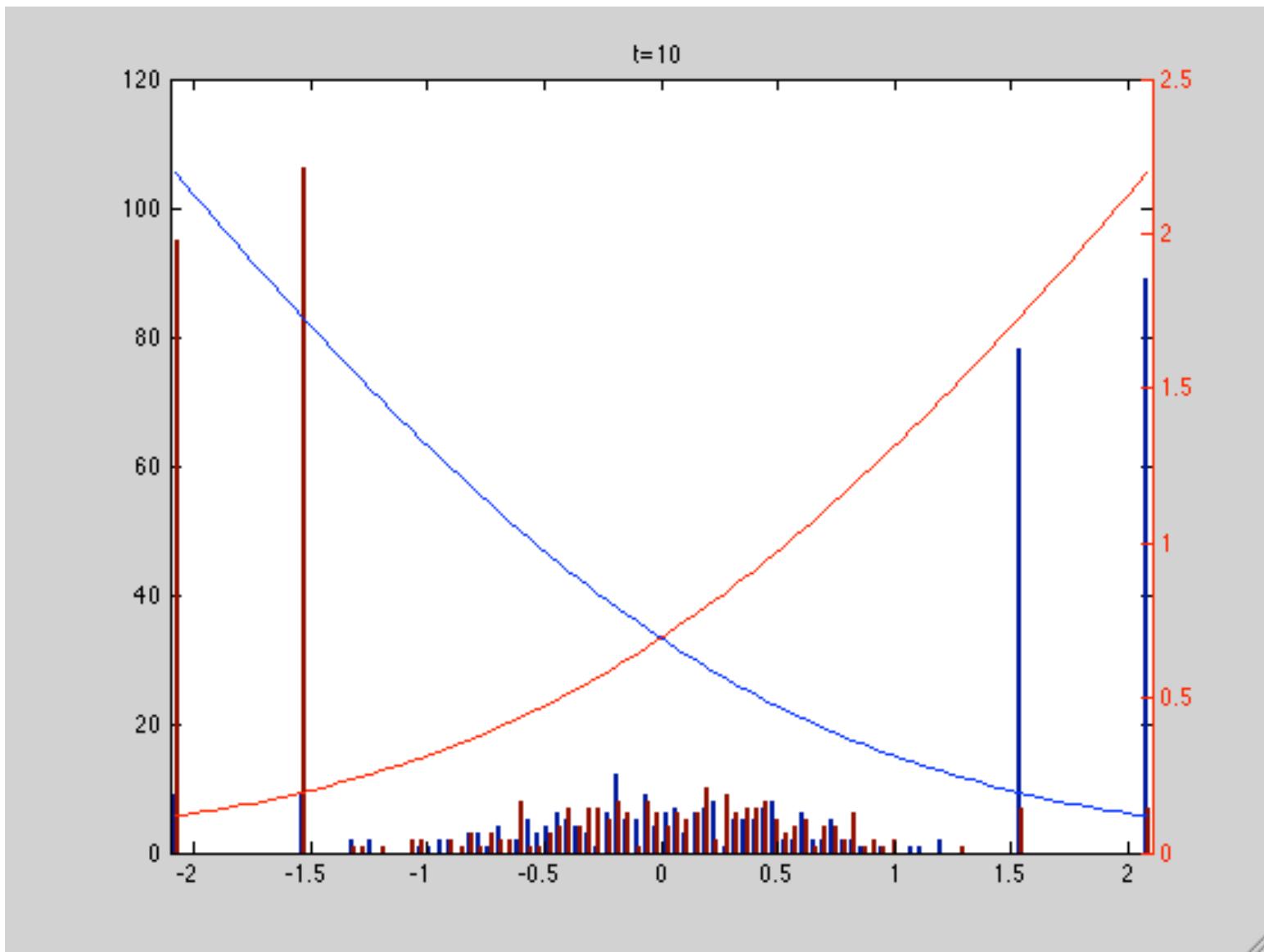


Experimental Results on Long/Servedio synthetic example

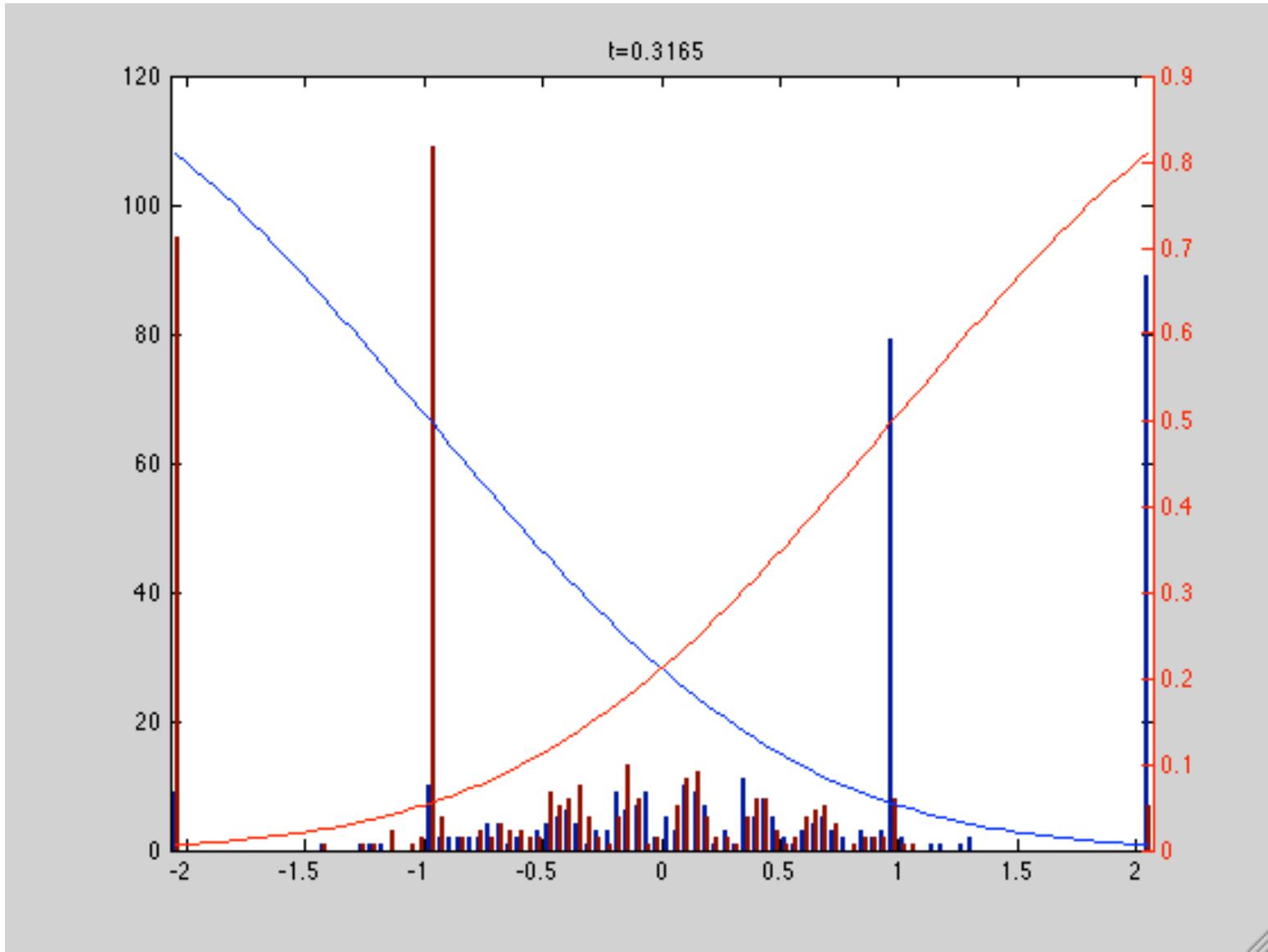
Adaboost on Long/Servedio



LogitBoost on Long/Servedio



Robustboost on Long/Servedio



Experimental Results on real-world data

Robustboost - A new boosting algorithm

Label Noise	Adaboost	Logitboost	Robustboost
0%	0.8% \pm 0.2%	0.8% \pm 0.1%	2.9% \pm 0.2%
20%	33.3% \pm 0.7%	31.6% \pm 0.6%	22.2 \pm 0.8%

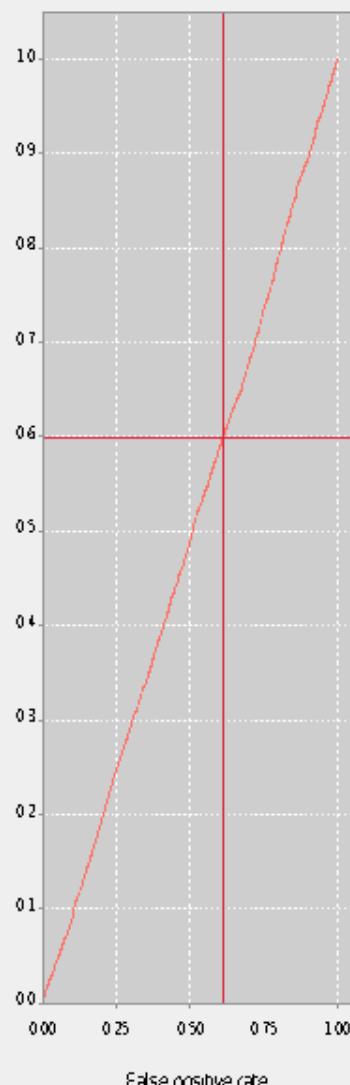
error with respect to original (noiseless) labels

20%	22.1% \pm 1.2%	19.4% \pm 1.3%	3.7% \pm 0.4%
-----	------------------	------------------	-----------------

