AdaBoost

Jiri Matas and Jan Šochman

Centre for Machine Perception
Czech Technical University, Prague
http://cmp.felk.cvut.cz

Outline:

- AdaBoost algorithm
  - Why is of interest?
  - How it works?
  - Why it works?
- AdaBoost variants
- AdaBoost with a Totally Corrective Step (TCS)
- Experiments with a Totally Corrective Step

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Introduction

- 1990 – Boost-by-majority algorithm (Freund)
- 1995 – AdaBoost (Freund & Schapire)
- 1997 – Generalized version of AdaBoost (Schapire & Singer)
- 2001 – AdaBoost in Face Detection (Viola & Jones)

Interesting properties:

- AB is a linear classifier with all its desirable properties.
- AB output converges to the logarithm of likelihood ratio.
- AB has good generalization properties.
- AB is a feature selector with a principled strategy (minimisation of upper bound on empirical error).
- AB close to sequential decision making (it produces a sequence of gradually more complex classifiers).

What is AdaBoost?

- AdaBoost is an algorithm for constructing a ”strong” classifier as linear combination

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

of “simple” “weak” classifiers $h_t(x)$.

What is AdaBoost?

- AdaBoost is an algorithm for constructing a ”strong” classifier as linear combination

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

  of “simple” “weak” classifiers $h_t(x)$.

Terminology

- $h_t(x)$ ... “weak” or basis classifier, hypothesis, ”feature”
- $H(x) = \textit{sign}(f(x))$ ... ‘’strong” or final classifier/hypothesis

- AdaBoost is an algorithm for constructing a ”strong” classifier as linear combination

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

of “simple” “weak” classifiers $h_t(x)$.

Terminology

- $h_t(x)$ ... “weak” or basis classifier, hypothesis, ”feature”
- $H(x) = \textit{sign}(f(x))$ ... ‘’strong” or final classifier/hypothesis

Comments

- The $h_t(x)$’s can be thought of as features.
- Often (typically) the set $\mathcal{H} = \{h(x)\}$ is infinite.

(Discrete) AdaBoost Algorithm – Singer & Schapire (1997)

Given: $(x_1, y_1), ..., (x_m, y_m)$; $x_i \in \mathcal{X}, y_i \in \{-1, 1\}$

Initialize weights $D_1(i) = 1/m$

For $t = 1, ..., T$:

1. (Call *WeakLearn*), which returns the weak classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ with minimum error w.r.t. distribution D_t;
2. Choose $\alpha_t \in R$,
3. Update

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that D_{t+1} is a distribution

Output the strong classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

(Discrete) AdaBoost Algorithm – Singer & Schapire (1997)

Given: $(x_1, y_1), ..., (x_m, y_m)$; $x_i \in \mathcal{X}, y_i \in \{-1, 1\}$

Initialize weights $D_1(i) = 1/m$

For $t = 1, ..., T$:

1. (Call *WeakLearn*), which returns the weak classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ with minimum error w.r.t. distribution D_t;
2. Choose $\alpha_t \in R$,
3. Update

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that D_{t+1} is a distribution

Output the strong classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Comments

- The computational complexity of selecting h_t is independent of t.
- All information about previously selected “features” is captured in D_t!

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Loop step: Call *WeakLearn*, given distribution D_t;
returns weak classifier $h_t : \mathcal{X} \to \{-1, 1\}$ from $\mathcal{H} = \{h(x)\}$

- Select a weak classifier with the smallest weighted error
$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- Prerequisite: $\epsilon_t < 1/2$ (otherwise stop)
- *WeakLearn* examples:
  - Decision tree builder, perceptron learning rule – $\mathcal{H}$ *infinite*
  - Selecting the best one from given *finite* set $\mathcal{H}$

WeakLearn

Loop step: Call *WeakLearn*, given distribution D_t;
returns weak classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ from $\mathcal{H} = \{h(x)\}$

- Select a weak classifier with the smallest weighted error
  $h_t = \arg \min\limits_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- Prerequisite: $\epsilon_t < 1/2$ (otherwise stop)
- *WeakLearn* examples:
  - Decision tree builder, perceptron learning rule – $\mathcal{H}$ *infinite*
  - Selecting the best one from given *finite* set $\mathcal{H}$

Demonstration example

Weak classifier = perceptron

● $\sim N(0,1)$ ● $\sim \frac{1}{r\sqrt{8\pi^3}} e^{-1/2(r-4)^2}$

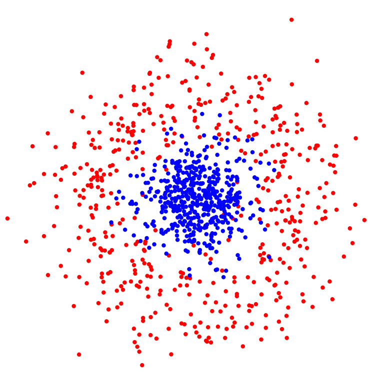
WeakLearn

Loop step: Call *WeakLearn*, given distribution D_t;
returns weak classifier $h_t : \mathcal{X} \to \{-1, 1\}$ from $\mathcal{H} = \{h(x)\}$

- Select a weak classifier with the smallest weighted error
  $h_t = \arg\min\limits_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- Prerequisite: $\epsilon_t < 1/2$ (otherwise stop)
- *WeakLearn* examples:
  - Decision tree builder, perceptron learning rule – $\mathcal{H}$ *infinite*
  - Selecting the best one from given *finite* set $\mathcal{H}$

Demonstration example

Training set



Weak classifier = perceptron

● $\sim N(0, 1)$ ● $\sim \frac{1}{r\sqrt{8\pi^3}} e^{-1/2(r-4)^2}$

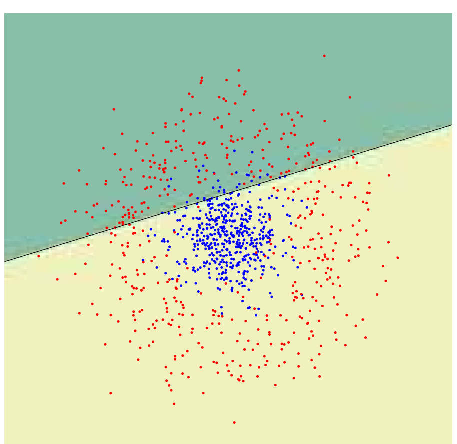
WeakLearn

Loop step: Call *WeakLearn*, given distribution D_t;
returns weak classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ from $\mathcal{H} = \{h(x)\}$

- Select a weak classifier with the smallest weighted error
  $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- Prerequisite: $\epsilon_t < 1/2$ (otherwise stop)
- *WeakLearn* examples:
  - Decision tree builder, perceptron learning rule – $\mathcal{H}$ *infinite*
  - Selecting the best one from given *finite* set $\mathcal{H}$

Demonstration example

Training set



Weak classifier = perceptron

● $\sim N(0, 1)$ ● $\sim \frac{1}{r\sqrt{8\pi^3}} e^{-1/2(r-4)^2}$

AdaBoost as a Minimiser of an Upper Bound on the Empirical Error

- The main objective is to minimize $\varepsilon_{tr} = \frac{1}{m}|\{i : H(x_i) \neq y_i\}|$
- It can be upper bounded by $\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t$

- The main objective is to minimize $\varepsilon_{tr} = \frac{1}{m}|\{i : H(x_i) \neq y_i\}|$
- It can be upper bounded by $\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t$

How to set α_t?

- Select α_t to greedily minimize $Z_t(\alpha)$ in each step
- $Z_t(\alpha)$ is convex differentiable function with one extremum

  $\Rightarrow$ $h_t(x) \in \{-1, 1\}$ then optimal $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
  where $r_t = \sum_{i=1}^{m} D_t(i)h_t(x_i)y_i$
- $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \leq 1$ for optimal α_t

  $\Rightarrow$ Justification of selection of h_t according to ϵ_t

- The main objective is to minimize $\varepsilon_{tr} = \frac{1}{m}|\{i : H(x_i) \neq y_i\}|$
- It can be upper bounded by $\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t$

How to set α_t?

- Select α_t to greedily minimize $Z_t(\alpha)$ in each step
- $Z_t(\alpha)$ is convex differentiable function with one extremum
  - $\Rightarrow$ $h_t(x) \in \{-1, 1\}$ then optimal $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
    where $r_t = \sum_{i=1}^{m} D_t(i) h_t(x_i) y_i$
- $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \leq 1$ for optimal α_t
  - $\Rightarrow$ Justification of selection of h_t according to ϵ_t

Comments

- The process of selecting α_t and $h_t(x)$ can be interpreted as a single optimization step minimising the upper bound on the empirical error. Improvement of the bound is guaranteed, provided that $\epsilon_t < 1/2$.
- The process can be interpreted as a component-wise local optimization (Gauss-Southwell iteration) in the (possibly infinite dimensional!) space of $\bar{\alpha} = (\alpha_1, \alpha_2, \dots)$ starting from. $\bar{\alpha}_0 = (0, 0, \dots)$.

Effect on the training set

Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \frac{exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))}{m \prod_{q=1}^{t} Z_q}$$

$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$

$\}$

$\Rightarrow$ Increase (decrease) weight of wrongly (correctly) classified examples. The weight is the upper bound on the error of a given example!

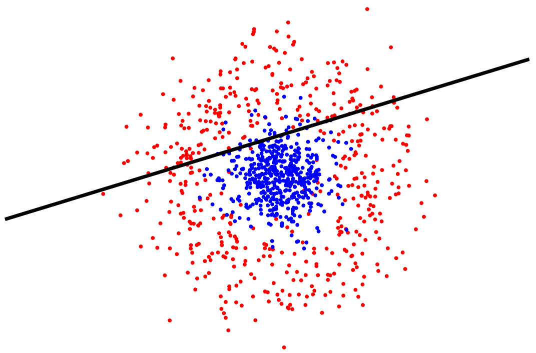
Effect on the training set

Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \frac{exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))}{m \prod_{q=1}^{t} Z_q}$$

$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$

}



⇒ Increase (decrease) weight of wrongly (correctly) classified examples. The weight is the upper bound on the error of a given example!

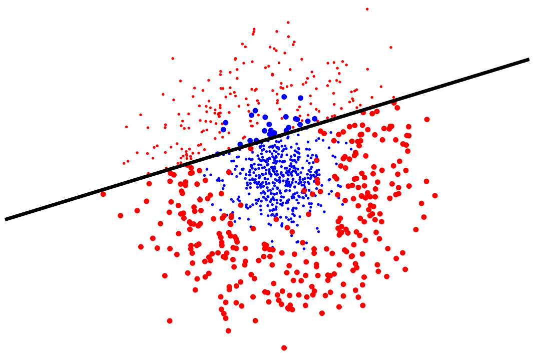
Effect on the training set

Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \frac{exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))}{m \prod_{q=1}^{t} Z_q}$$

$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$

}



$\Rightarrow$ Increase (decrease) weight of wrongly (correctly) classified examples. The weight is the upper bound on the error of a given example!

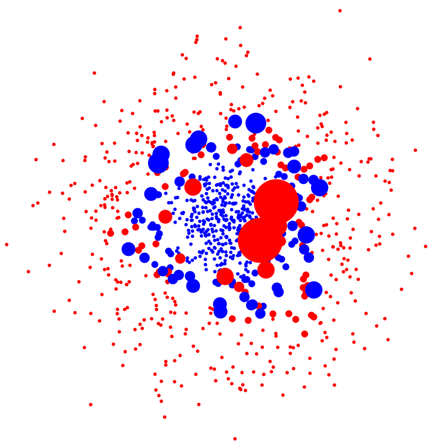
Effect on the training set

Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \frac{exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))}{m \prod_{q=1}^{t} Z_q}$$

$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$

$\}$



$\Rightarrow$ Increase (decrease) weight of wrongly (correctly) classified examples. The weight is the upper bound on the error of a given example!

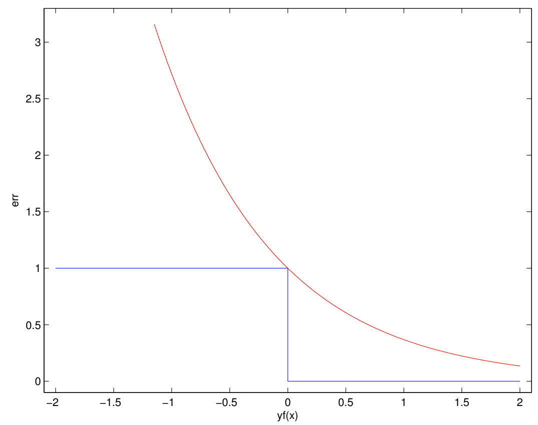
Effect on the training set

Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \frac{exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))}{m \prod_{q=1}^{t} Z_q}$$

$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$

}



$\Rightarrow$ Increase (decrease) weight of wrongly (correctly) classified examples. The weight is the upper bound on the error of a given example!

Effect on the training set

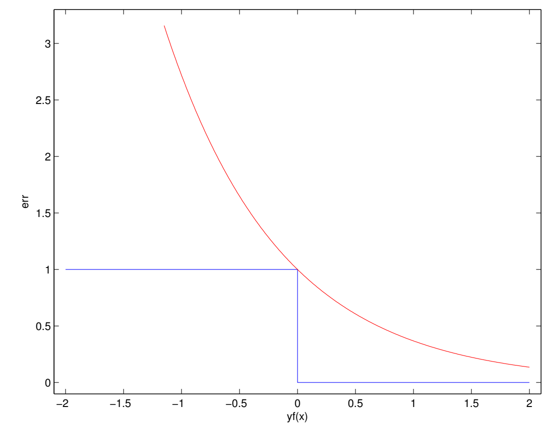
Reweighting formula:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \frac{exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))}{m \prod_{q=1}^{t} Z_q}$$

$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, & y_i = h_t(x_i) \\ > 1, & y_i \neq h_t(x_i) \end{cases}$$

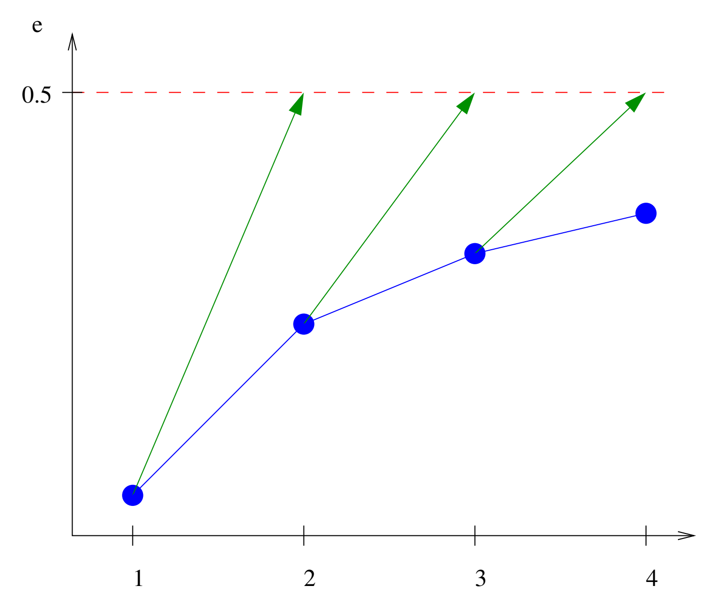
}

$\Rightarrow$ Increase (decrease) weight of wrongly (correctly) classified examples. The weight is the upper bound on the error of a given example!



Effect on h_t

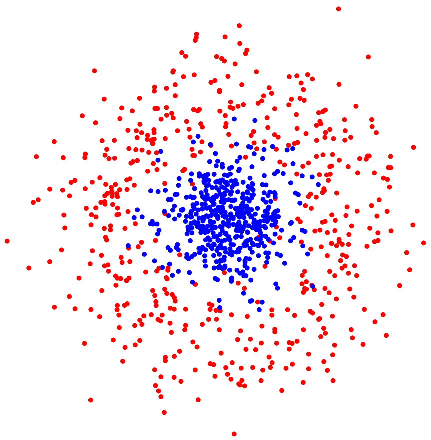
- α_t minimize $Z_t \Rightarrow$
  $$\sum_{i:h_t(x_i)=y_i} D_{t+1}(i) = \sum_{i:h_t(x_i)\neq y_i} D_{t+1}(i)$$
- Error of h_t on D_{t+1} is $1/2$
- Next weak classifier is the most "independent" one



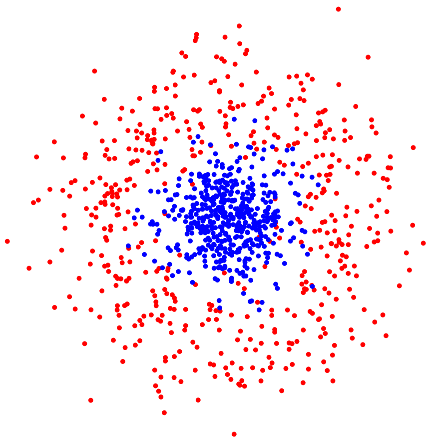
Summary of the Algorithm

Summary of the Algorithm

Initialization...



Initialization...
For $t = 1, ..., T$:

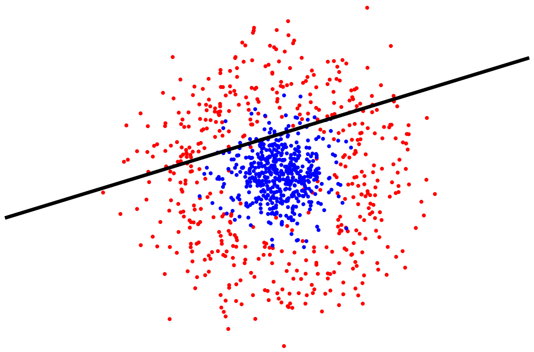


Initialization...

For $t = 1, ..., T$:

- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$

$t = 1$

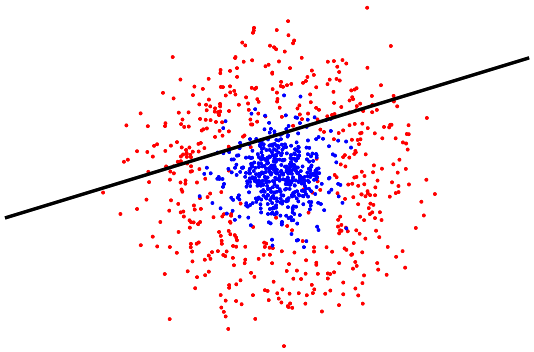


Initialization...

For $t = 1, ..., T$:

- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop

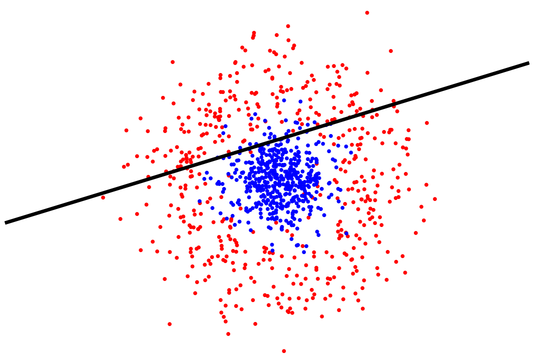
$t = 1$



Initialization...
For $t = 1, ..., T$:

- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$

$t = 1$

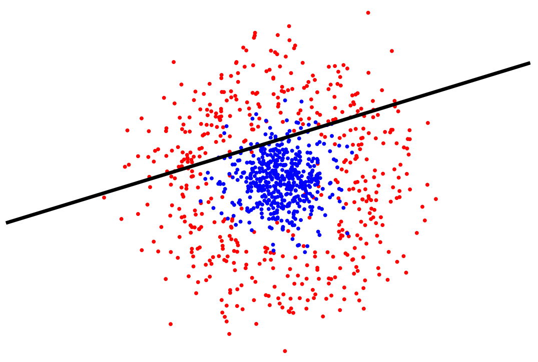


Initialization...
For $t = 1, ..., T$:

- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$t = 1$



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Summary of the Algorithm

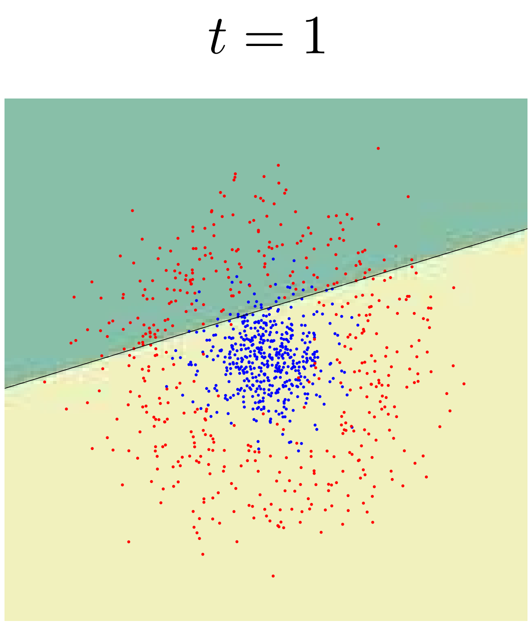
Initialization...
For $t = 1, ..., T$:

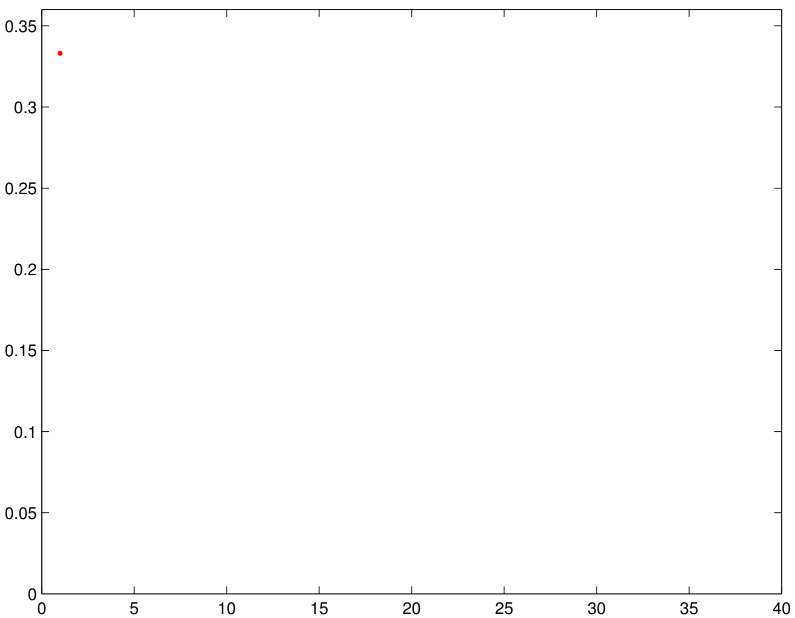
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$





Summary of the Algorithm

Initialization...
For $t = 1, ..., T$:

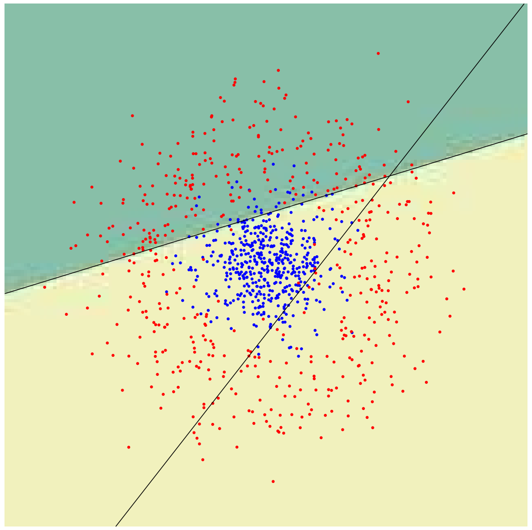
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

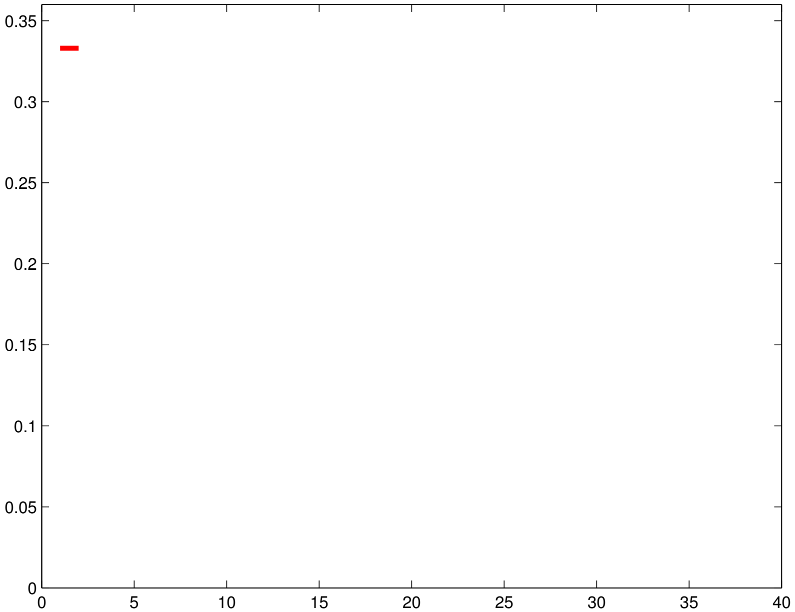
$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$t = 2$





Summary of the Algorithm

Initialization...

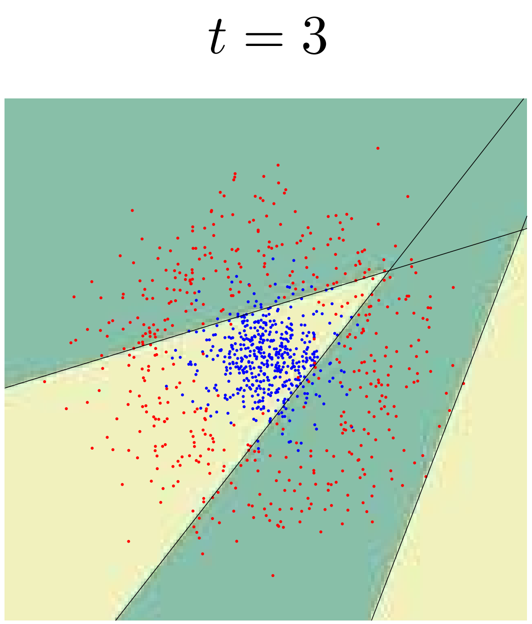
For $t = 1, ..., T$:

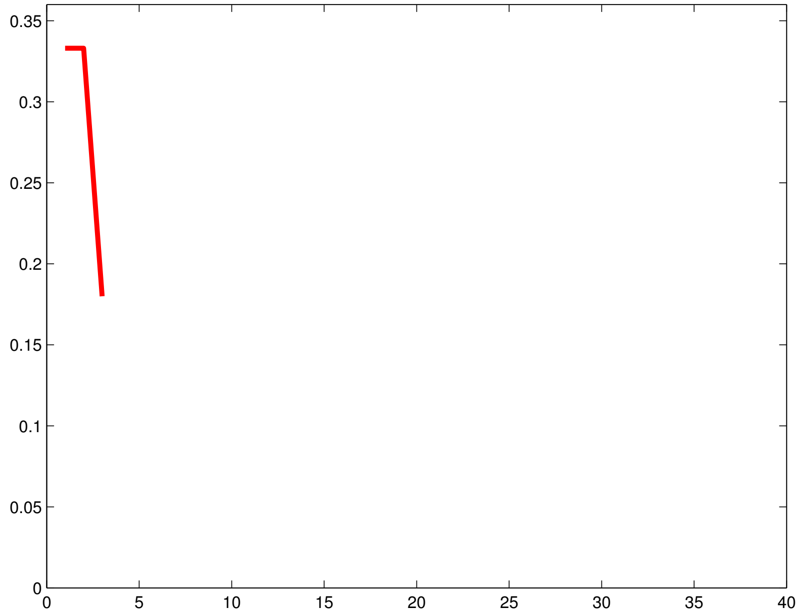
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$





Summary of the Algorithm

Initialization...
For $t = 1, ..., T$:

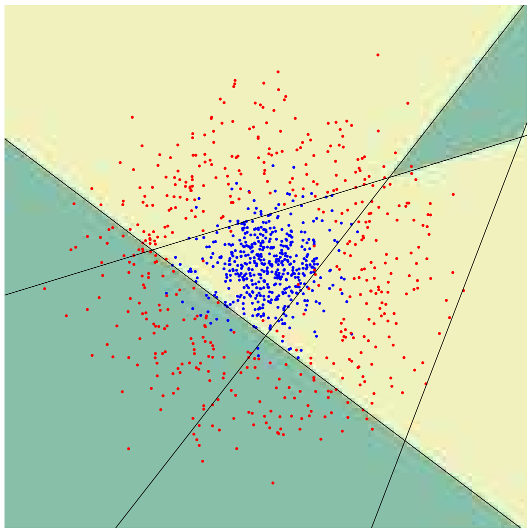
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

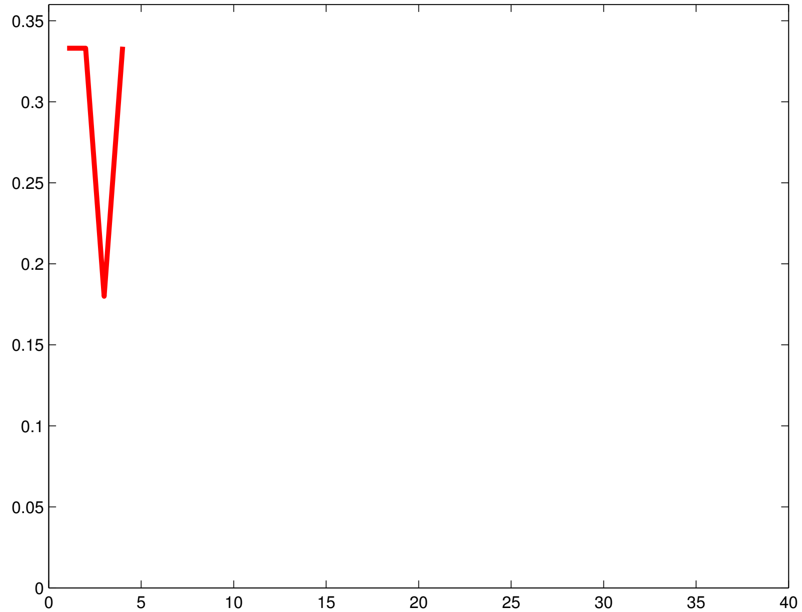
$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$t = 4$





Initialization...

For $t = 1, ..., T$:

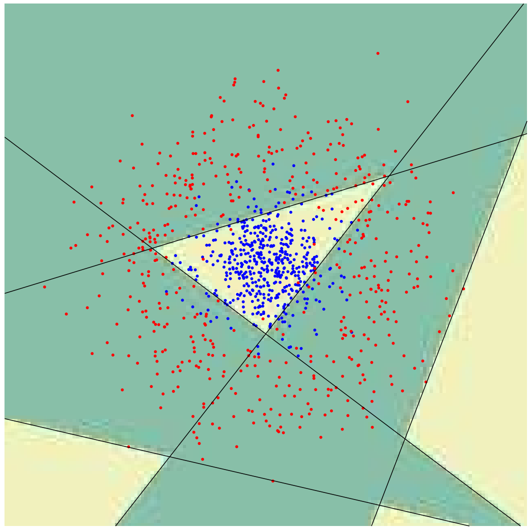
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

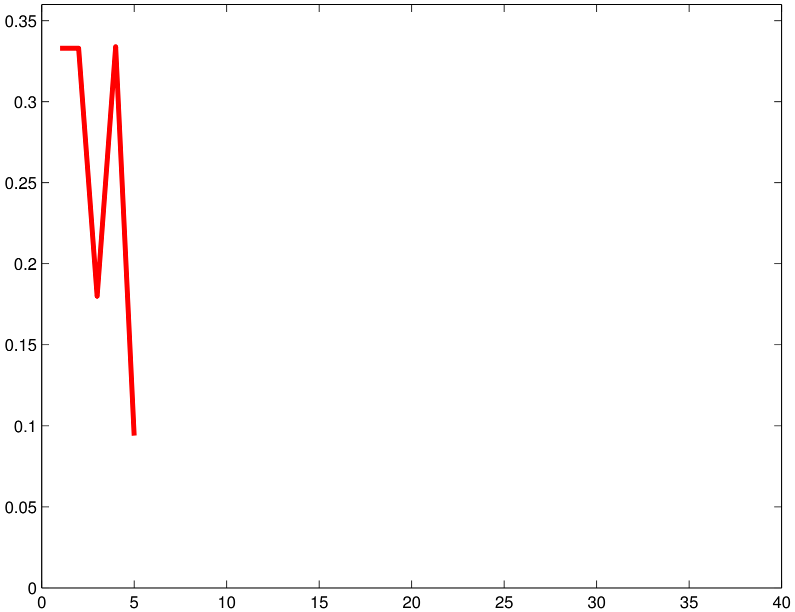
$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$t = 5$





Summary of the Algorithm

Initialization...