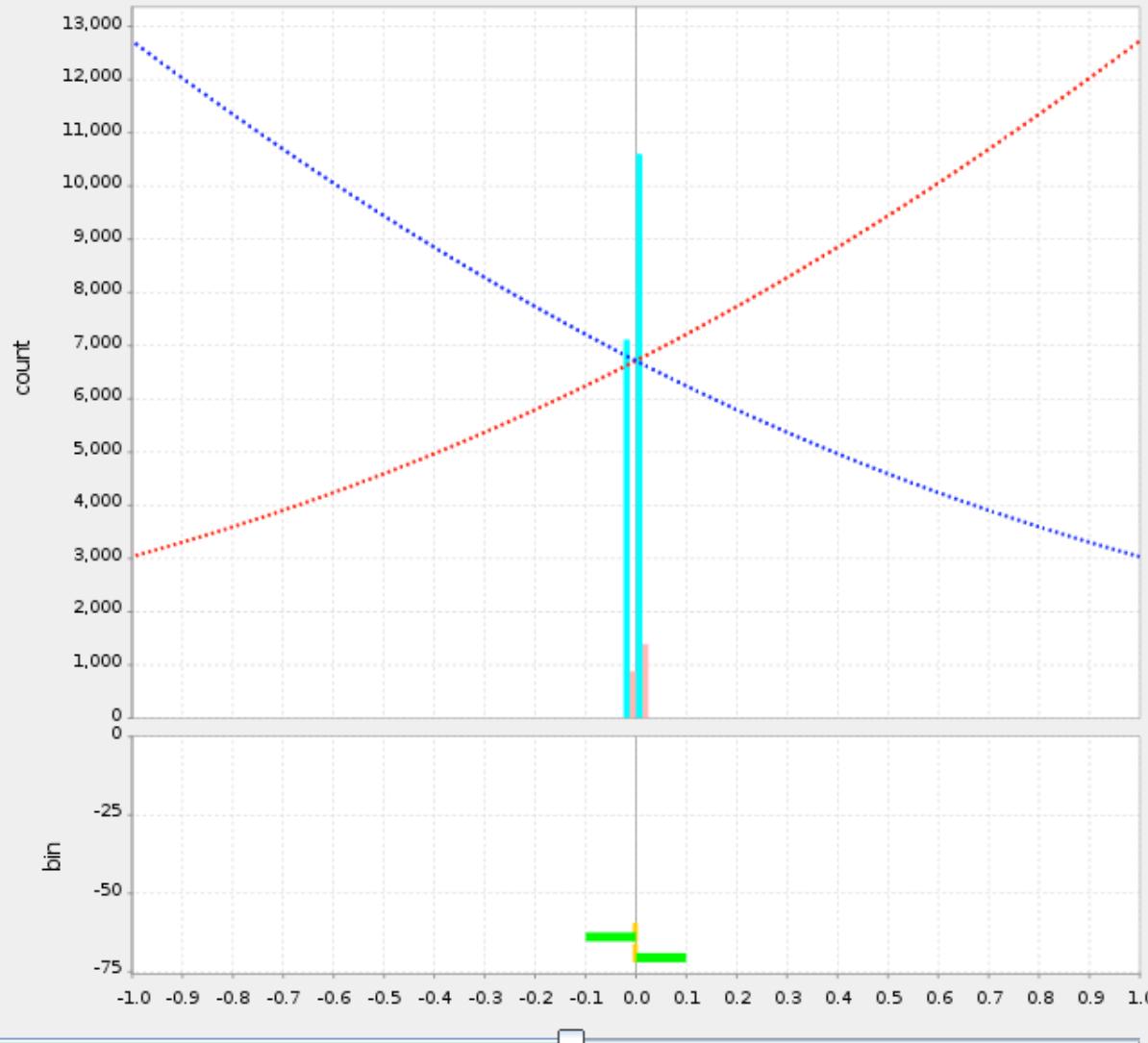
Logitboost
0% Noise

ROC

True positive rate



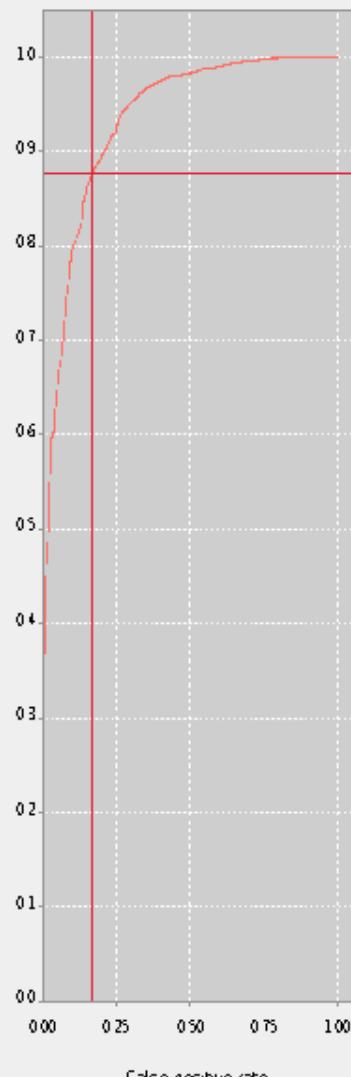
Histogram



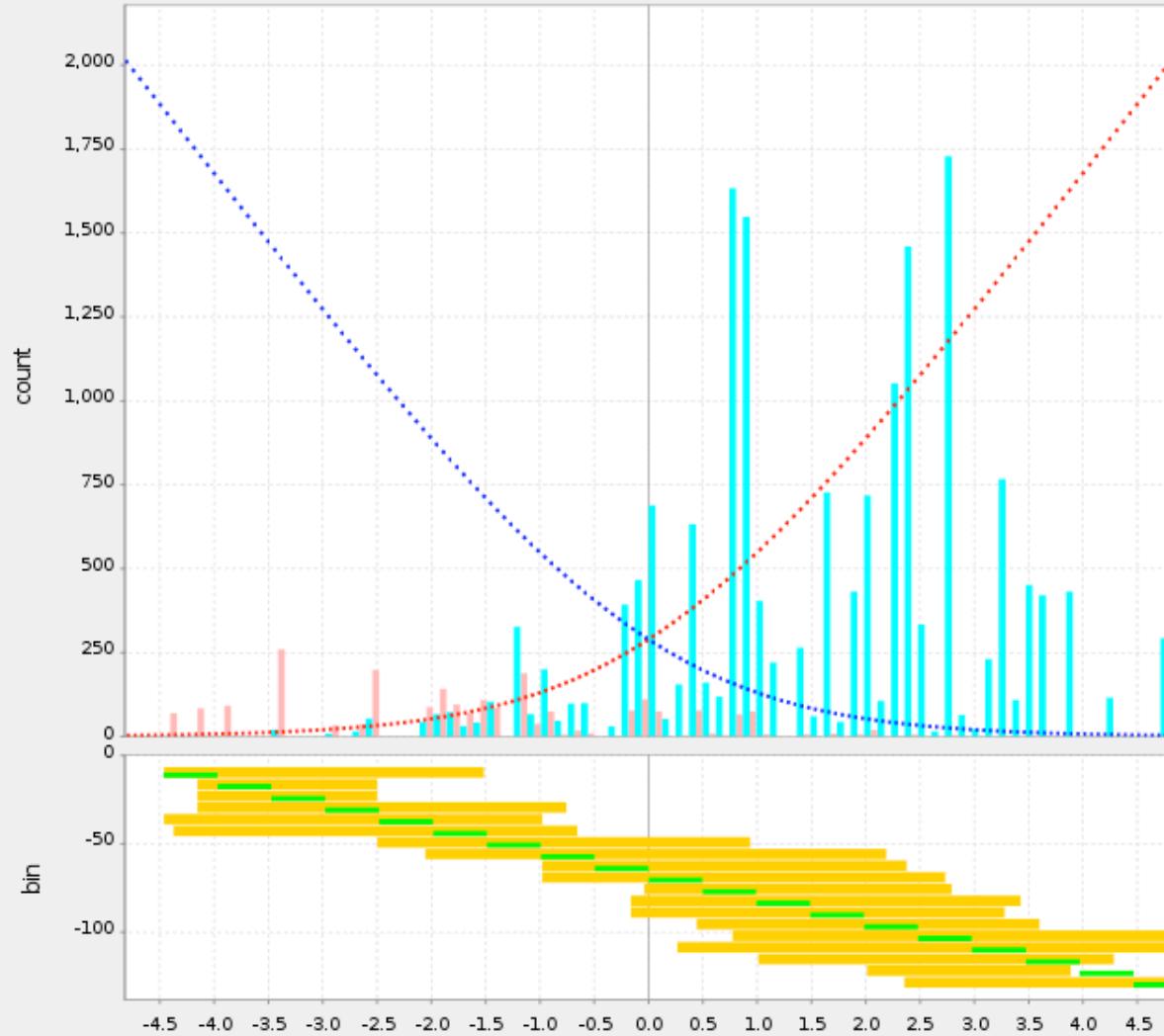
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 20
Iteration 30
Iteration 40
Iteration 50
Iteration 60
Iteration 70
Iteration 80
Iteration 90
Iteration 100
Iteration 200

ROC

True positive rate



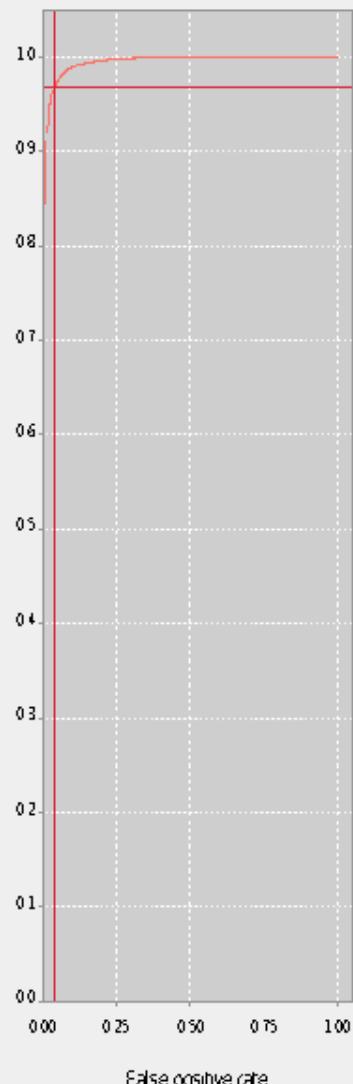
Histogram



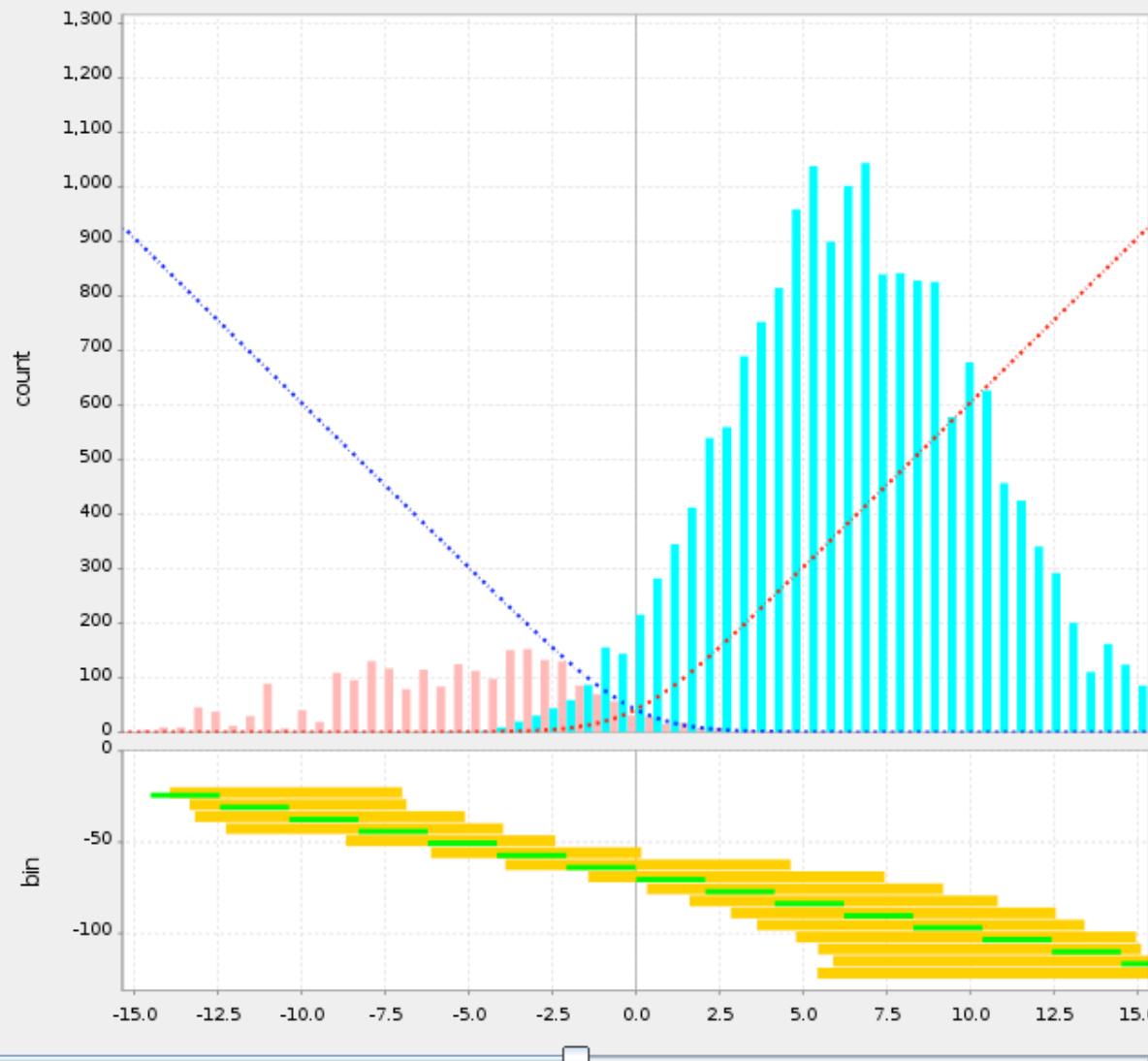
- Iteration 0
- Iteration 1
- Iteration 2
- Iteration 3
- Iteration 4
- Iteration 5
- Iteration 6
- Iteration 7
- Iteration 8
- Iteration 9**
- Iteration 10
- Iteration 20
- Iteration 30
- Iteration 40
- Iteration 50
- Iteration 60
- Iteration 70
- Iteration 80
- Iteration 90
- Iteration 100
- Iteration 200

ROC

True positive rate



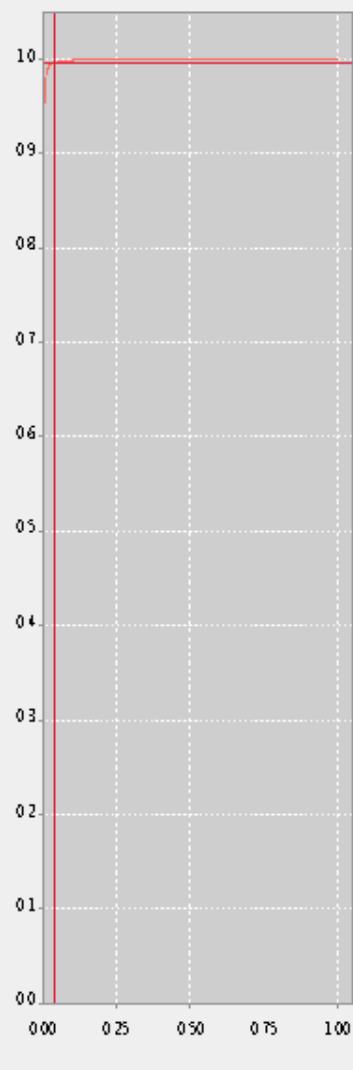
Histogram



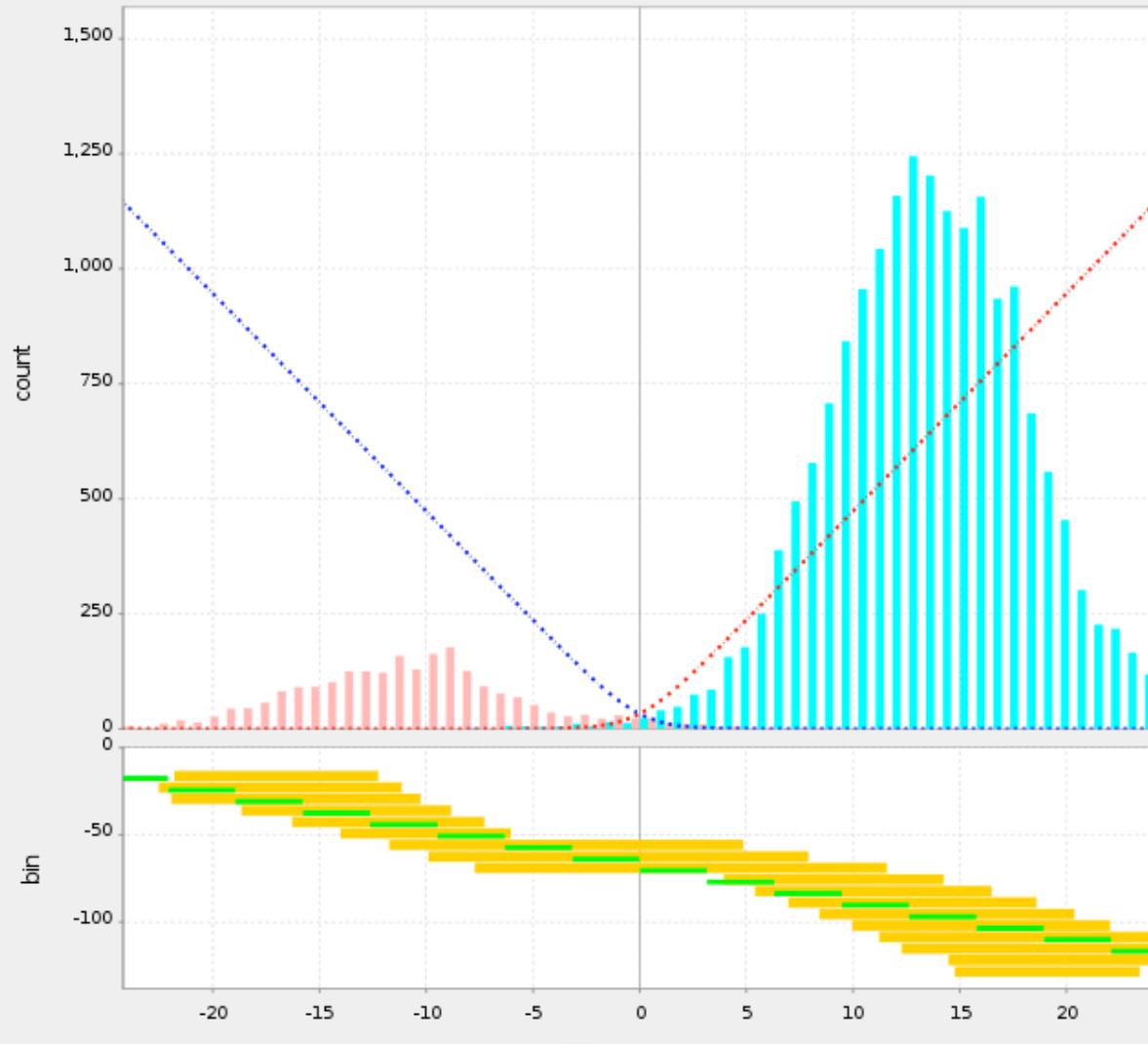
- Iteration 0
- Iteration 1
- Iteration 2
- Iteration 3
- Iteration 4
- Iteration 5
- Iteration 6
- Iteration 7
- Iteration 8
- Iteration 9
- Iteration 10
- Iteration 20
- Iteration 30
- Iteration 40
- Iteration 50
- Iteration 60
- Iteration 70
- Iteration 80
- Iteration 90
- Iteration 100**
- Iteration 200

ROC

True positive rate



Histogram

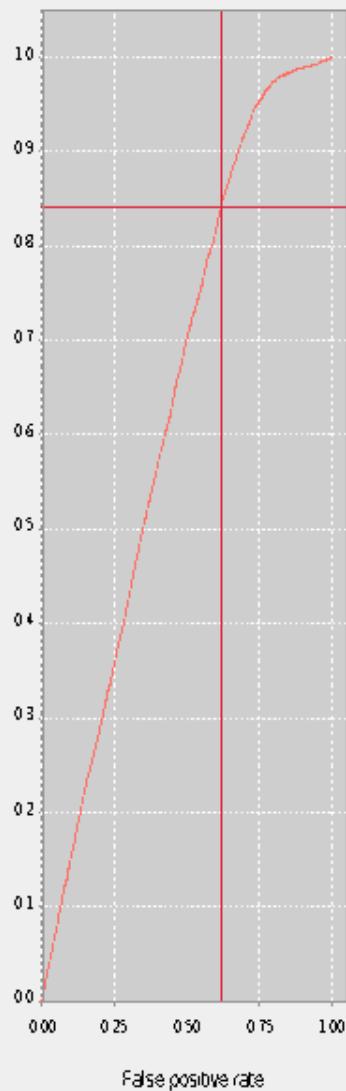


- Iteration 9
- Iteration 10
- Iteration 20
- Iteration 30
- Iteration 40
- Iteration 50
- Iteration 60
- Iteration 70
- Iteration 80
- Iteration 90
- Iteration 100
- Iteration 200
- Iteration 300
- Iteration 400
- Iteration 500
- Iteration 600
- Iteration 700
- Iteration 800
- Iteration 900
- Iteration 1000
- Iteration 2000**