For $t = 1, ..., T$:

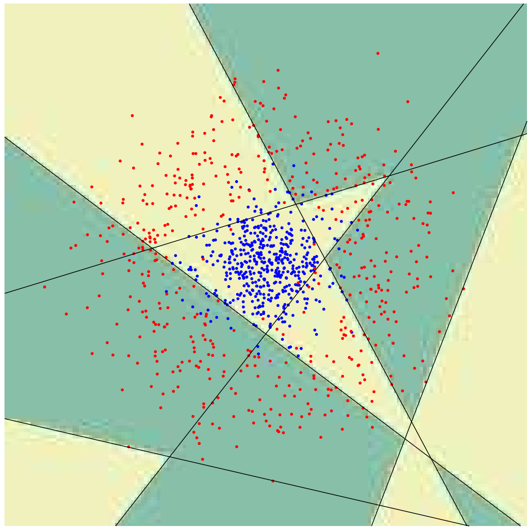
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

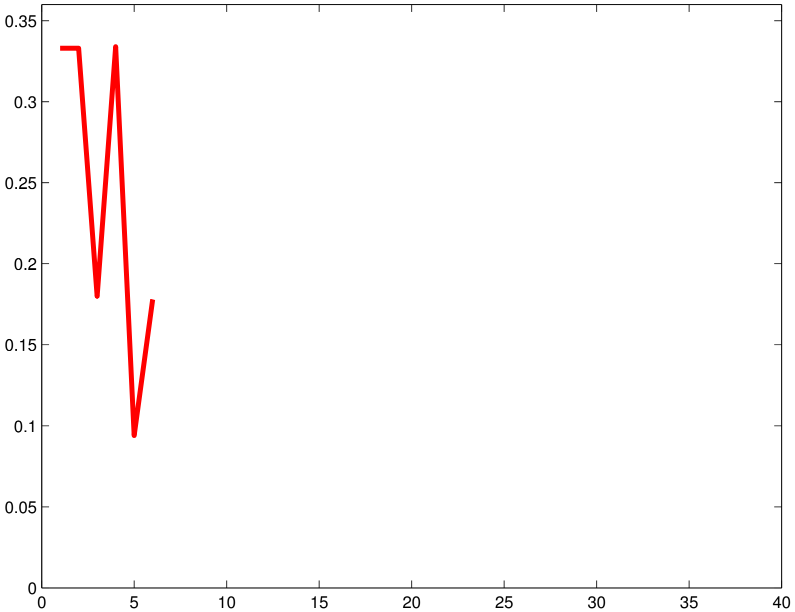
$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$t = 6$





Summary of the Algorithm

Initialization...
For $t = 1, ..., T$:

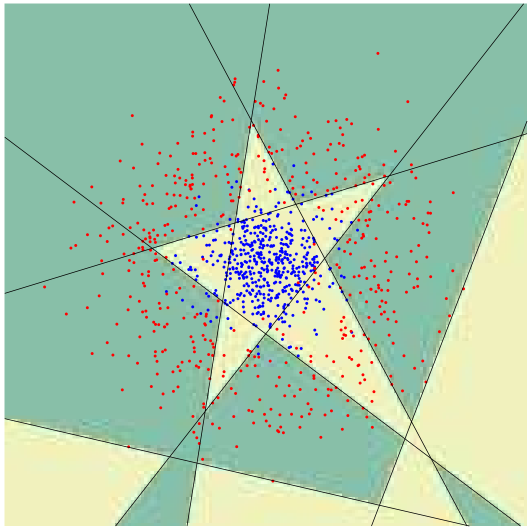
- Find $h_t = \arg \min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2} \log(\frac{1+r_t}{1-r_t})$
- Update

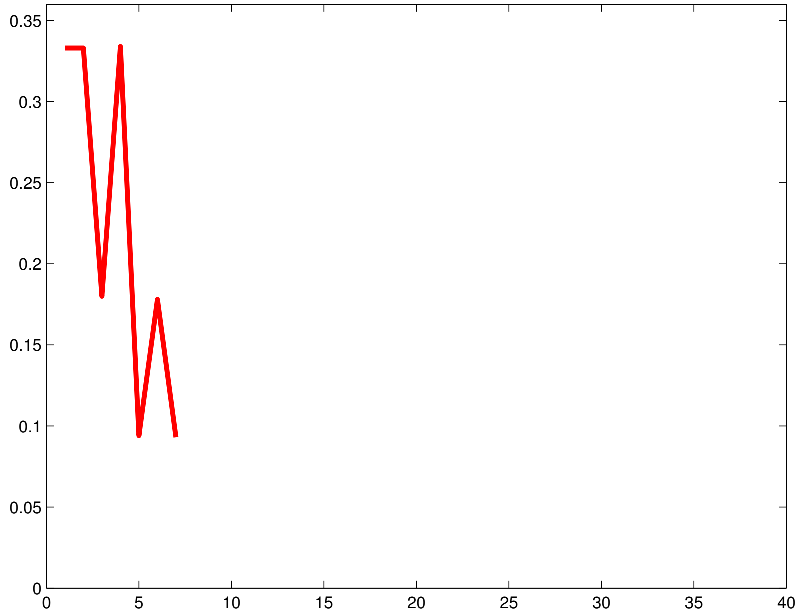
$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$t = 7$





Summary of the Algorithm

Initialization...
For $t = 1, ..., T$:

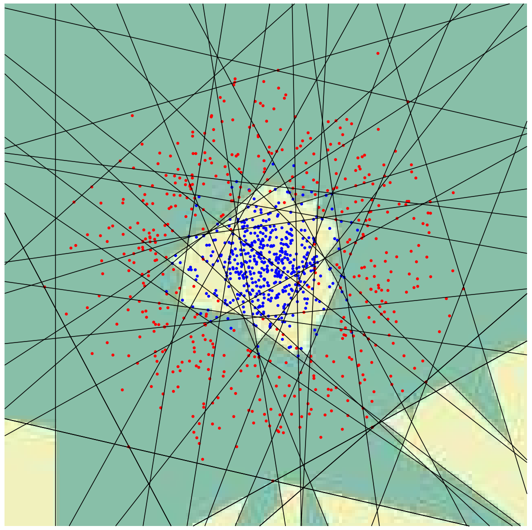
- Find $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$
- Update

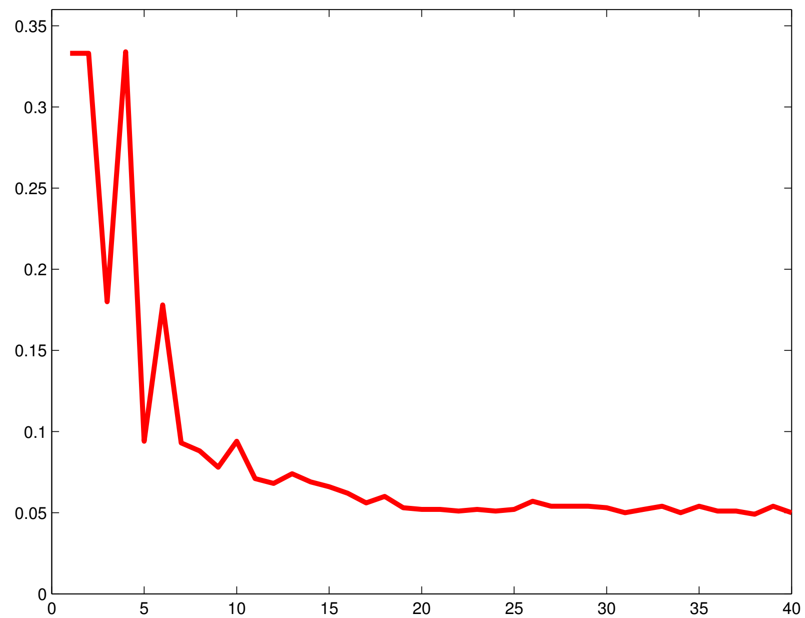
$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$t = 40$





1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Margins in SVM

$$\max \min_{(x,y)\in S} \frac{y(\vec{\alpha}\cdot\vec{h}(x))}{\|\vec{\alpha}\|_2}$$

Margins in AdaBoost

$$\max \min_{(x,y)\in S} \frac{y(\vec{\alpha}\cdot\vec{h}(x))}{\|\vec{\alpha}\|_1}$$

Maximizing margins in AdaBoost

$$P_S[yf(x)\le\theta] \le 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \qquad \text{where } f(x) = \frac{\vec{\alpha}\cdot\vec{h}(x)}{\|\vec{\alpha}\|_1}$$

Upper bounds based on margin

$$P_{\mathcal{D}}[yf(x)\le 0] \le P_S[yf(x)\le\theta] + \mathcal{O}\left(\frac{1}{\sqrt{m}}\left(\frac{d\log^2(m/d)}{\theta^2}+\log(1/\delta)\right)^{1/2}\right)$$

Freund & Schapire 1995

- Discrete ($h : \mathcal{X} \to \{0,1\}$)
- Multiclass AdaBoost.M1 ($h : \mathcal{X} \to \{0,1,...,k\}$)
- Multiclass AdaBoost.M2 ($h : \mathcal{X} \to [0,1]^k$)
- Real valued AdaBoost.R ($Y = [0,1]$, $h : \mathcal{X} \to [0,1]$)

Schapire & Singer 1997

- Confidence rated prediction ($h : \mathcal{X} \to R$, two-class)
- Multilabel AdaBoost.MR, AdaBoost.MH (different formulation of minimized loss)

... Many other modifications since then (Totally Corrective AB, Cascaded AB)

Advantages

- Very simple to implement
- Feature selection on very large sets of features
- Fairly good generalization

Disadvantages

- Suboptimal solution for $\bar{\alpha}$
- Can overfit in presence of noise

Adaboost with a Totally Corrective Step (TCA)

Given: $(x_1, y_1), ..., (x_m, y_m)$; $x_i \in \mathcal{X}, y_i \in \{-1, 1\}$
Initialize weights $D_1(i) = 1/m$

For $t = 1, ..., T$:

1. (Call *WeakLearn*), which returns the weak classifier $h_t : \mathcal{X} \to \{-1, 1\}$ with minimum error w.r.t. distribution D_t;
2. Choose $\alpha_t \in R$,
3. Update D_{t+1}
4. (Call *WeakLearn*) on the set of h_m's with non zero α's . Update $\alpha_.$. Update D_{t+1}. Repeat till $|\epsilon_t - 1/2| < \delta, \forall t$.

Comments

- **All** weak classifiers have $\epsilon_t \approx 1/2$, therefore the classifier selected at $t + 1$ is "independent" of all classifiers selected so far.
- It can be easily shown, that the totally corrective step reduces the upper bound on the empirical error without increasing classifier complexity.
- The TCA was first proposed by Kivinen and Warmuth, but their α_t is set as in stadard Adaboost.
- Generalization of TCA is an open question.

- Discrete AdaBoost, Real AdaBoost, and Discrete and Real TCA evaluated
- Weak learner: stumps.
- Data from the IDA repository (Ratsch:2000):

	Input dimension	Training patterns	Testing patterns	Number of realizations
Banana	2	400	4900	100
Breast cancer	9	200	77	100
Diabetes	8	468	300	100
German	20	700	300	100
Heart	13	170	100	100
Image segment	18	1300	1010	20
Ringnorm	20	400	7000	100
Flare solar	9	666	400	100
Splice	60	1000	2175	20
Thyroid	5	140	75	100
Titanic	3	150	2051	100
Twonorm	20	400	7000	100
Waveform	21	400	4600	100

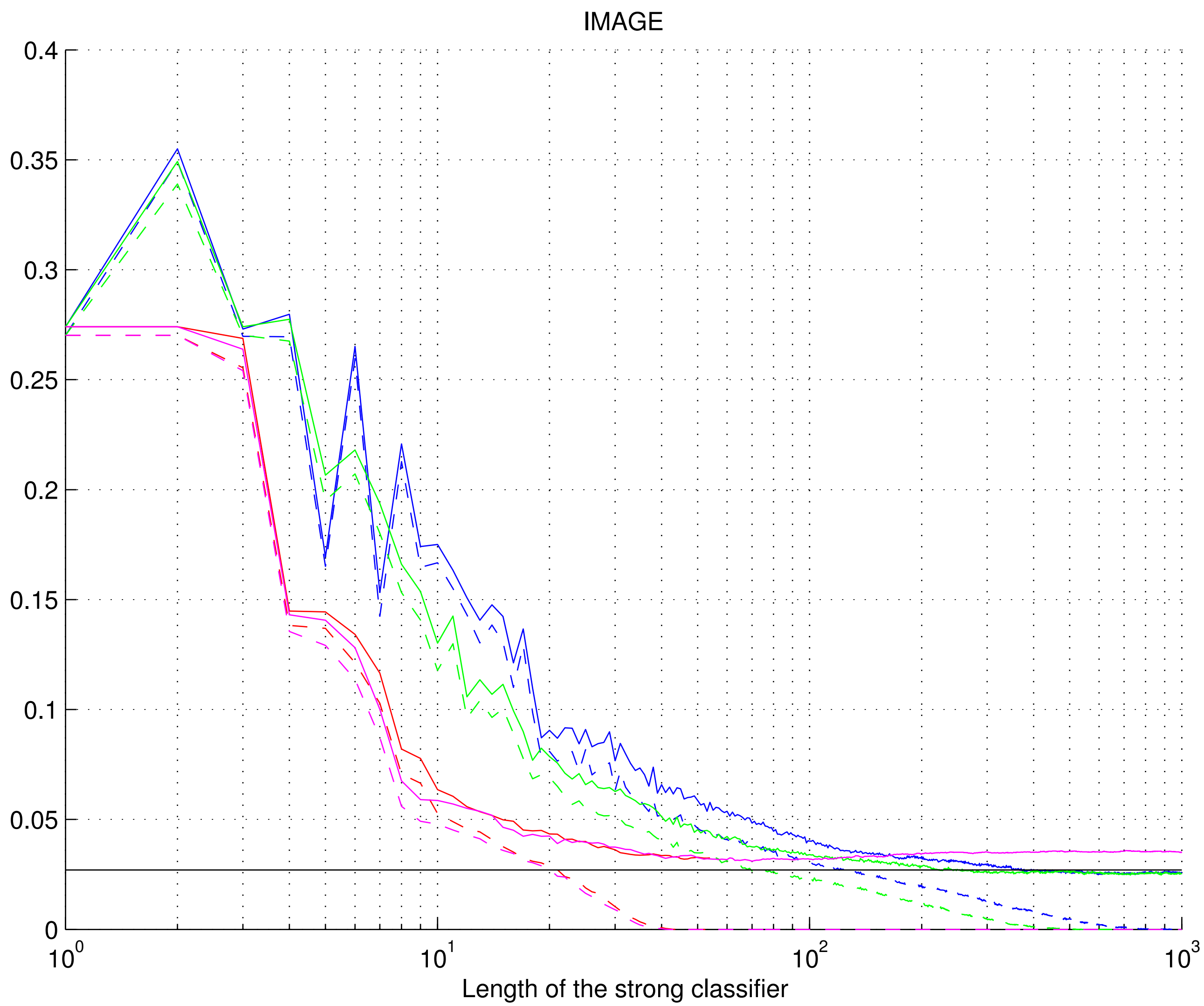
- Note that the training sets are fairly small

Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)

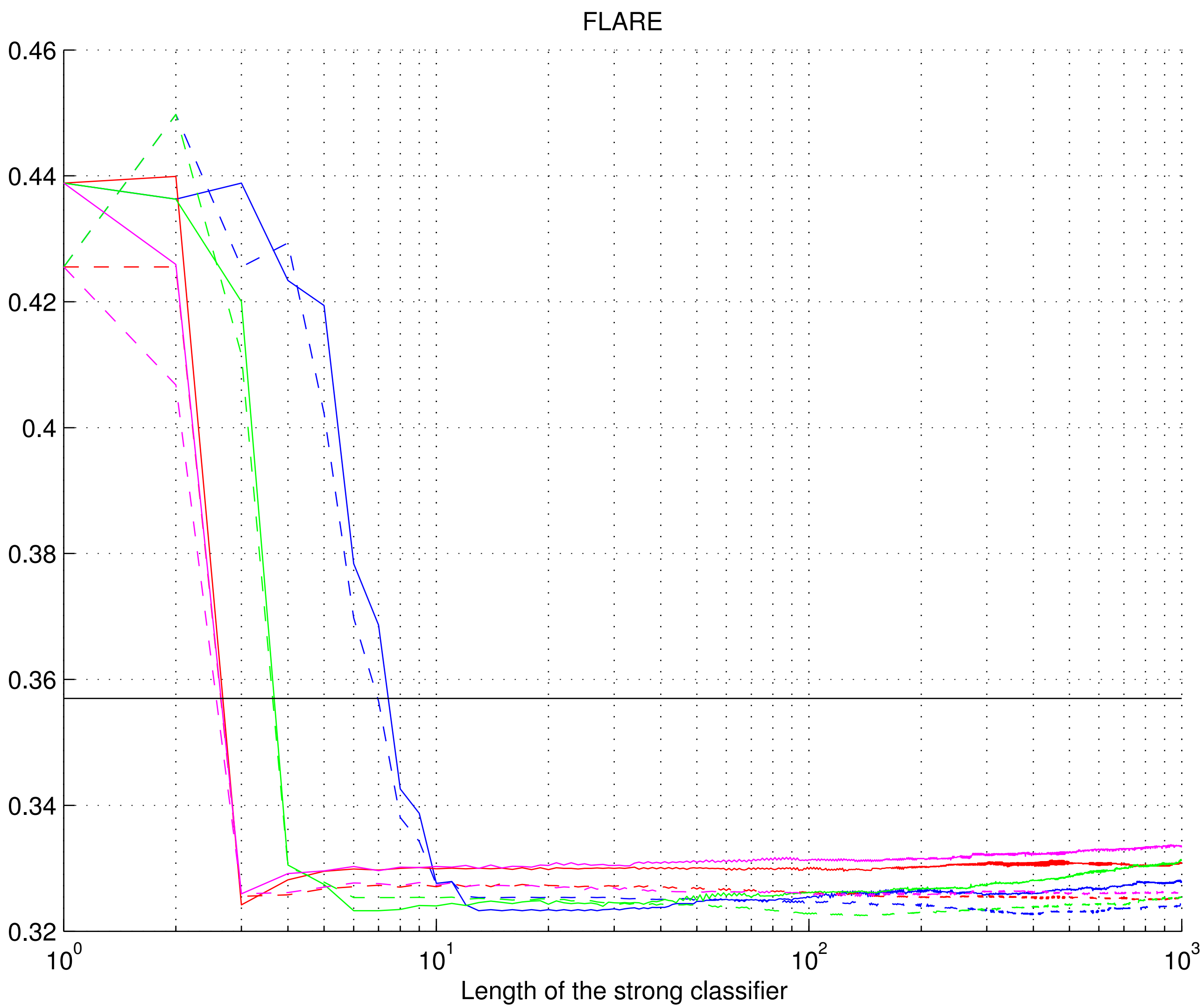
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



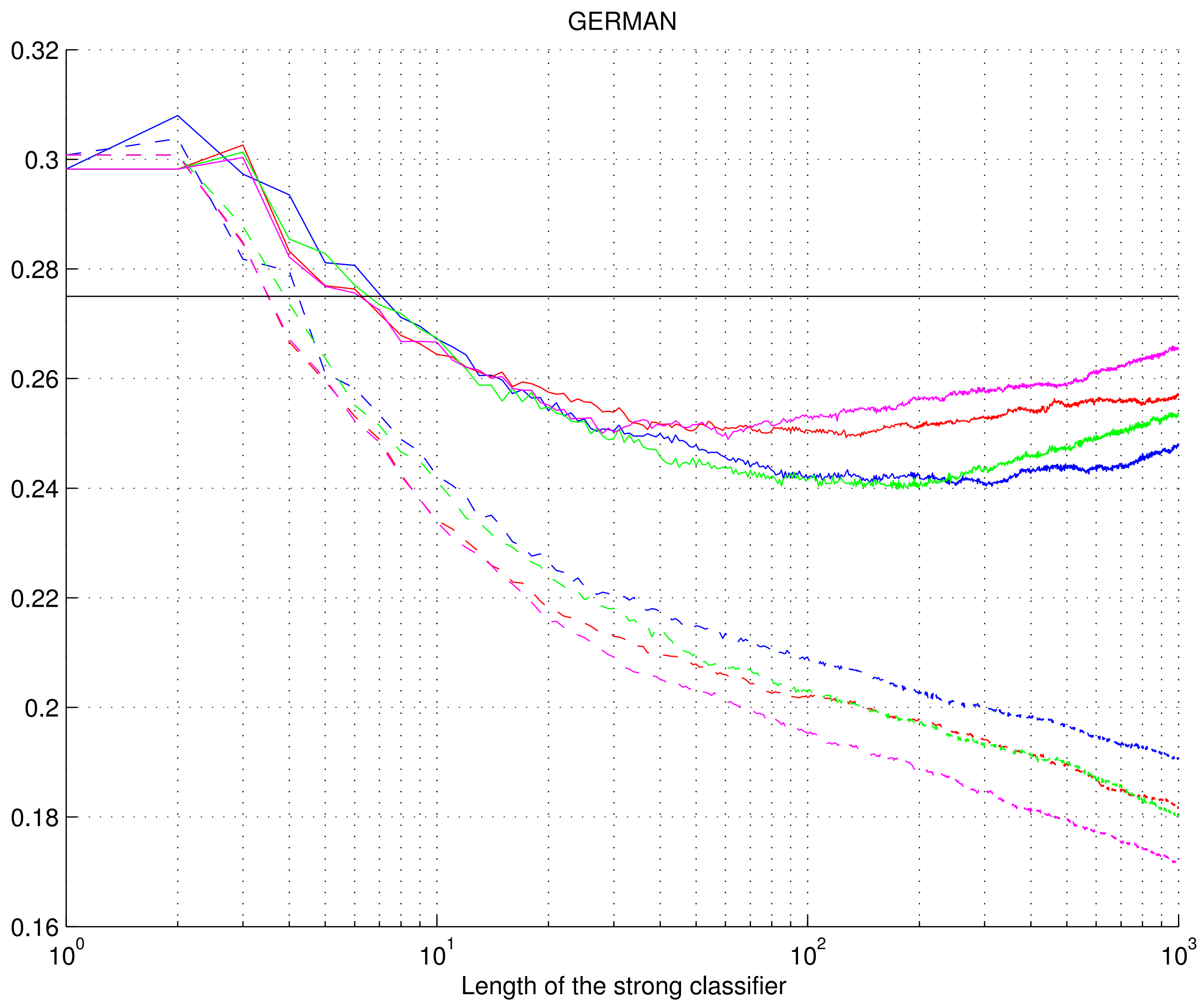
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



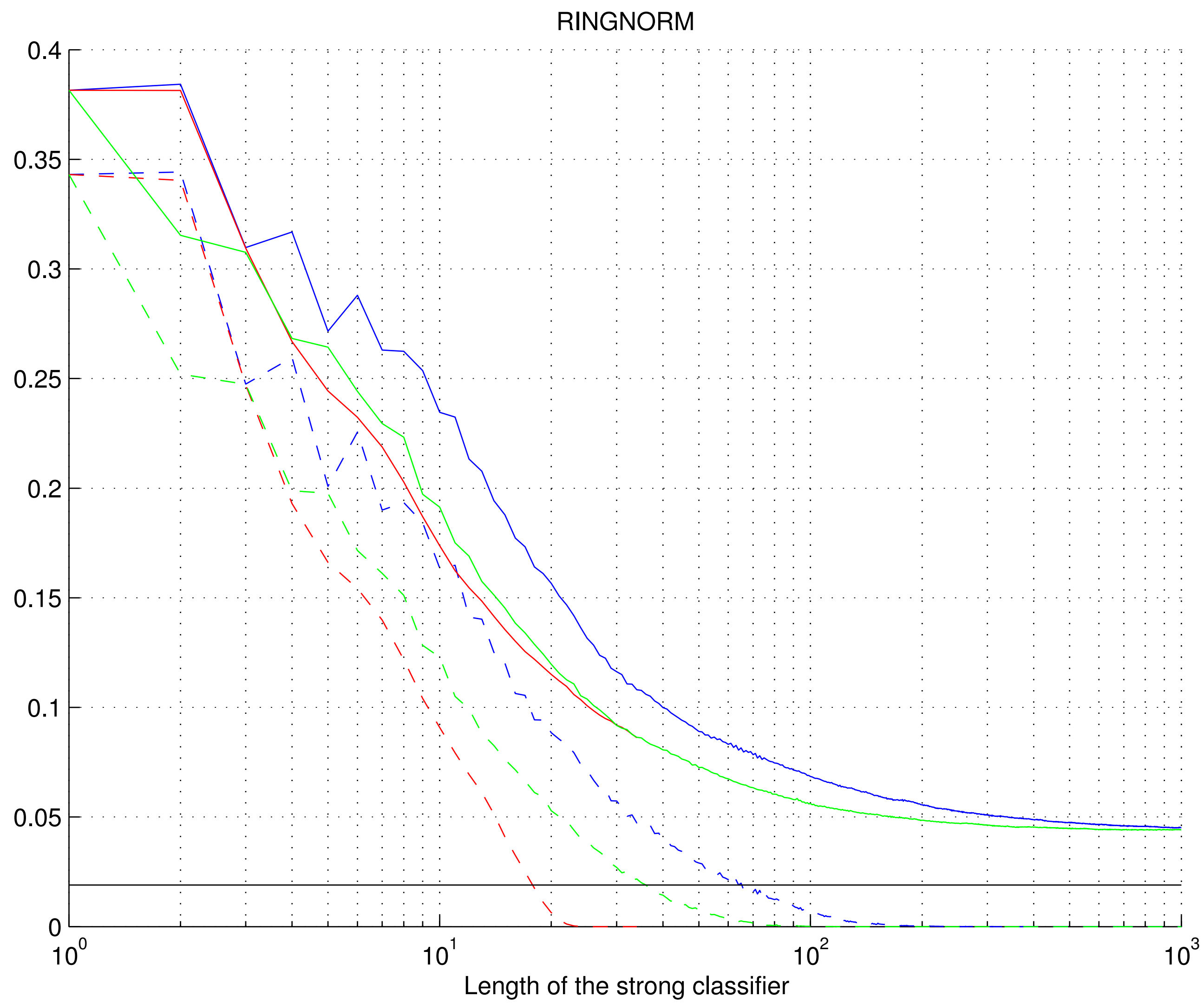
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



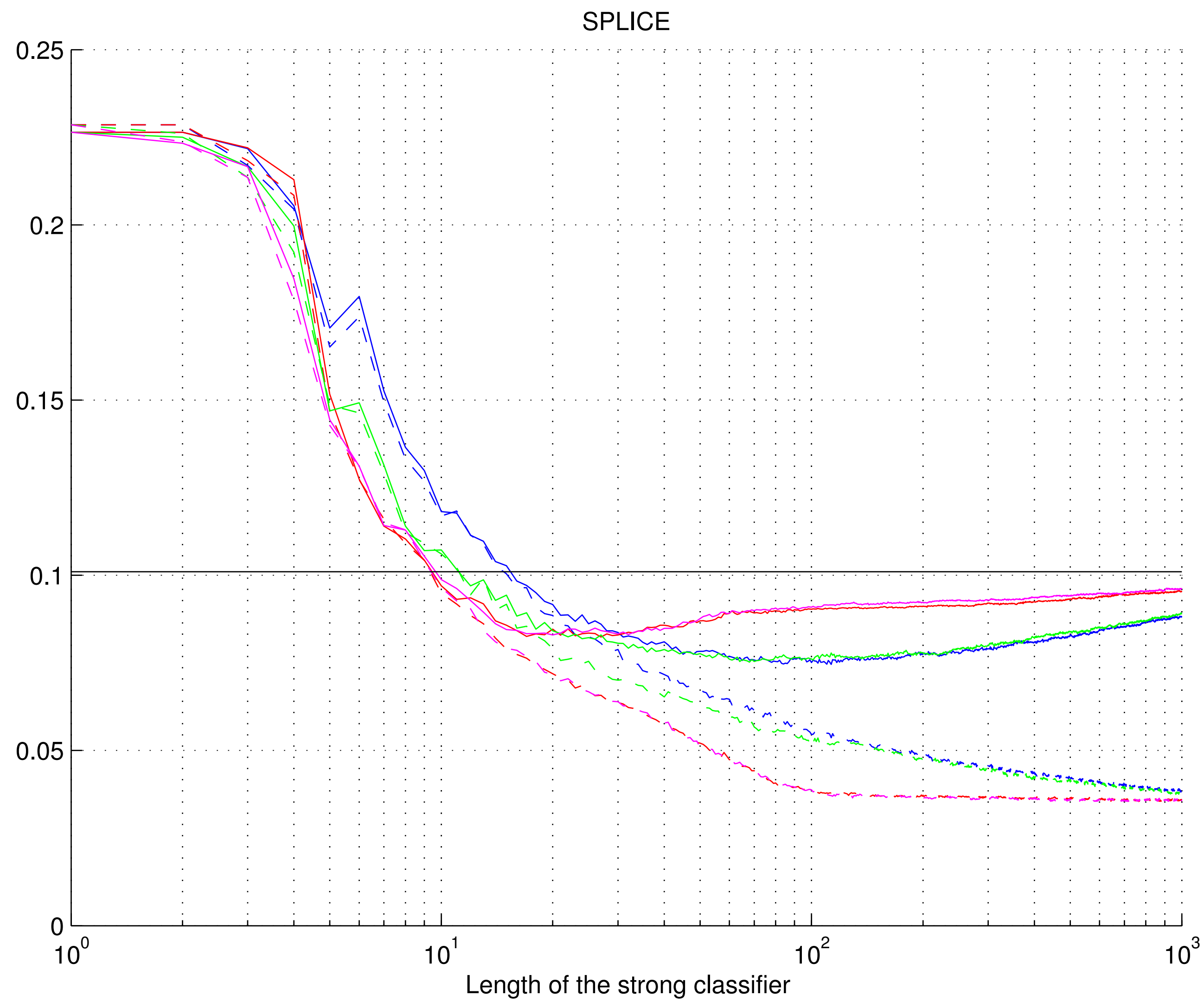
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



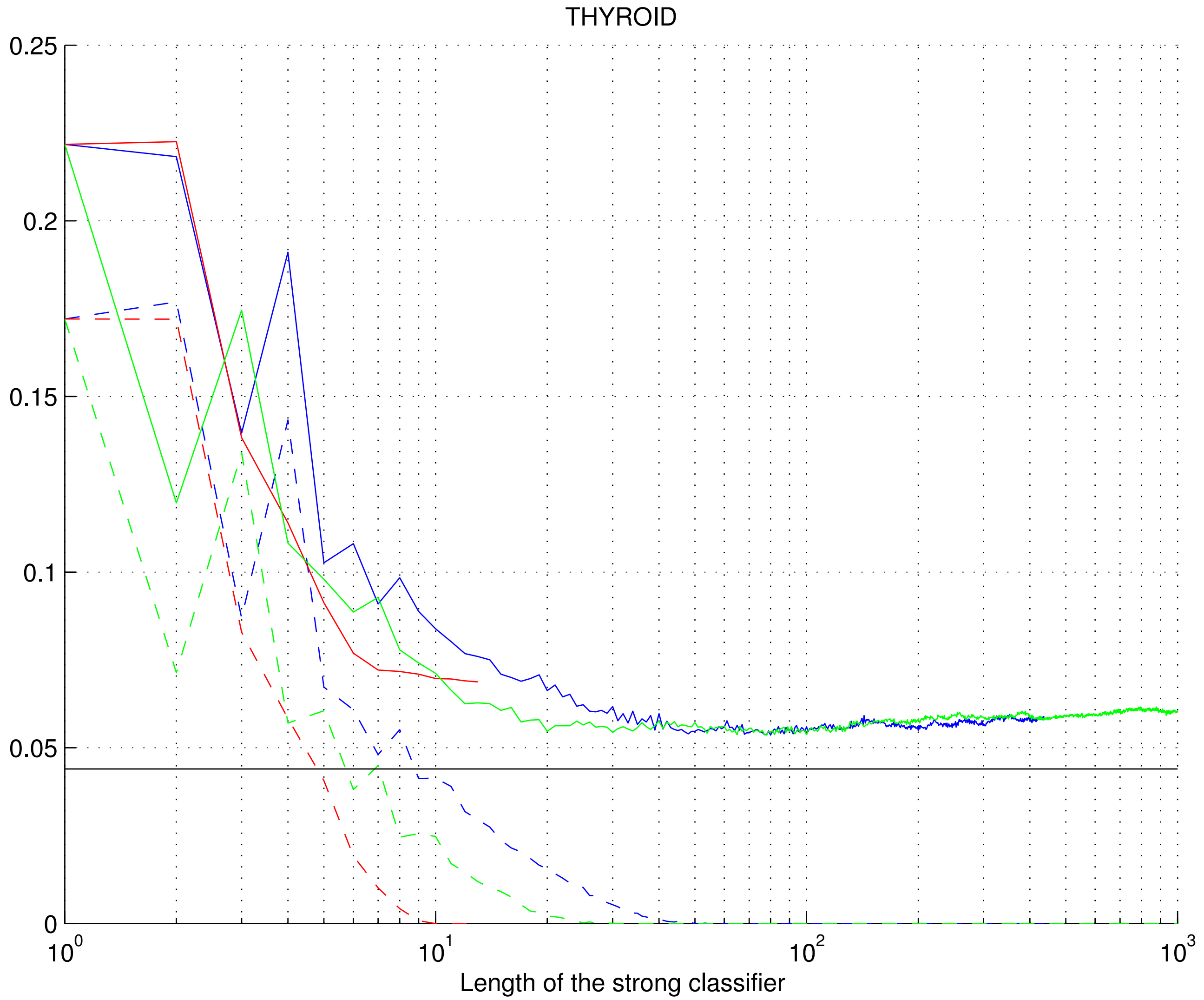
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



Results with TCA on the IDA Database

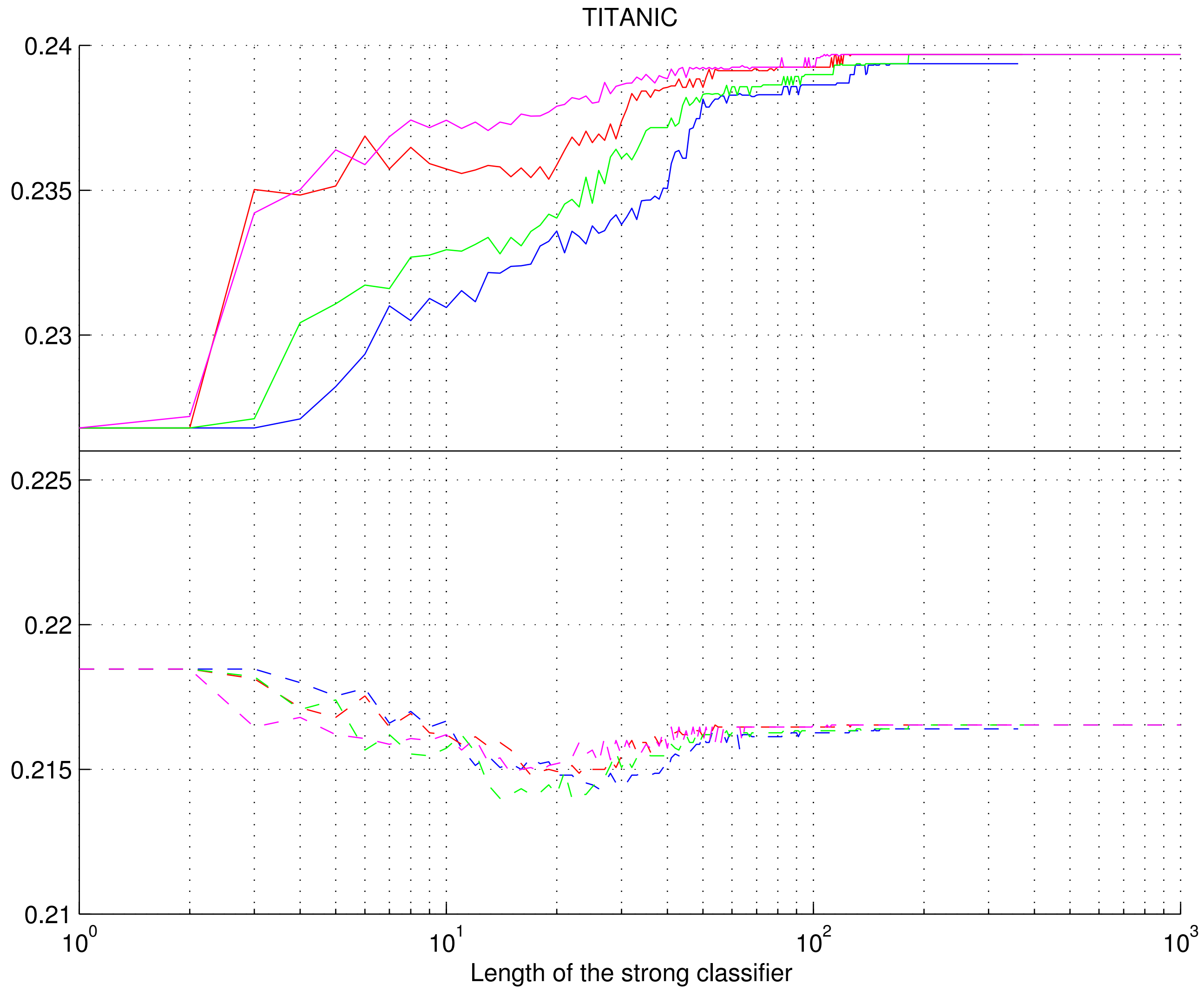
- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