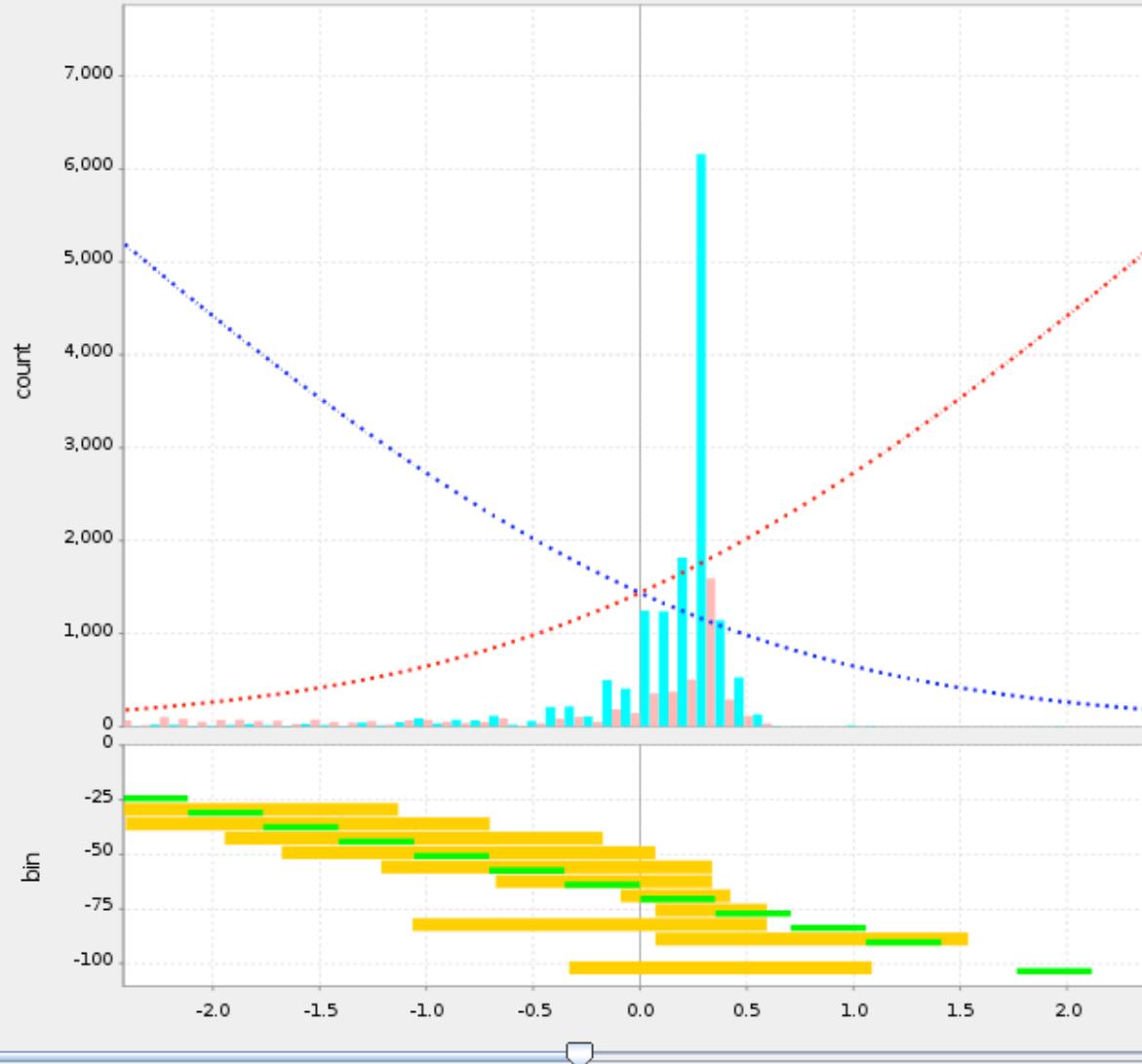
**Logitboost
20% Noise**

ROC

True positive rate



Histogram



Iteration 0

Iteration 1

Iteration 2

Iteration 3

Iteration 4

Iteration 5

Iteration 6

Iteration 7

Iteration 8

Iteration 9

Iteration 10

Iteration 20

Iteration 30

Iteration 40

Iteration 50

Iteration 60

Iteration 70

Iteration 80

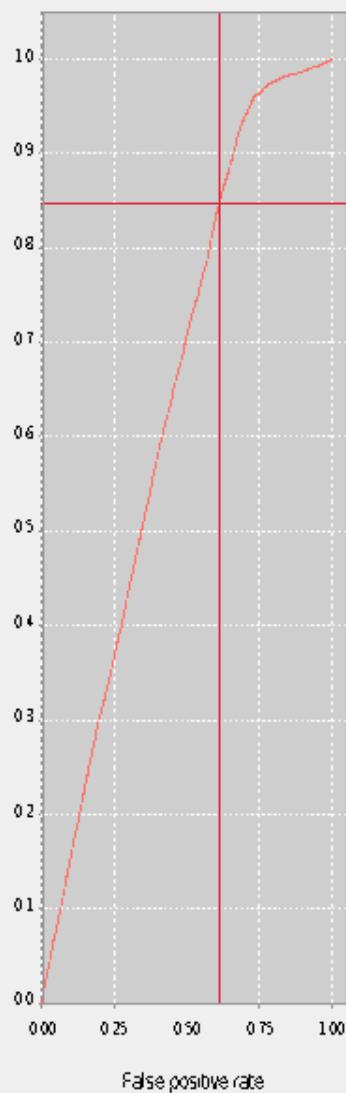
Iteration 90

Iteration 100

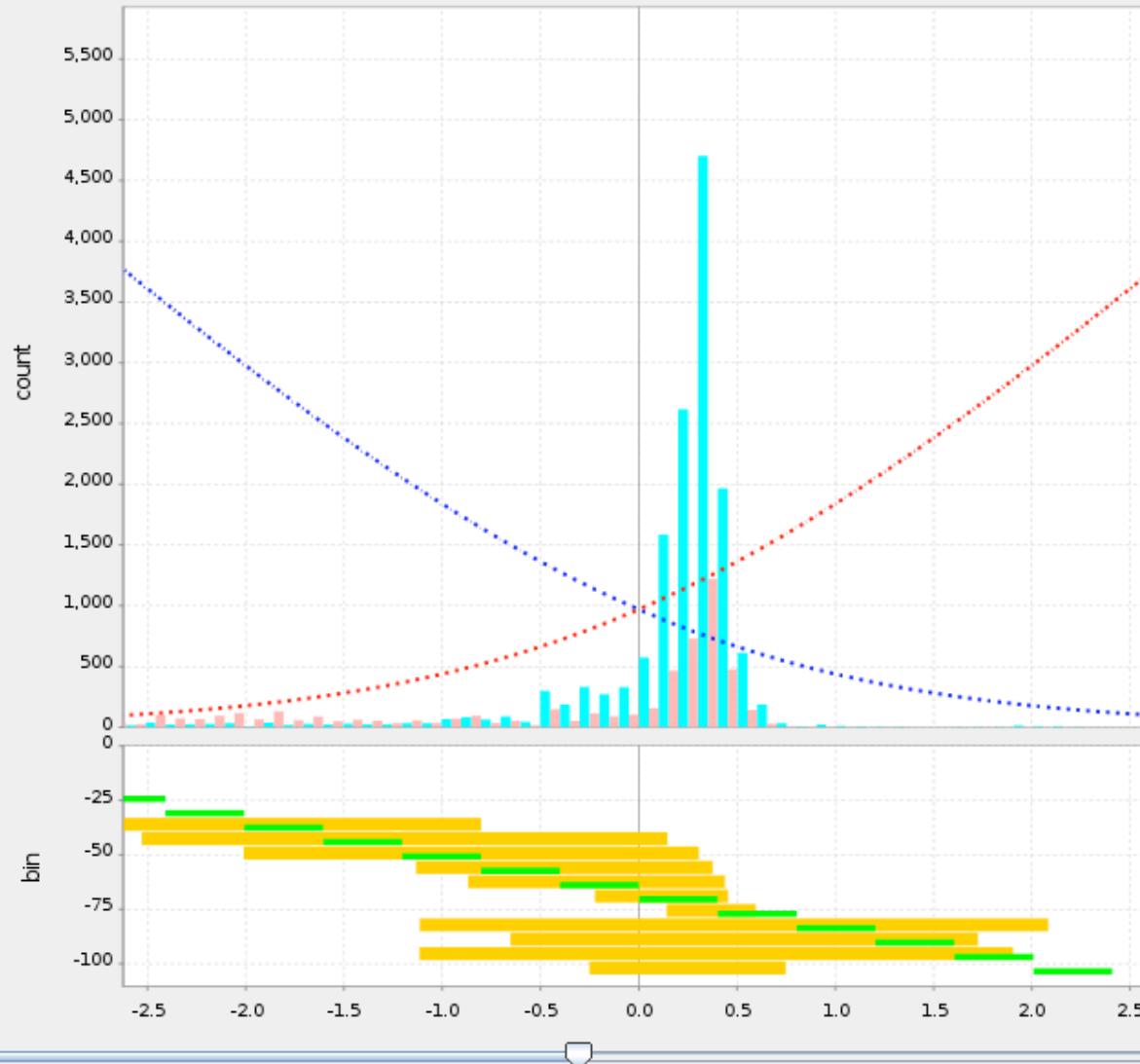
Iteration 200

ROC

True positive rate



Histogram



Iteration 0

Iteration 1

Iteration 2

Iteration 3

Iteration 4

Iteration 5

Iteration 6

Iteration 7

Iteration 8

Iteration 9

Iteration 10

Iteration 20

Iteration 30

Iteration 40

Iteration 50

Iteration 60

Iteration 70

Iteration 80

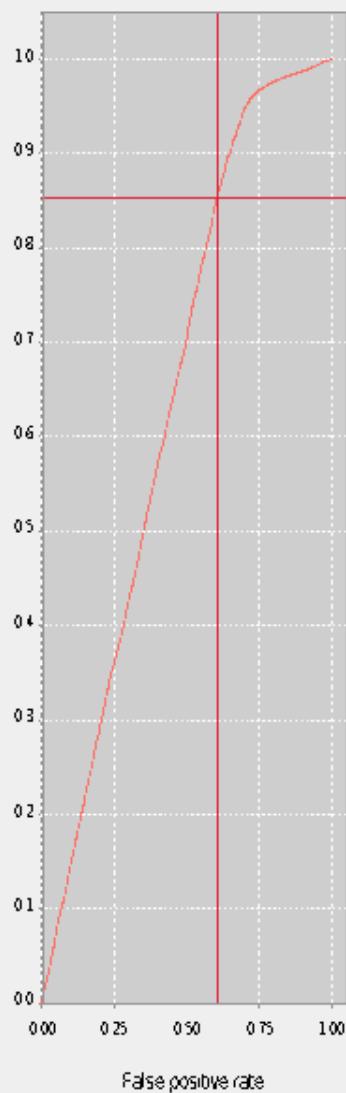
Iteration 90

Iteration 100

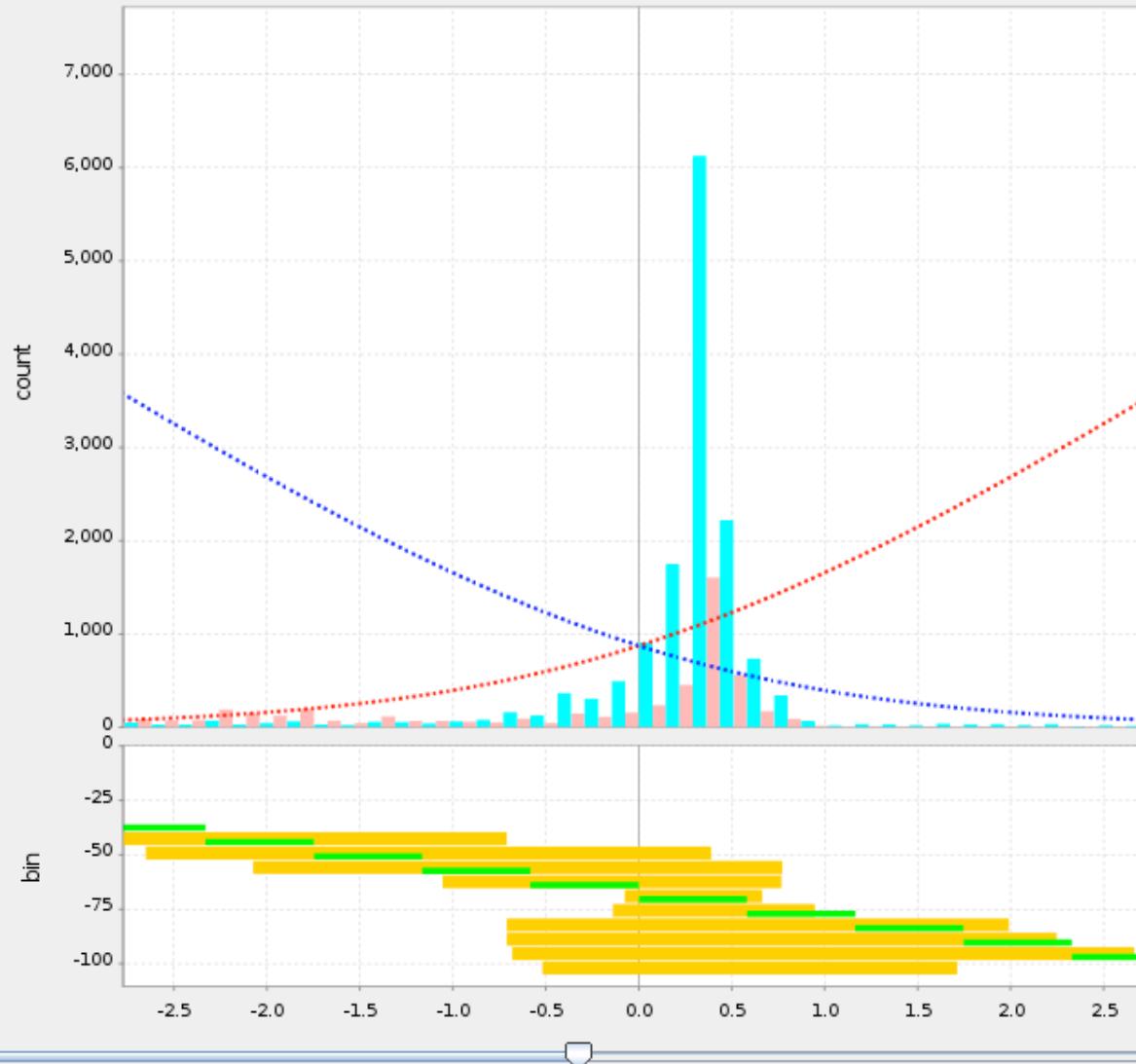
Iteration 200

ROC

True positive rate



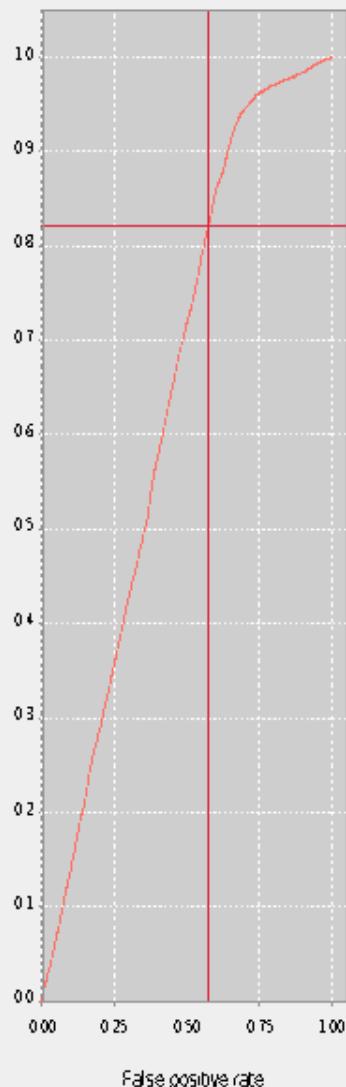
Histogram



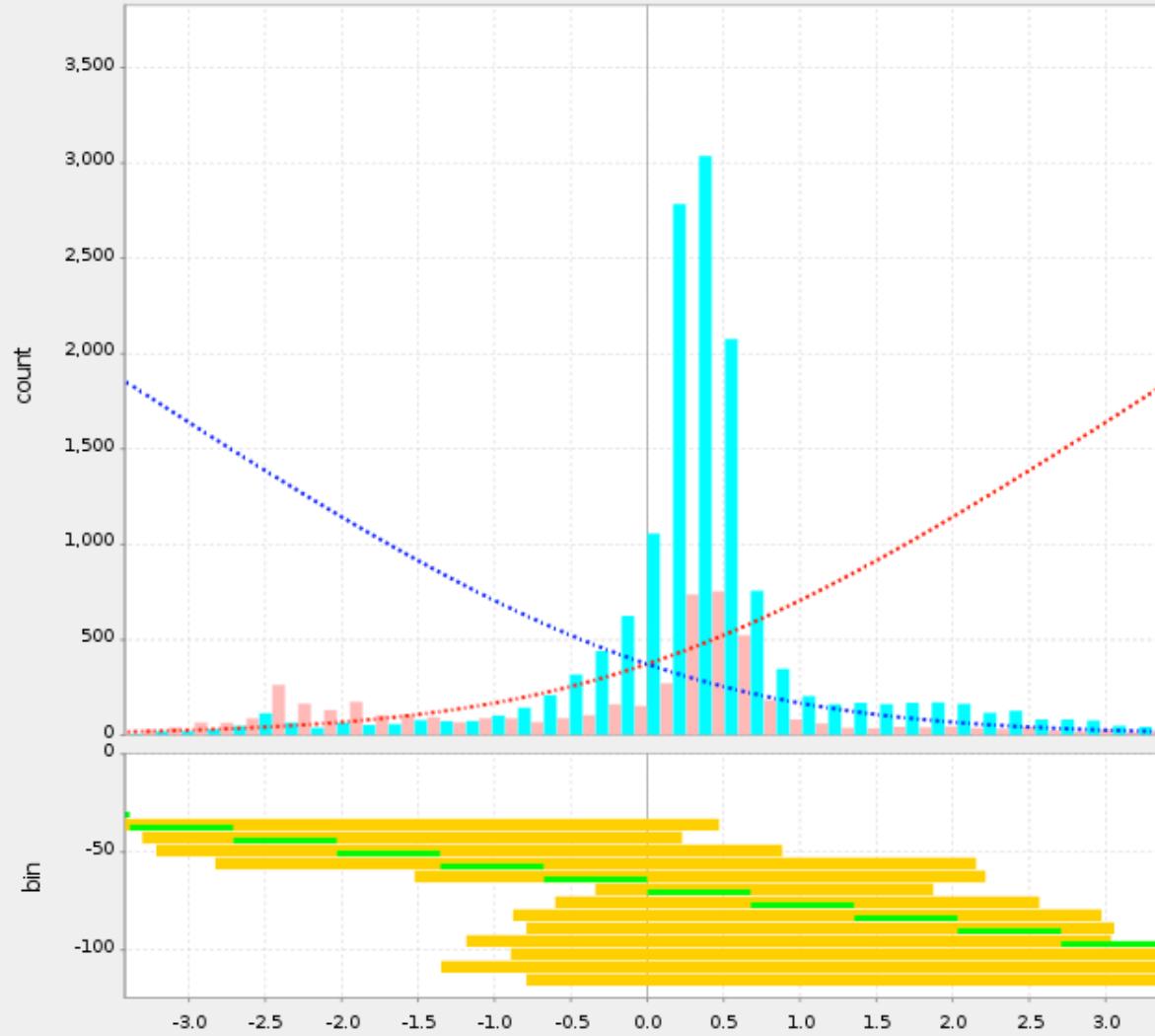
- Iteration 0
- Iteration 1
- Iteration 2
- Iteration 3
- Iteration 4
- Iteration 5