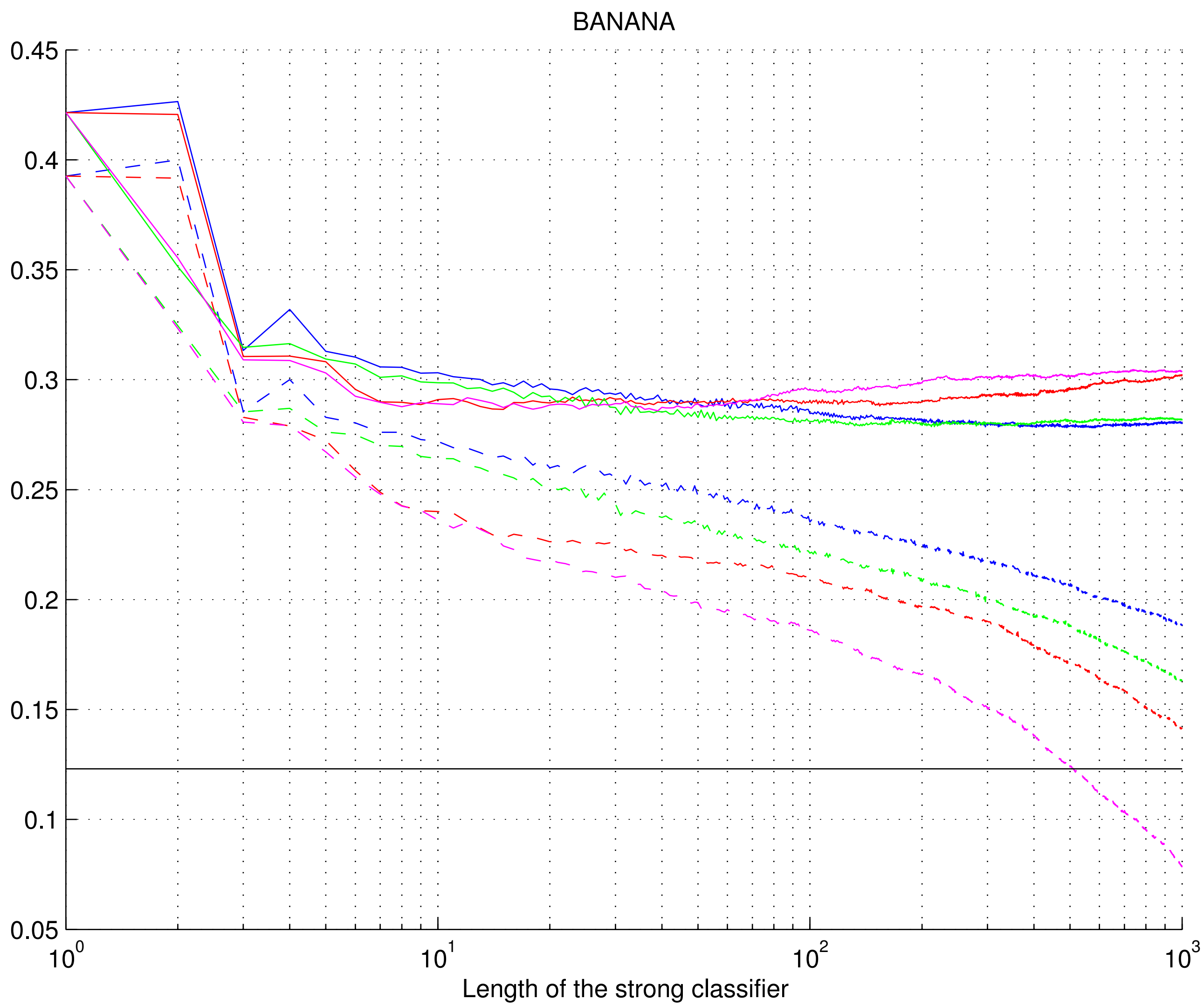
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



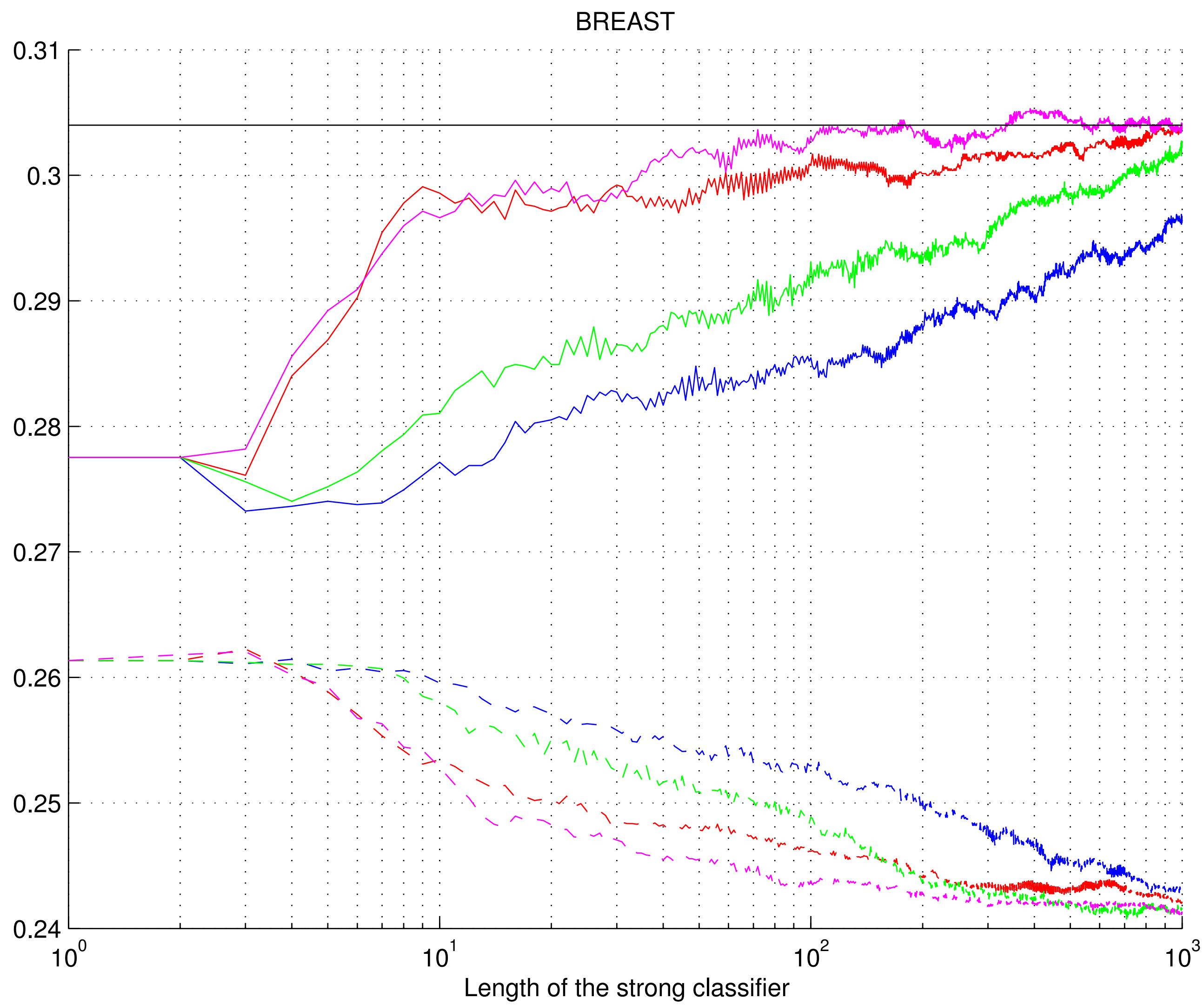
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



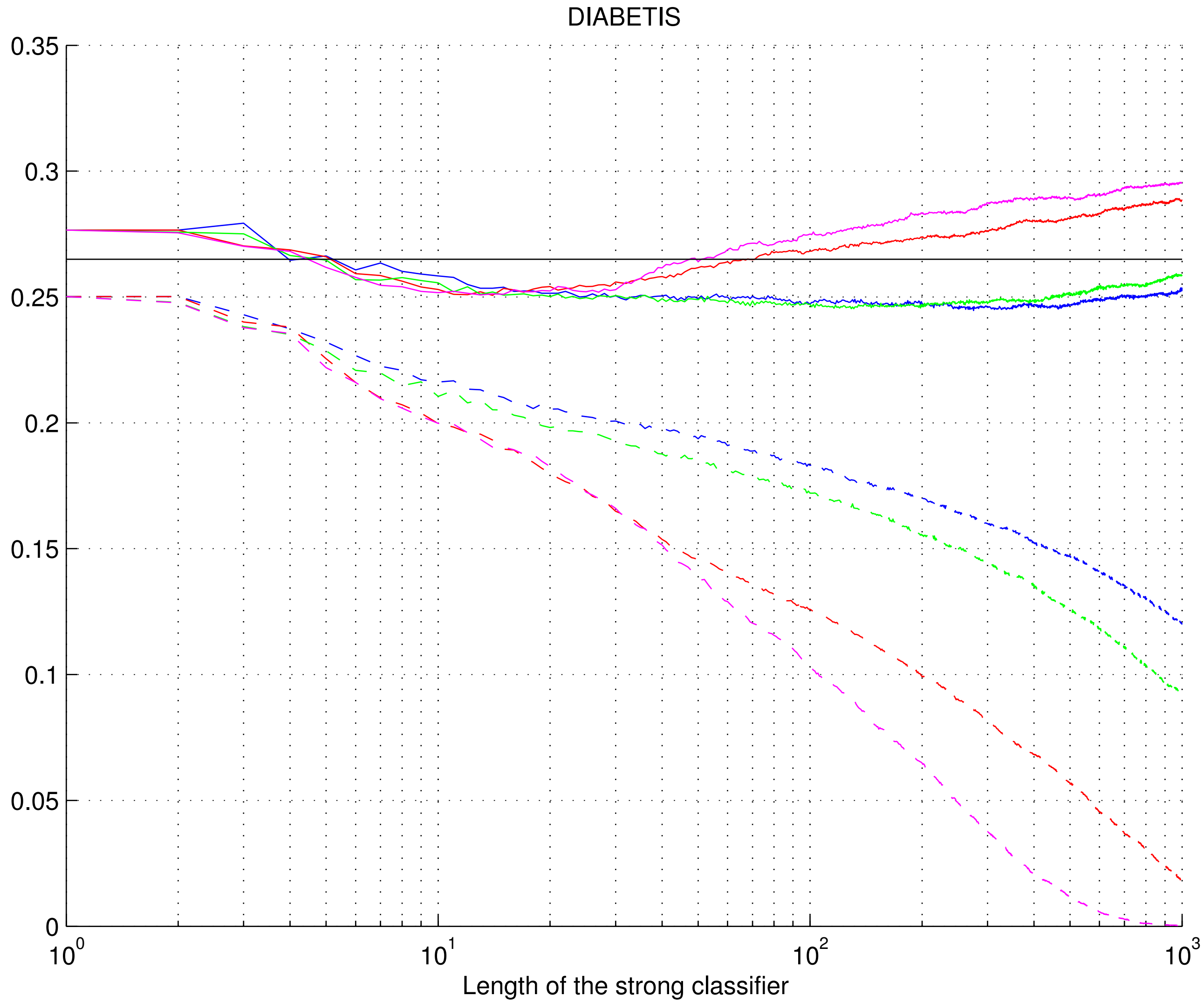
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



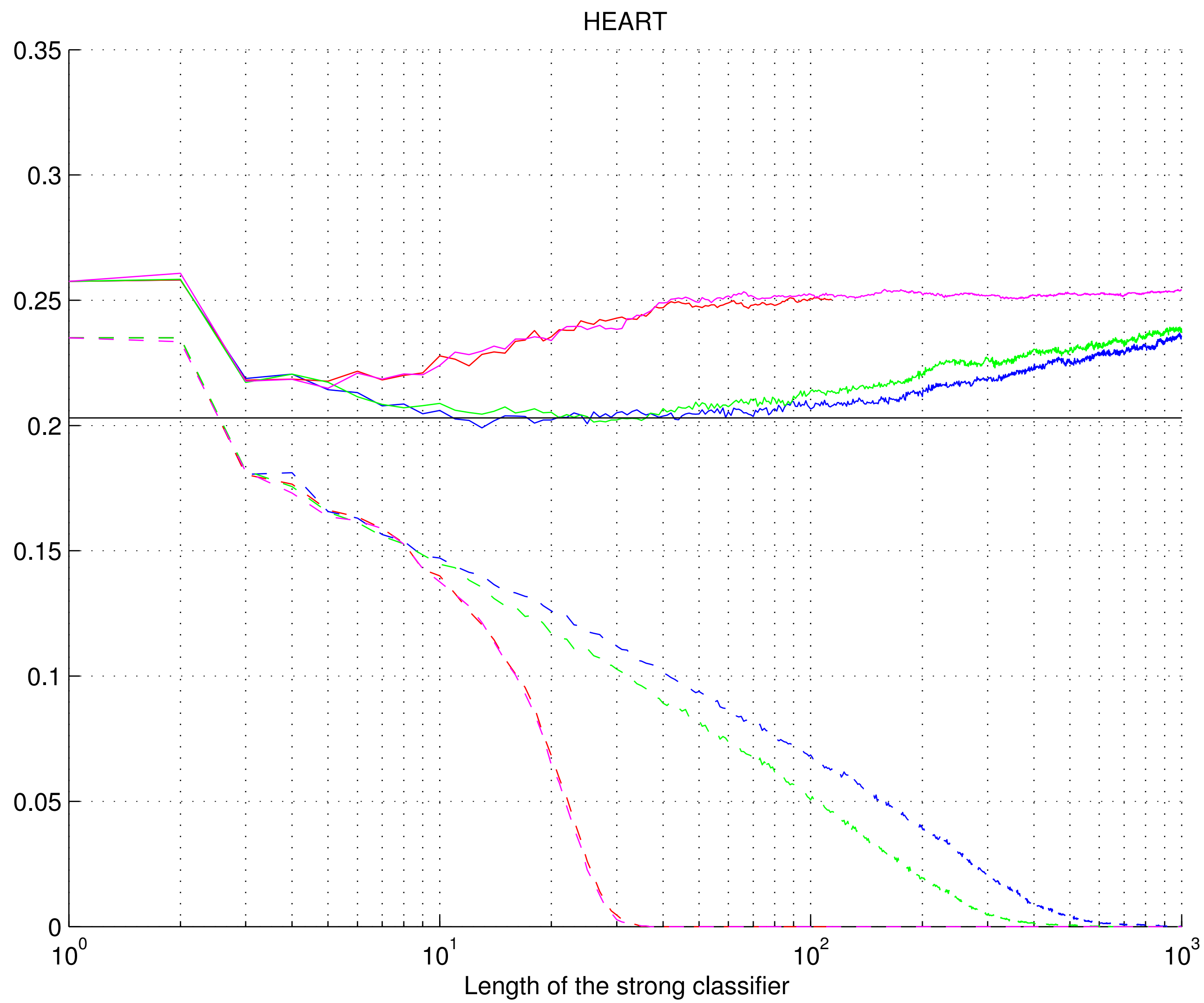
Results with TCA on the IDA Database

- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)



Results with TCA on the IDA Database

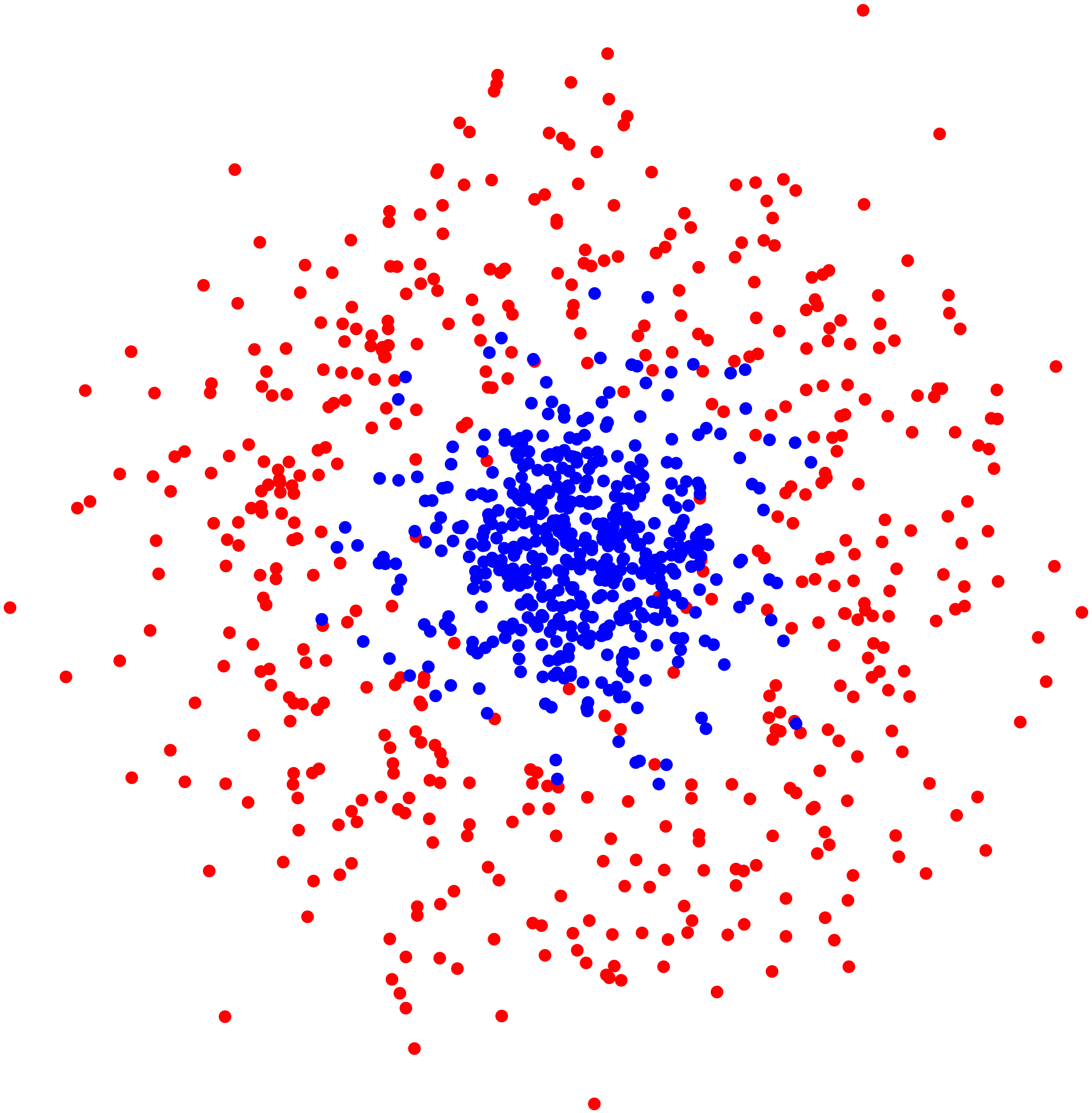
- Training error (dashed line), test error (solid line)
- Discrete AdaBoost (blue), Real AdaBoost (green),
- Discrete AdaBoost with TCA (red), Real AdaBoost with TCA (cyan)
- the black horizontal line: the error of AdaBoost with RBF network weak classifiers from (Ratsch-ML:2000)

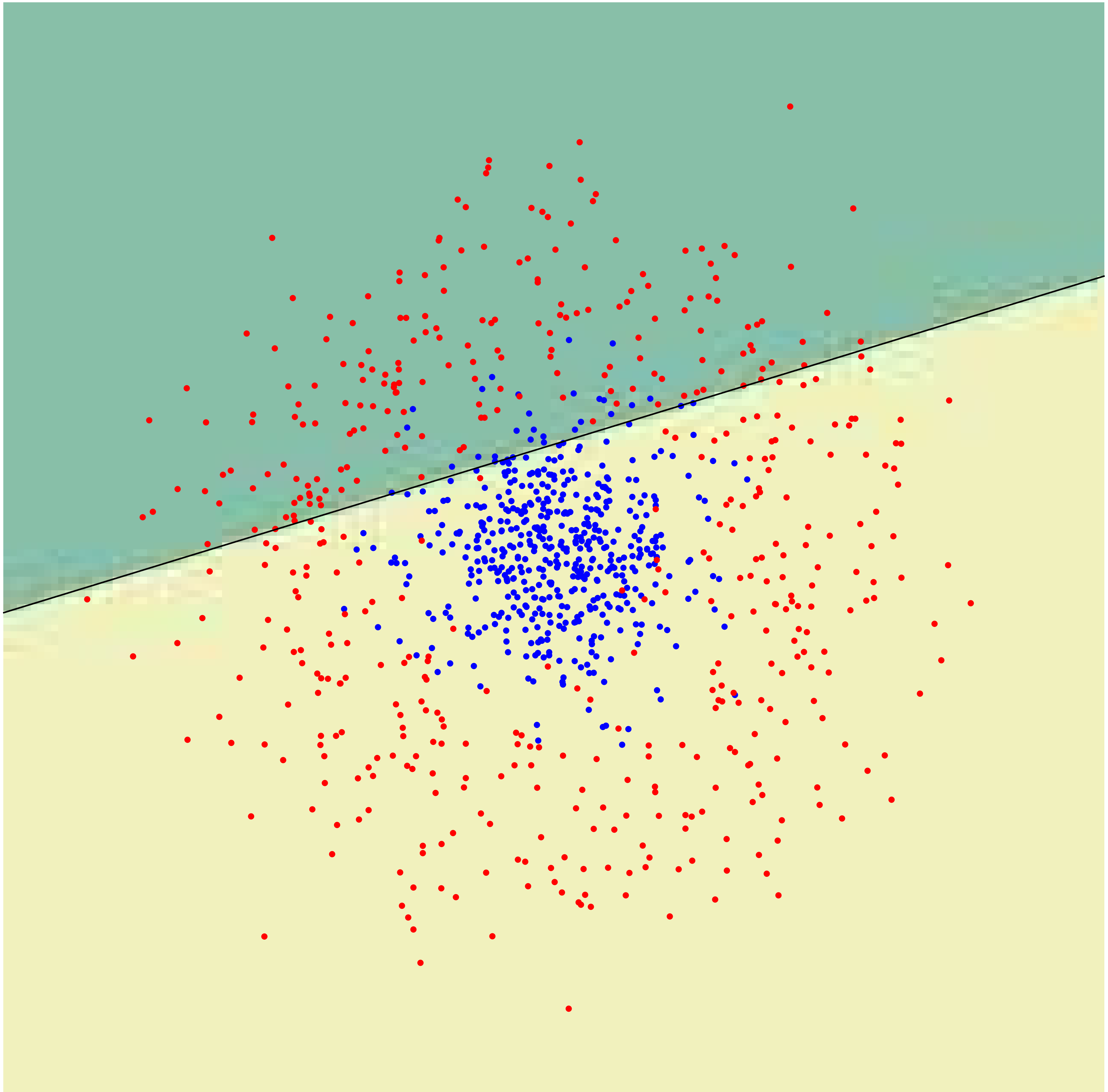


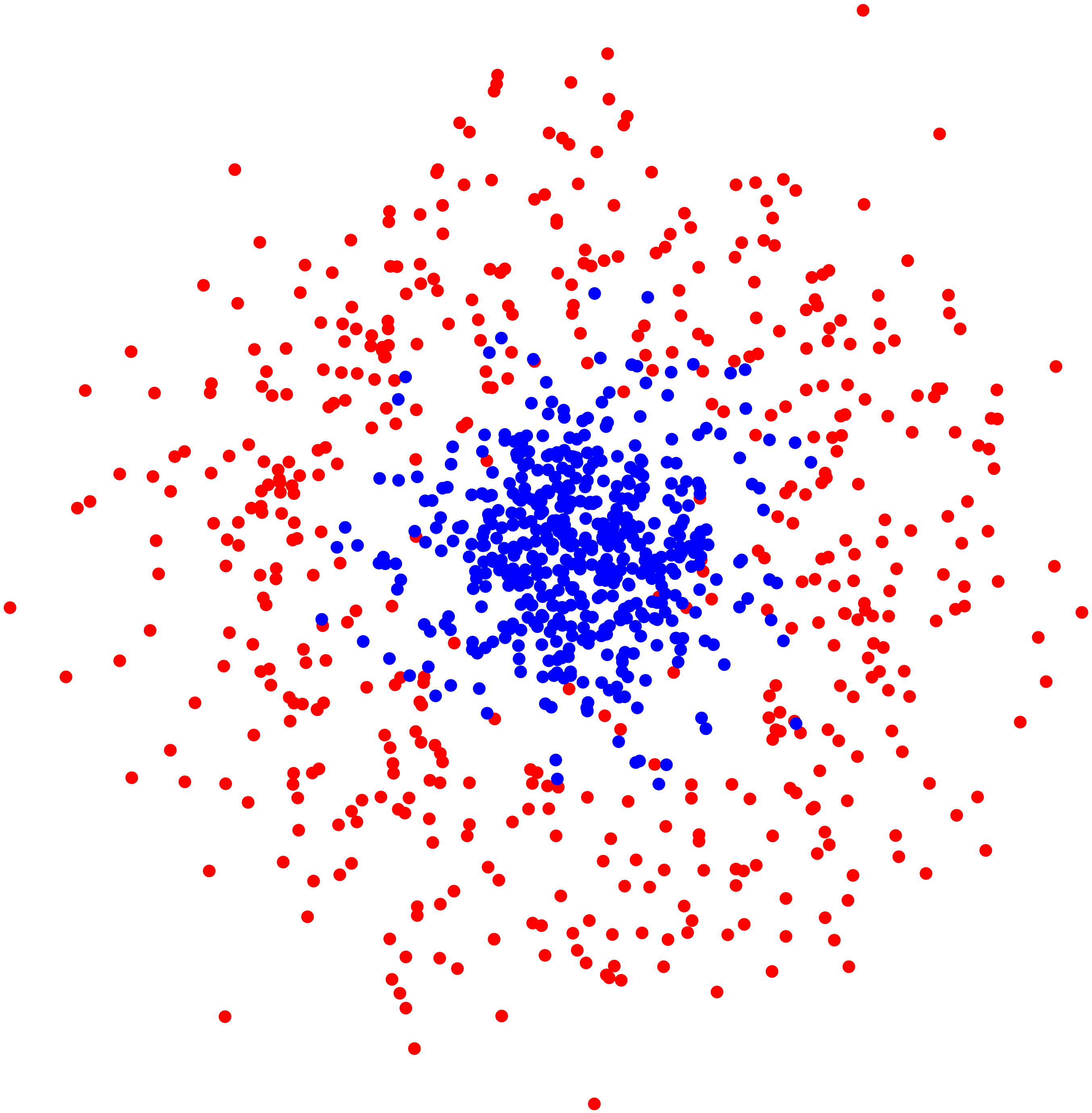
Conclusions

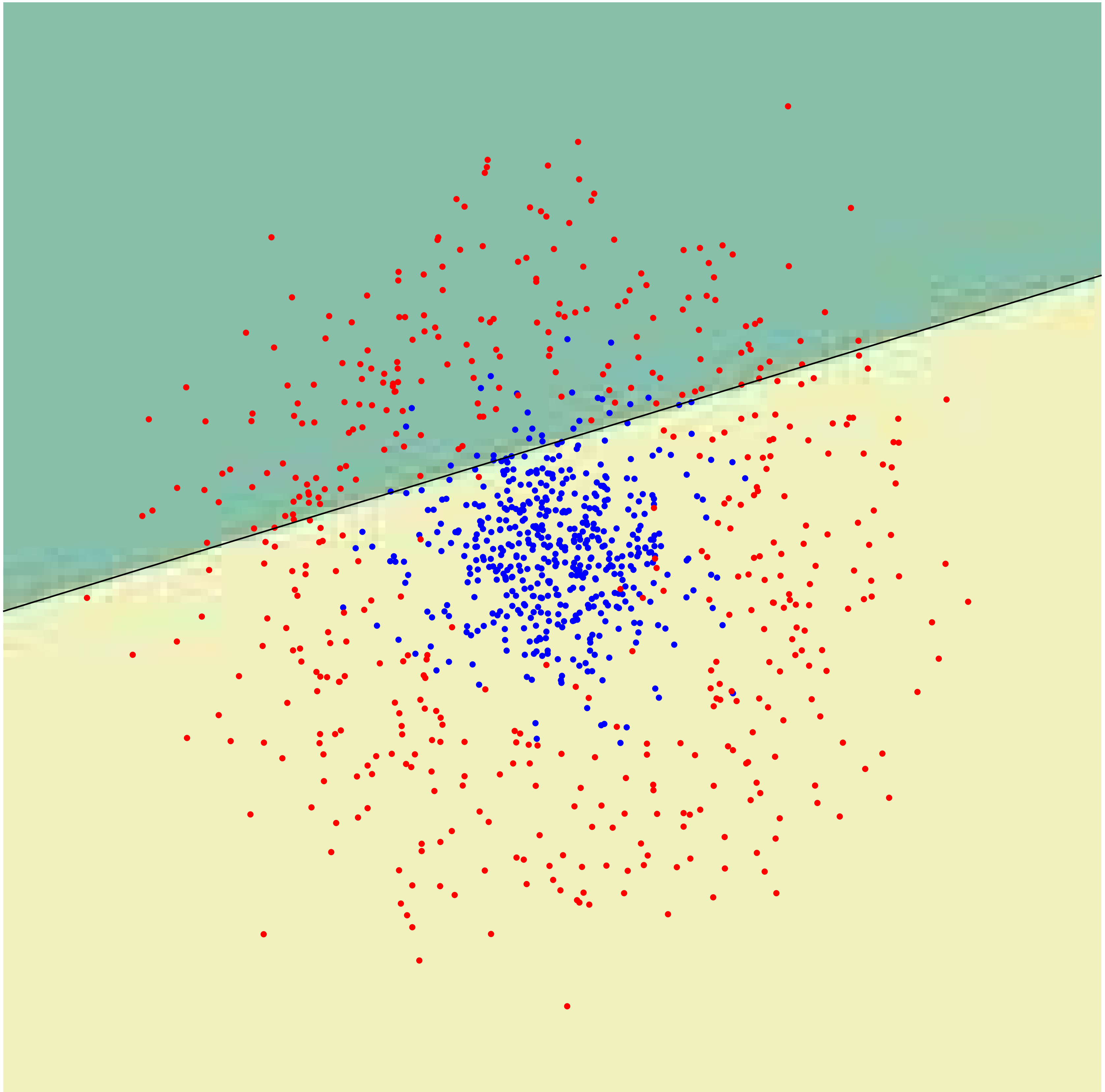
- The AdaBoost algorithm was presented and analysed
- A modification of the Totally Corrective AdaBoost was introduced
- Initial test show that the TCA outperforms AB on some standard data sets.