- Iteration 6
- Iteration 7
- Iteration 8
- Iteration 9
- Iteration 10
- Iteration 20
- Iteration 30
- Iteration 40
- Iteration 50
- Iteration 60
- Iteration 70
- Iteration 80
- Iteration 90
- Iteration 100
- Iteration 200

ROC

True positive rate



Histogram



Iteration 3

Iteration 4

Iteration 5

Iteration 6

Iteration 7

Iteration 8

Iteration 9

Iteration 10

Iteration 20

Iteration 30

Iteration 40

Iteration 50

Iteration 60

Iteration 70

Iteration 80

Iteration 90

Iteration 100

Iteration 200

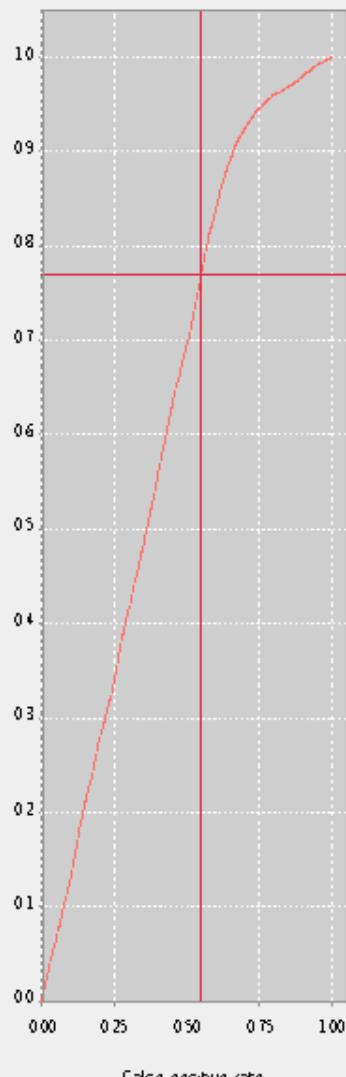
Iteration 300

Iteration 400

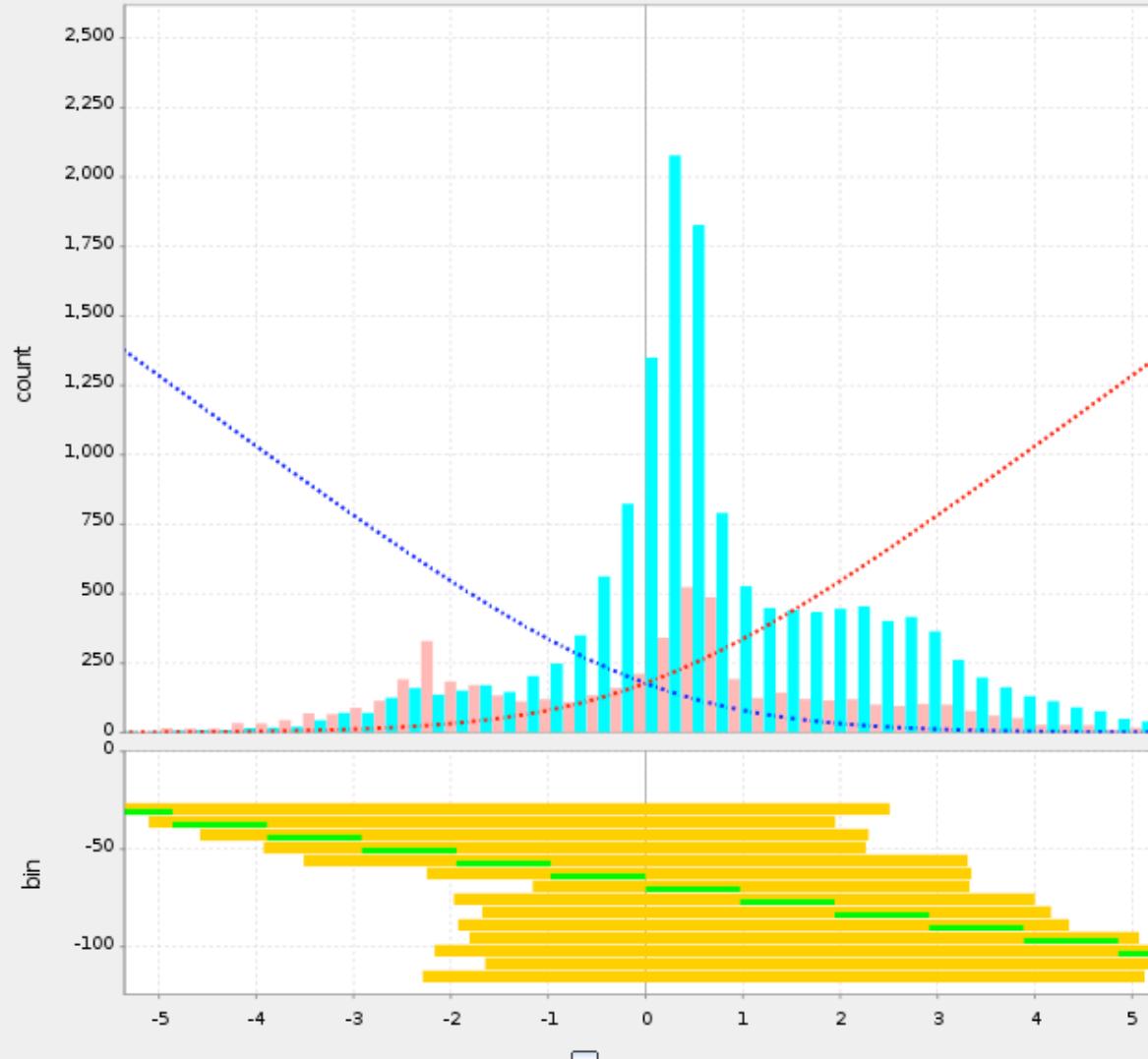
Iteration 500

ROC

True positive rate



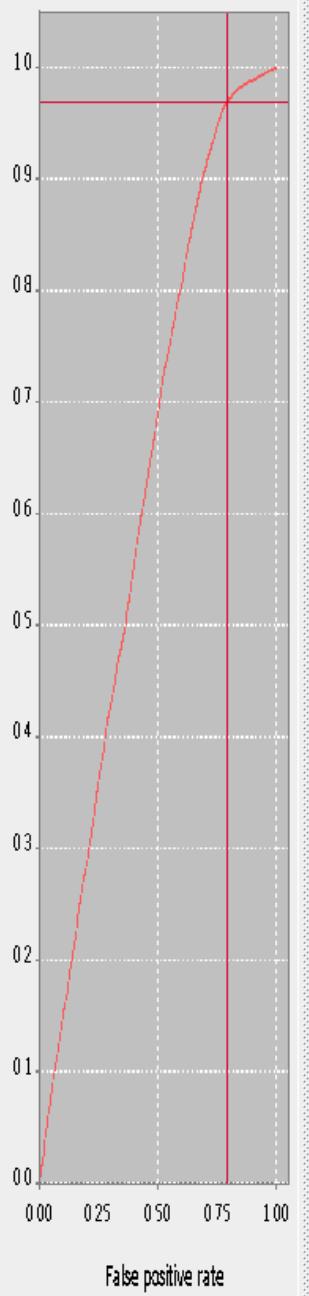
Histogram



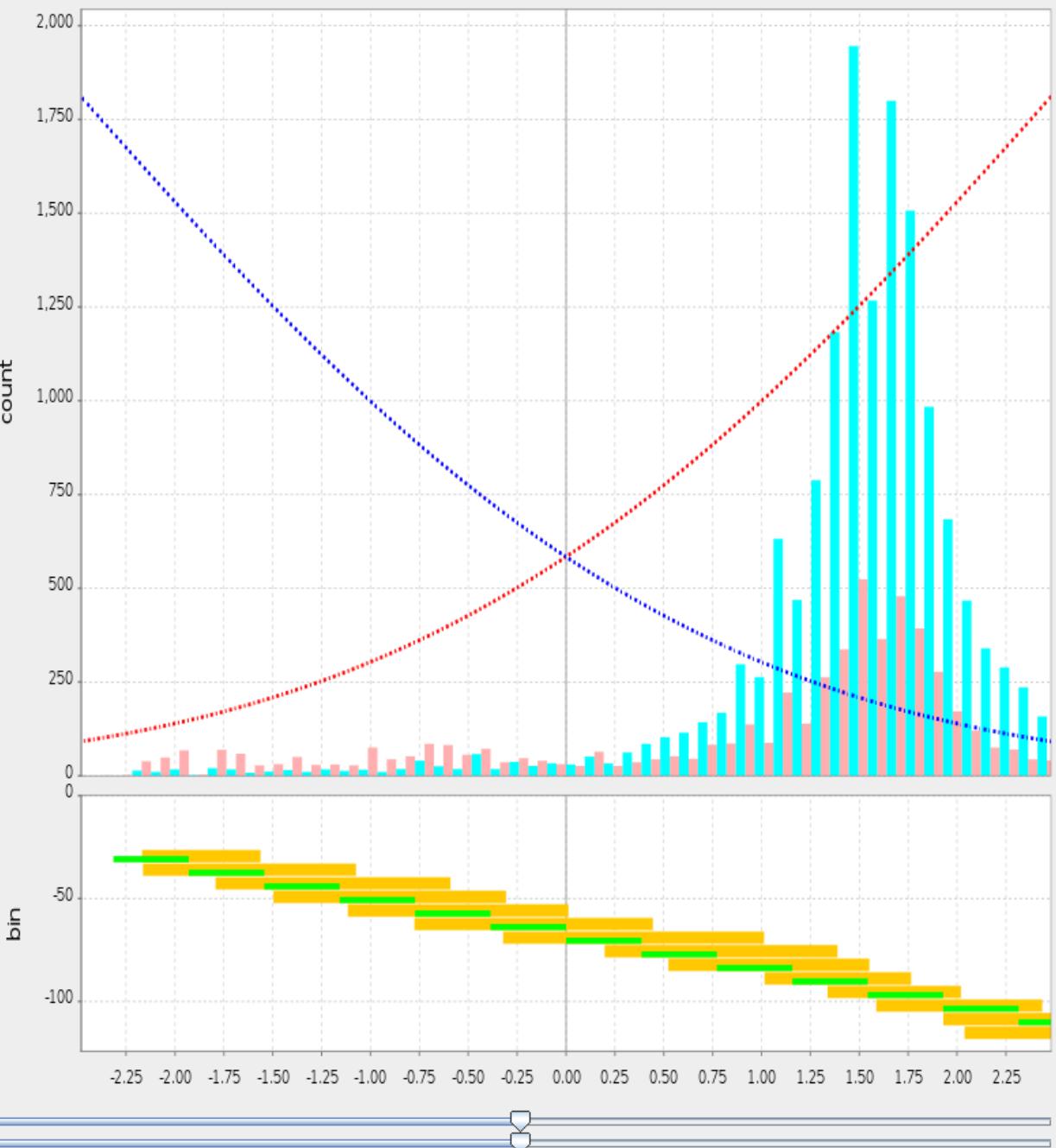
- Iteration 9
- Iteration 10
- Iteration 20
- Iteration 30
- Iteration 40
- Iteration 50
- Iteration 60
- Iteration 70
- Iteration 80
- Iteration 90
- Iteration 100
- Iteration 200
- Iteration 300
- Iteration 400
- Iteration 500
- Iteration 600
- Iteration 700
- Iteration 800
- Iteration 900
- Iteration 1000
- Iteration 2000**