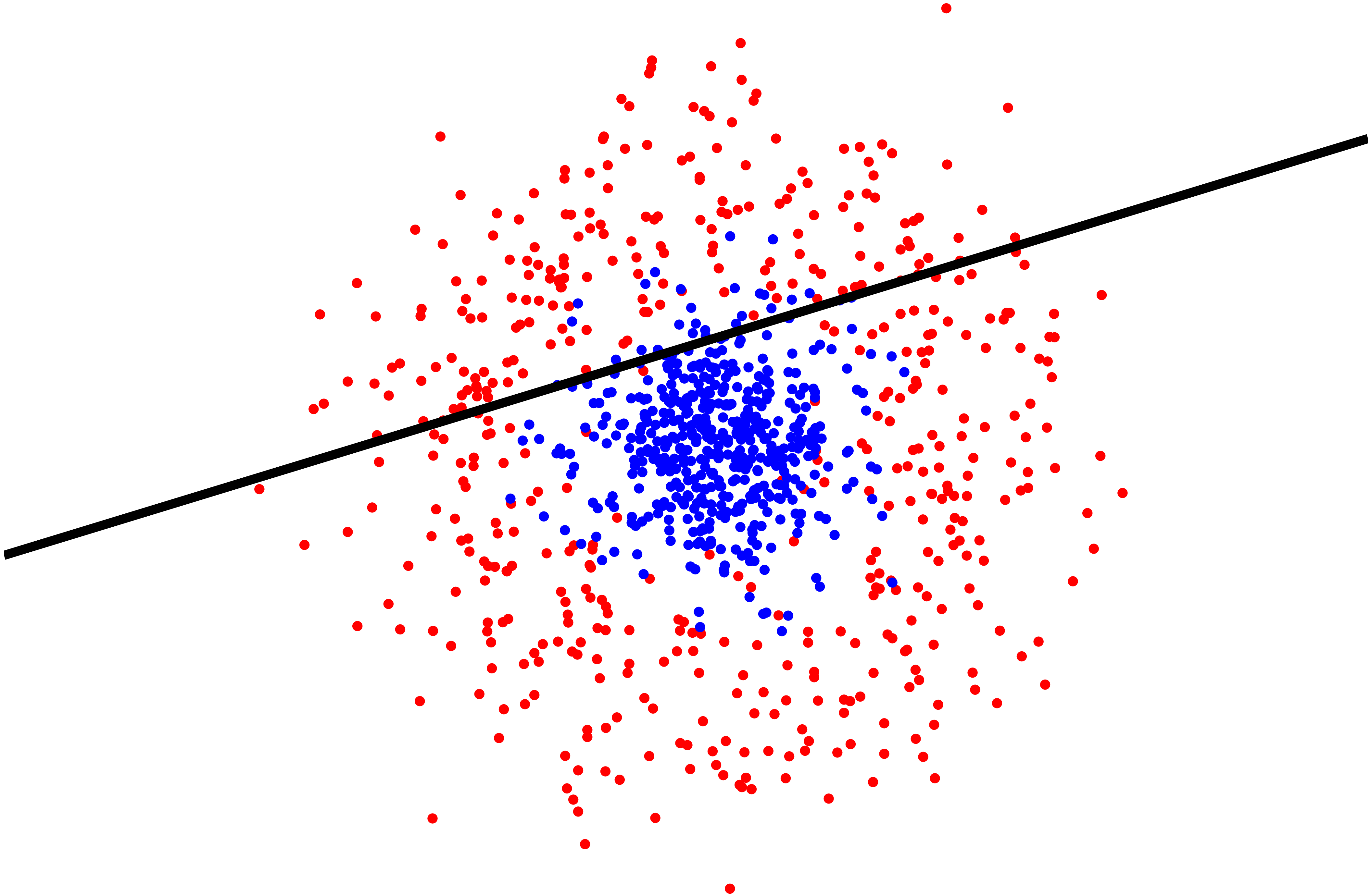
m p

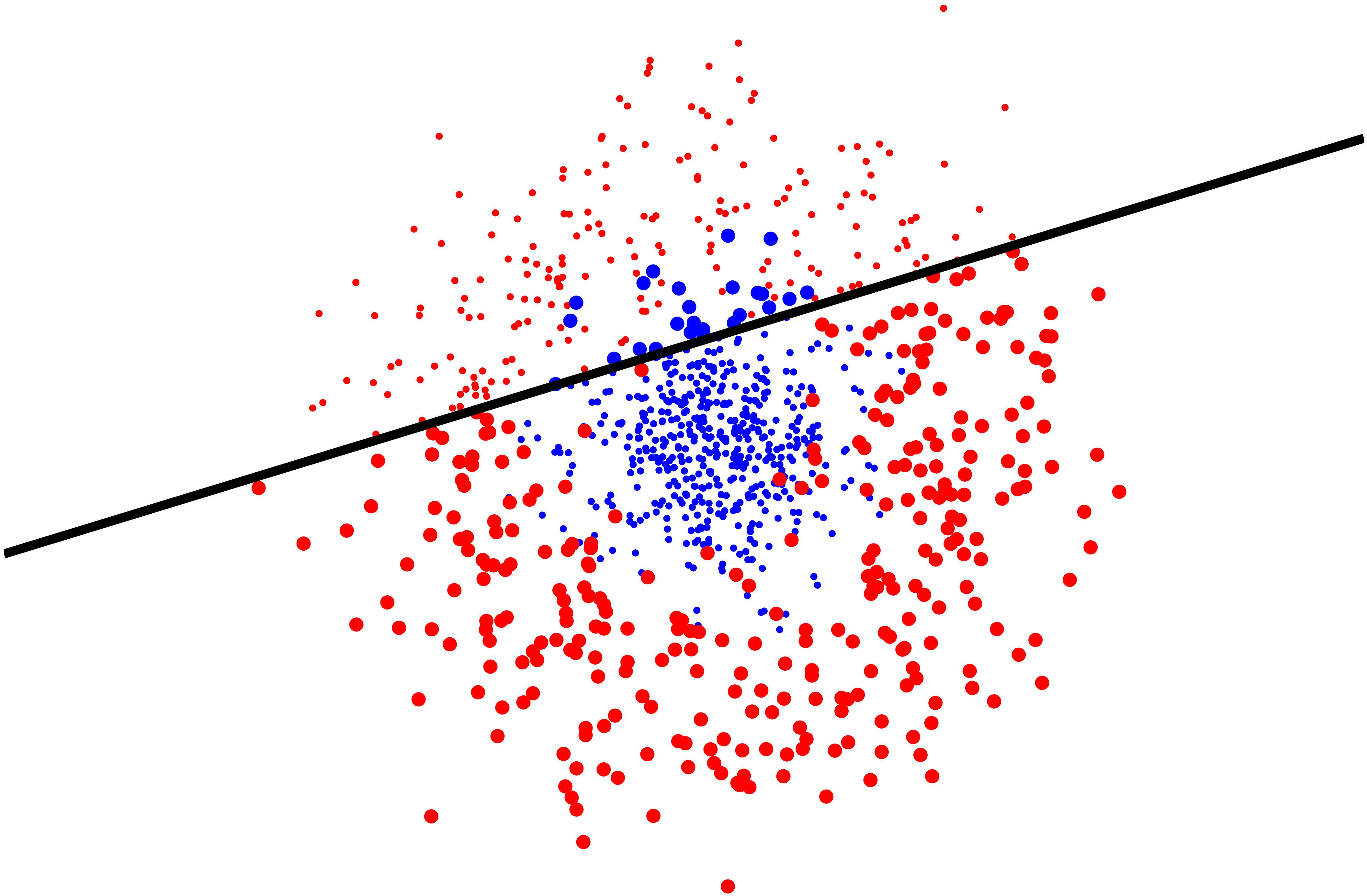


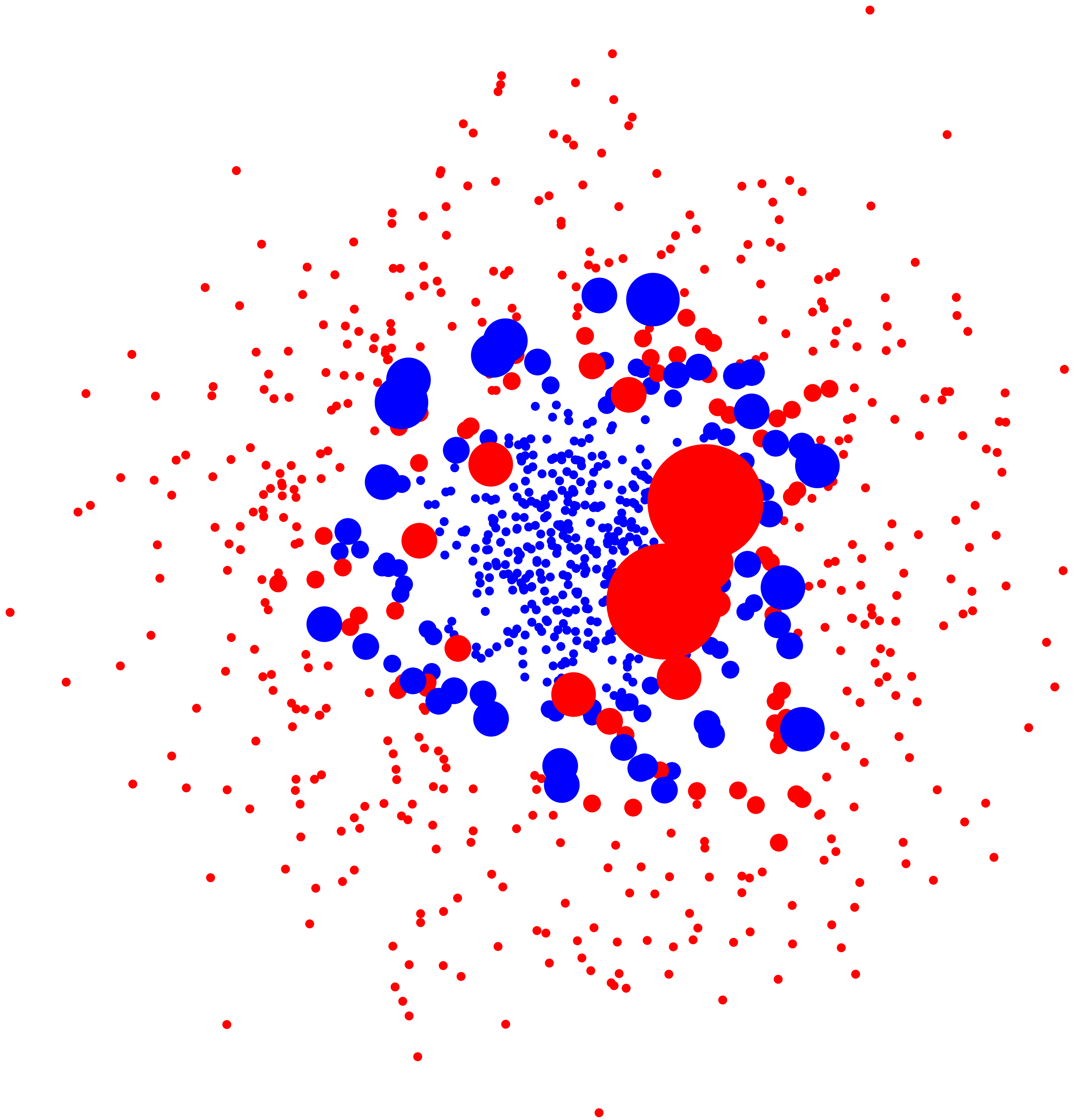


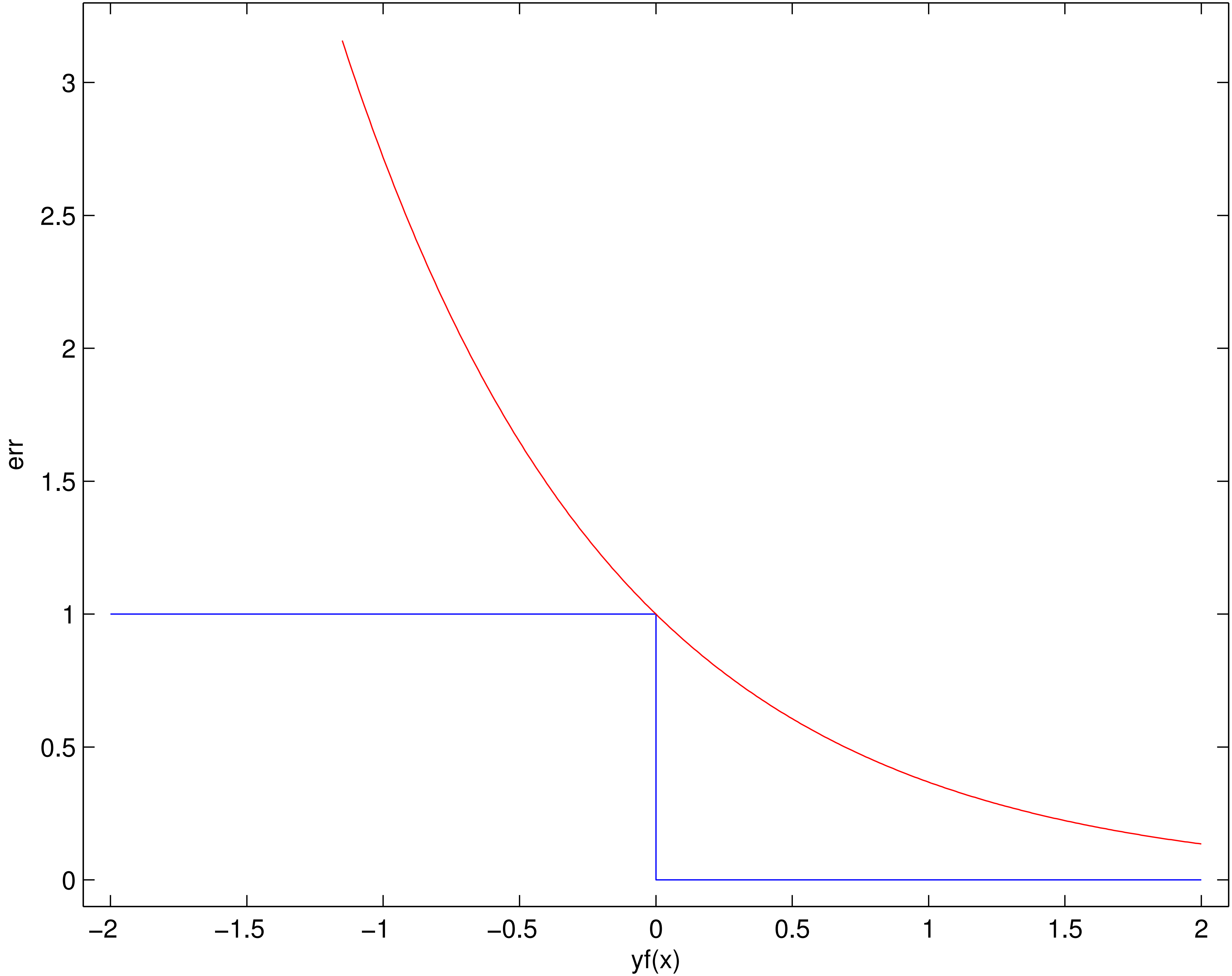




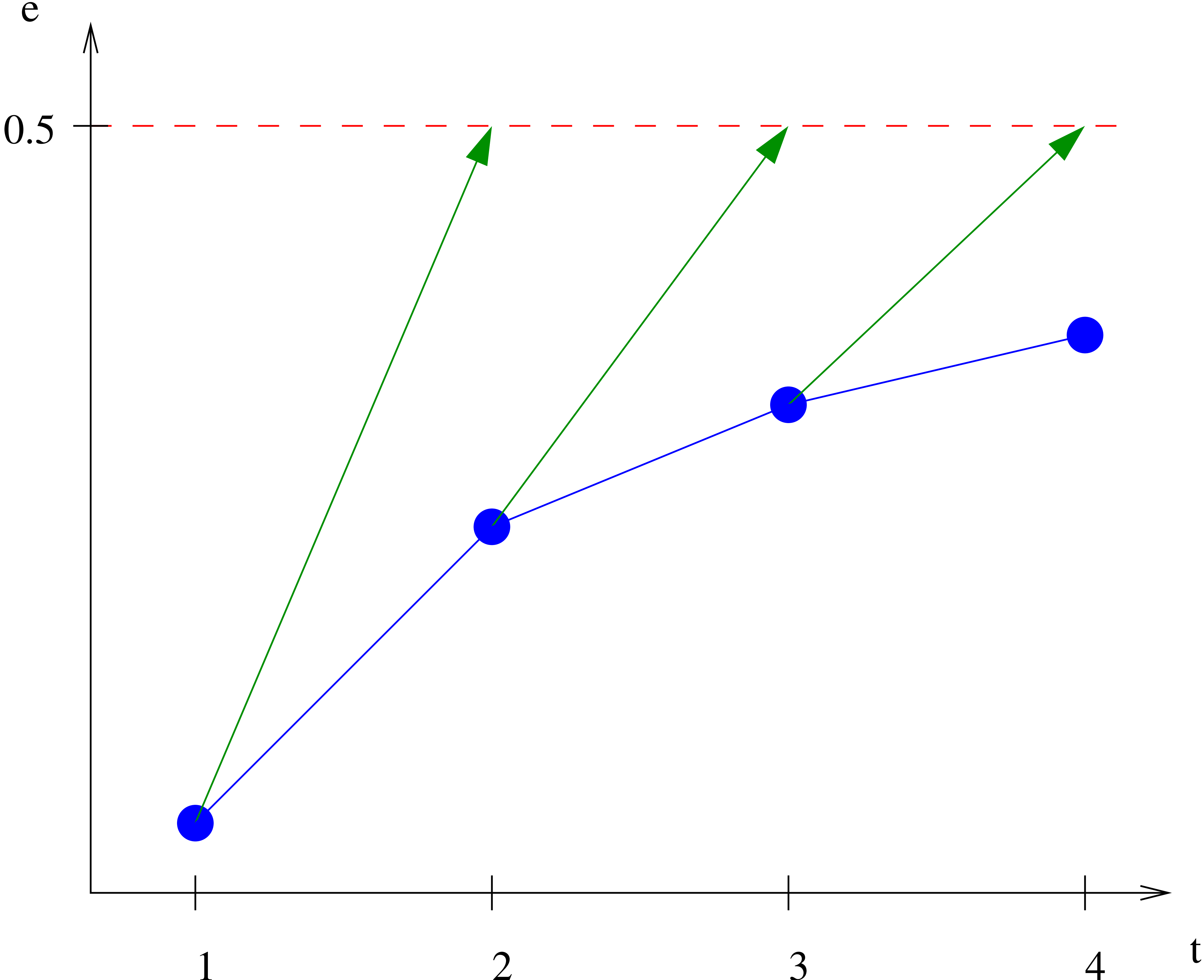


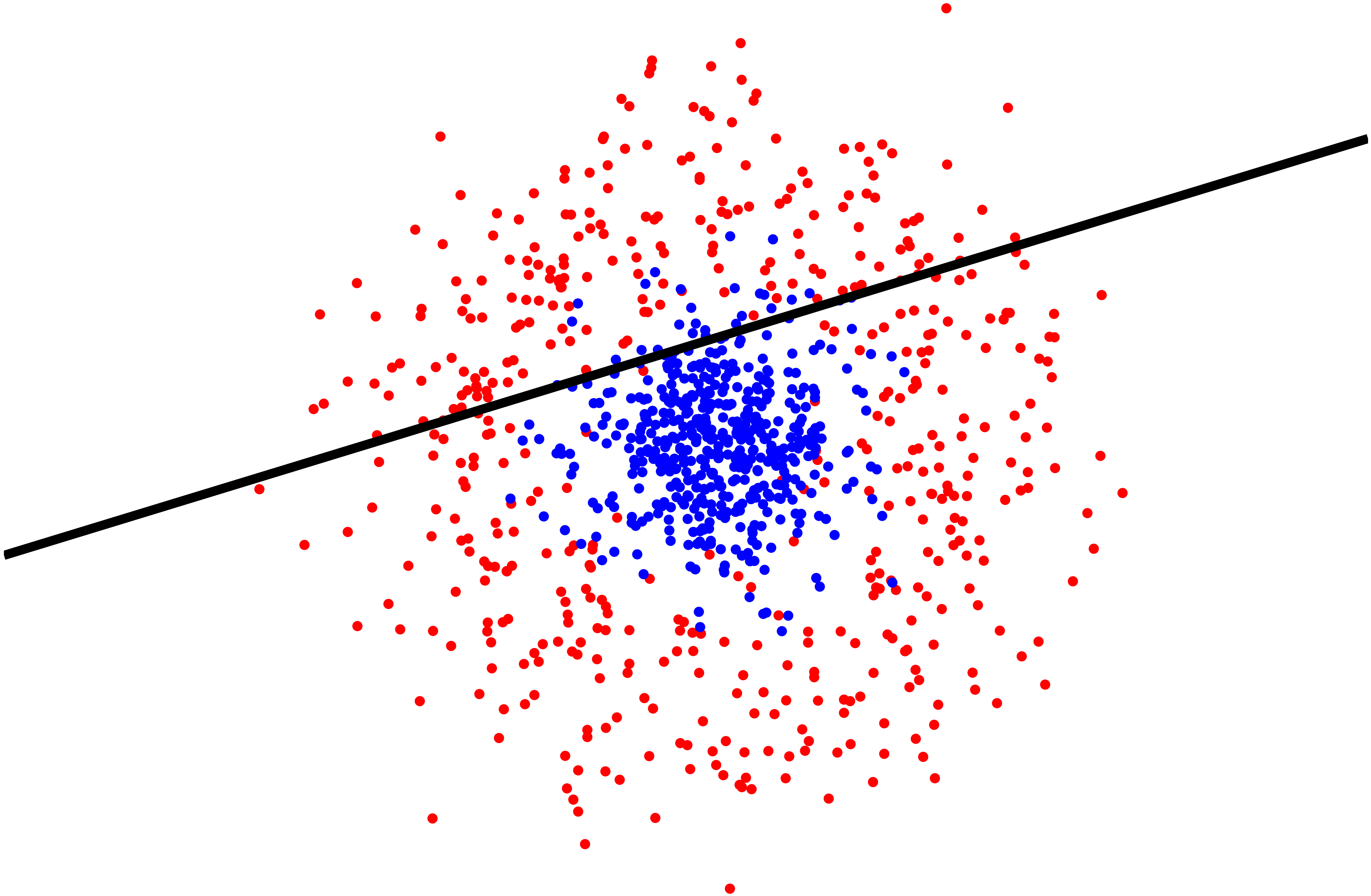


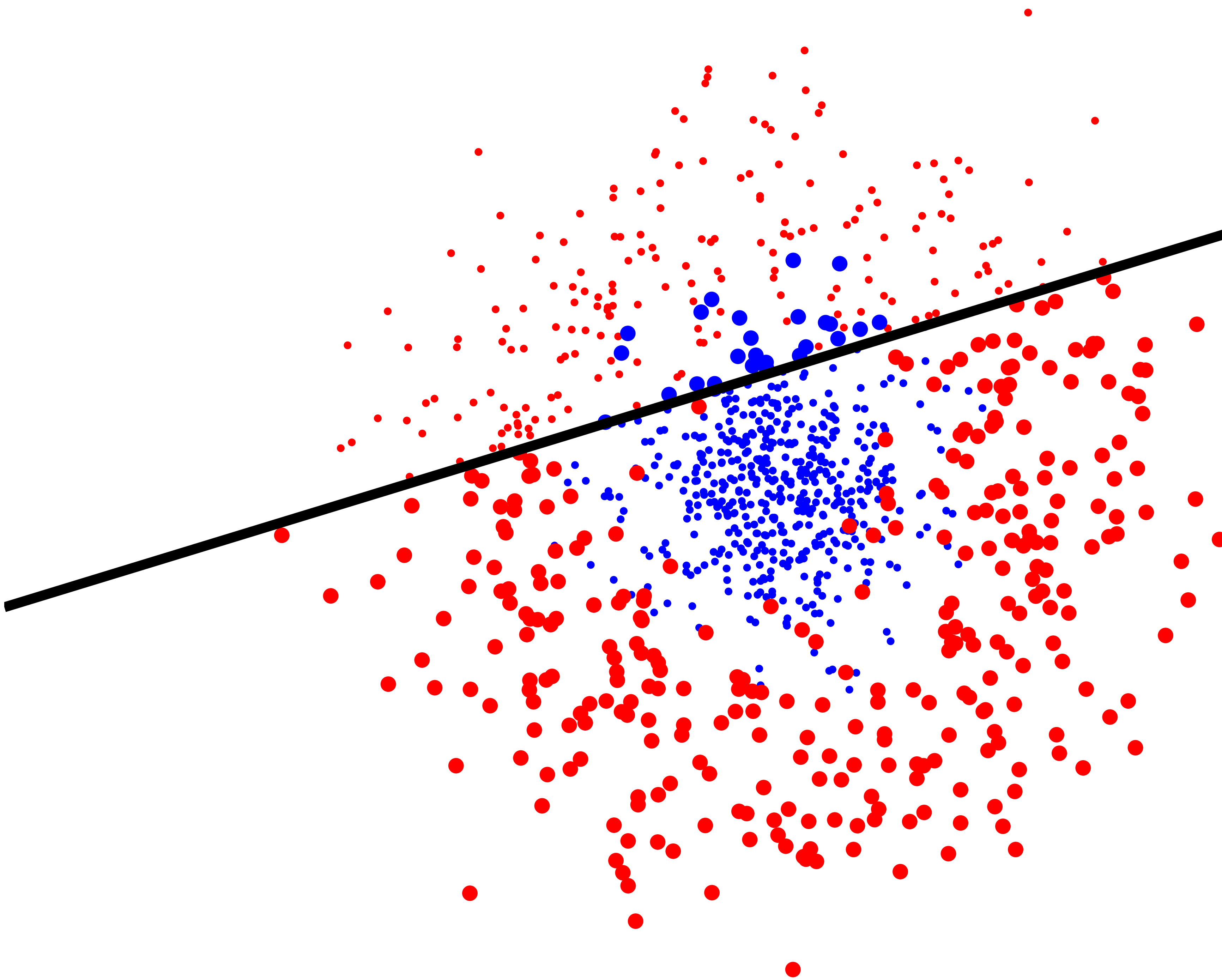


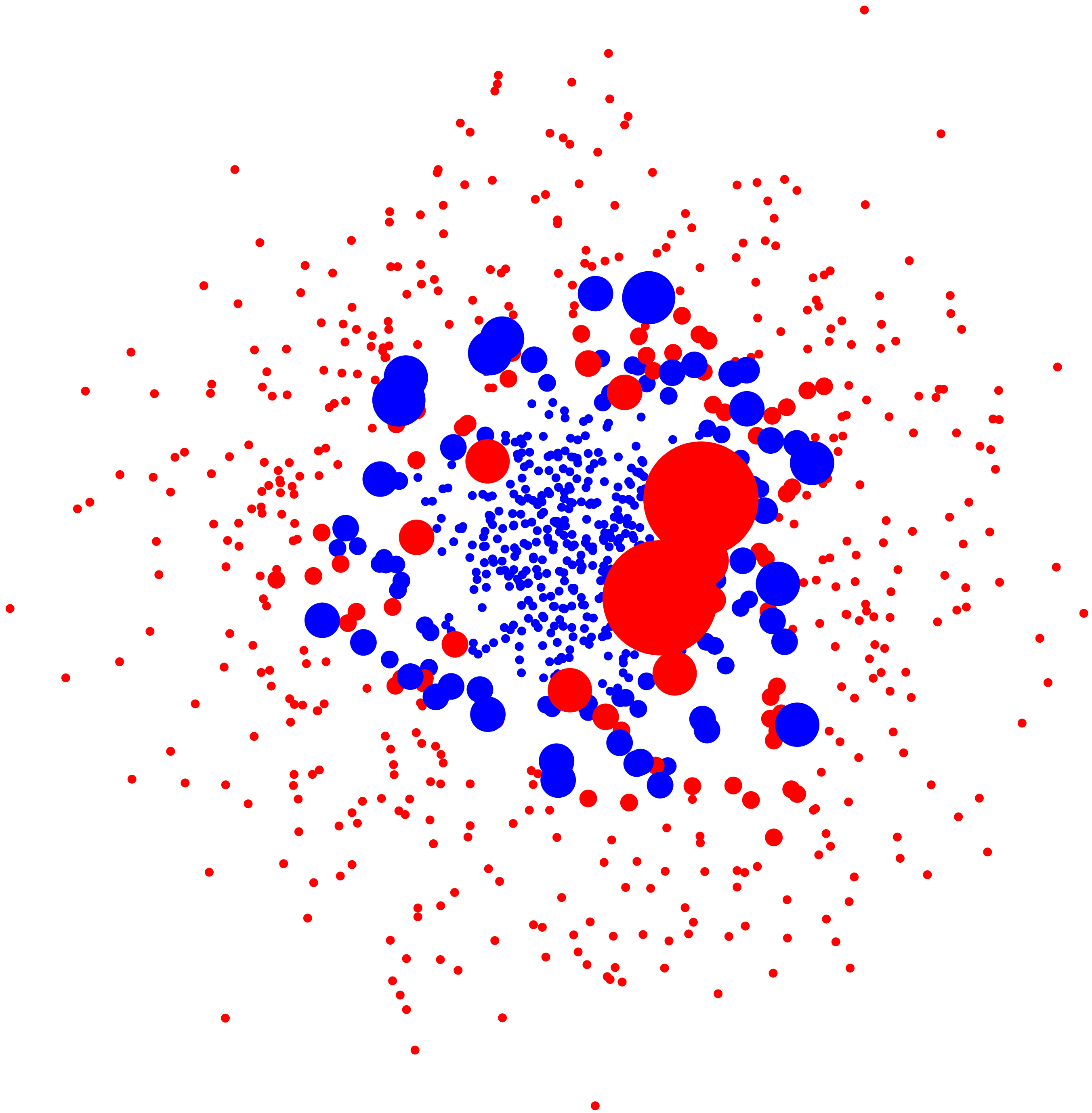


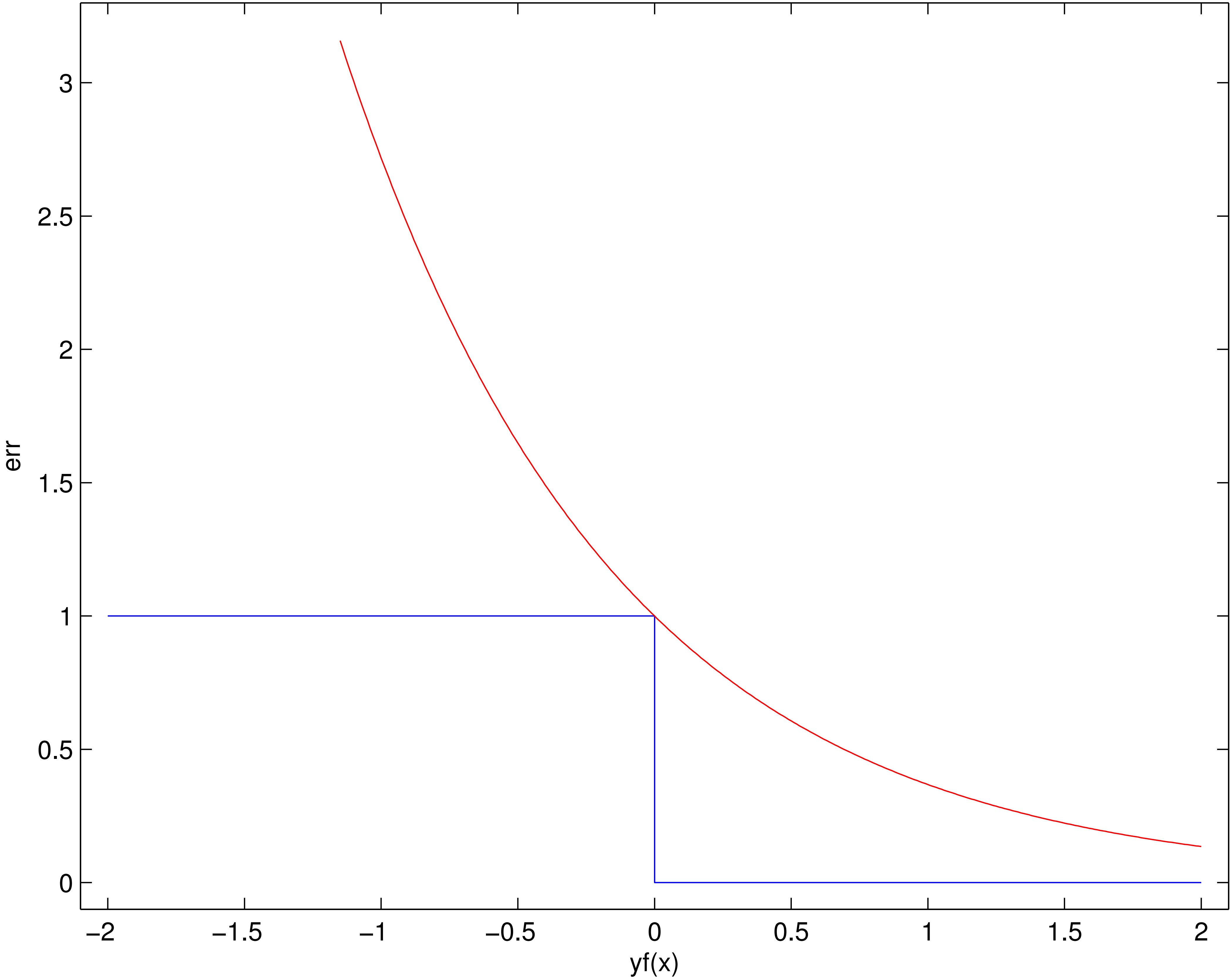
err
3
2.5
2
1.5
1
0.5
0
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
yf(x)