Robustboost
20% Noise

ROC

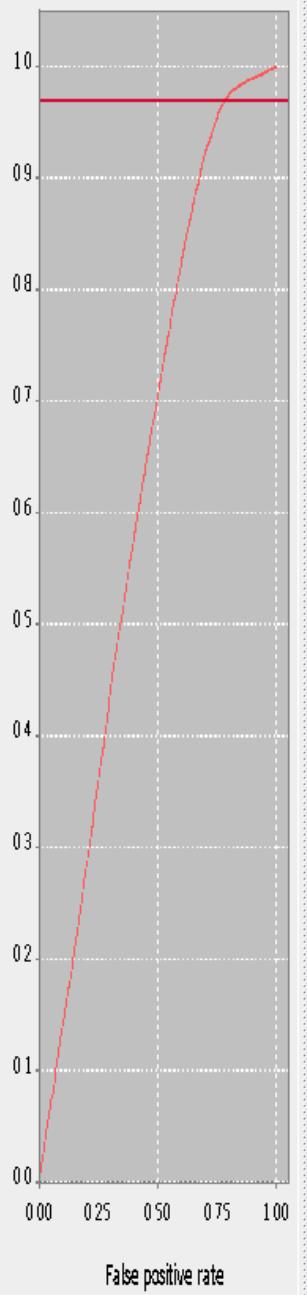


Histogram

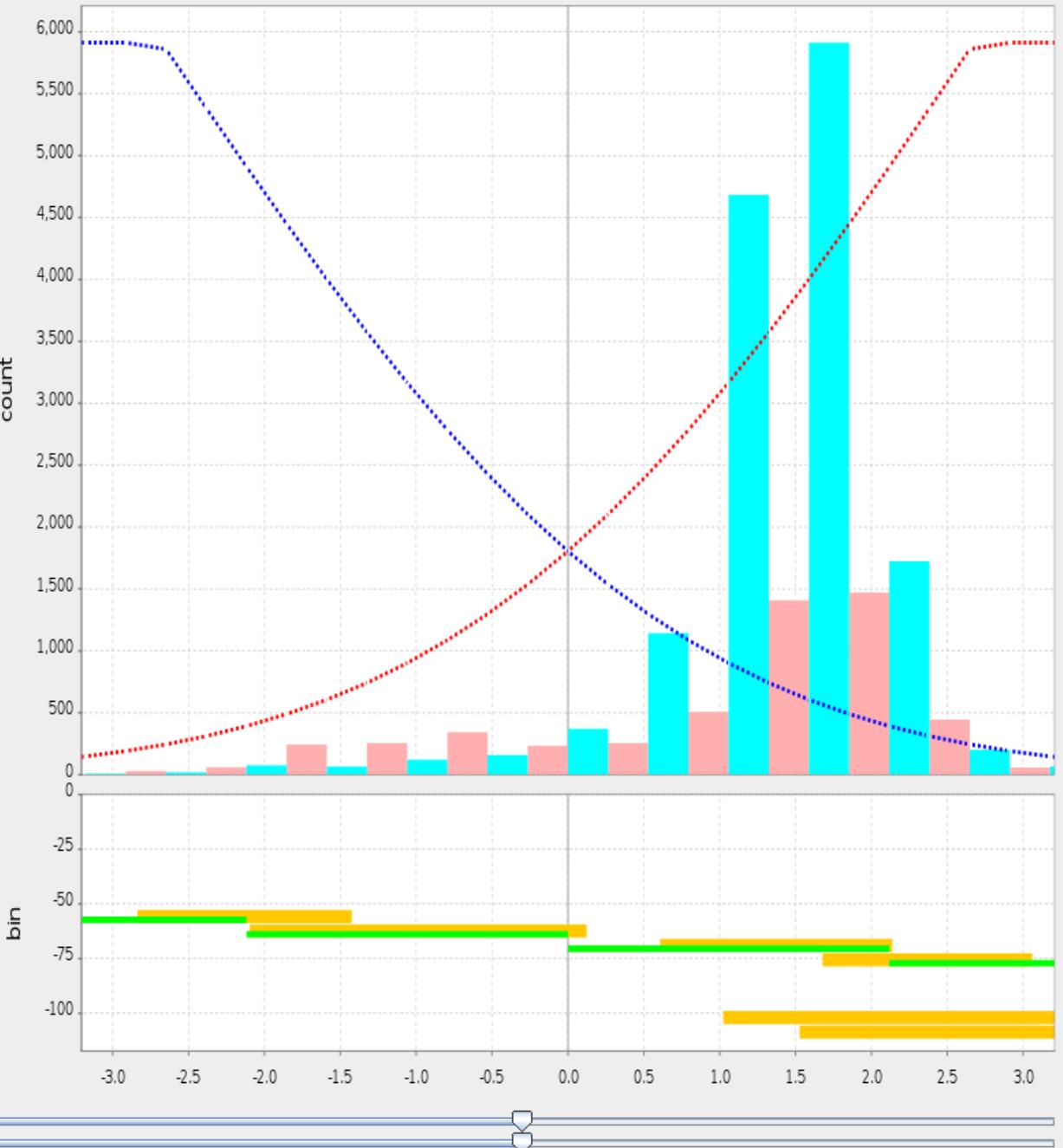


Iteration 0 [T=0.003+-0.000]
Iteration 1 [T=0.003+-0.000]
Iteration 2 [T=0.004+-0.000]
Iteration 3 [T=0.004+-0.000]
Iteration 4 [T=0.004+-0.000]
Iteration 5 [T=0.004+-0.000]
Iteration 6 [T=0.004+-0.000]
Iteration 7 [T=0.004+-0.000]
Iteration 8 [T=0.004+-0.000]
Iteration 9 [T=0.004+-0.000]
Iteration 10 [T=0.004+-0.000]
Iteration 11 [T=0.004+-0.000]
Iteration 12 [T=0.004+-0.000]
Iteration 13 [T=0.004+-0.000]
Iteration 14 [T=0.004+-0.000]
Iteration 15 [T=0.004+-0.000]
Iteration 16 [T=0.004+-0.000]
Iteration 17 [T=0.004+-0.000]
Iteration 18 [T=0.004+-0.000]
Iteration 19 [T=0.004+-0.000]
Iteration 20 [T=0.004+-0.000]
Iteration 21 [T=0.004+-0.000]
Iteration 22 [T=0.004+-0.000]
Iteration 23 [T=0.004+-0.000]
Iteration 24 [T=0.004+-0.000]
Iteration 25 [T=0.004+-0.000]
Iteration 26 [T=0.004+-0.000]
Iteration 27 [T=0.004+-0.000]
Iteration 28 [T=0.004+-0.000]

ROC

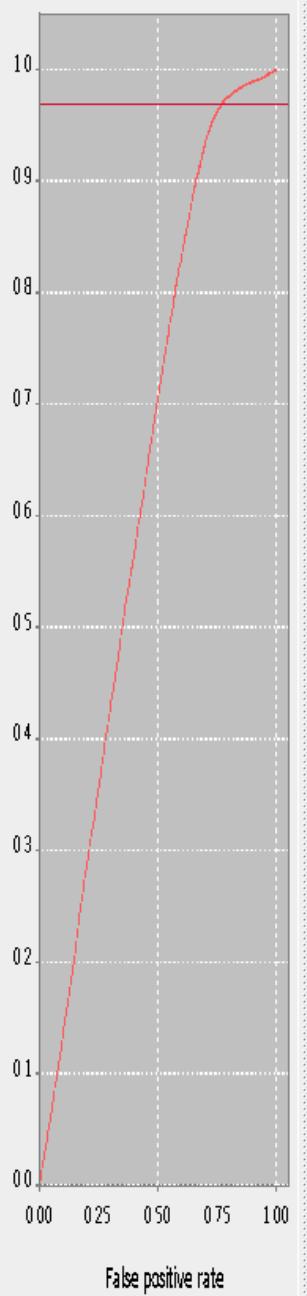


Histogram

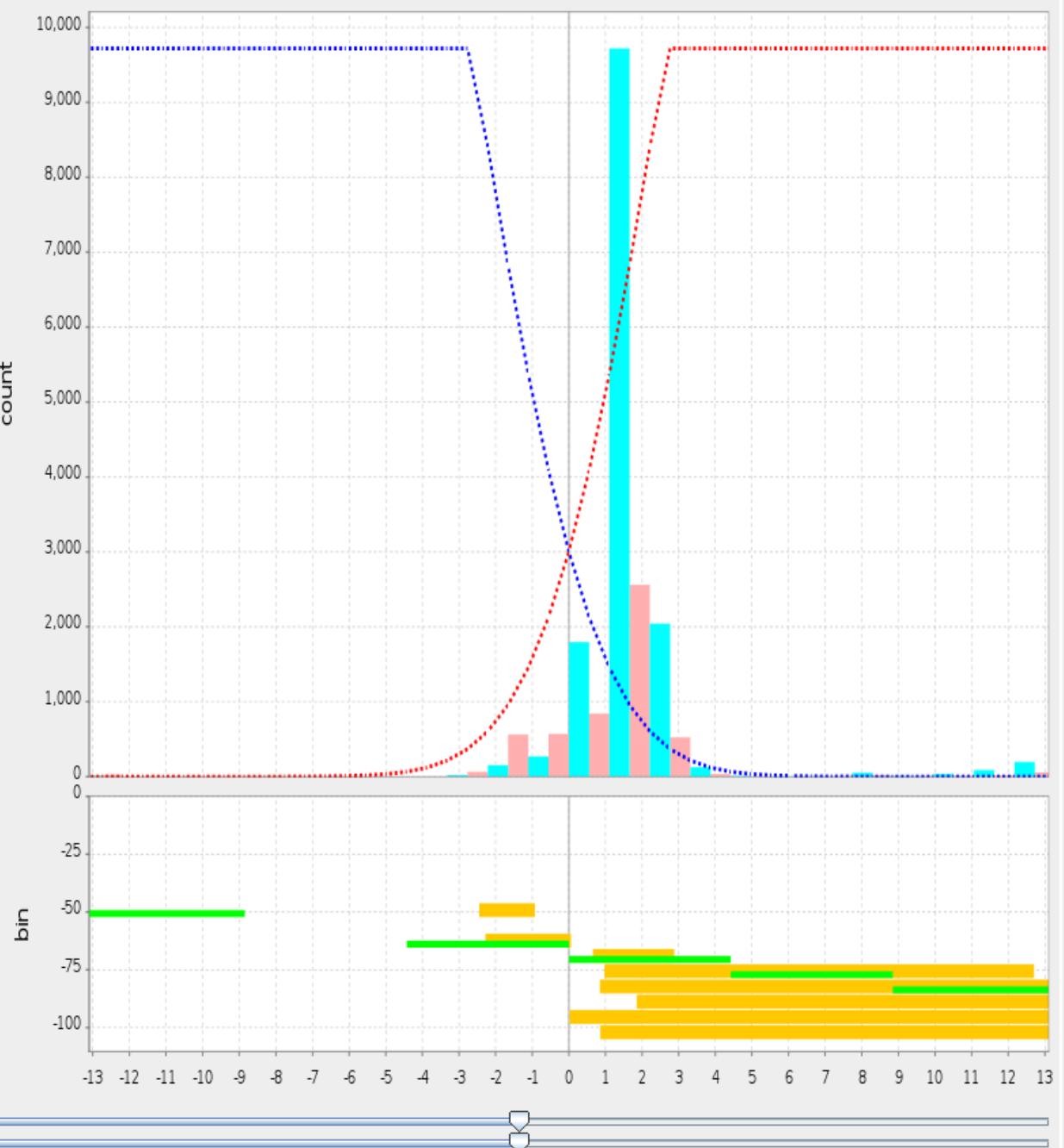


Iteration 21 [T=0.004+-0.000]
Iteration 22 [T=0.004+-0.000]
Iteration 23 [T=0.004+-0.000]
Iteration 24 [T=0.004+-0.000]
Iteration 25 [T=0.004+-0.000]
Iteration 26 [T=0.004+-0.000]
Iteration 27 [T=0.004+-0.000]
Iteration 28 [T=0.004+-0.000]
Iteration 29 [T=0.004+-0.000]
Iteration 30 [T=0.004+-0.000]
Iteration 31 [T=0.004+-0.000]
Iteration 32 [T=0.004+-0.000]
Iteration 33 [T=0.004+-0.000]
Iteration 34 [T=0.004+-0.000]
Iteration 35 [T=0.004+-0.000]
Iteration 36 [T=0.004+-0.000]
Iteration 37 [T=0.004+-0.000]
Iteration 38 [T=0.004+-0.000]
Iteration 39 [T=0.004+-0.000]
Iteration 40 [T=0.004+-0.000]
Iteration 41 [T=0.004+-0.000]
Iteration 42 [T=0.004+-0.000]
Iteration 43 [T=0.004+-0.000]
Iteration 44 [T=0.004+-0.000]
Iteration 45 [T=0.004+-0.000]
Iteration 46 [T=0.004+-0.000]
Iteration 47 [T=0.004+-0.000]
Iteration 48 [T=0.004+-0.000]
Iteration 49 [T=0.004+-0.000]

ROC

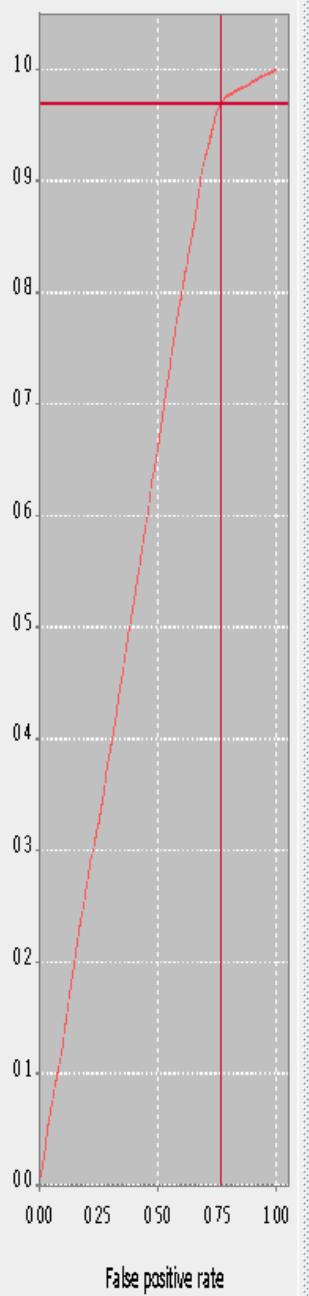


Histogram

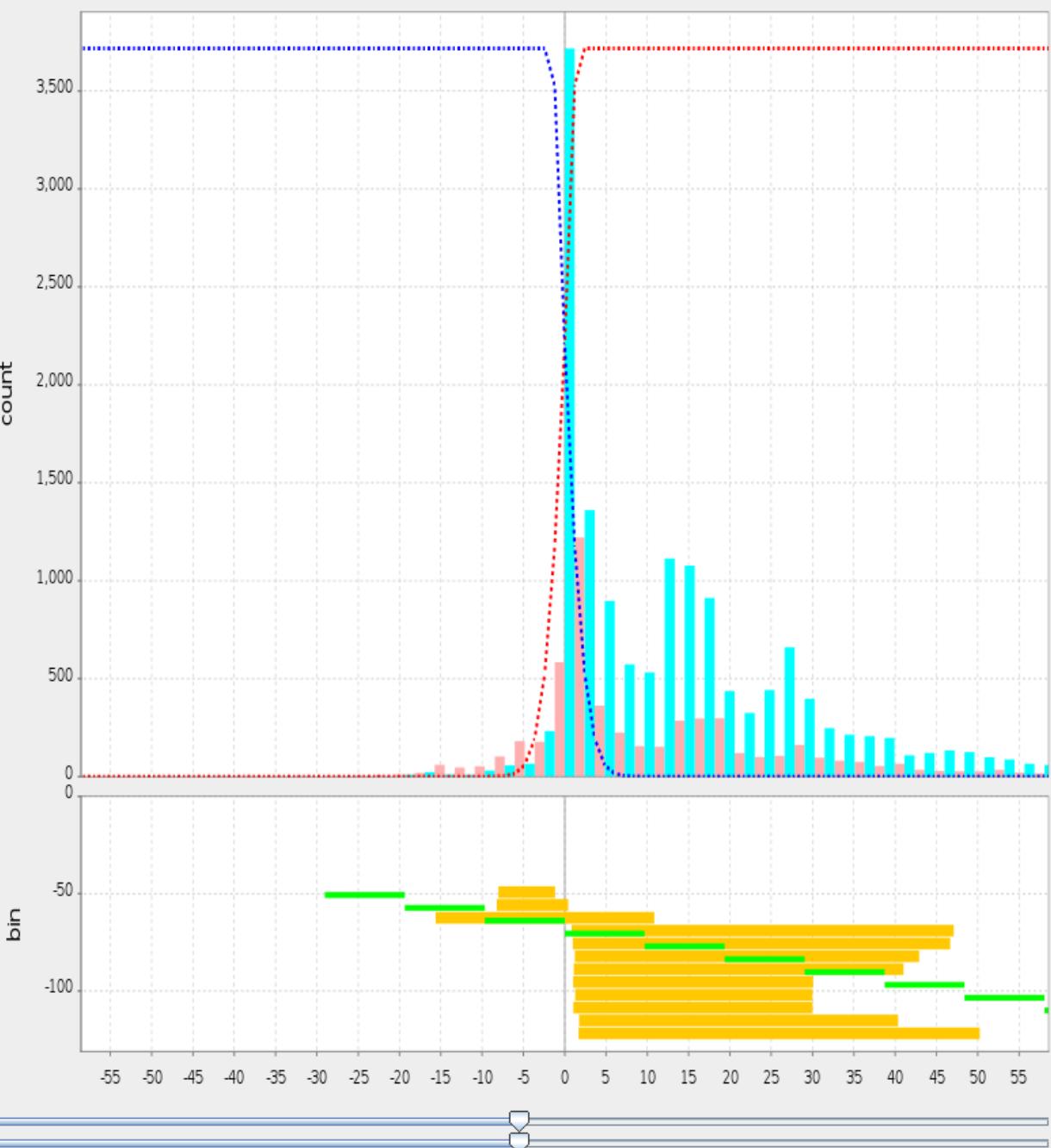


Iteration 72 [T=0.005+-0.000]
Iteration 73 [T=0.005+-0.000]
Iteration 74 [T=0.005+-0.000]
Iteration 75 [T=0.005+-0.000]
Iteration 76 [T=0.005+-0.000]
Iteration 77 [T=0.005+-0.000]
Iteration 78 [T=0.005+-0.000]
Iteration 79 [T=0.005+-0.000]
Iteration 80 [T=0.005+-0.000]
Iteration 81 [T=0.005+-0.000]
Iteration 82 [T=0.005+-0.000]
Iteration 83 [T=0.005+-0.000]
Iteration 84 [T=0.005+-0.000]
Iteration 85 [T=0.005+-0.000]
Iteration 86 [T=0.005+-0.000]
Iteration 87 [T=0.005+-0.000]
Iteration 88 [T=0.005+-0.000]
Iteration 89 [T=0.005+-0.000]
Iteration 90 [T=0.005+-0.000]
Iteration 91 [T=0.005+-0.000]
Iteration 92 [T=0.005+-0.000]
Iteration 93 [T=0.005+-0.000]
Iteration 94 [T=0.005+-0.000]
Iteration 95 [T=0.005+-0.000]
Iteration 96 [T=0.005+-0.000]
Iteration 97 [T=0.005+-0.000]
Iteration 98 [T=0.005+-0.000]
Iteration 99 [T=0.005+-0.000]
Iteration 100 [T=0.005+-0.000]

ROC

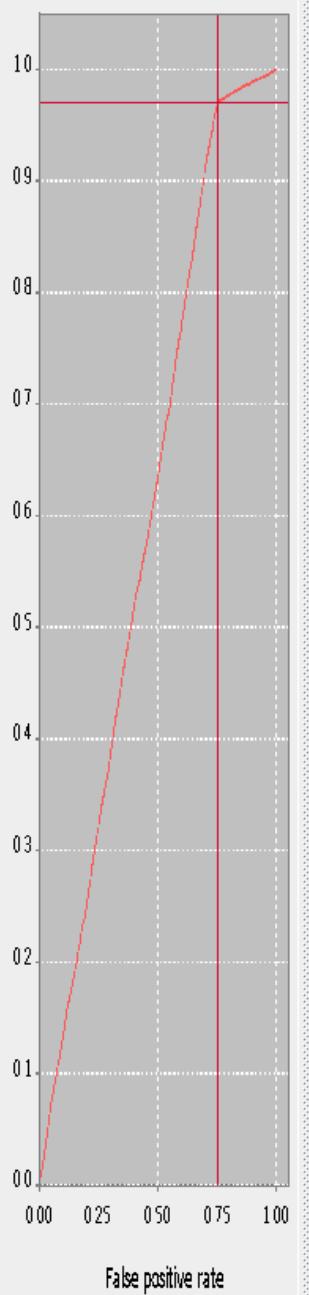


Histogram

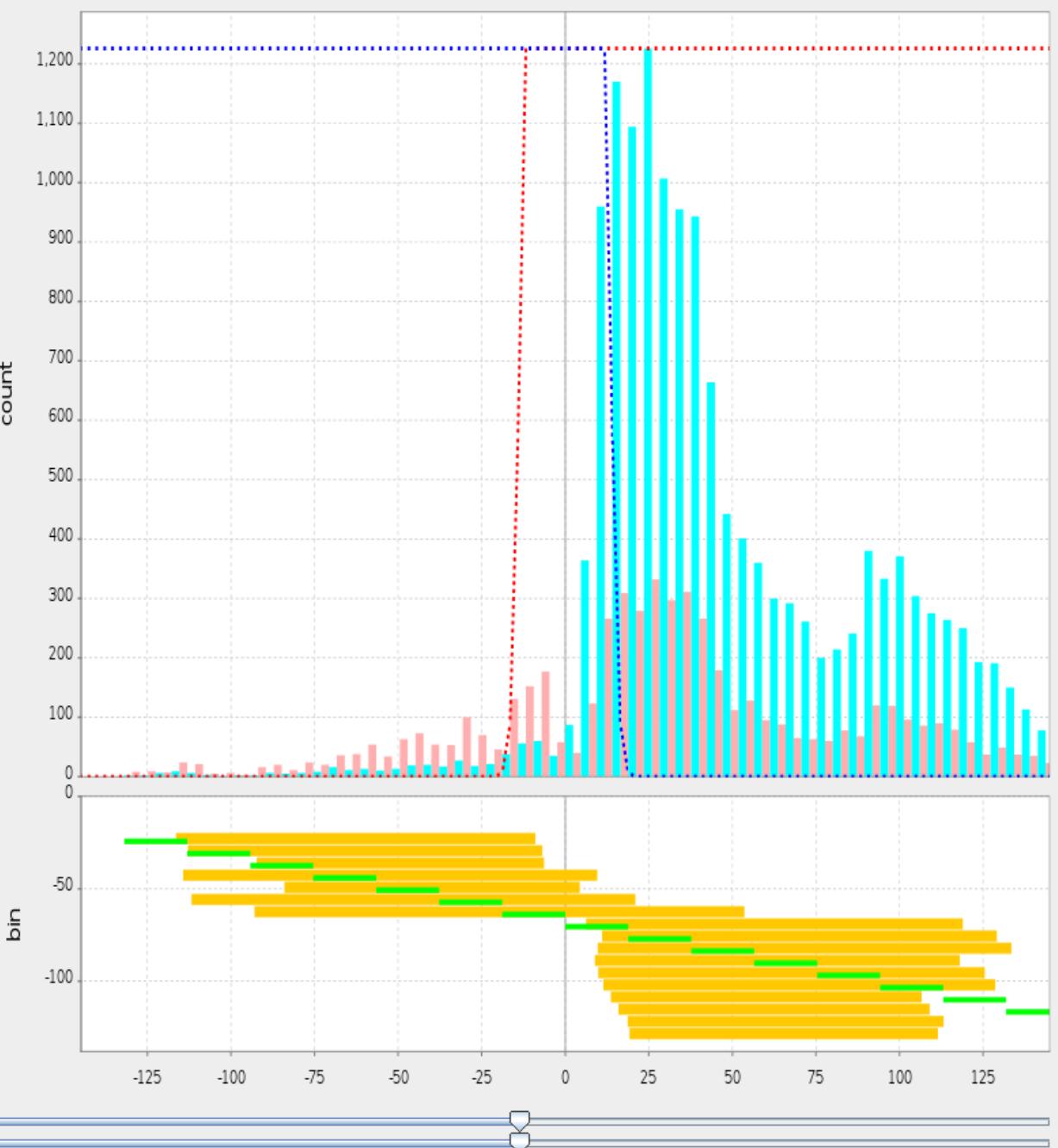


Iteration 172 [T=0.006+-0.002]
Iteration 173 [T=0.006+-0.002]
Iteration 174 [T=0.006+-0.003]
Iteration 175 [T=0.007+-0.003]
Iteration 176 [T=0.007+-0.003]
Iteration 177 [T=0.007+-0.003]
Iteration 178 [T=0.007+-0.004]
Iteration 179 [T=0.007+-0.004]
Iteration 180 [T=0.007+-0.004]
Iteration 181 [T=0.008+-0.004]
Iteration 182 [T=0.008+-0.004]
Iteration 183 [T=0.008+-0.004]
Iteration 184 [T=0.008+-0.005]
Iteration 185 [T=0.008+-0.005]
Iteration 186 [T=0.009+-0.005]
Iteration 187 [T=0.009+-0.005]
Iteration 188 [T=0.009+-0.006]
Iteration 189 [T=0.010+-0.007]
Iteration 190 [T=0.011+-0.008]
Iteration 191 [T=0.011+-0.009]
Iteration 192 [T=0.012+-0.010]
Iteration 193 [T=0.012+-0.011]
Iteration 194 [T=0.012+-0.012]
Iteration 195 [T=0.013+-0.012]
Iteration 196 [T=0.014+-0.014]
Iteration 197 [T=0.015+-0.015]
Iteration 198 [T=0.016+-0.016]
Iteration 199 [T=0.016+-0.017]
Iteration 200 [T=0.017+-0.018]

ROC

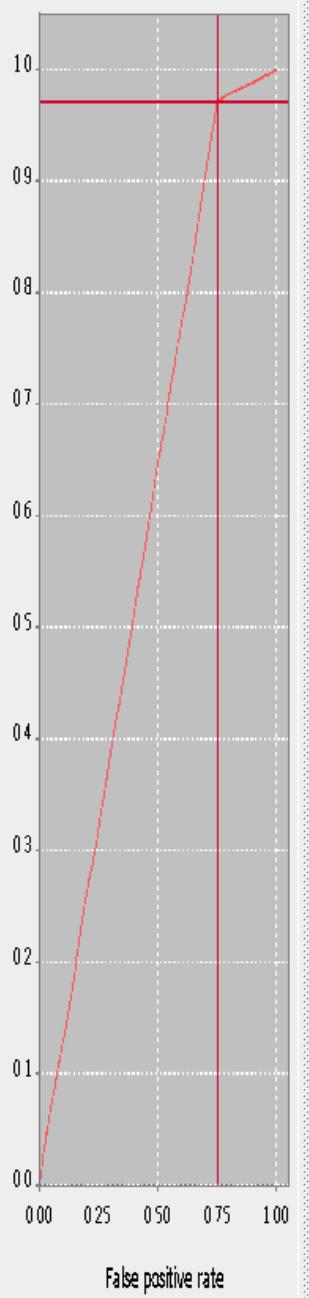


Histogram

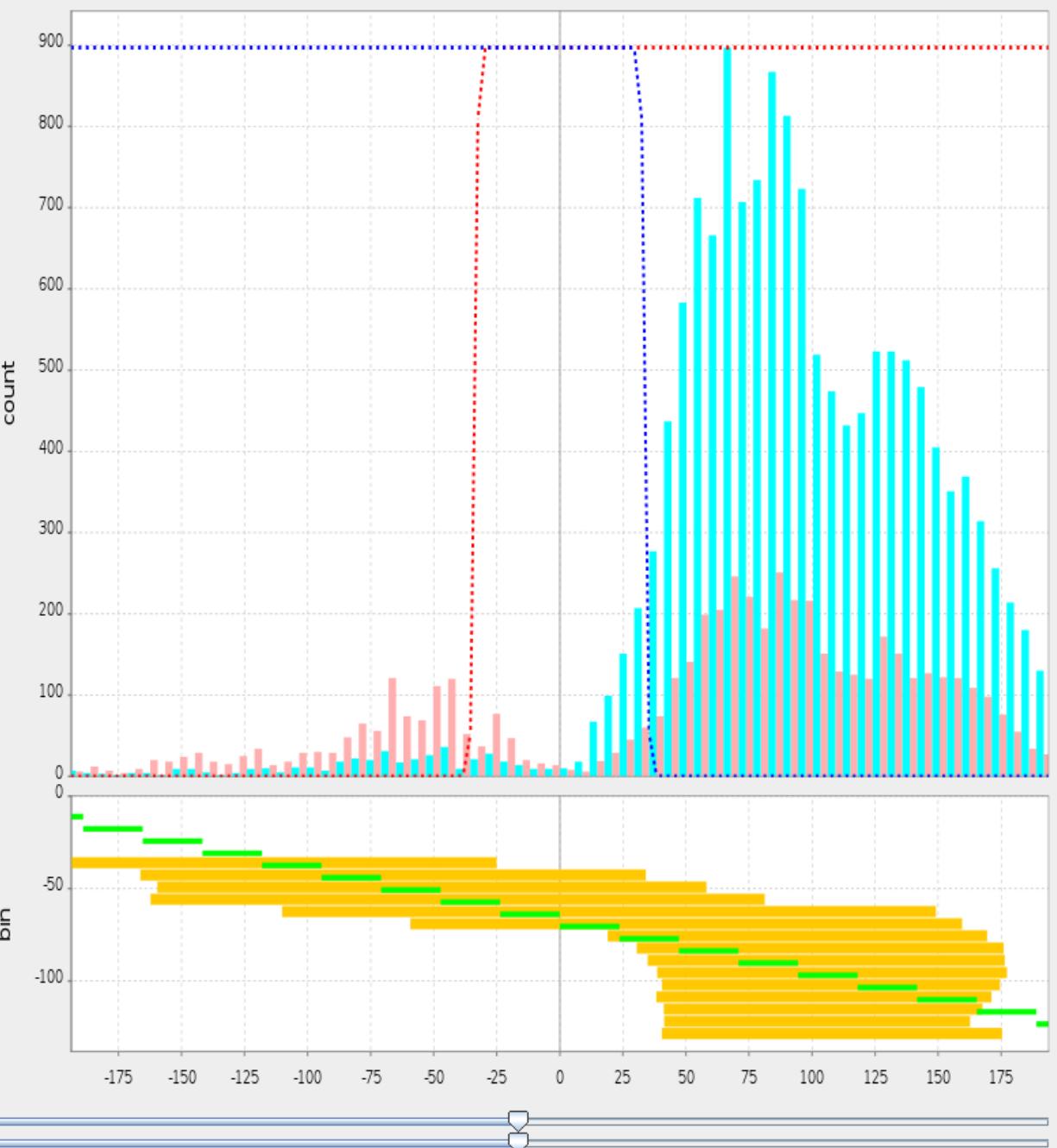


Iteration 222 [T=0.060+-0.082]
Iteration 223 [T=0.062+-0.083]
Iteration 224 [T=0.066+-0.088]
Iteration 225 [T=0.067+-0.090]
Iteration 226 [T=0.070+-0.092]
Iteration 227 [T=0.073+-0.096]
Iteration 228 [T=0.078+-0.098]
Iteration 229 [T=0.080+-0.099]
Iteration 230 [T=0.082+-0.100]
Iteration 231 [T=0.086+-0.102]
Iteration 232 [T=0.089+-0.103]
Iteration 233 [T=0.092+-0.105]
Iteration 234 [T=0.096+-0.106]
Iteration 235 [T=0.098+-0.107]
Iteration 236 [T=0.100+-0.108]
Iteration 237 [T=0.103+-0.109]
Iteration 238 [T=0.106+-0.112]
Iteration 239 [T=0.109+-0.112]
Iteration 240 [T=0.114+-0.113]
Iteration 241 [T=0.117+-0.115]
Iteration 242 [T=0.124+-0.117]
Iteration 243 [T=0.128+-0.119]
Iteration 244 [T=0.132+-0.123]
Iteration 245 [T=0.139+-0.126]
Iteration 246 [T=0.143+-0.128]
Iteration 247 [T=0.147+-0.130]
Iteration 248 [T=0.152+-0.131]
Iteration 249 [T=0.157+-0.134]
Iteration 250 [T=0.160+-0.135]