e
0.5
t
1
2
3
4

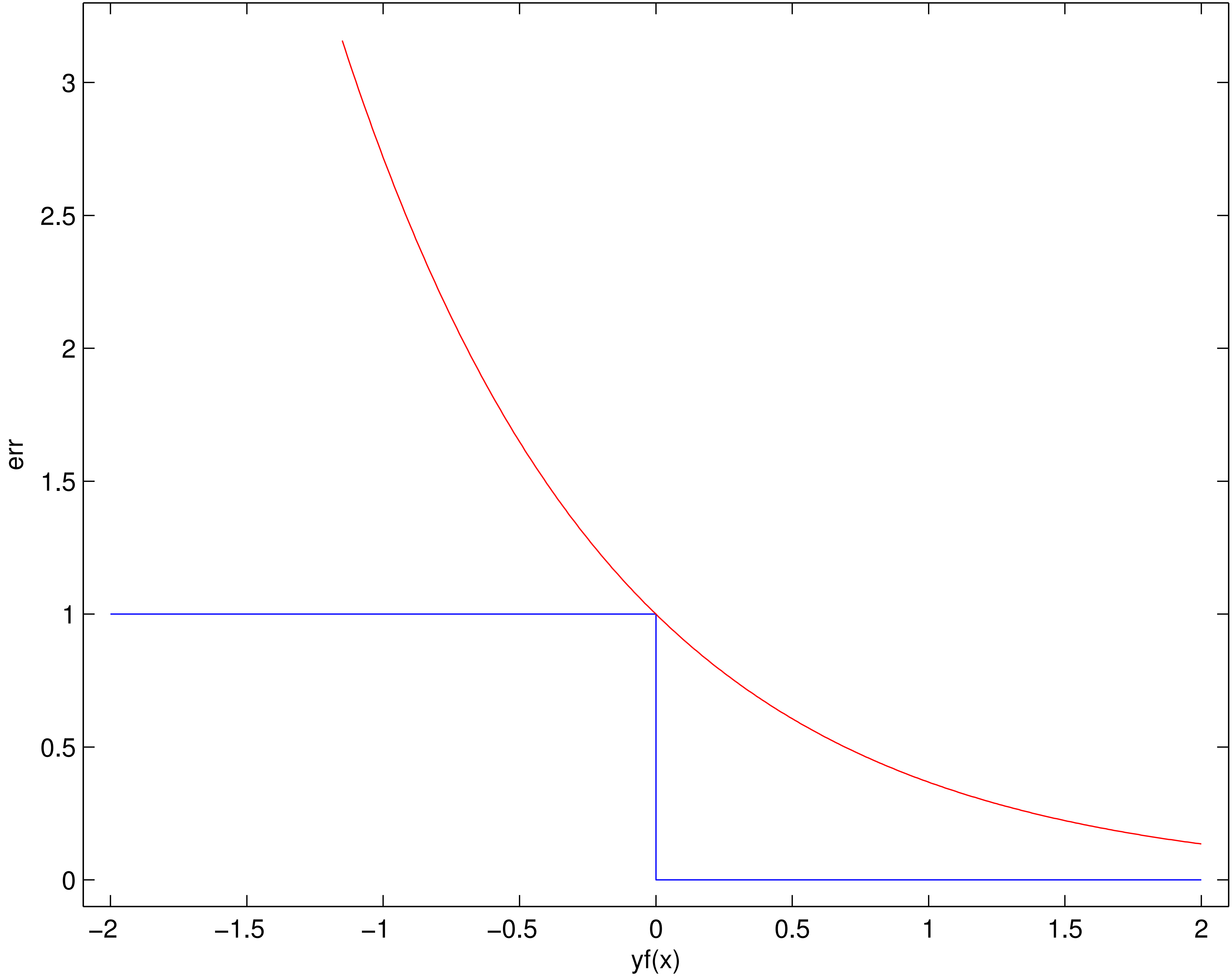




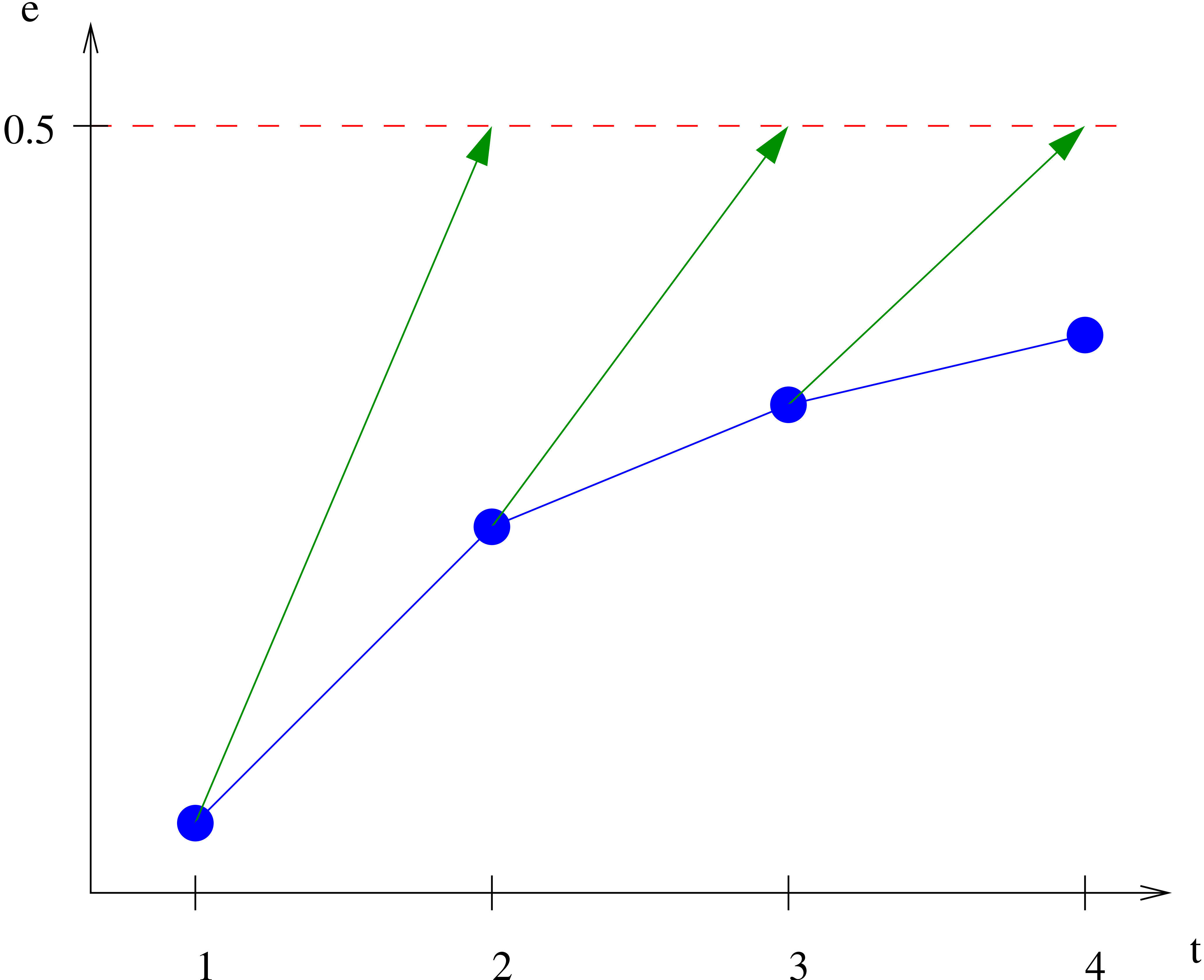


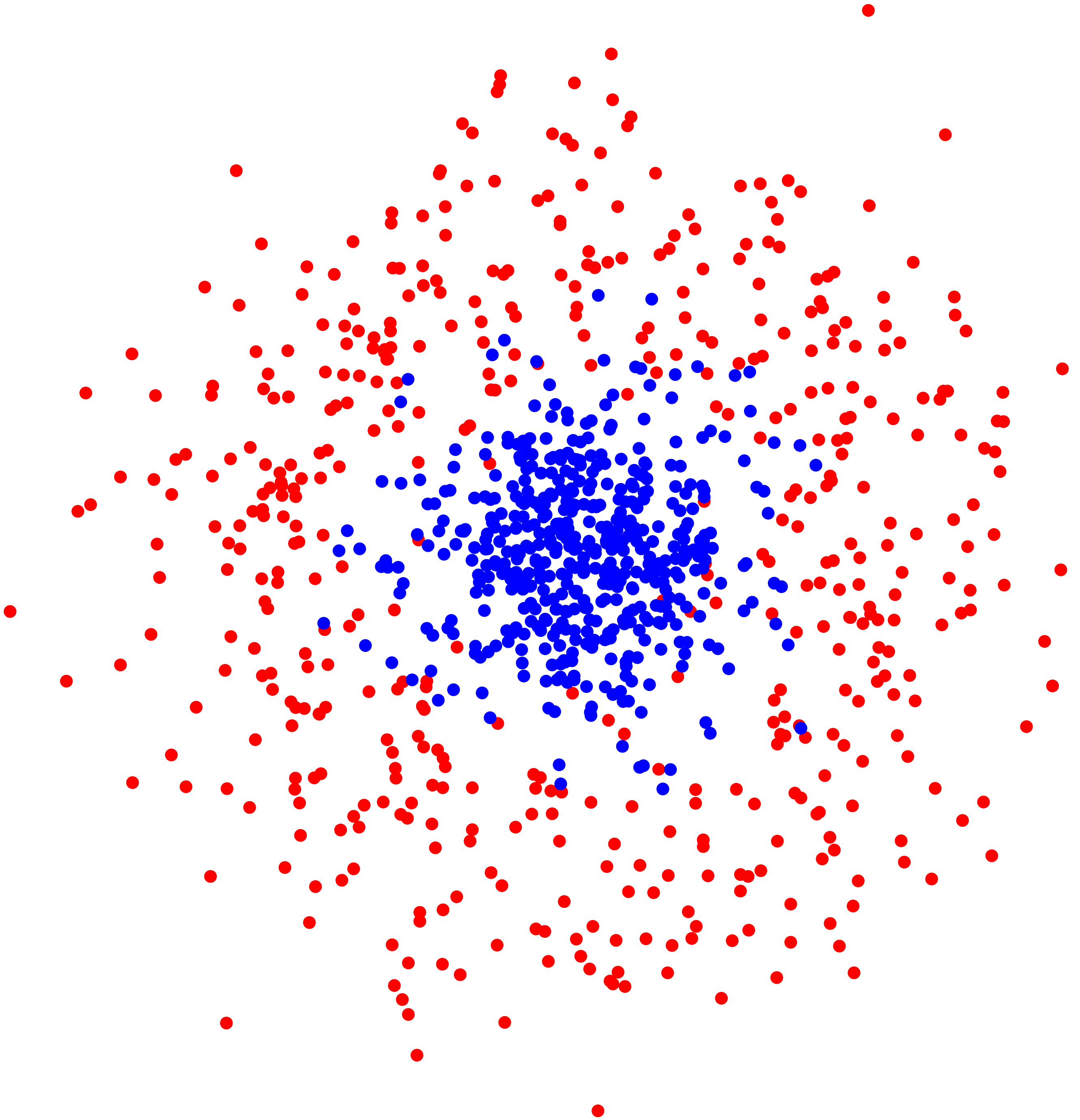


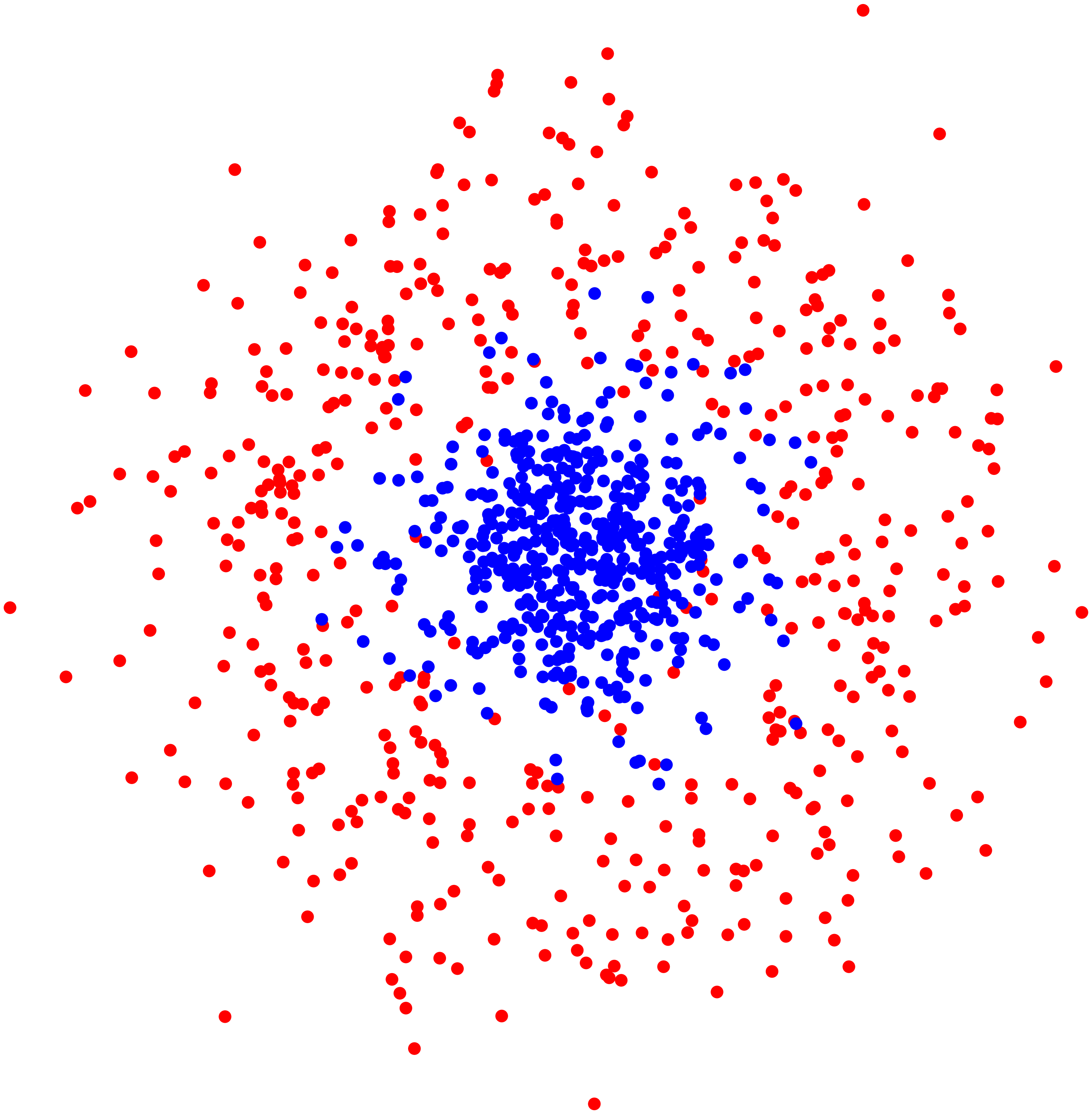
err
yf(x)
3
2.5
2
1.5
1
0.5
0
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2

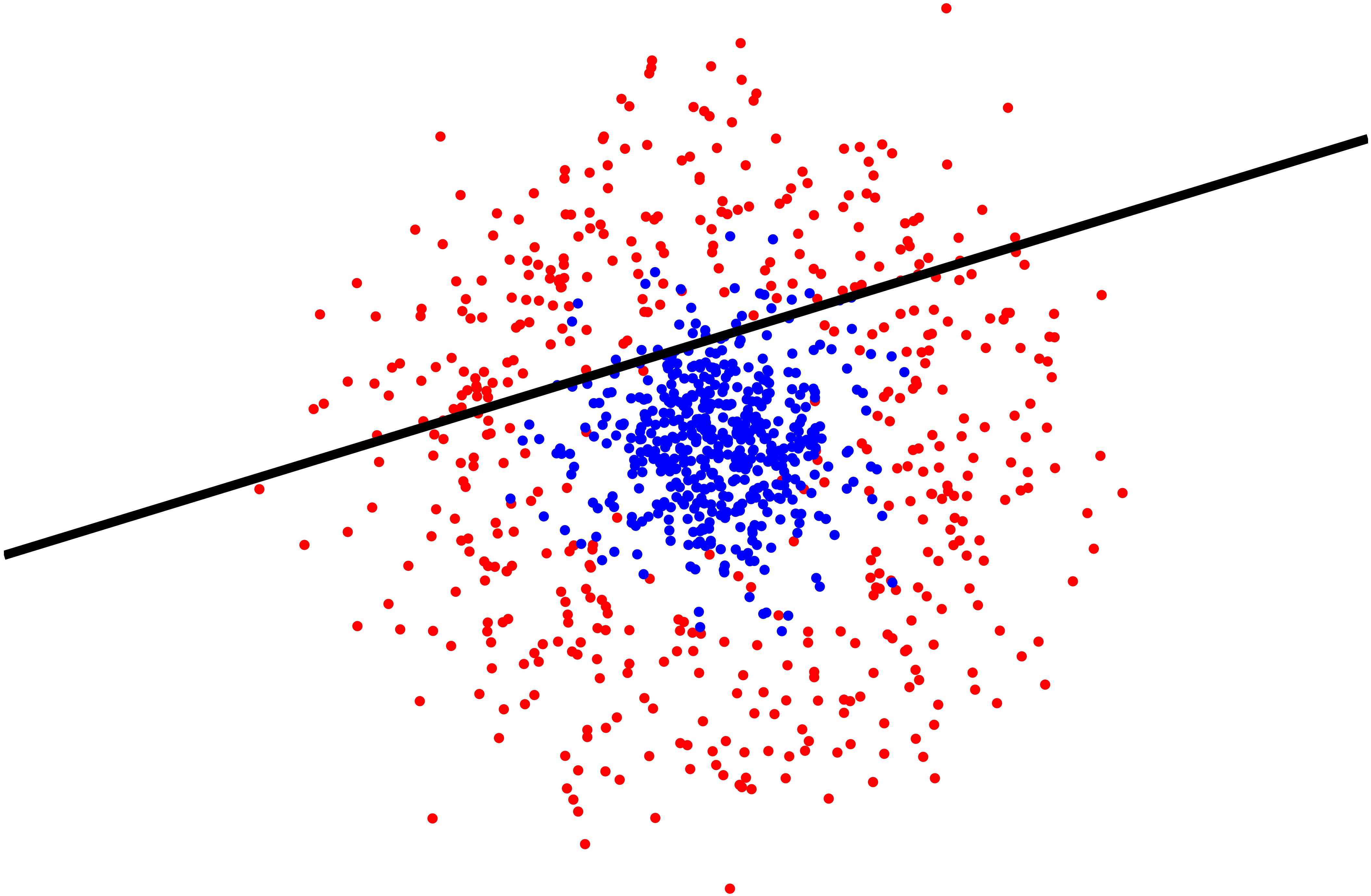


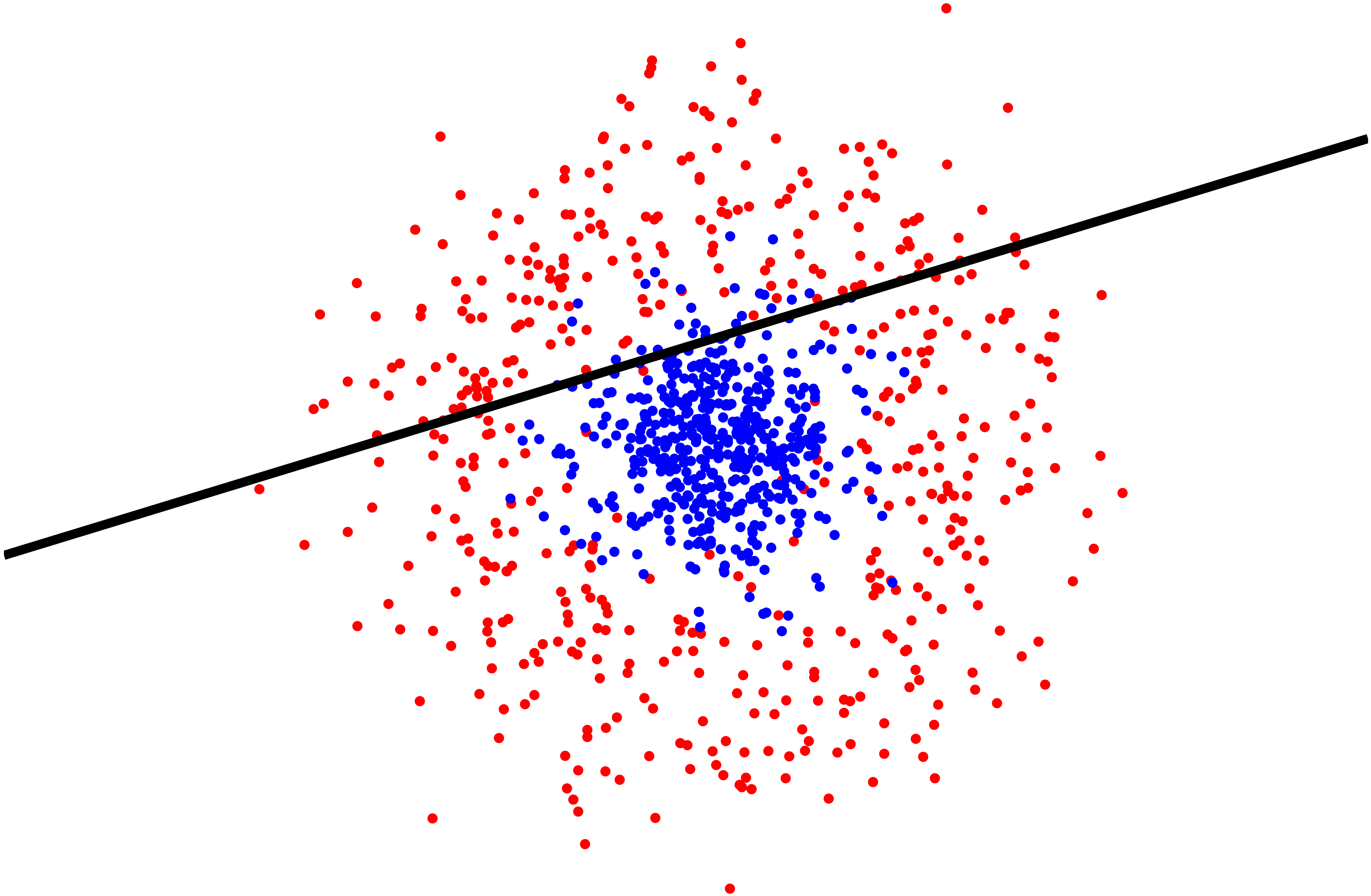
3
2.5
2
1.5
1
0.5
0
err
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
yf(x)

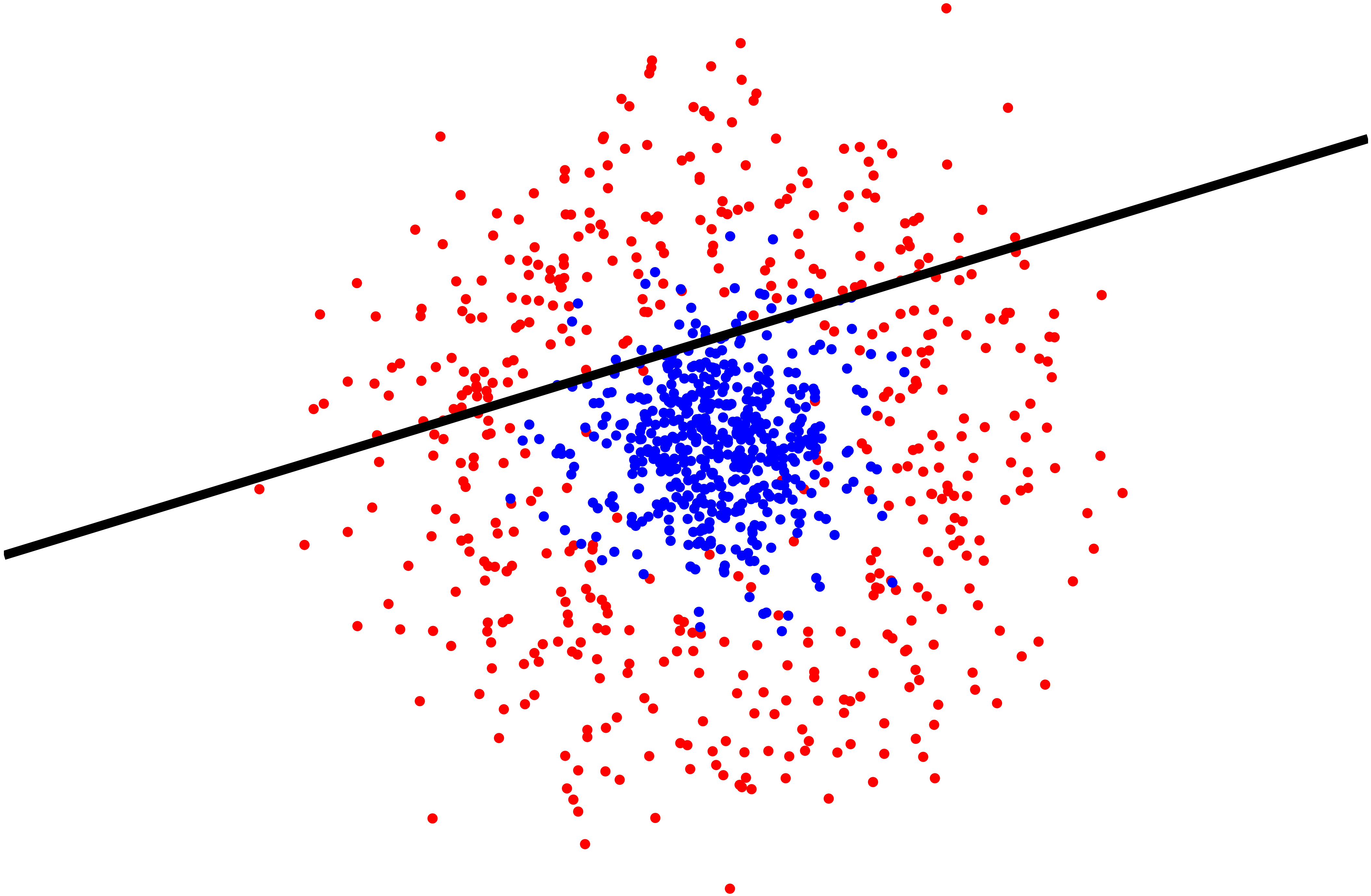

e
0.5
t
1
2
3
4

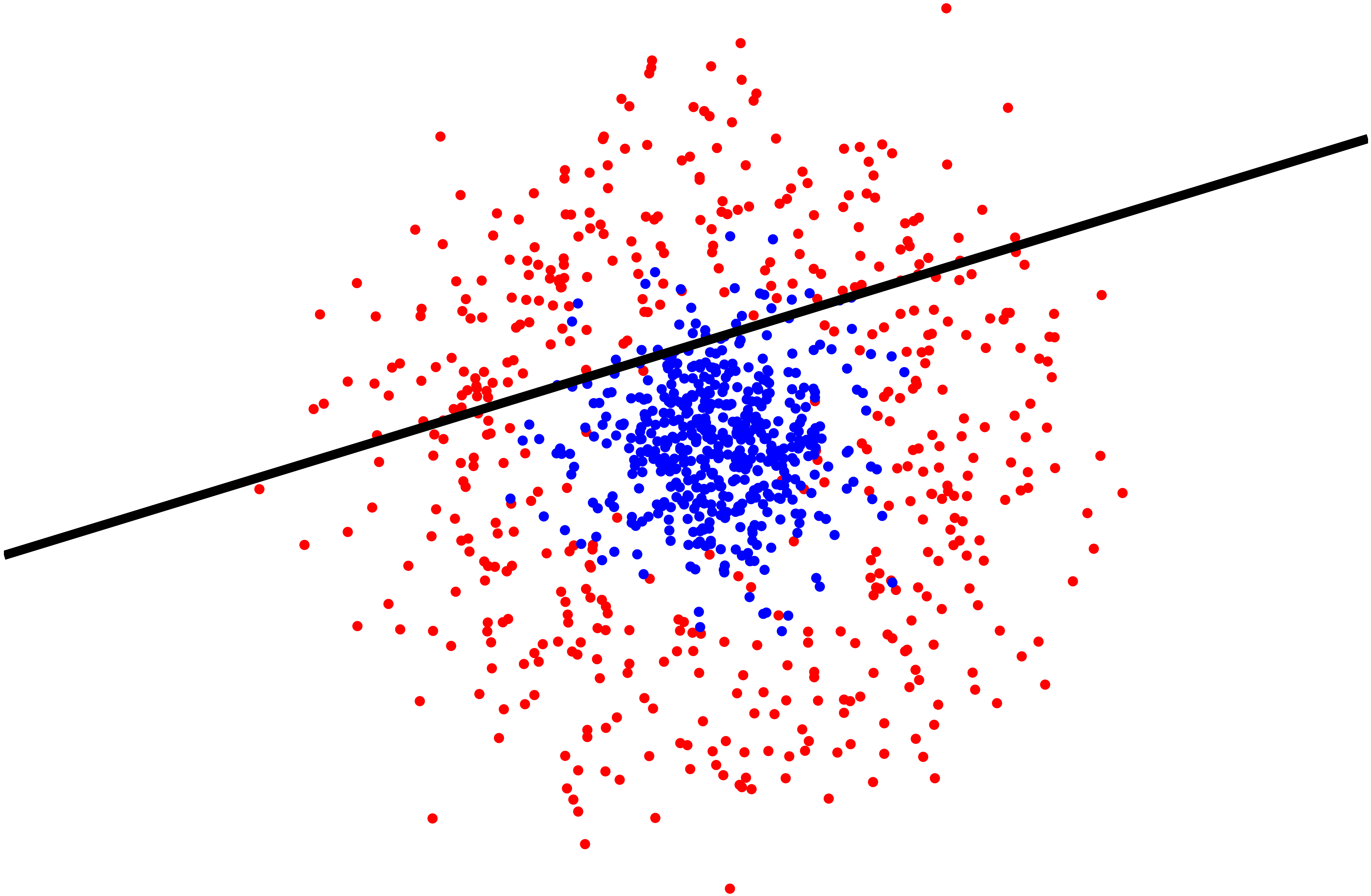


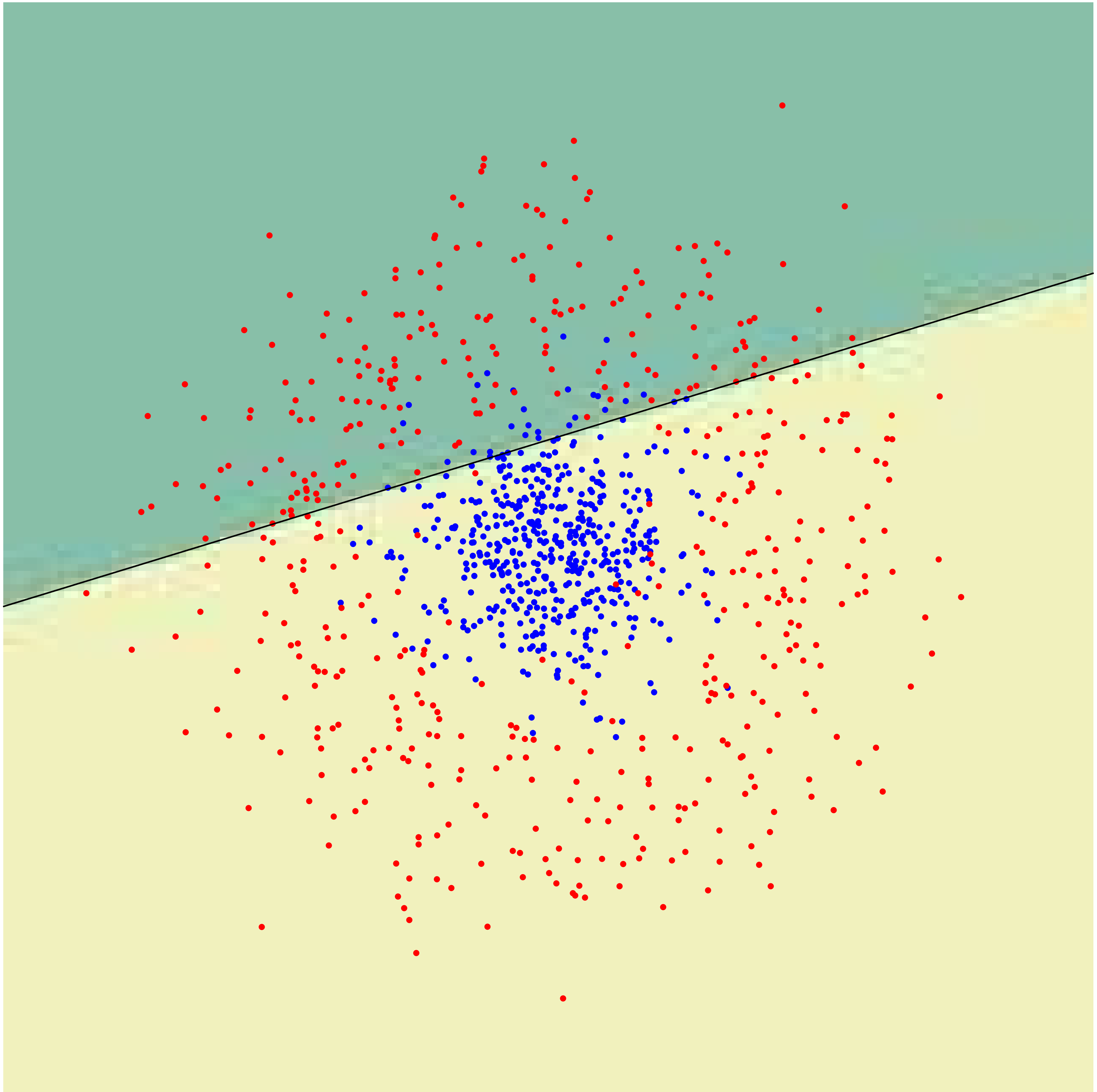


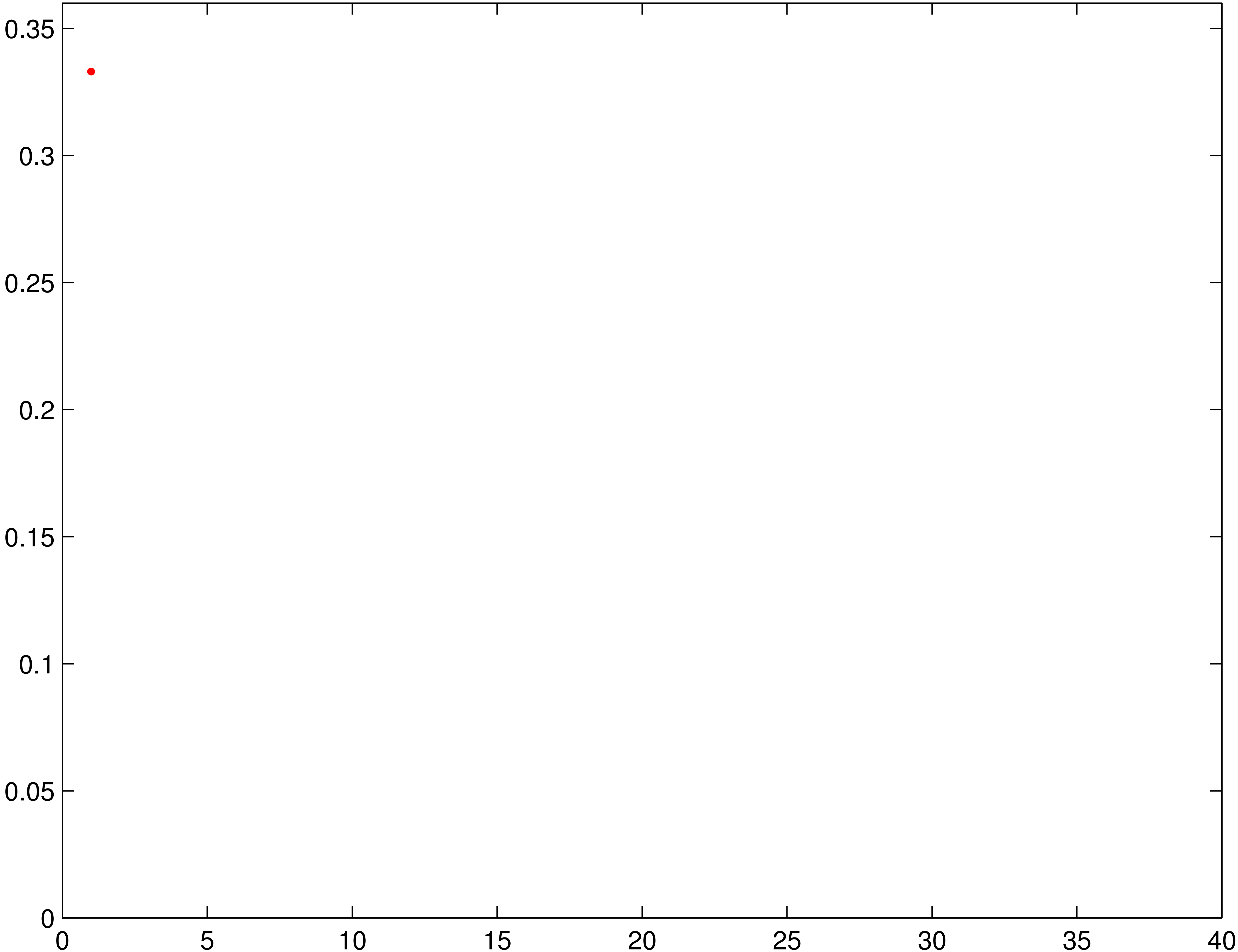


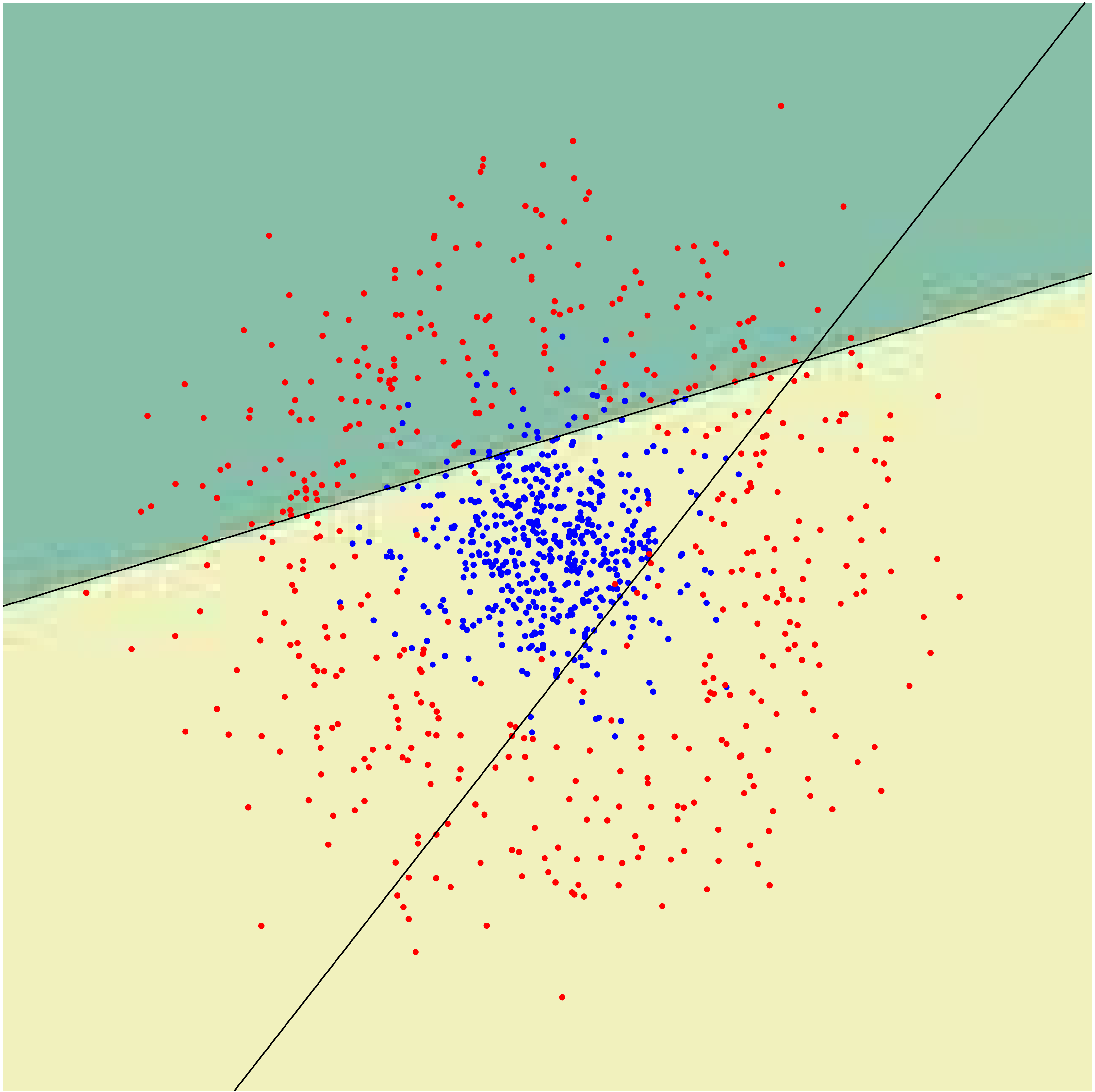


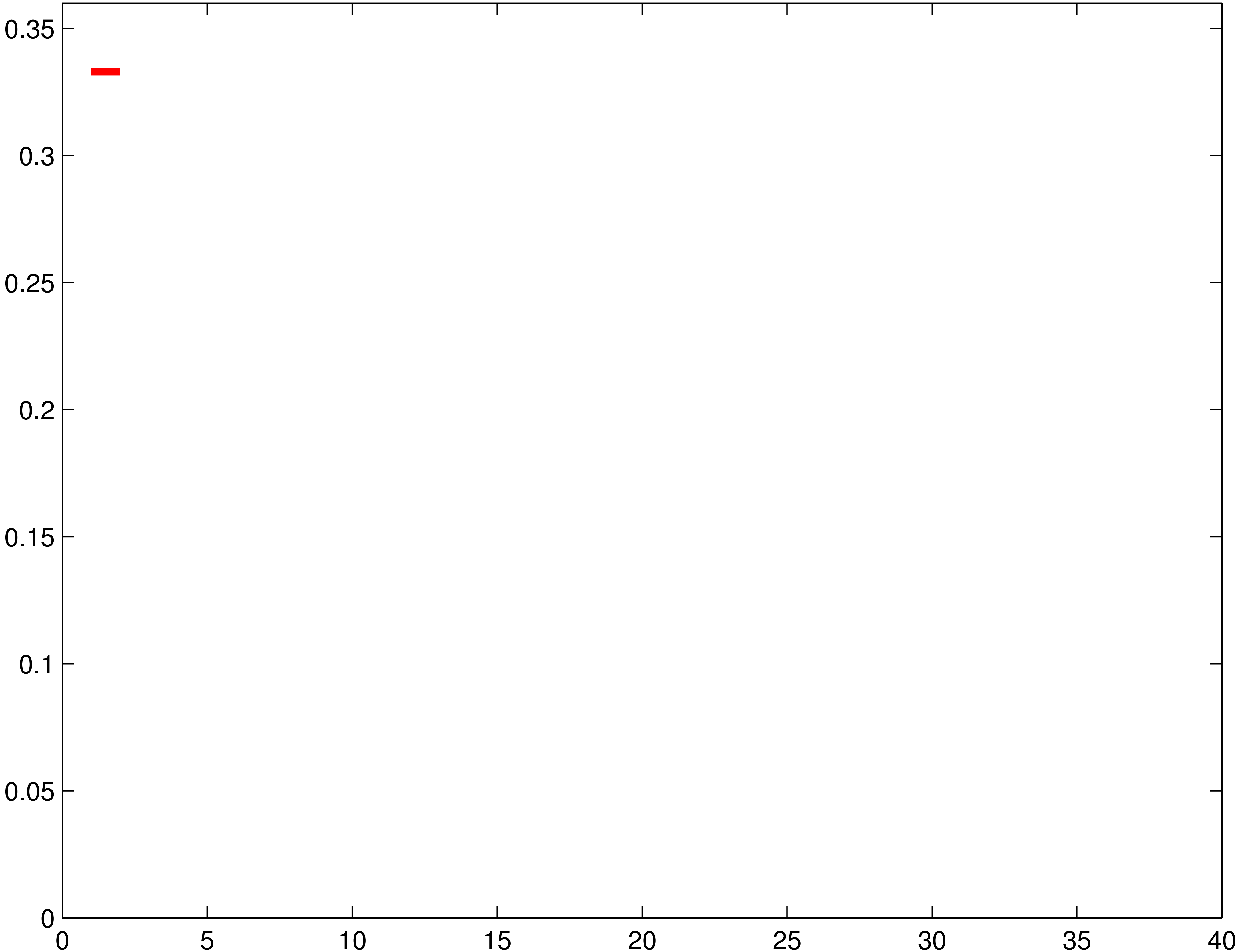


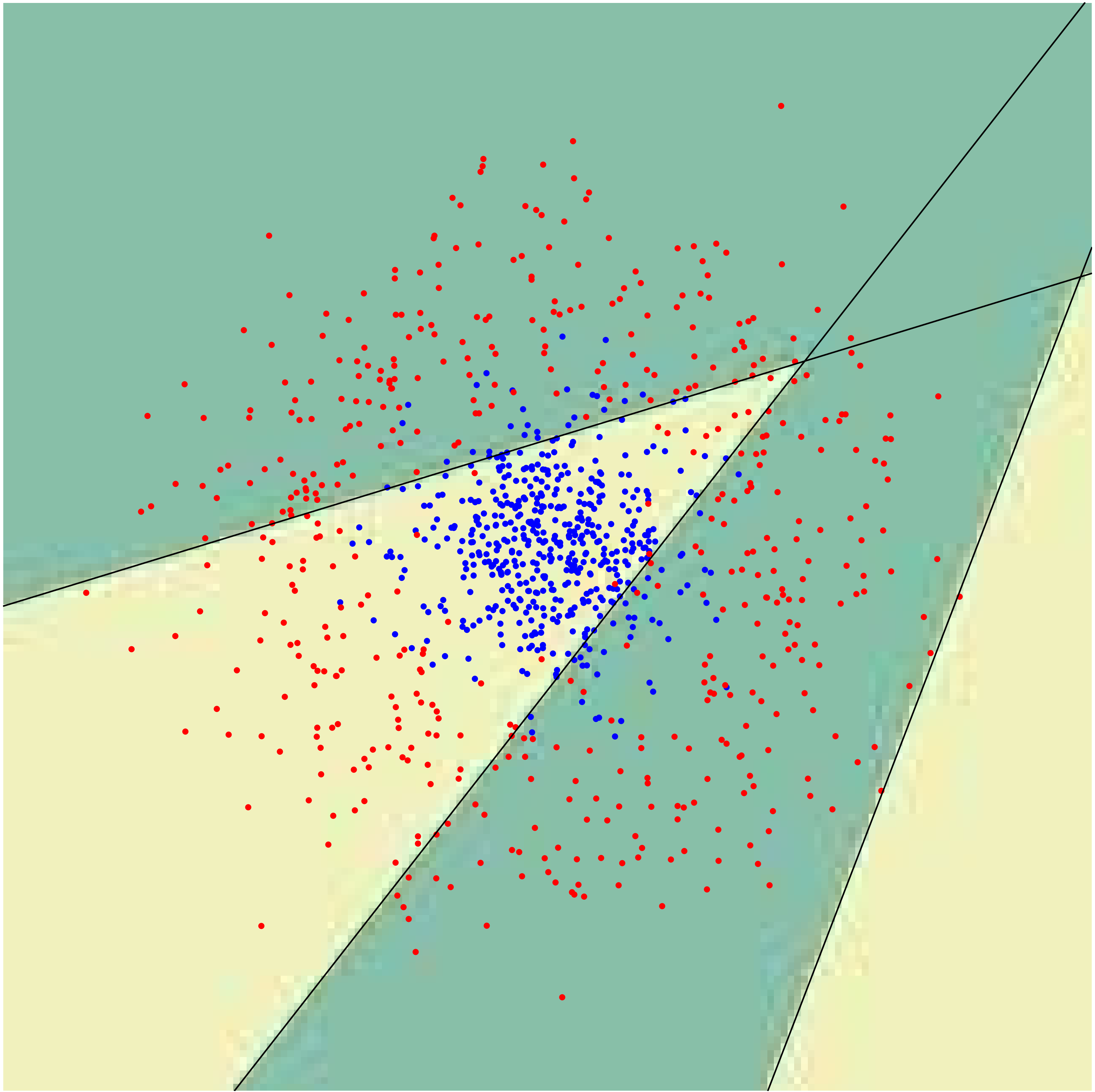


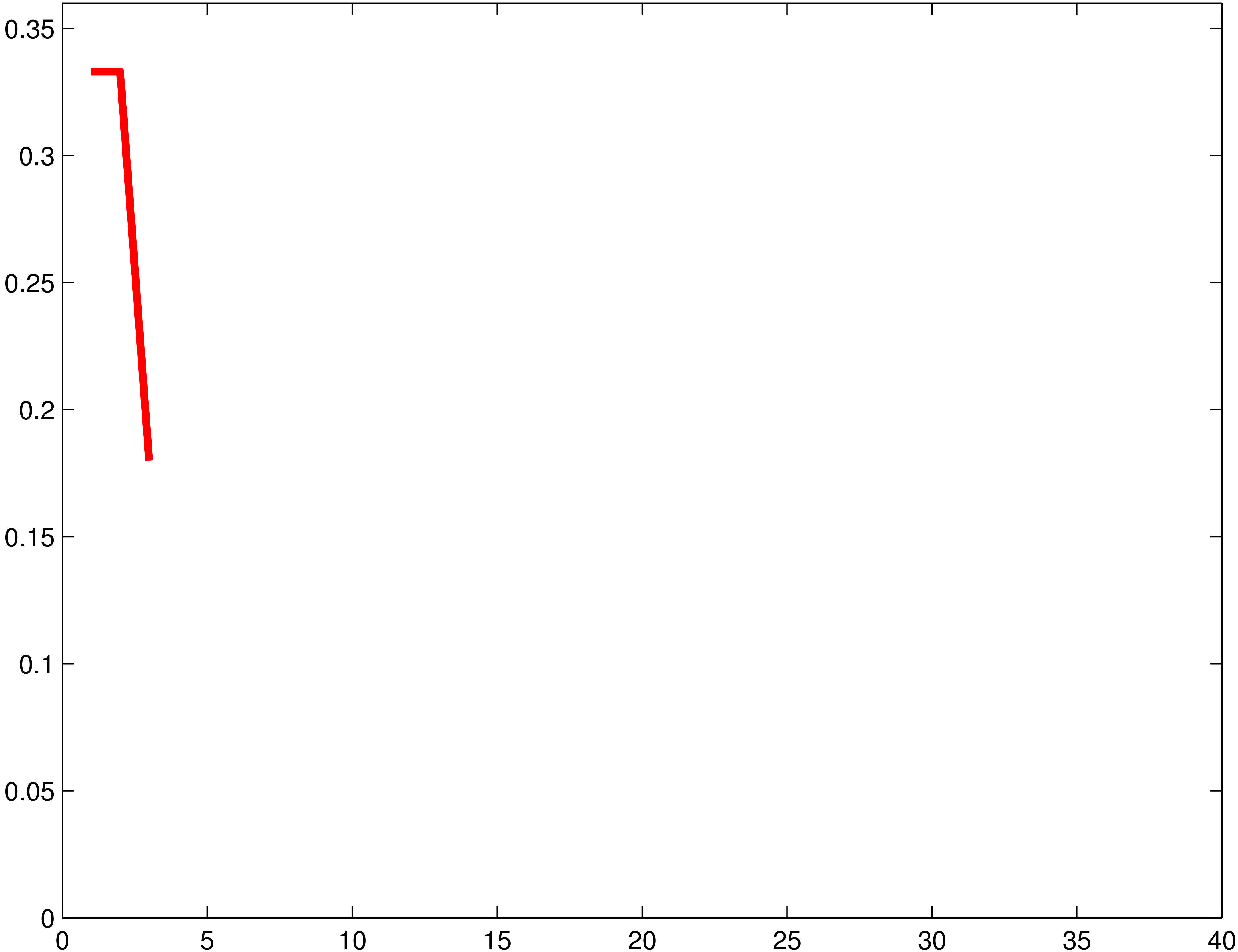


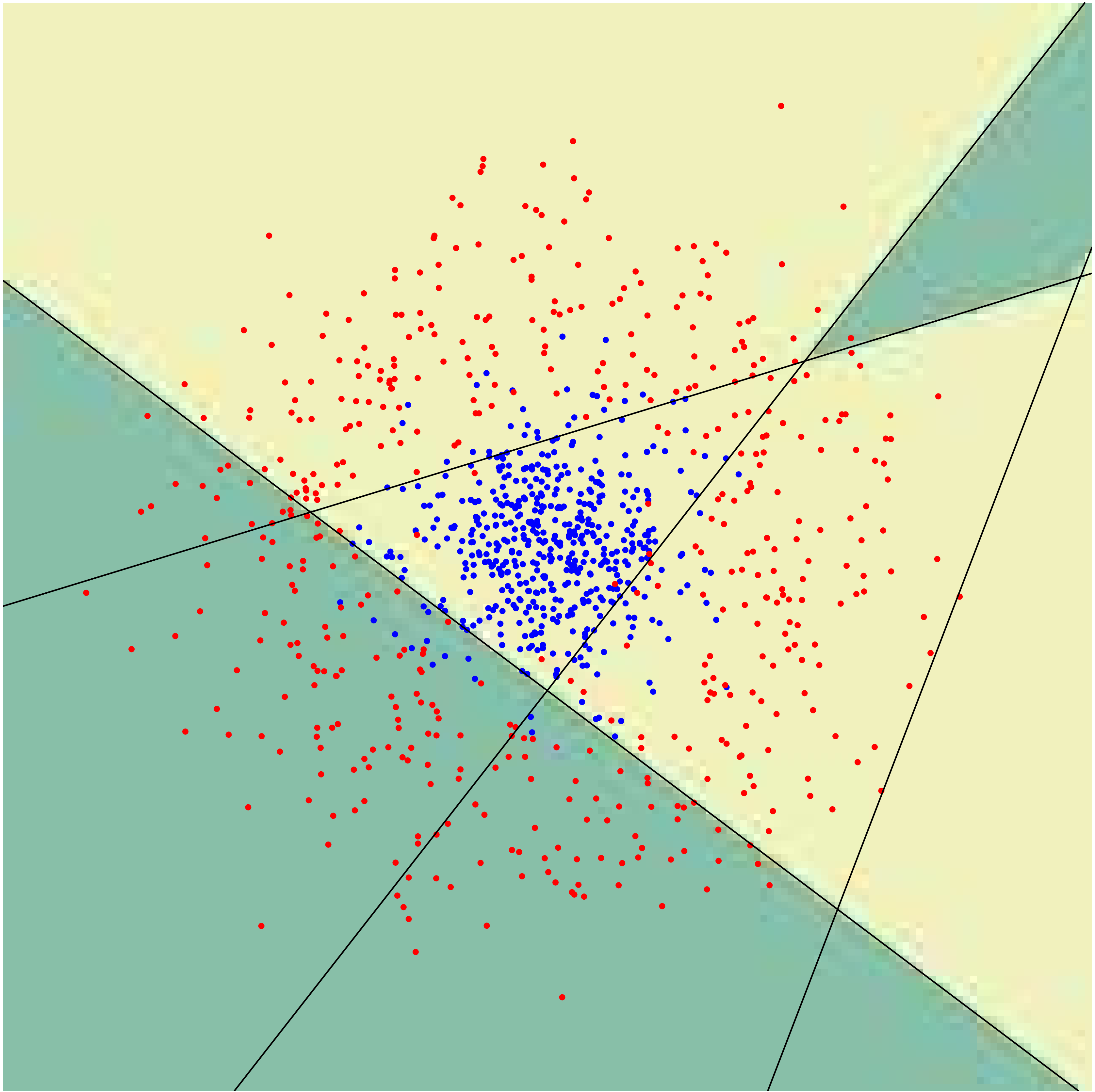

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

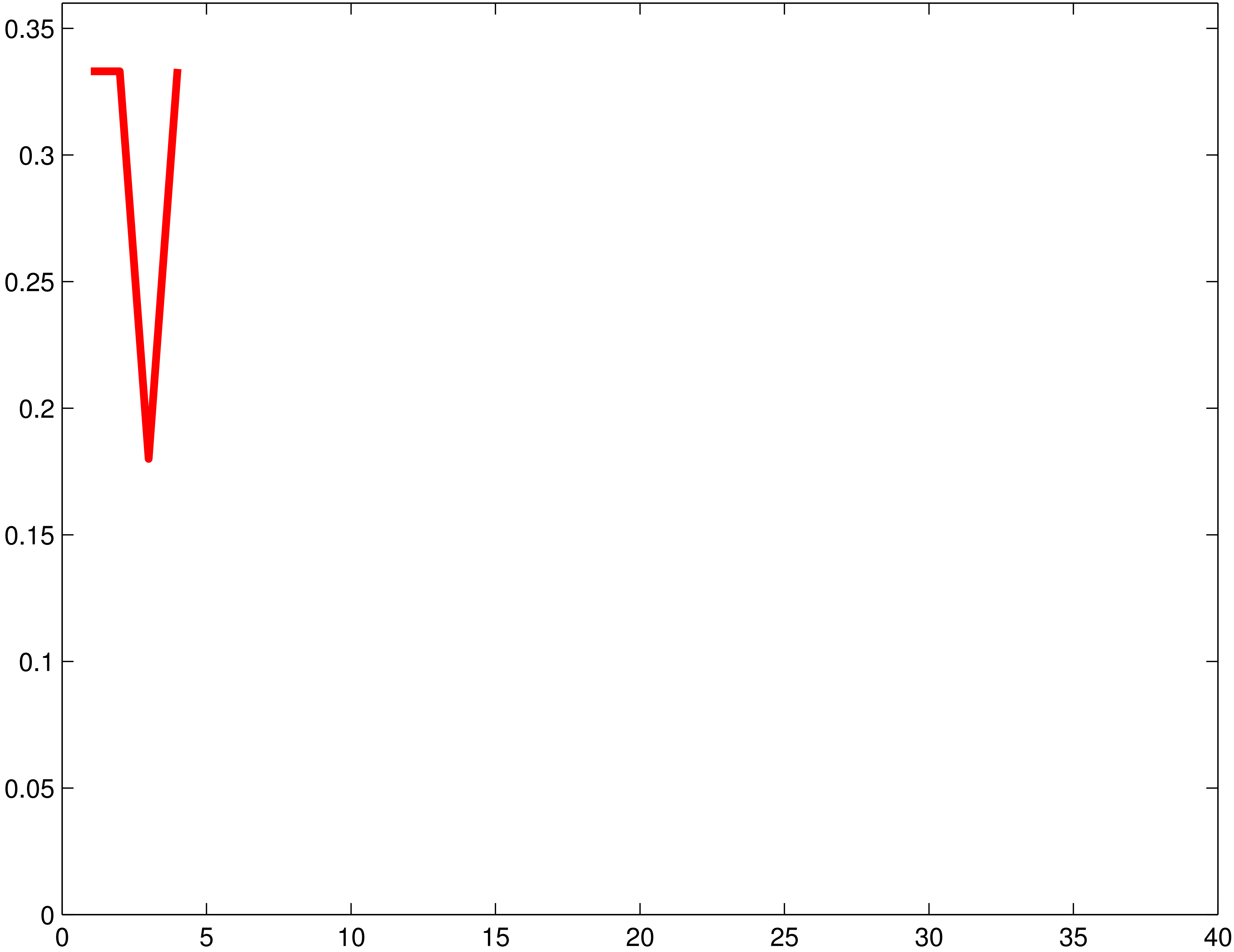



0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

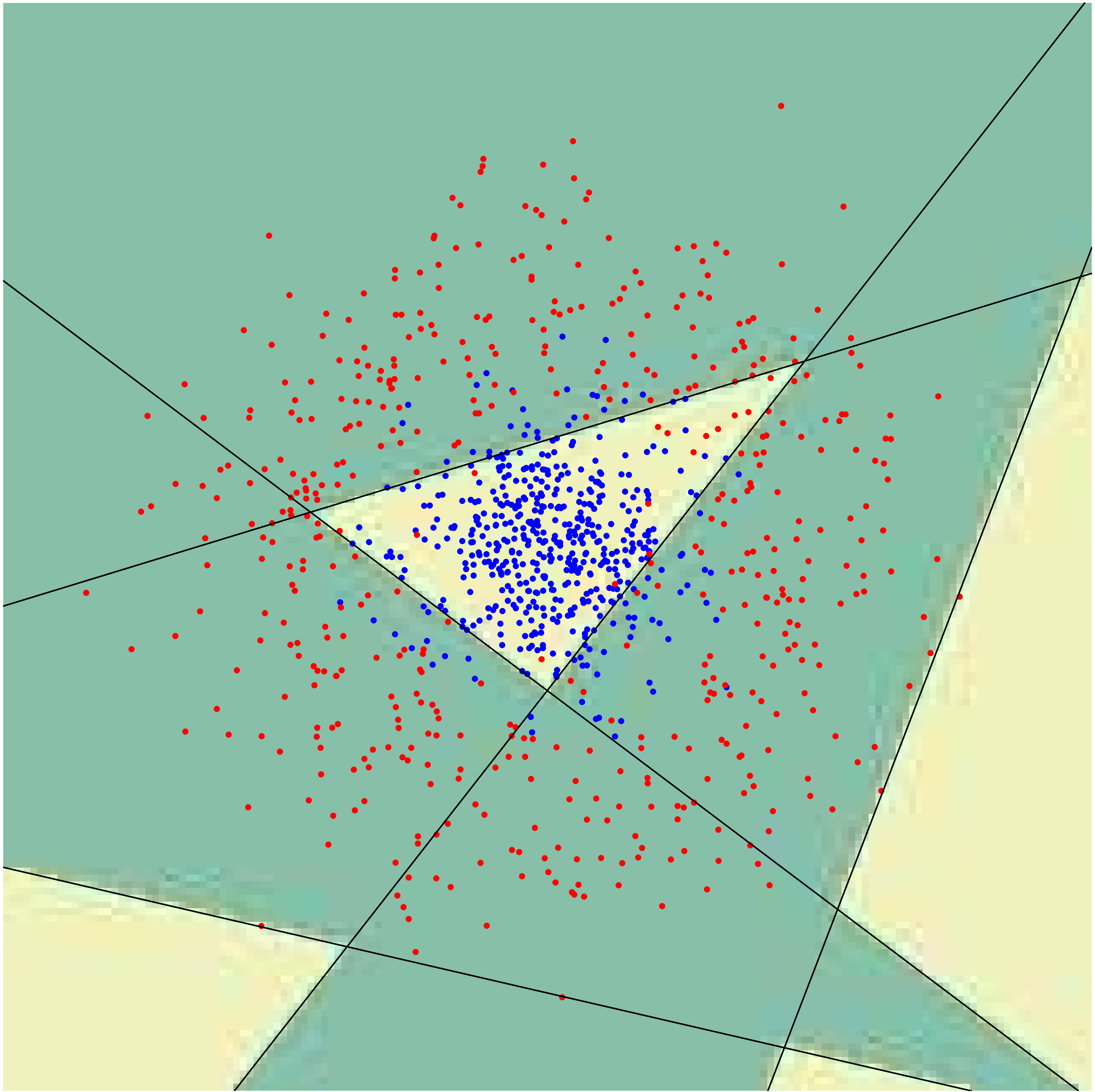


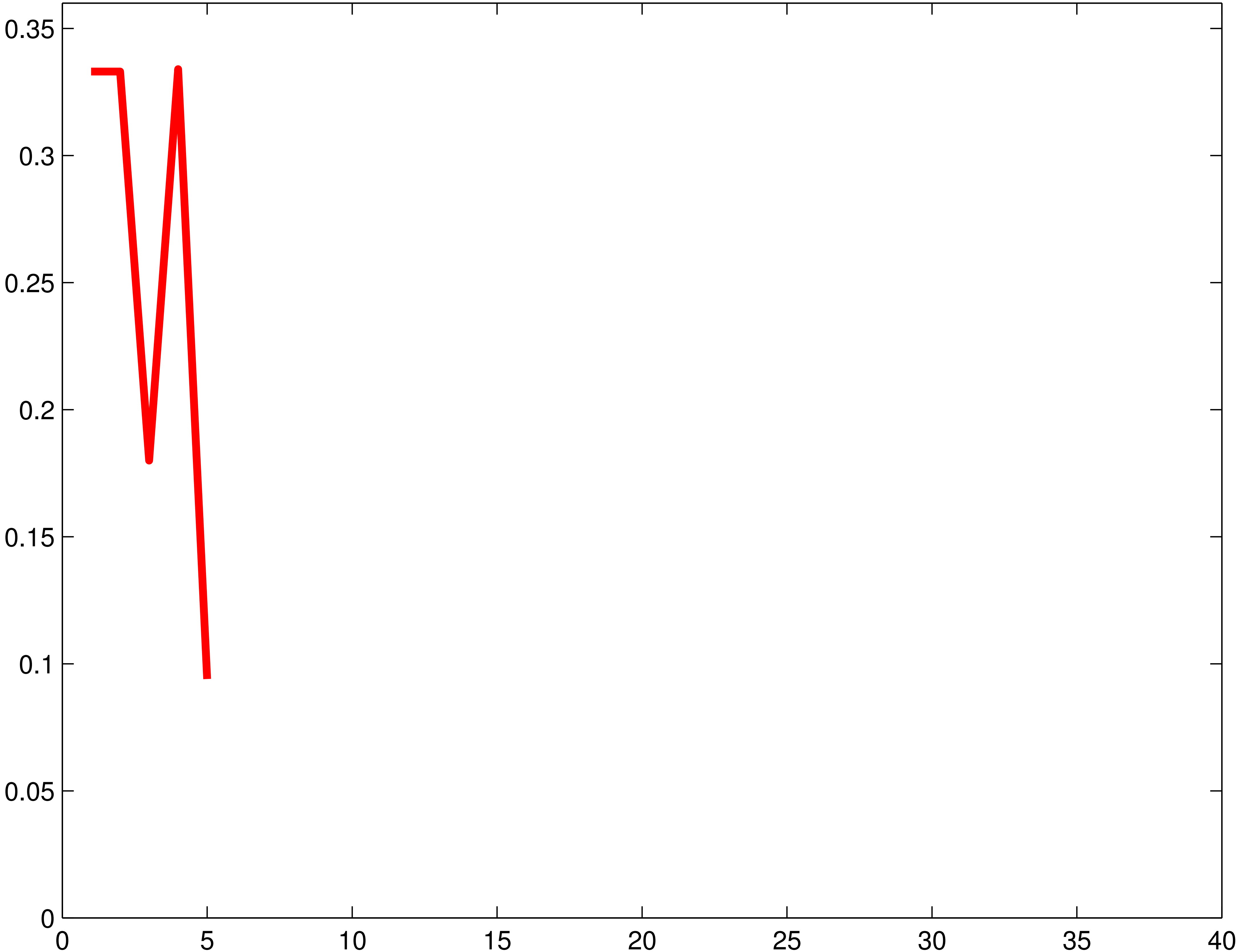

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

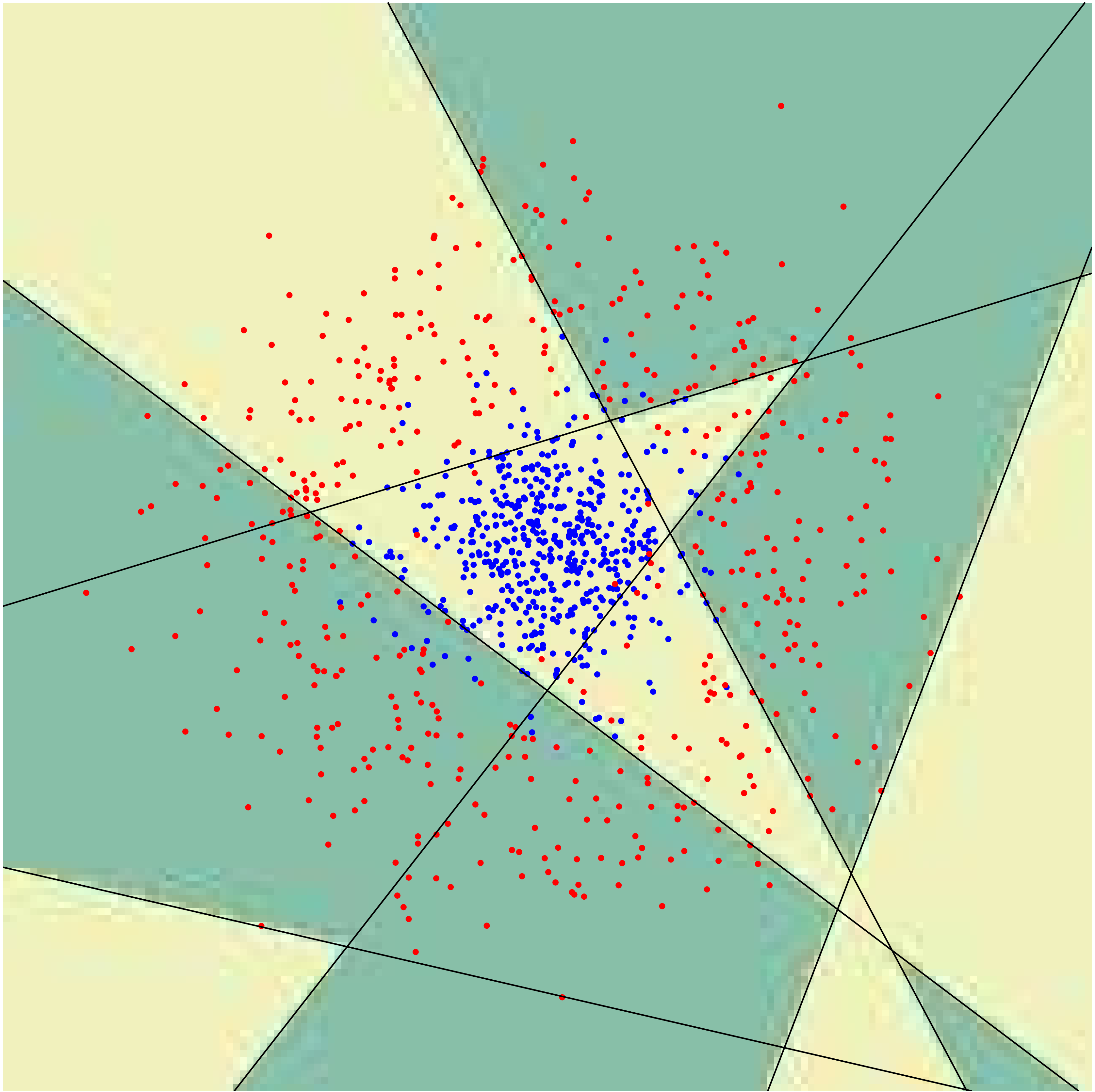


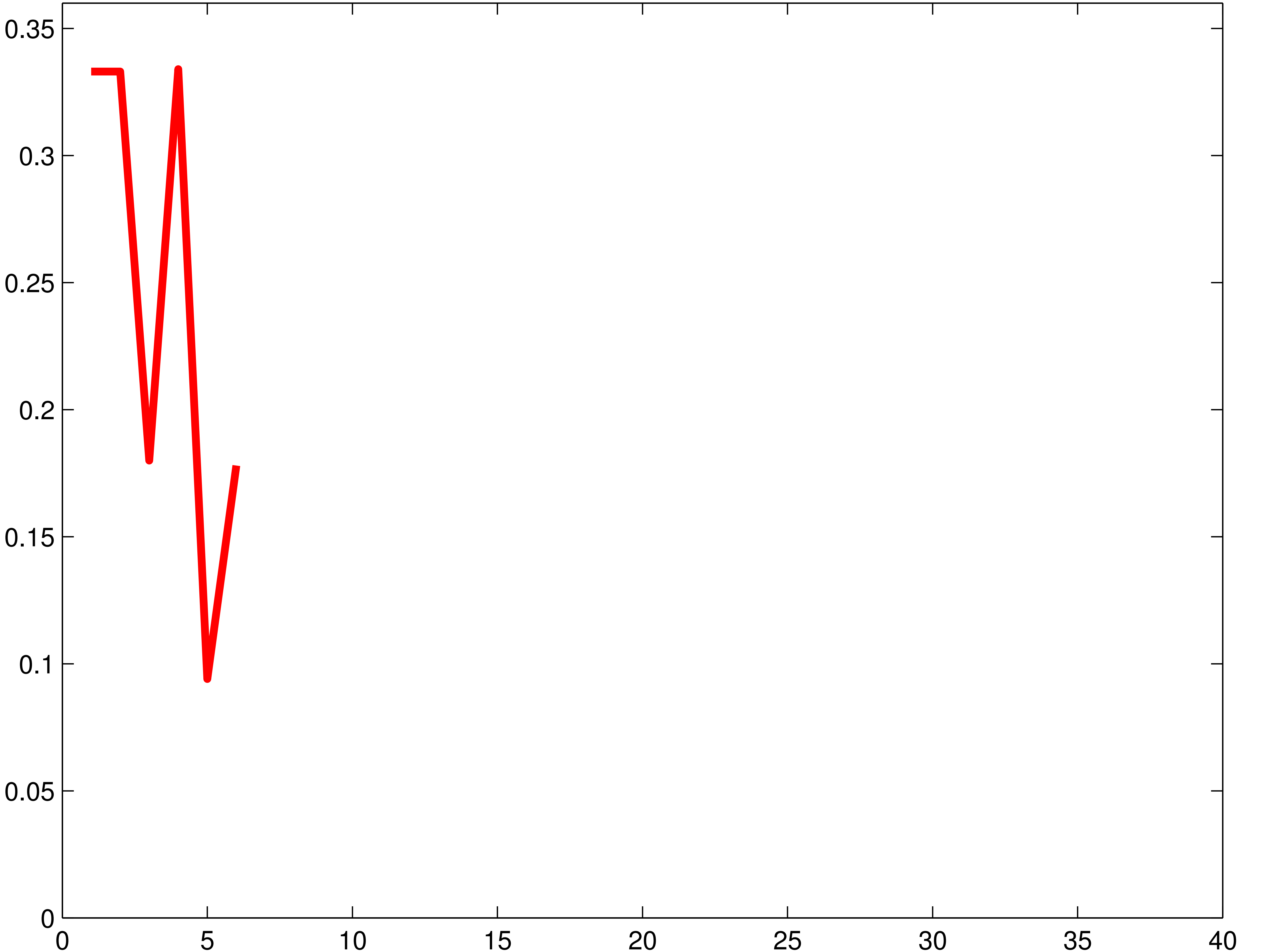


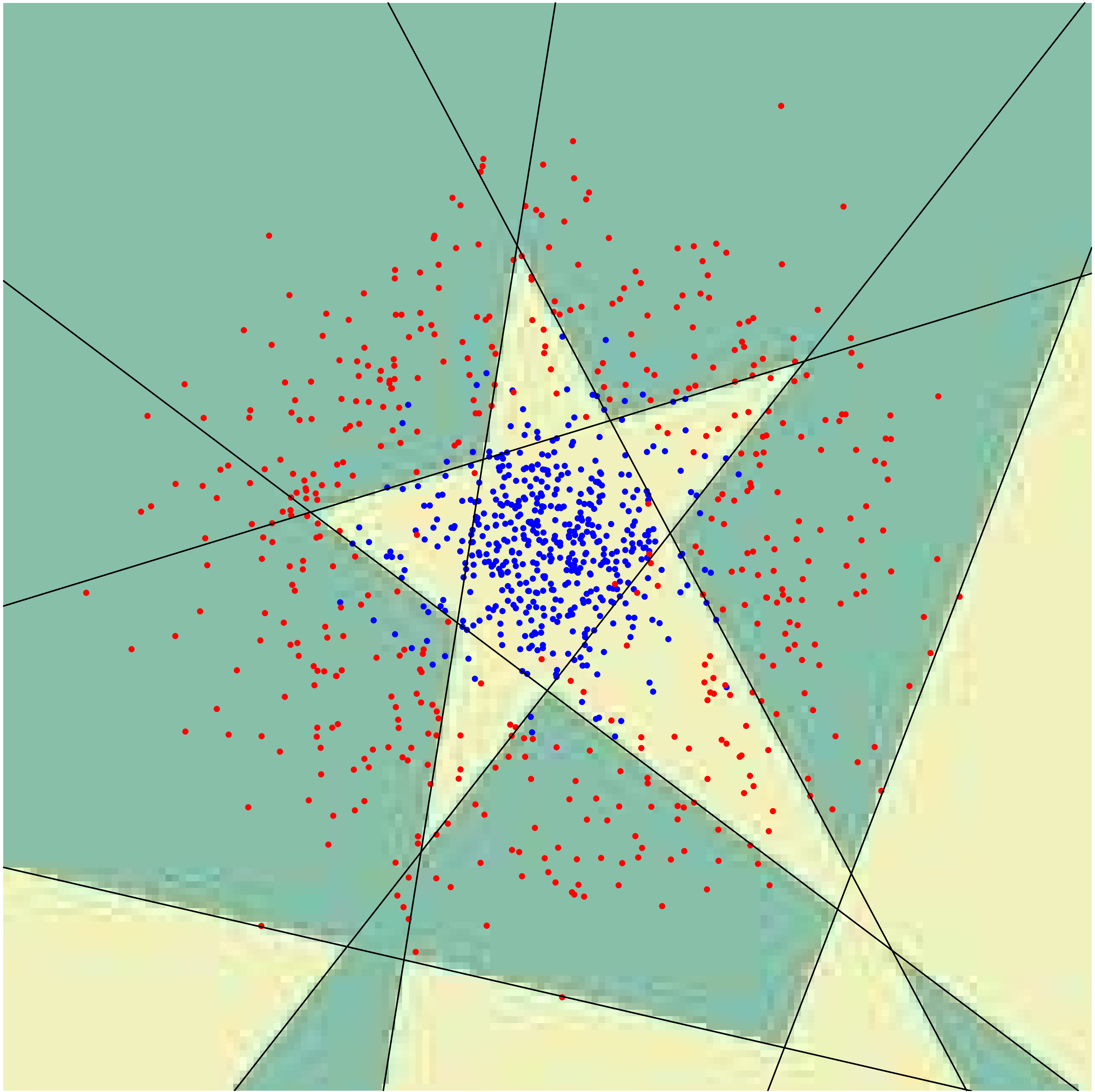
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