ROC



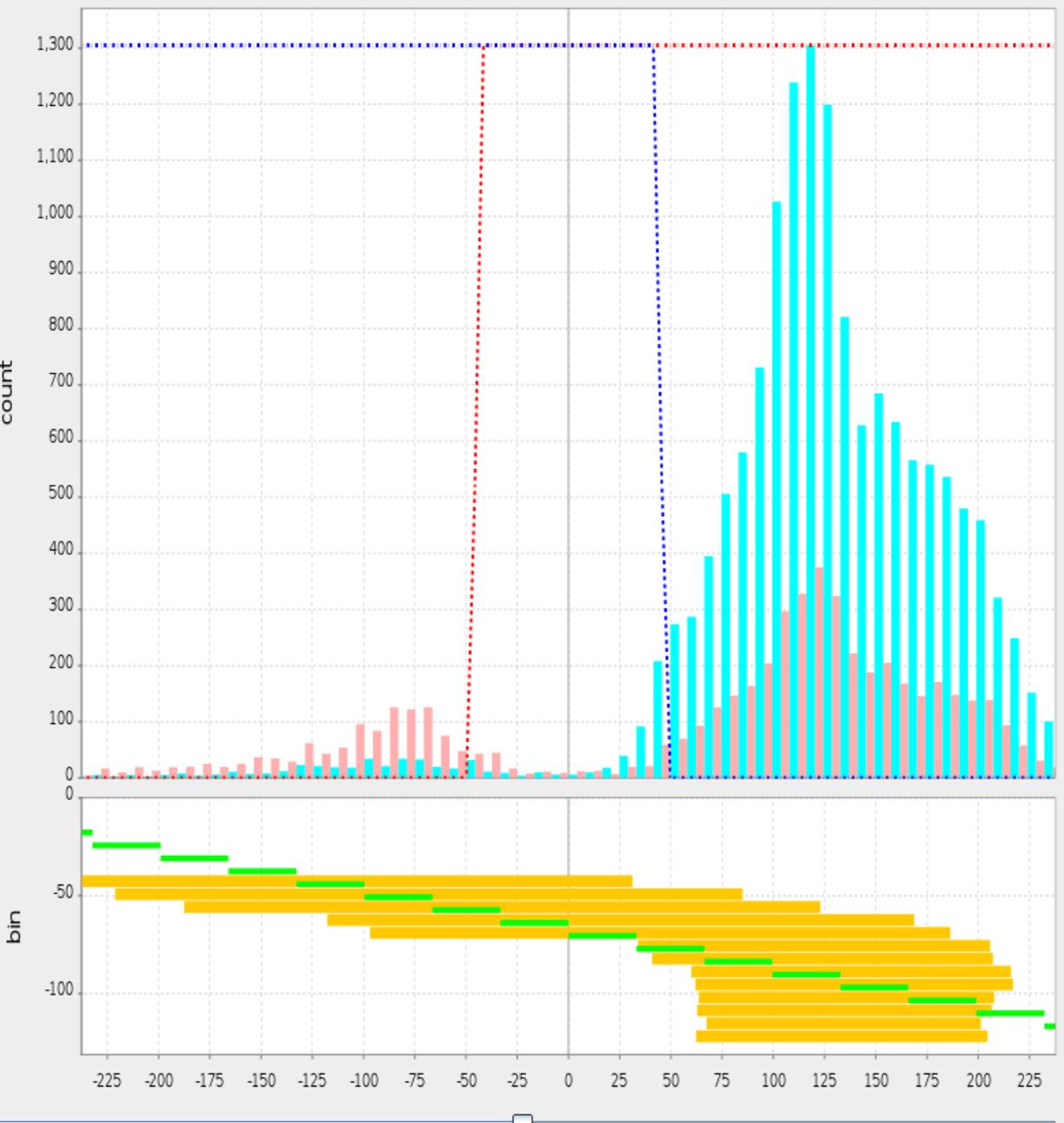
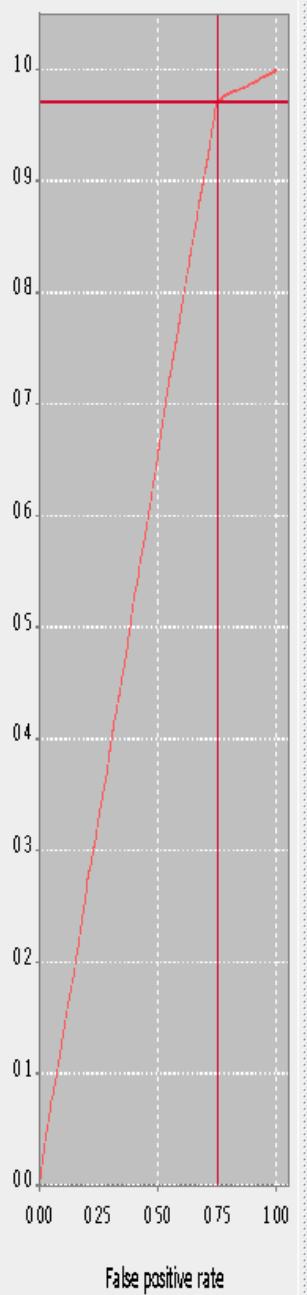
Histogram



Iteration 272 [T=0.272+-0.159]
Iteration 273 [T=0.275+-0.157]
Iteration 274 [T=0.282+-0.161]
Iteration 275 [T=0.286+-0.161]
Iteration 276 [T=0.291+-0.162]
Iteration 277 [T=0.294+-0.163]
Iteration 278 [T=0.300+-0.163]
Iteration 279 [T=0.305+-0.164]
Iteration 280 [T=0.311+-0.165]
Iteration 281 [T=0.318+-0.165]
Iteration 282 [T=0.324+-0.165]
Iteration 283 [T=0.328+-0.166]
Iteration 284 [T=0.333+-0.166]
Iteration 285 [T=0.340+-0.164]
Iteration 286 [T=0.344+-0.165]
Iteration 287 [T=0.347+-0.166]
Iteration 288 [T=0.353+-0.163]
Iteration 289 [T=0.359+-0.165]
Iteration 290 [T=0.363+-0.164]
Iteration 291 [T=0.367+-0.164]
Iteration 292 [T=0.371+-0.165]
Iteration 293 [T=0.375+-0.165]
Iteration 294 [T=0.379+-0.164]
Iteration 295 [T=0.382+-0.165]
Iteration 296 [T=0.385+-0.165]
Iteration 297 [T=0.390+-0.165]
Iteration 298 [T=0.396+-0.165]
Iteration 299 [T=0.400+-0.165]
Iteration 300 [T=0.406+-0.164]

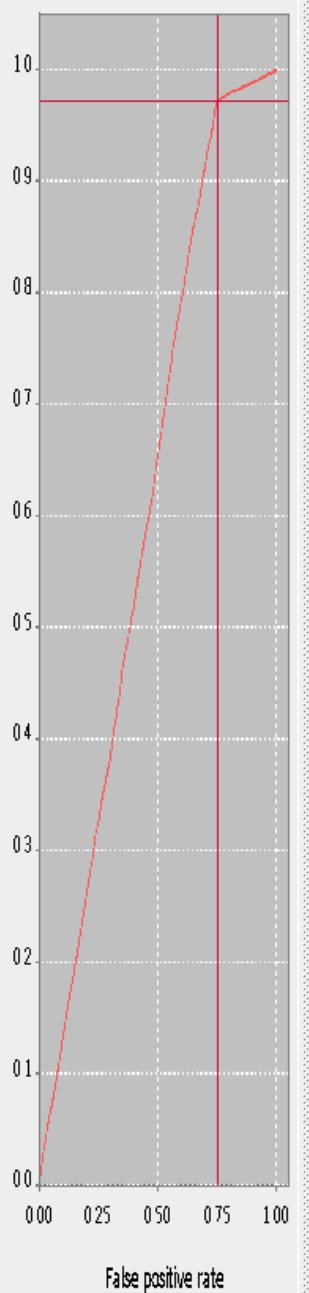
ROC

Histogram

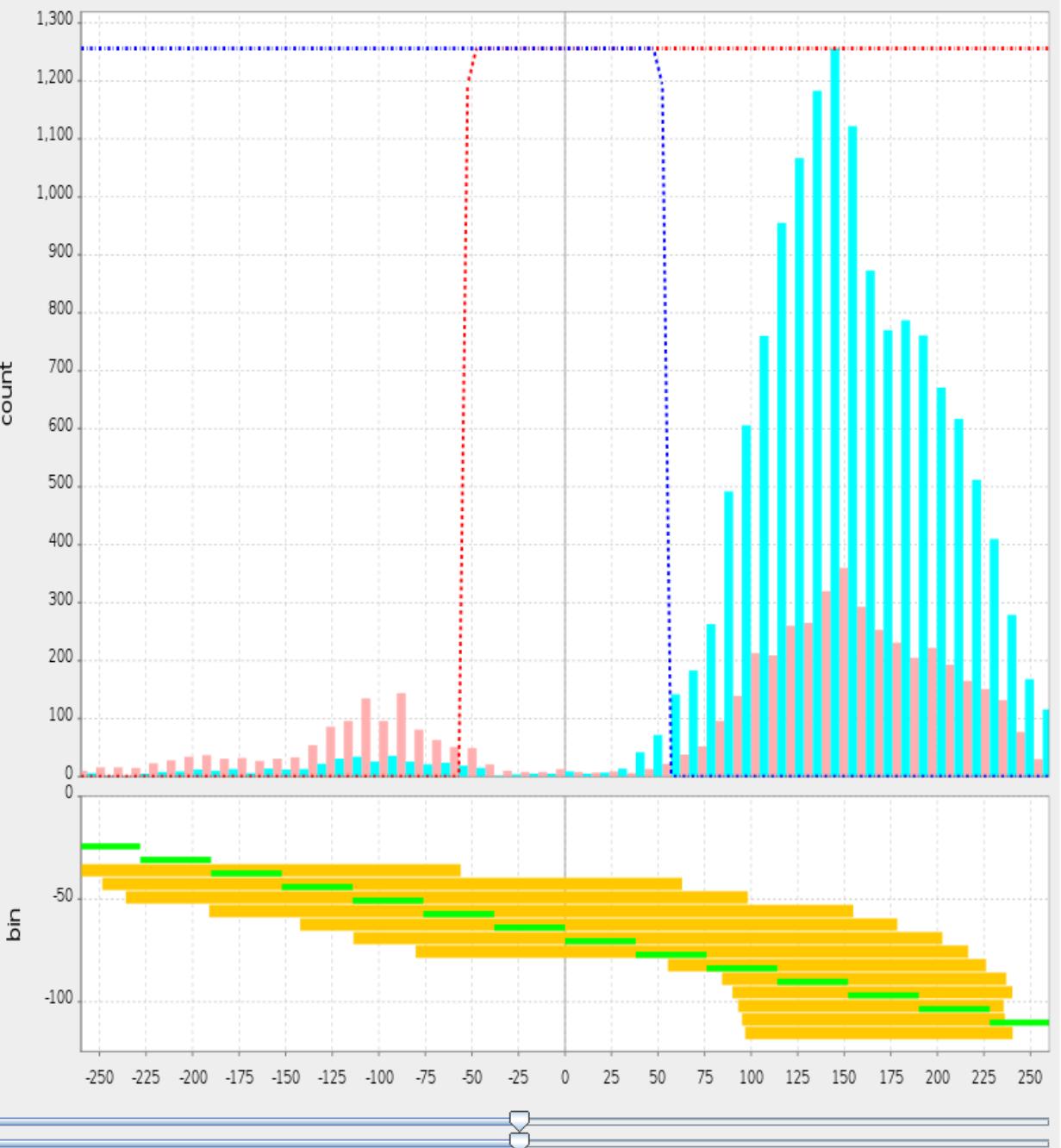


Iteration 322 [T=0.499+-0.166]
 Iteration 323 [T=0.502+-0.166]
 Iteration 324 [T=0.505+-0.166]
 Iteration 325 [T=0.508+-0.166]
 Iteration 326 [T=0.513+-0.168]
 Iteration 327 [T=0.518+-0.169]
 Iteration 328 [T=0.520+-0.169]
 Iteration 329 [T=0.525+-0.169]
 Iteration 330 [T=0.527+-0.169]
 Iteration 331 [T=0.531+-0.169]
 Iteration 332 [T=0.535+-0.172]
 Iteration 333 [T=0.538+-0.171]
 Iteration 334 [T=0.540+-0.172]
 Iteration 335 [T=0.542+-0.173]
 Iteration 336 [T=0.545+-0.173]
 Iteration 337 [T=0.550+-0.173]
 Iteration 338 [T=0.553+-0.173]
 Iteration 339 [T=0.557+-0.174]
 Iteration 340 [T=0.561+-0.175]
 Iteration 341 [T=0.563+-0.175]
 Iteration 342 [T=0.567+-0.177]
 Iteration 343 [T=0.571+-0.178]
 Iteration 344 [T=0.575+-0.179]
 Iteration 345 [T=0.579+-0.180]
 Iteration 346 [T=0.582+-0.181]
 Iteration 347 [T=0.586+-0.181]
 Iteration 348 [T=0.589+-0.181]
 Iteration 349 [T=0.592+-0.181]
 Iteration 350 [T=0.596+-0.182]