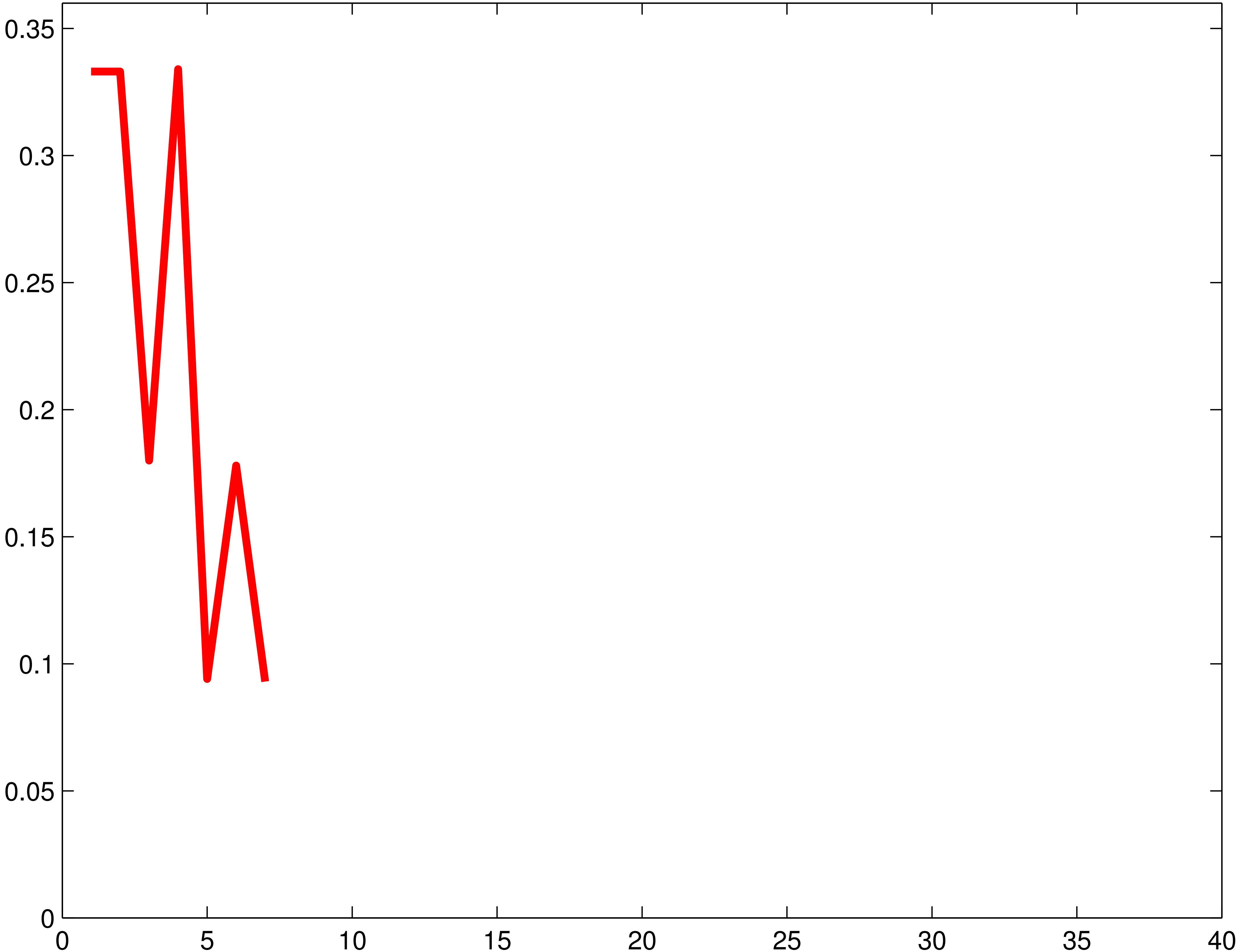


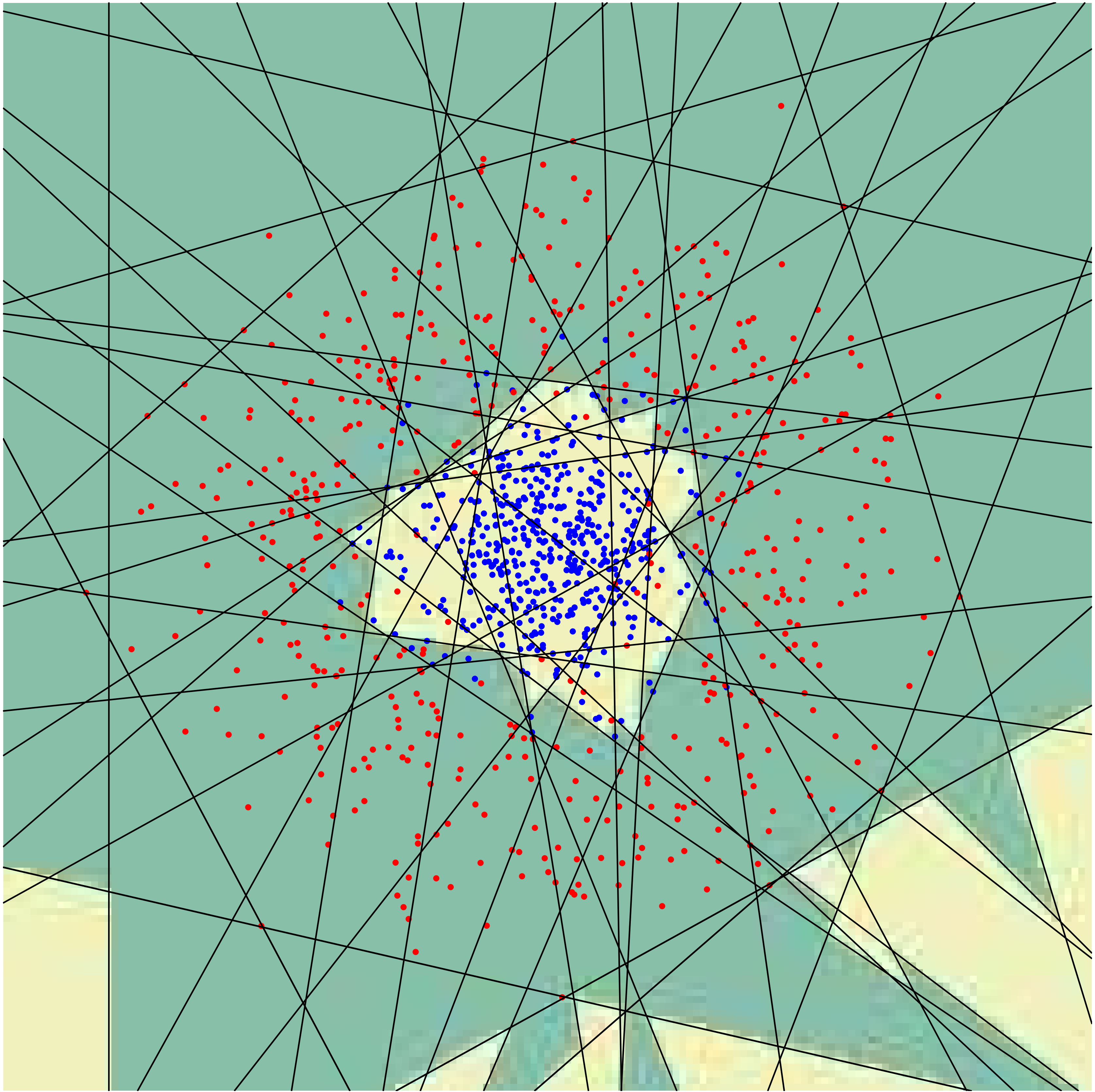

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

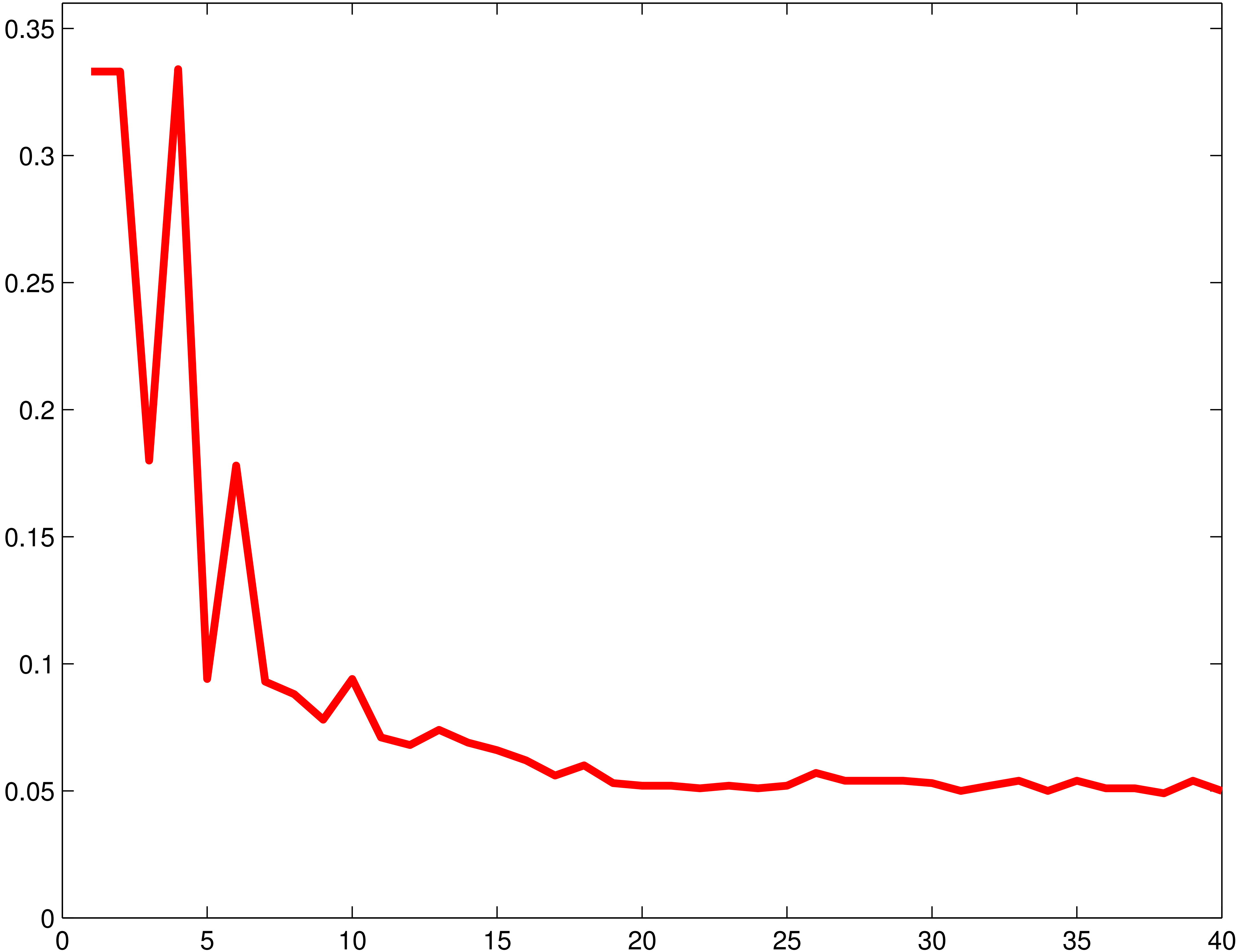


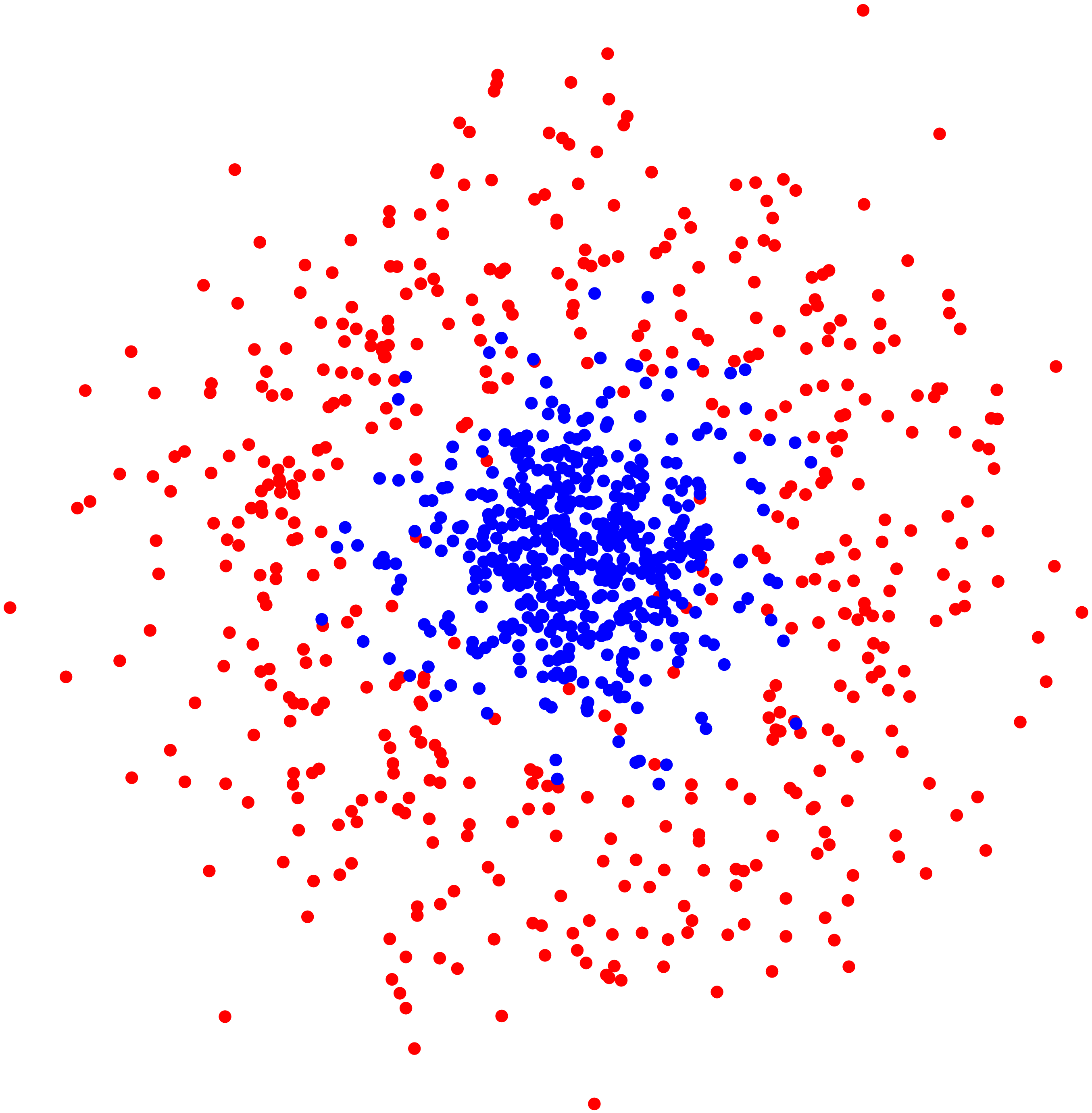

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

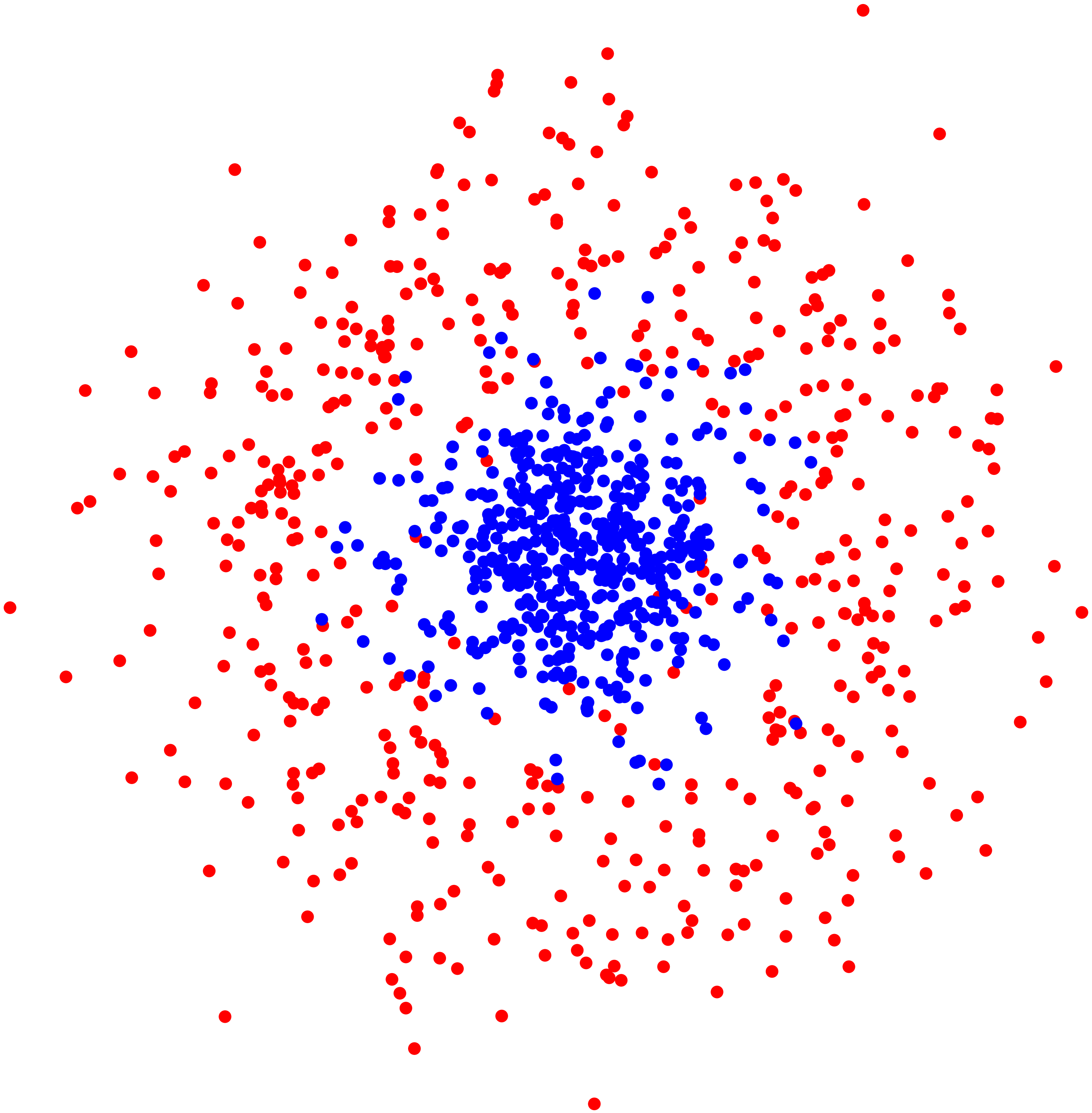


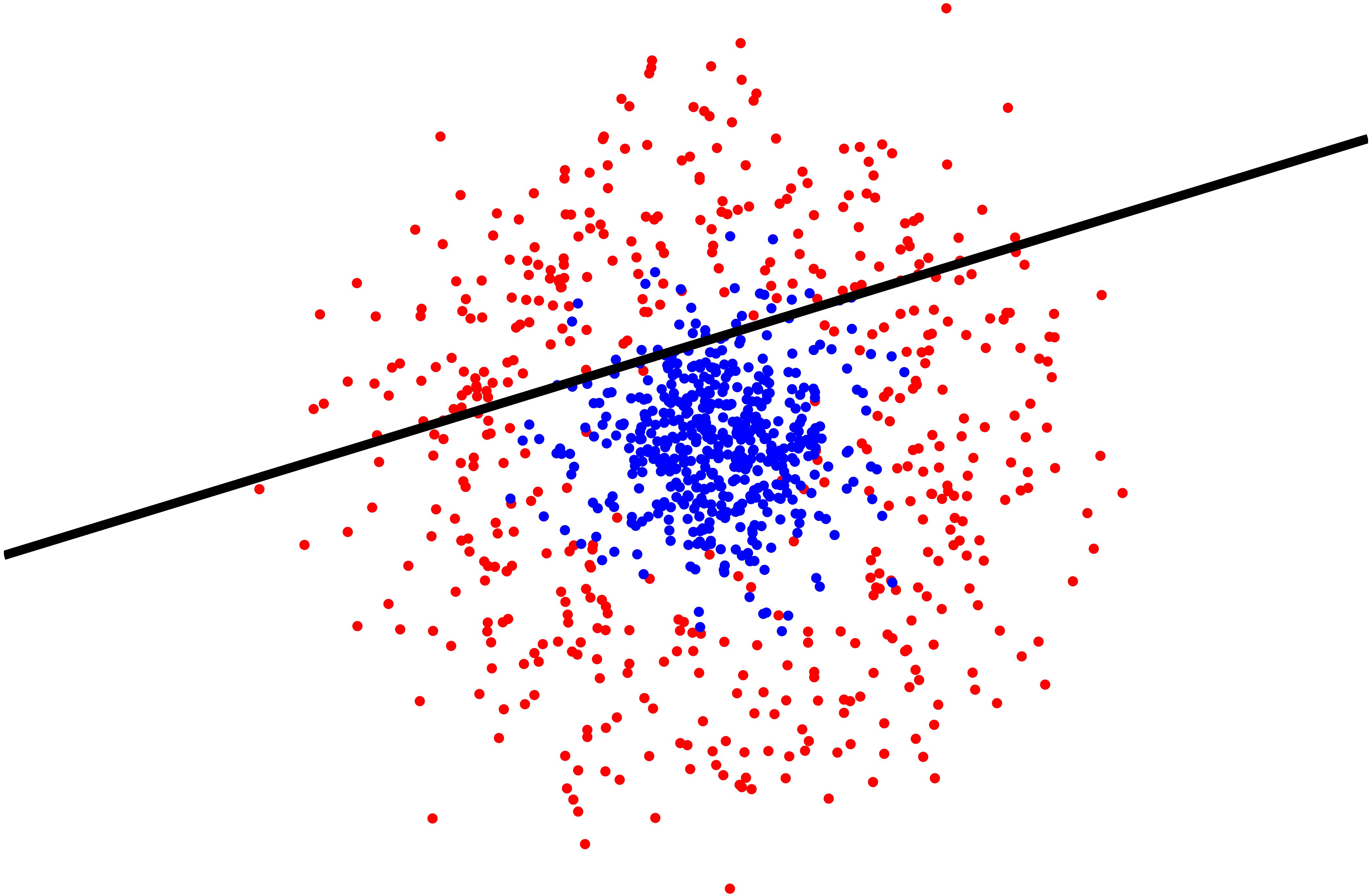

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

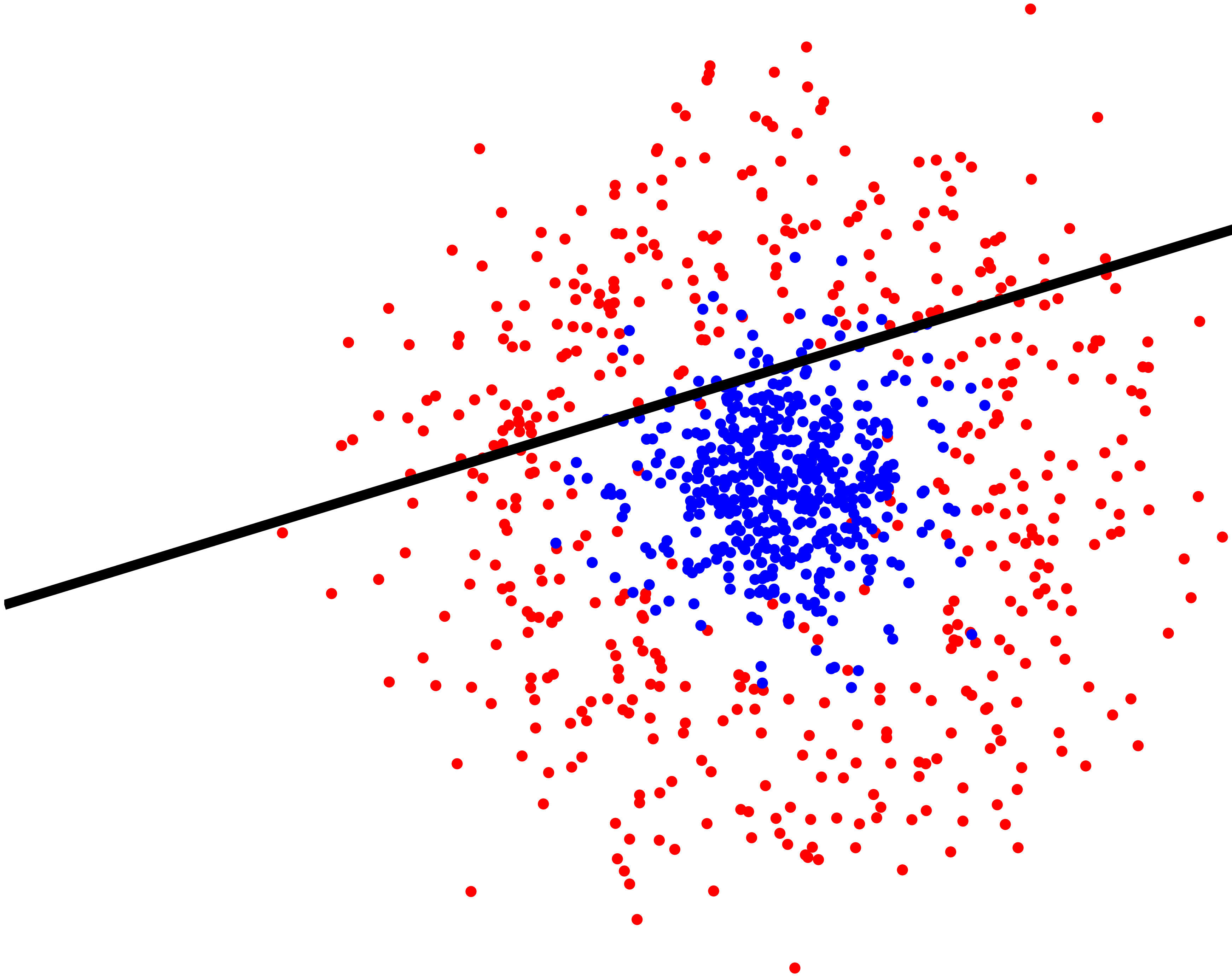


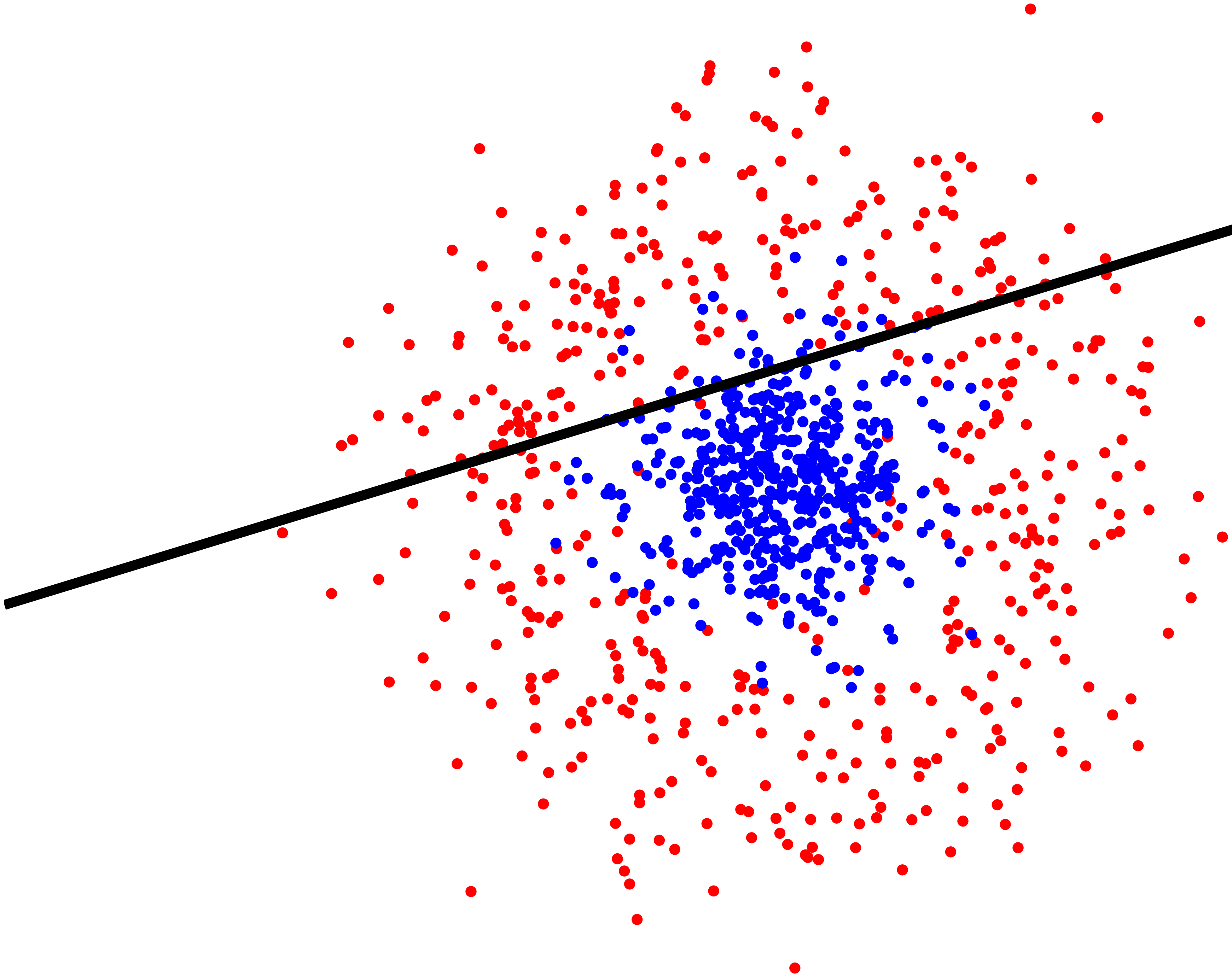

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

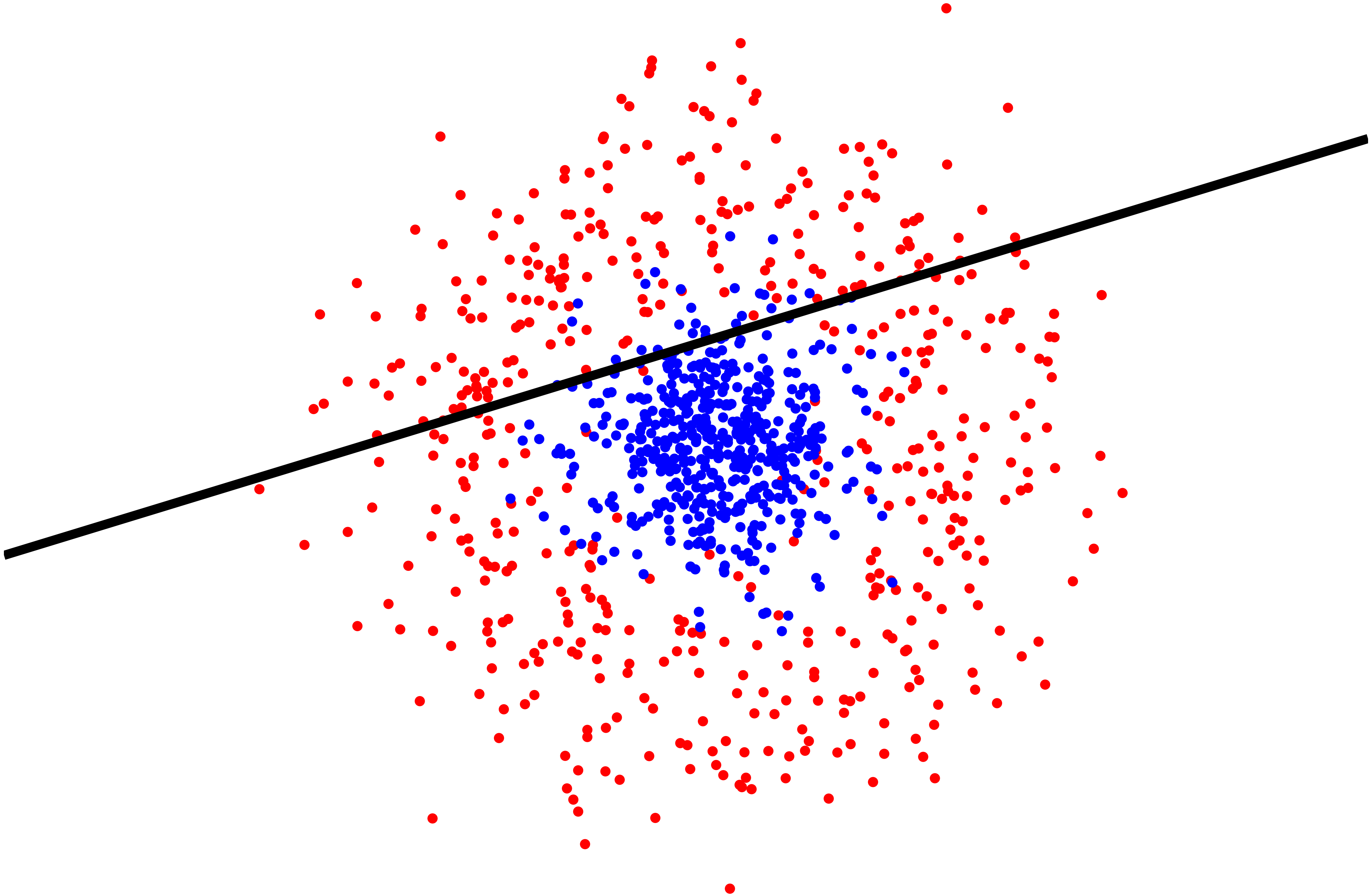