ROC



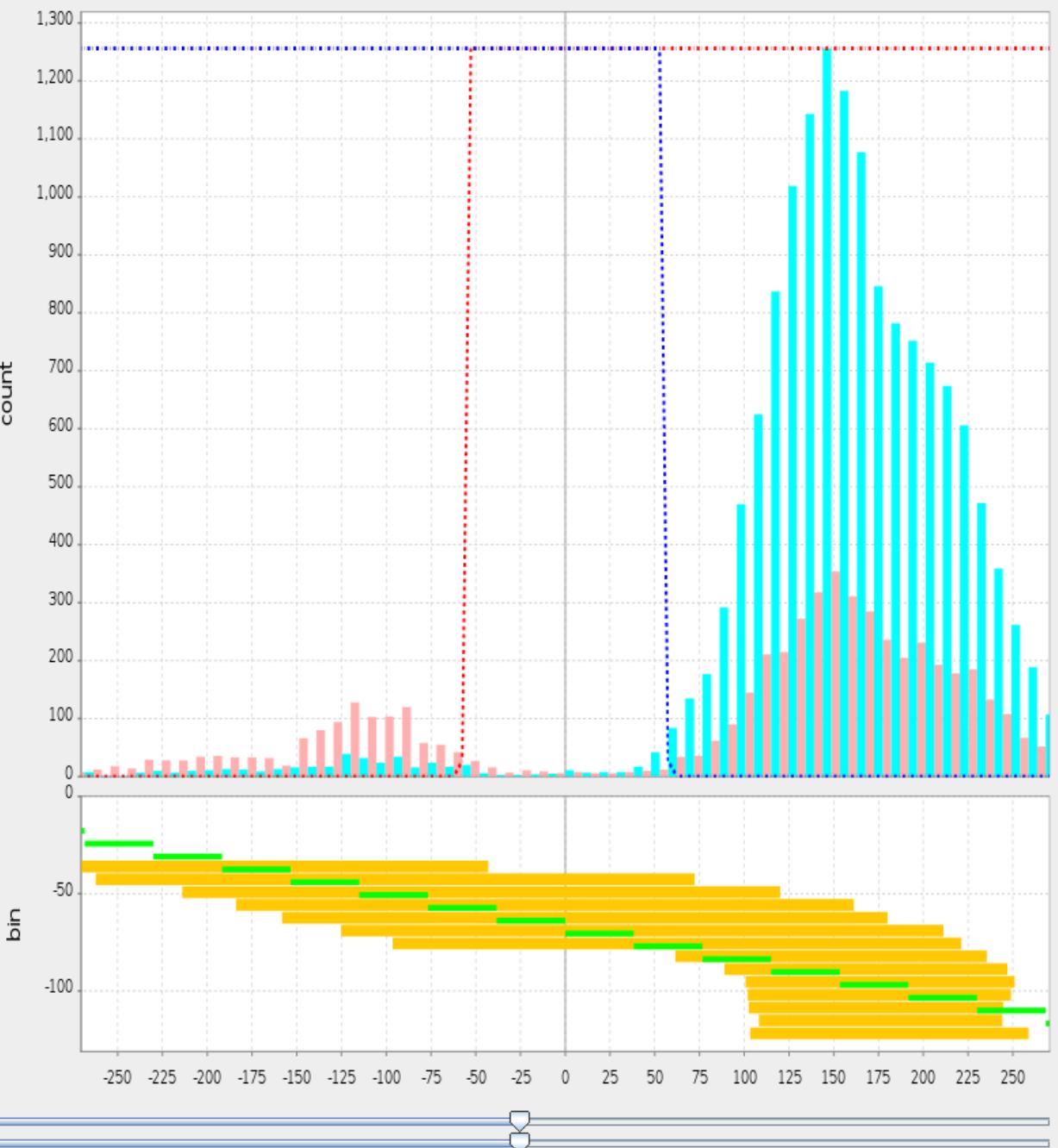
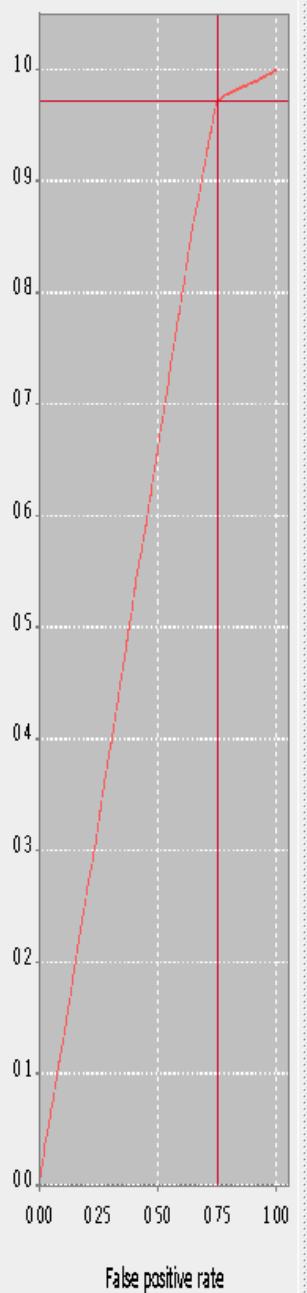
Histogram



Iteration 372 [T=0.670+-0.185]
Iteration 373 [T=0.672+-0.185]
Iteration 374 [T=0.674+-0.185]
Iteration 375 [T=0.678+-0.184]
Iteration 376 [T=0.680+-0.184]
Iteration 377 [T=0.682+-0.184]
Iteration 378 [T=0.684+-0.184]
Iteration 379 [T=0.687+-0.183]
Iteration 380 [T=0.689+-0.184]
Iteration 381 [T=0.692+-0.184]
Iteration 382 [T=0.694+-0.184]
Iteration 383 [T=0.696+-0.184]
Iteration 384 [T=0.699+-0.185]
Iteration 385 [T=0.701+-0.185]
Iteration 386 [T=0.704+-0.185]
Iteration 387 [T=0.706+-0.185]
Iteration 388 [T=0.710+-0.183]
Iteration 389 [T=0.712+-0.184]
Iteration 390 [T=0.714+-0.184]
Iteration 391 [T=0.717+-0.183]
Iteration 392 [T=0.719+-0.183]
Iteration 393 [T=0.721+-0.184]
Iteration 394 [T=0.723+-0.184]
Iteration 395 [T=0.726+-0.185]
Iteration 396 [T=0.727+-0.185]
Iteration 397 [T=0.730+-0.184]
Iteration 398 [T=0.733+-0.186]
Iteration 399 [T=0.735+-0.186]
Iteration 400 [T=0.736+-0.186]

ROC

Histogram



Iteration 403 [T=0.742+-0.187]
 Iteration 404 [T=0.744+-0.188]
 Iteration 405 [T=0.745+-0.188]
 Iteration 406 [T=0.747+-0.188]
 Iteration 407 [T=0.749+-0.188]
 Iteration 408 [T=0.751+-0.188]
 Iteration 409 [T=0.754+-0.187]
 Iteration 410 [T=0.755+-0.187]
 Iteration 411 [T=0.757+-0.187]
 Iteration 412 [T=0.759+-0.186]
 Iteration 413 [T=0.760+-0.186]
 Iteration 414 [T=0.762+-0.186]
 Iteration 415 [T=0.764+-0.186]
 Iteration 416 [T=0.767+-0.186]
 Iteration 417 [T=0.768+-0.187]
 Iteration 418 [T=0.770+-0.187]
 Iteration 419 [T=0.773+-0.187]
 Iteration 420 [T=0.775+-0.186]
 Iteration 421 [T=0.777+-0.186]
 Iteration 422 [T=0.779+-0.186]
 Iteration 423 [T=0.782+-0.186]
 Iteration 424 [T=0.784+-0.186]
 Iteration 425 [T=0.786+-0.185]
 Iteration 426 [T=0.787+-0.185]
 Iteration 427 [T=0.789+-0.185]
 Iteration 428 [T=0.791+-0.186]
 Iteration 429 [T=0.794+-0.185]
 Iteration 430 [T=0.796+-0.185]
 Iteration 431 [T=0.797+-0.185]

JBoost V2.0

The screenshot shows a web browser window with the title "JBoost" at the top. The address bar displays "http://jboost.sourceforge.net/". The toolbar includes standard buttons for Back, Forward, Reload, Stop, Home, and a "Sage" button. Below the toolbar is a menu bar with links like "Most Visited", "TWiki", "My Page", "Yahoo Mail", "UCSD", "banks", "Services", "Other", "vision and Biomed... Share on Facebook", "Facebook", "Google", "jboost", "Search", "Bookmarks", "PageRank", "AutoFill", "Settings", "Dictionary (Google)", "Scholar", "W Wikipedia via Google", and "Button Gal". The main content area features a large "JBoost" logo in a dark blue cloud-like shape. A navigation bar below it contains links for "Home", "Downloads", "Documentation", "FAQ", and "Publications". On the left, a sidebar under the heading "General" lists links for "Home", "Downloads", "About/Contacts", "Getting Started", "Installing", "Tutorial", "Tips", "Documentation", "Examples", "FAQ", "Developers", "Contributing", "TODO List", and "SourceForge Page". The main content area has two sections: "New in Version 2.0!" which lists new features like RobustBoost support, a visualization tool, and support for stopping and restarting the boosting process; and "Overview" which describes JBoost as an easy-to-use tool for boosting classification, mentioning AdaBoost, LogitBoost, BoosTexter, and RobustBoost algorithms, and their implementation using ADTrees for visualizing high-dimensional data. At the bottom, there is a link to download JBoost or read its documentation.

New in Version 2.0!

The following are the new features of JBoost 2.0:

- ▶ RobustBoost support added -- a new boosting algorithm that is resistant to label noise.
- ▶ A new visualization tool -- the [score visualizer](#)
- ▶ Support for stopping and restarting the boosting process while eliminating those examples with small weight from the restarted process.
- ▶ JBoost no longer supports Multi-class problems internally, but now offers a [wrapper script](#).

Overview

JBoost is an easy to use and modify tool for boosting classification. JBoost includes state-of-the-art algorithms and can be used by researchers to quickly implement new boosting algorithms. JBoost also includes a set of easy to use scripts so that machine learning novices can quickly learn and utilize the power of boosting.

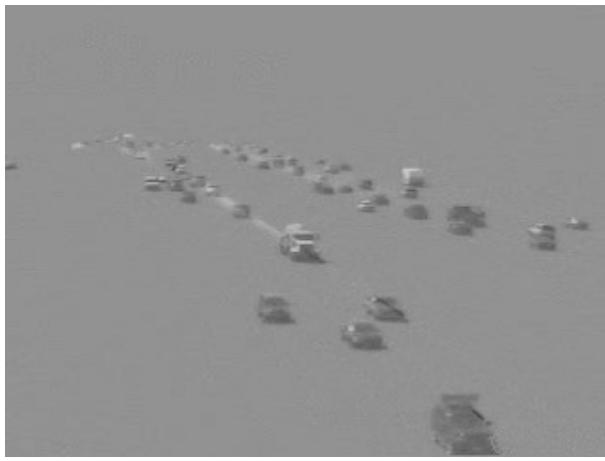
Some of the algorithms currently implemented include AdaBoost, LogitBoost, BoosTexter and RobustBoost. These algorithms are wrapped inside of an implementation of alternating decision trees (ADTrees), which allows for easy visualization of the final classifier, even for high dimensional data. Each of the algorithms comes with a set of options that allows for customization to your dataset.

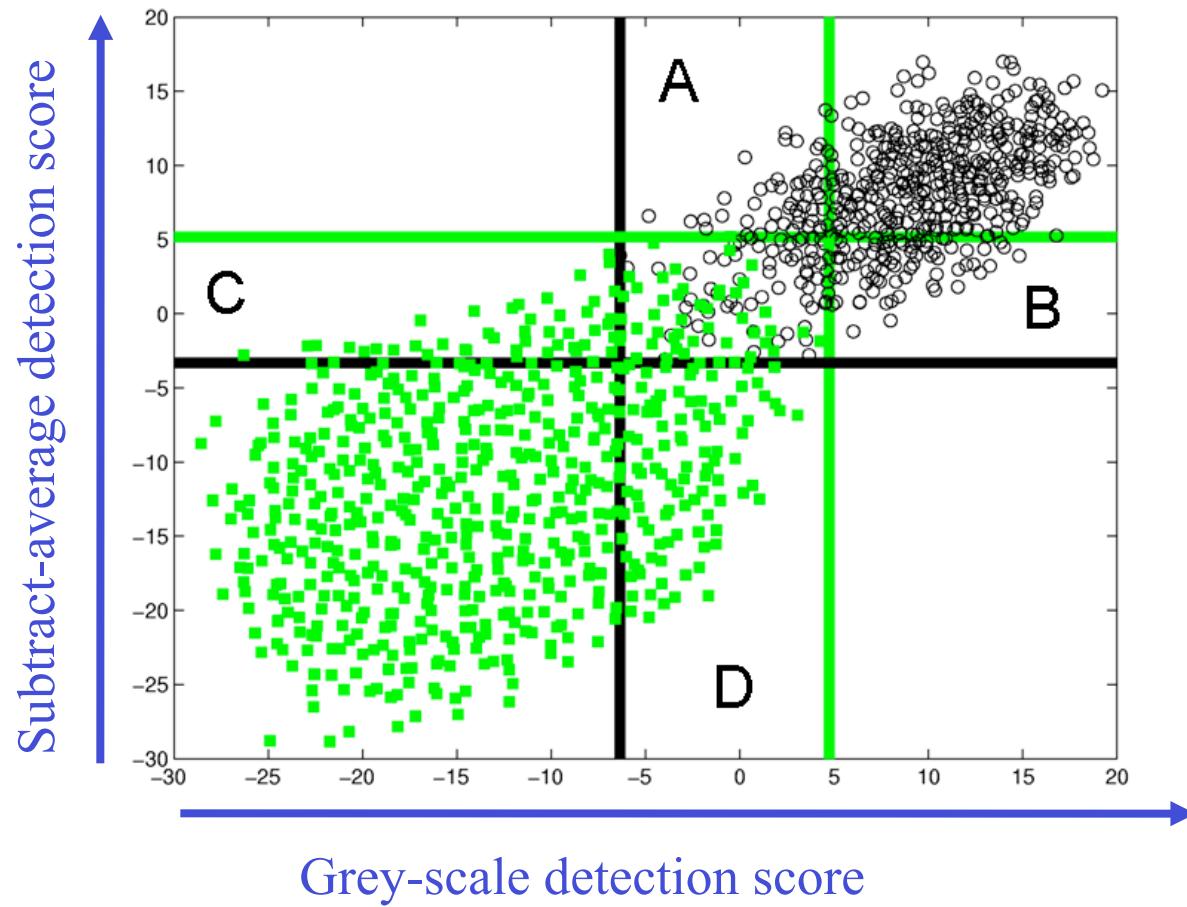
To learn more, [download JBoost](#) or read the [documentation](#).

Parallelizing Boosting

- Map-reduce not good because stateless
- Split features among slaves. Data stays in memory between iterations.
 - Each slave finds his best weak rule
 - Master chooses overall best weak rule.
- When weights get very skewed, flush memory and collect examples filtering them by weight.
 - Concentrates on the hard examples
- Can be combined with Co-training / Active learning.

Co-training





Summary

- **Boosting** is a method for learning an accurate classifiers by combining many weak classifiers.
- Boosting is **resistant to over-fitting**.
- **Margins** quantify prediction confidence.
- **High noise** is a serious problem for learning classifiers-
can't be solved by minimizing convex functions.
- **Robustboost** can solve some high noise problems. Exact characterization still unclear.
- **Jboost** - an implementation of ADTrees and various boosting algorithms in java.
- **Book** on boosting coming this spring.
- *Thank you, questions?*