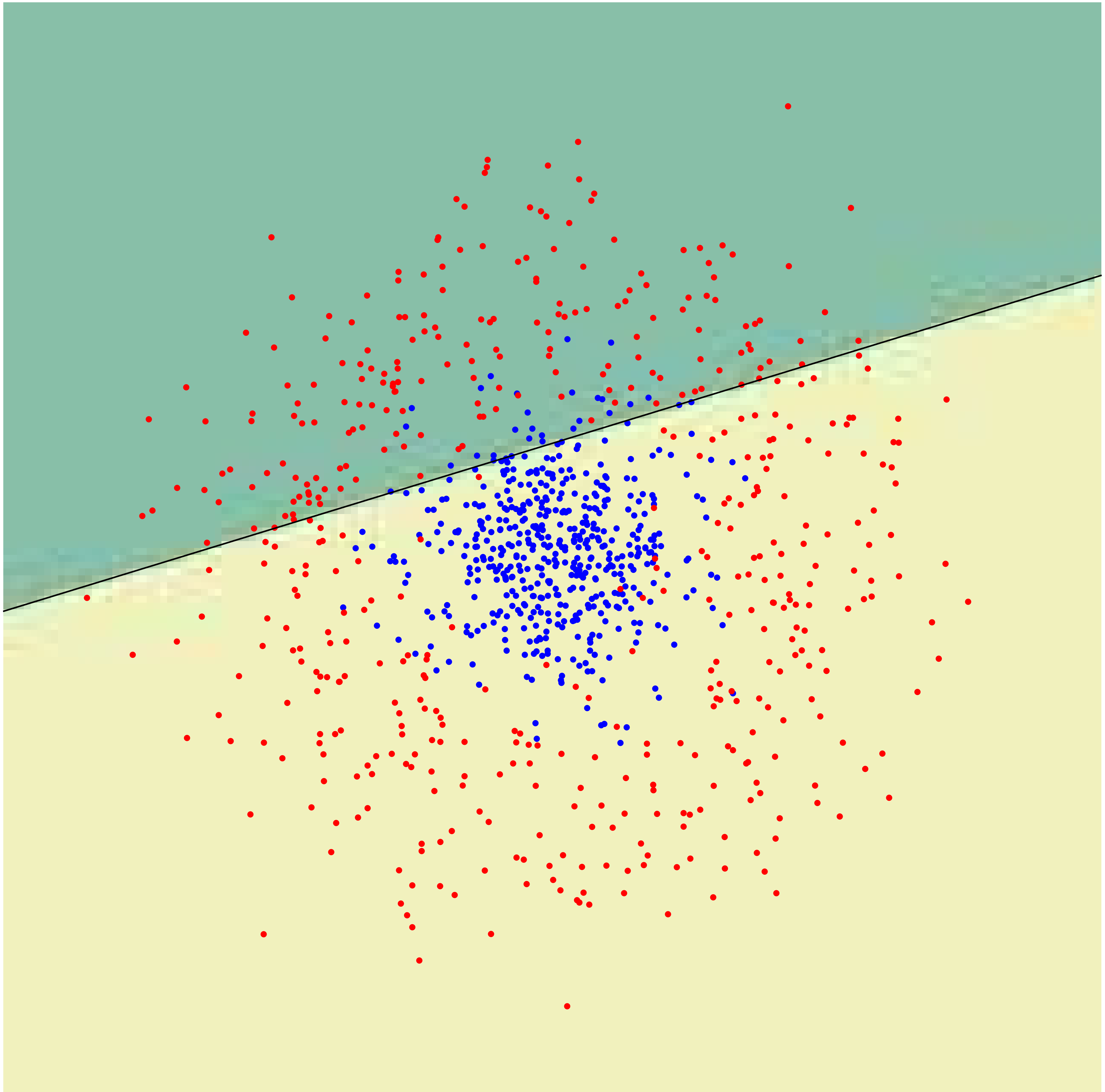


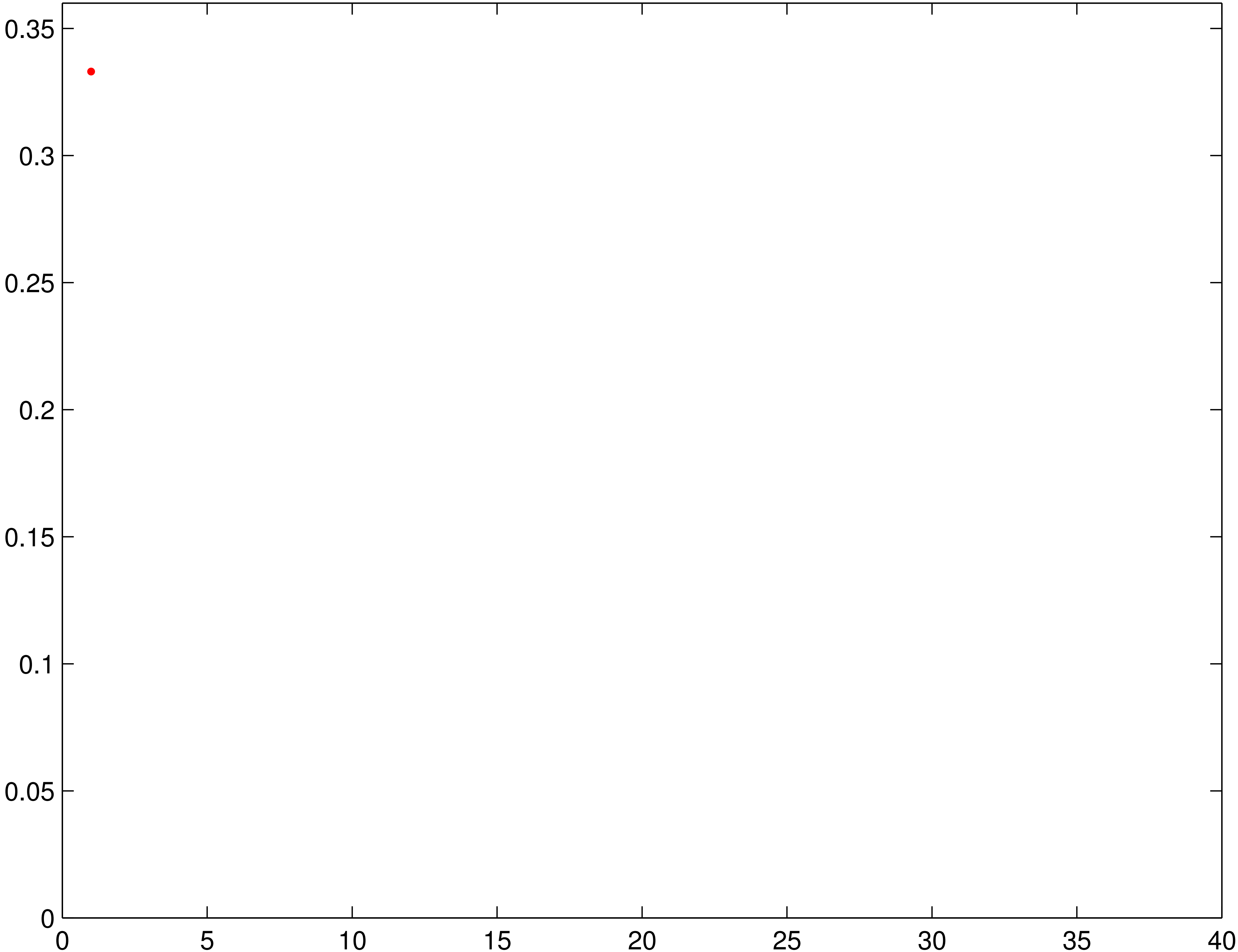


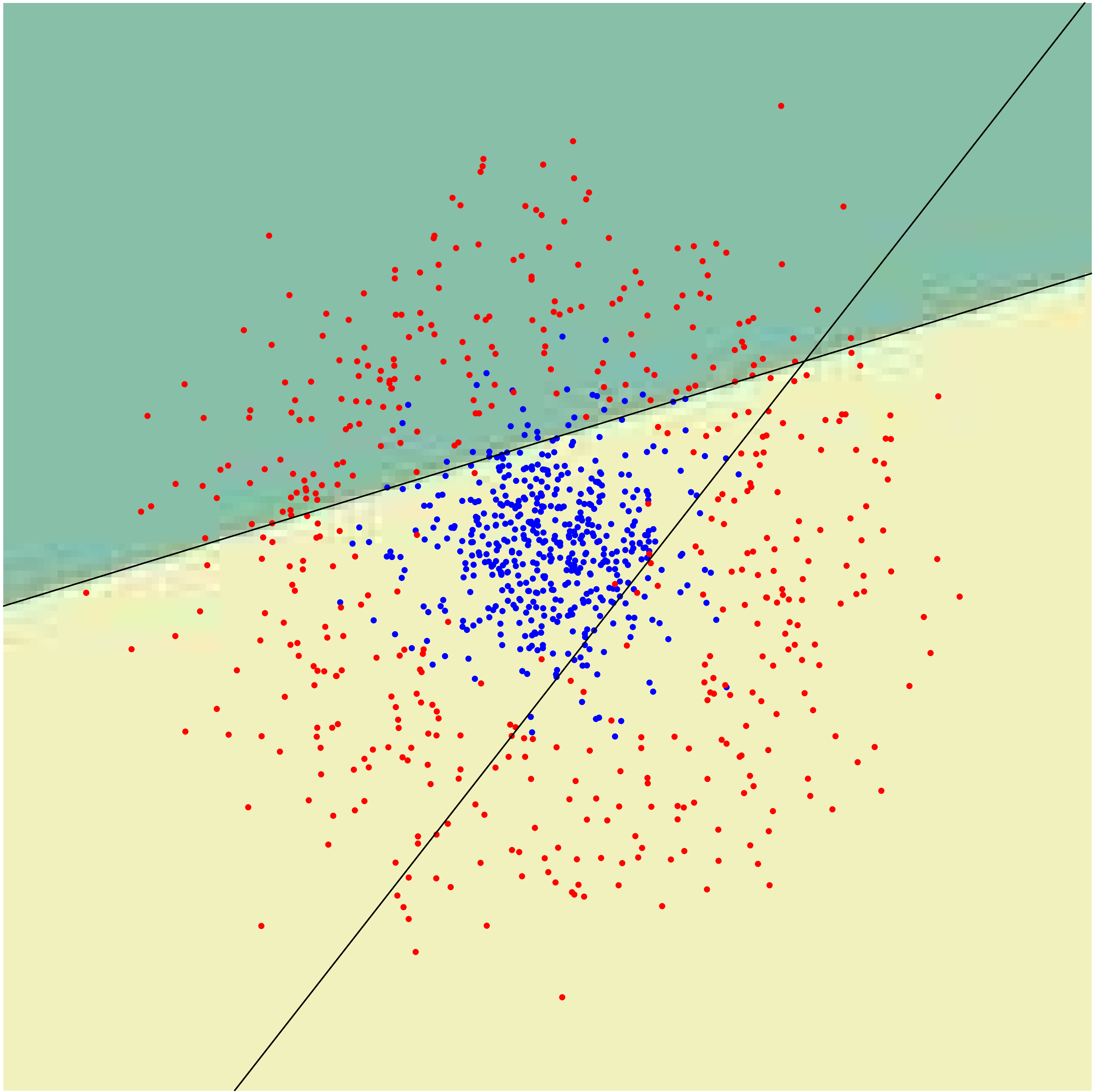


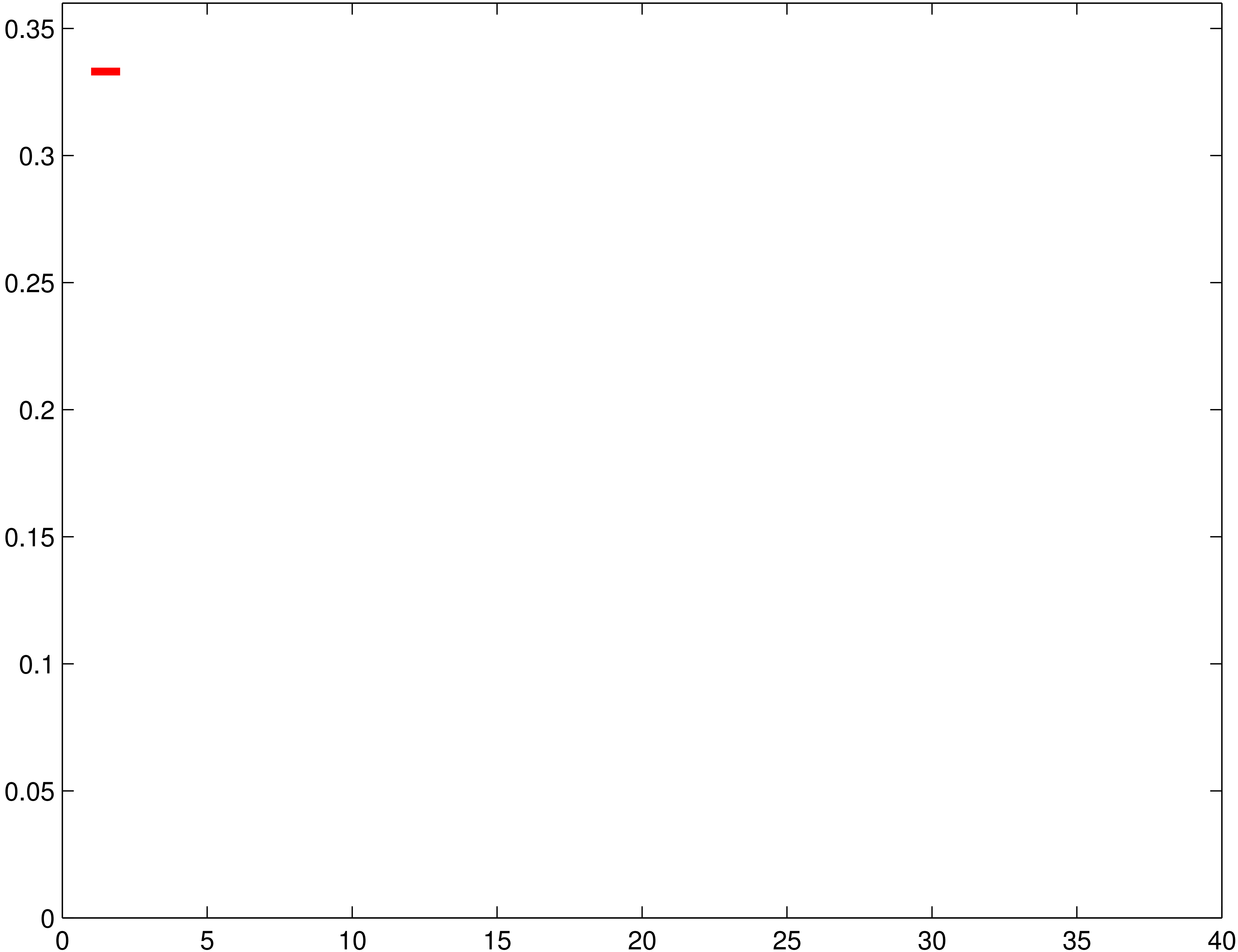




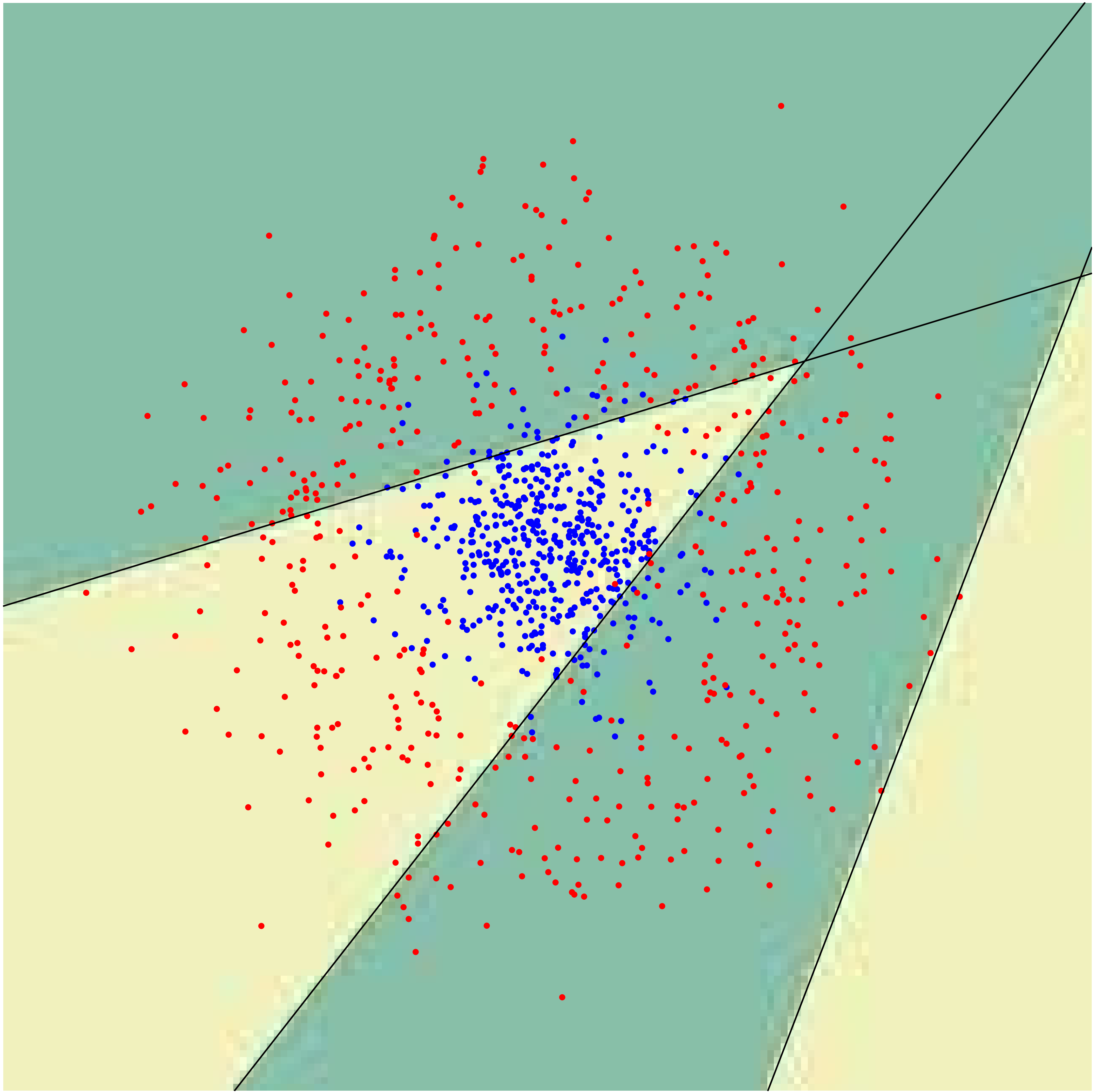


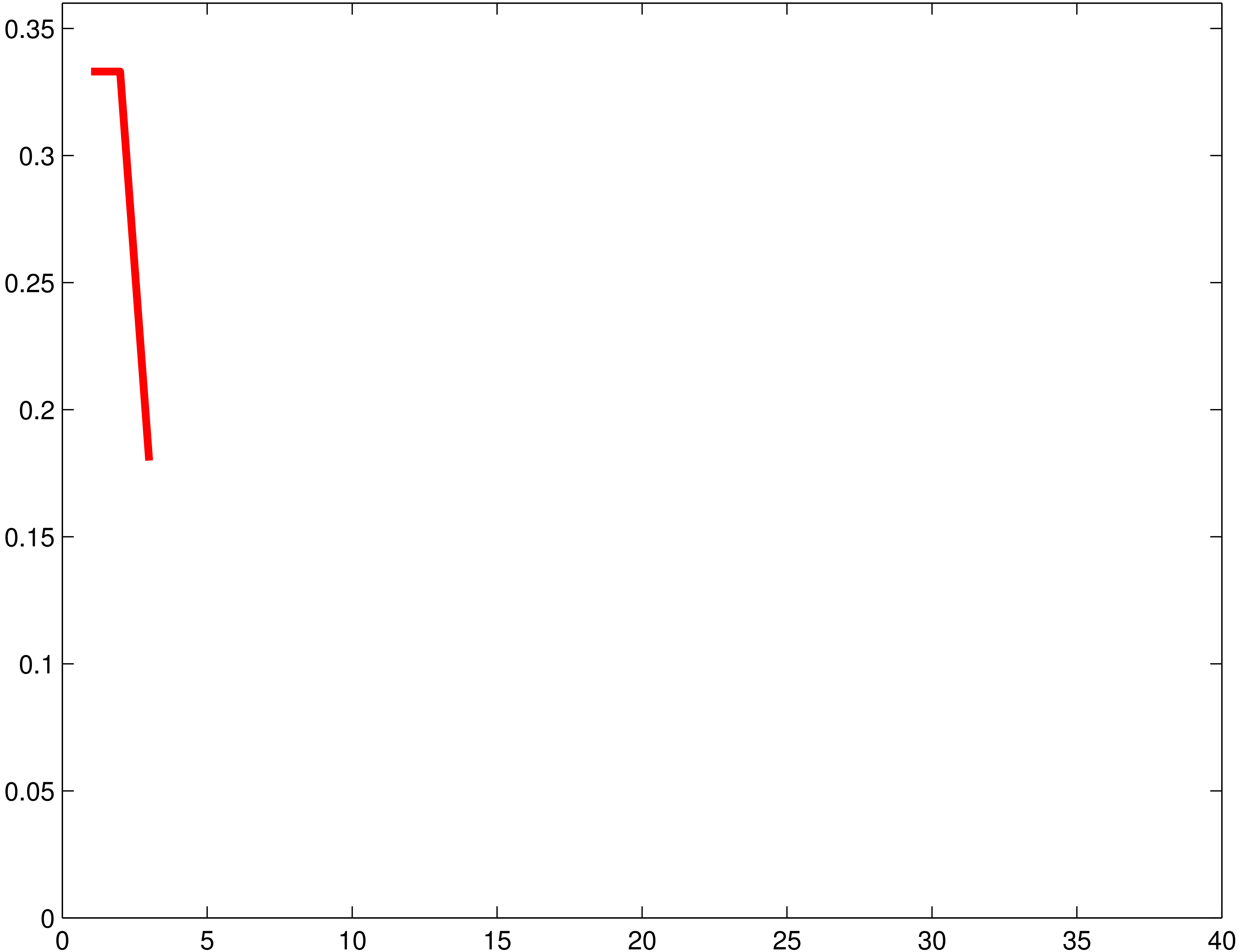

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

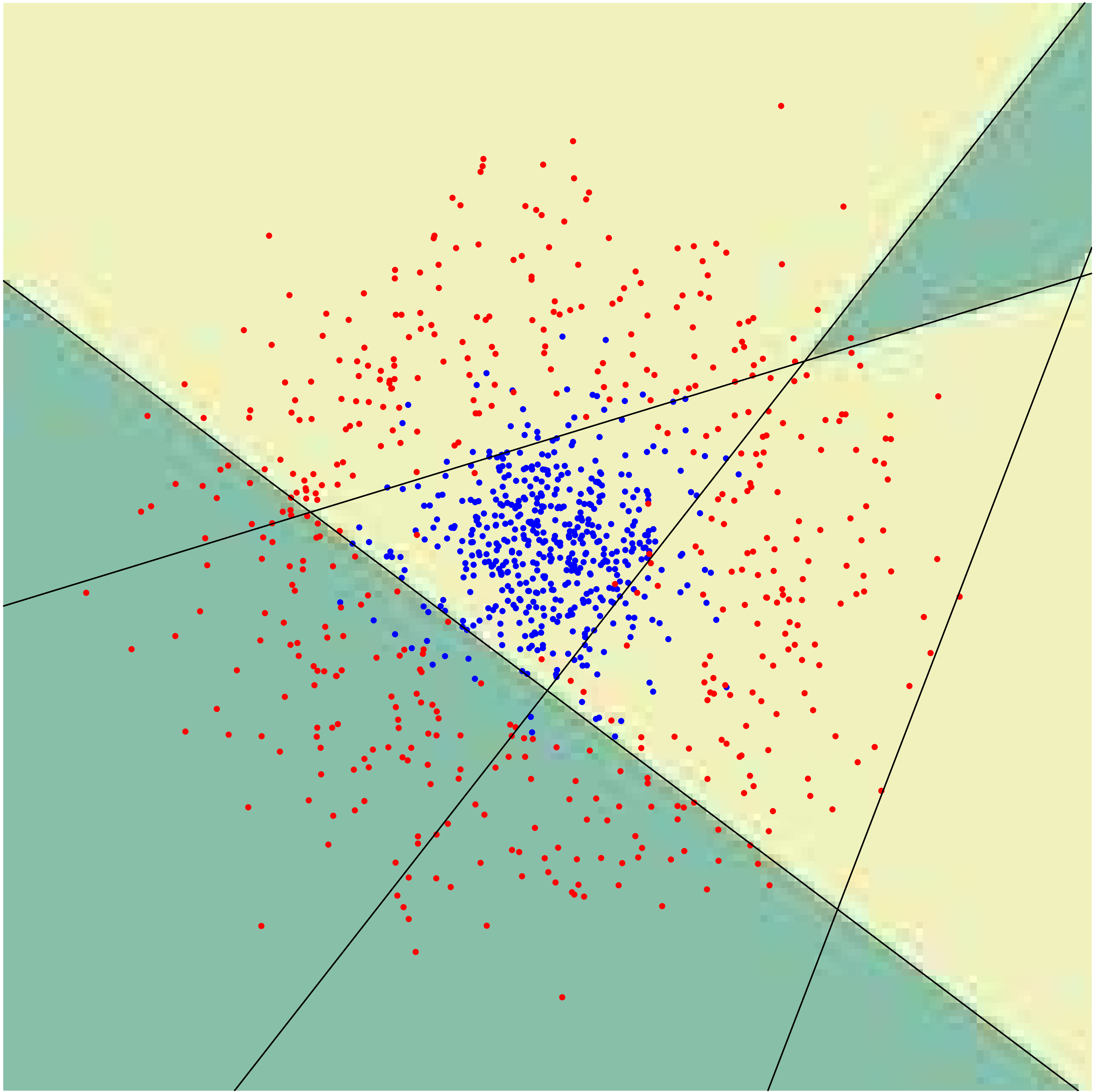


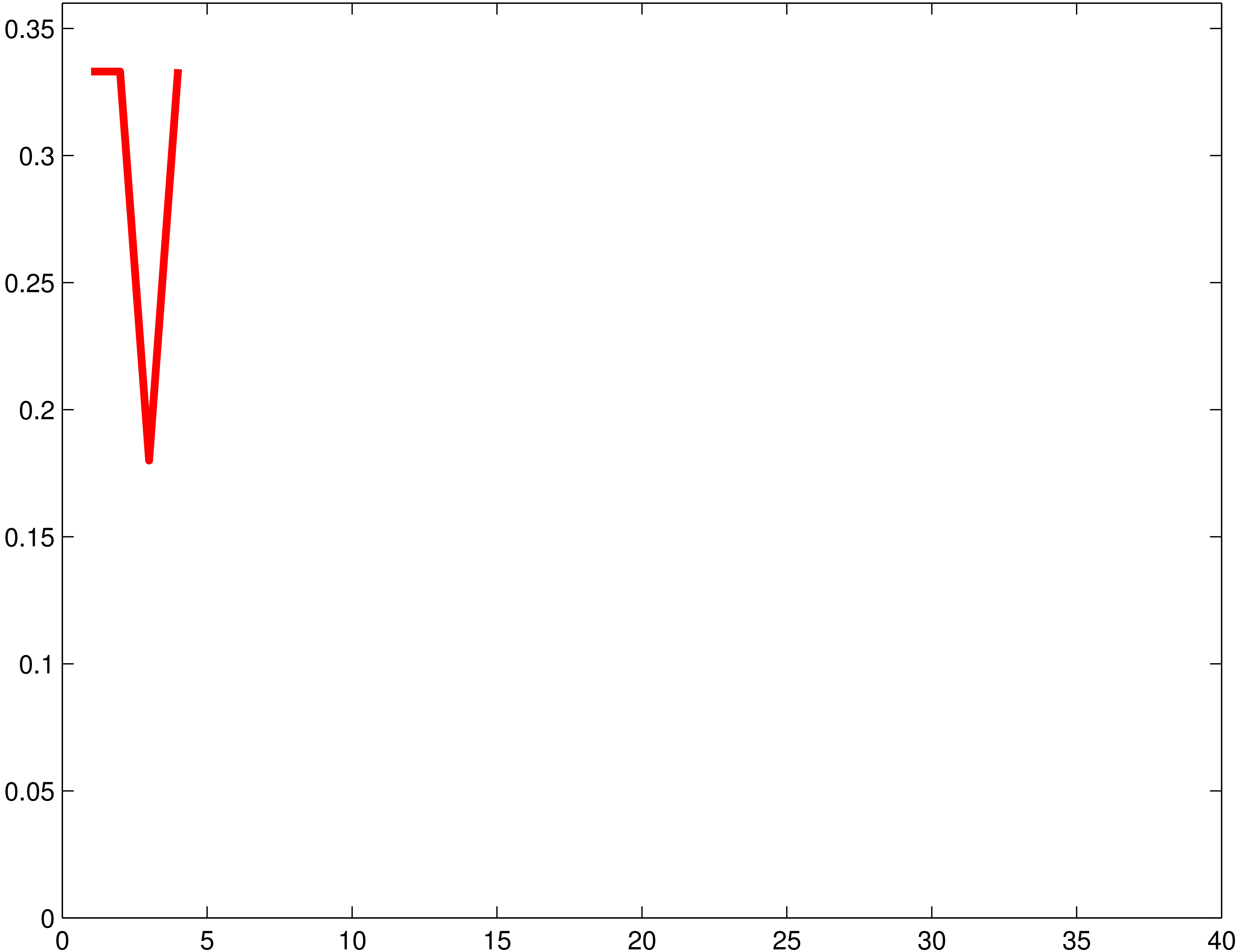


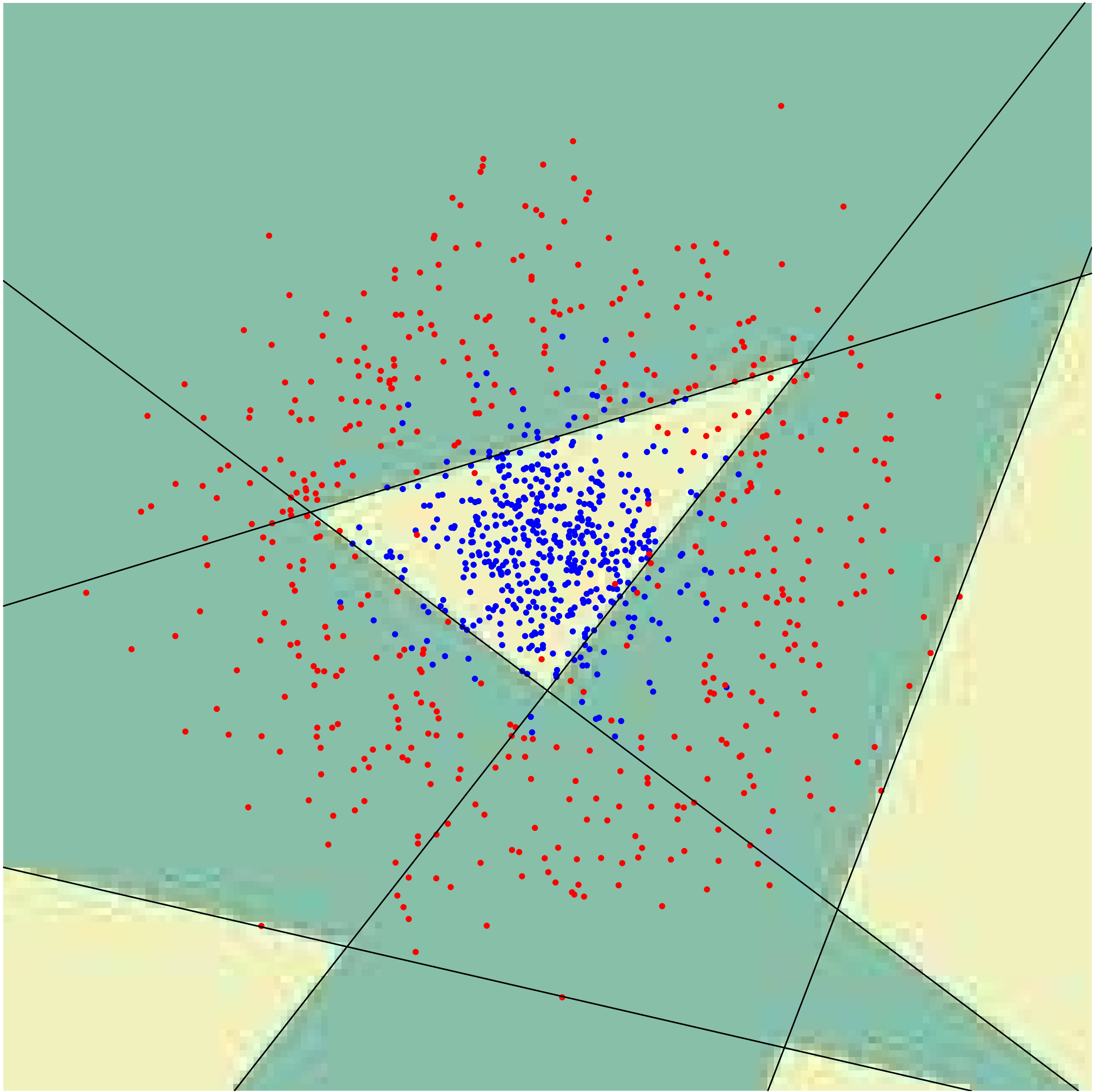
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

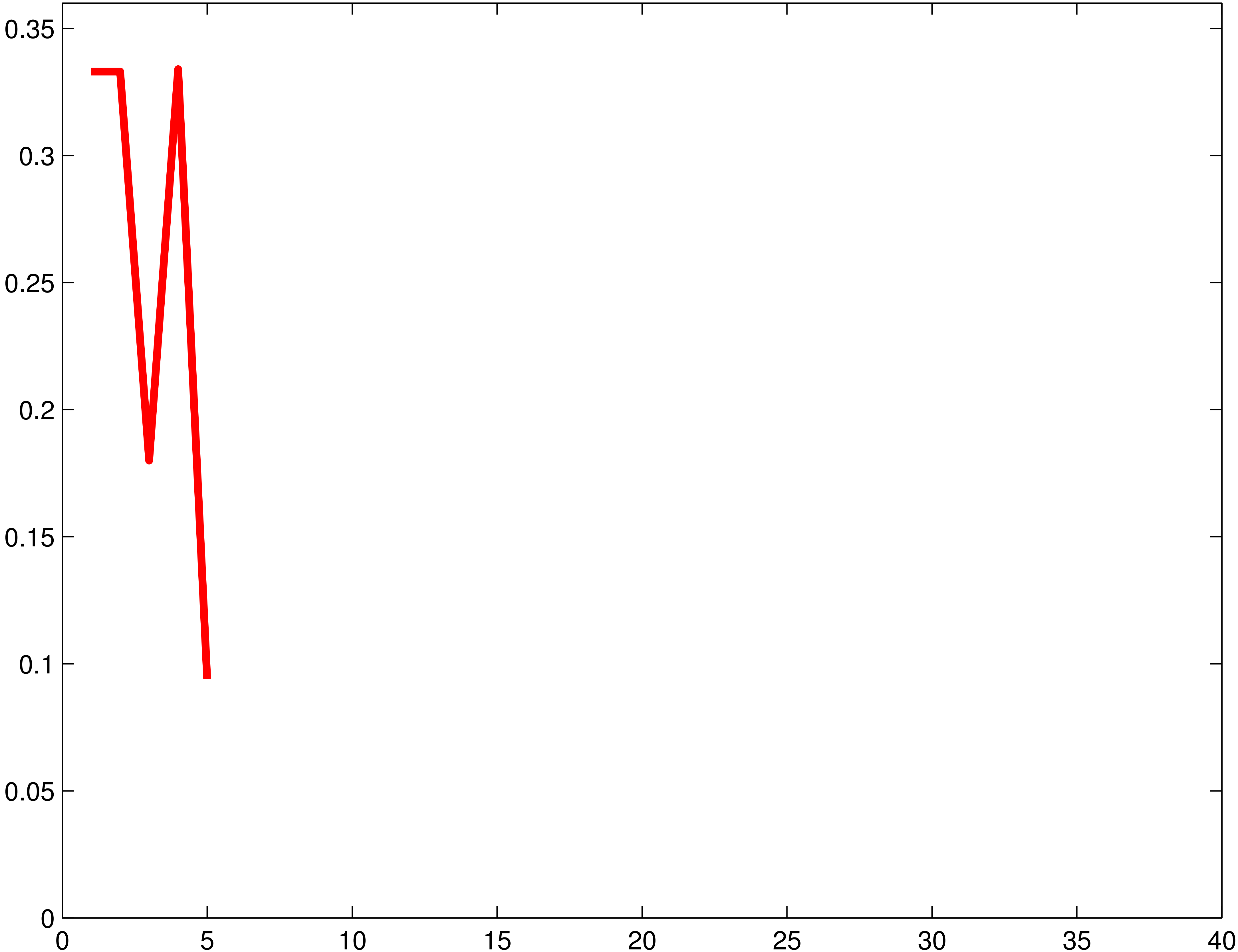



0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

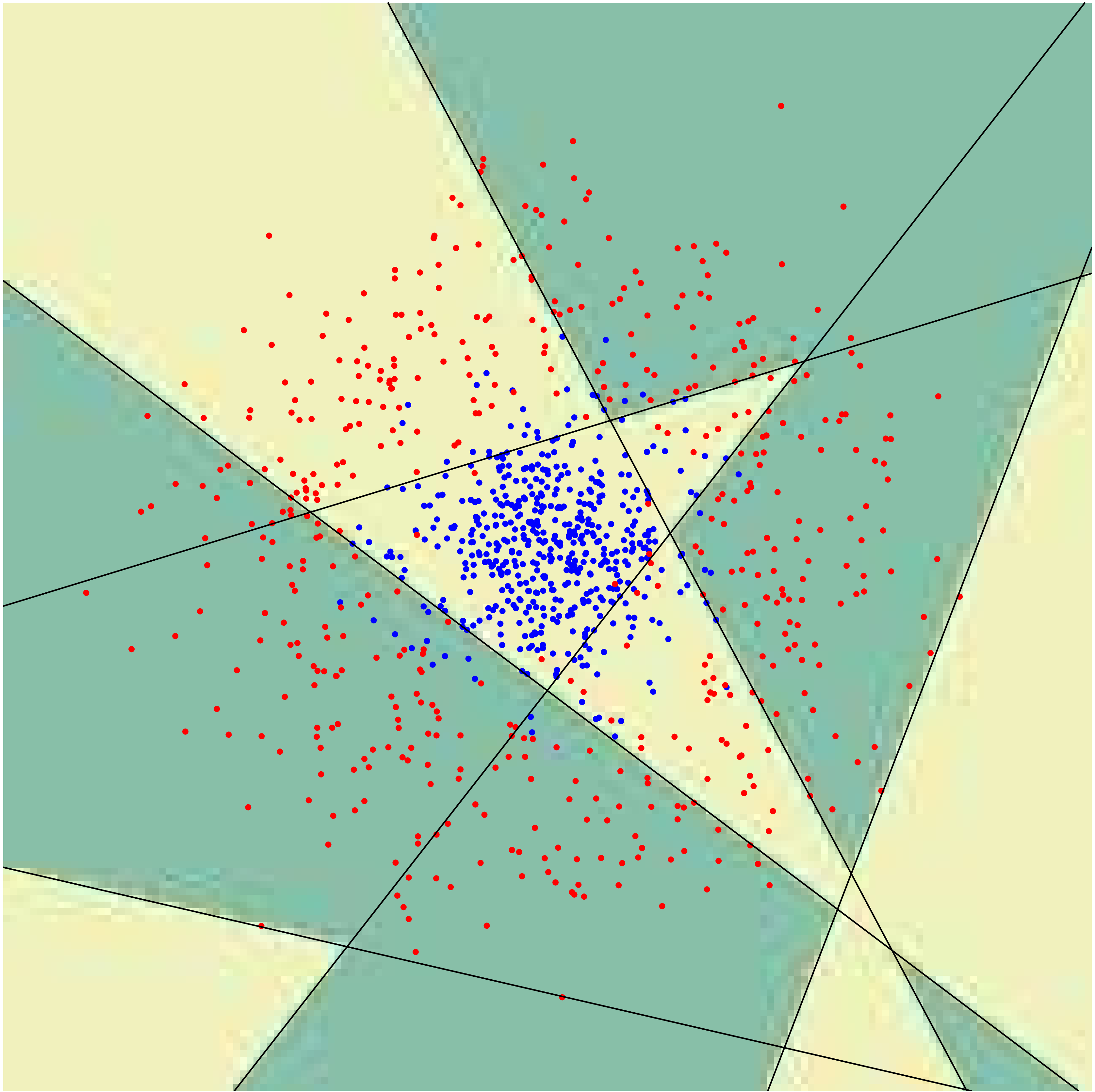


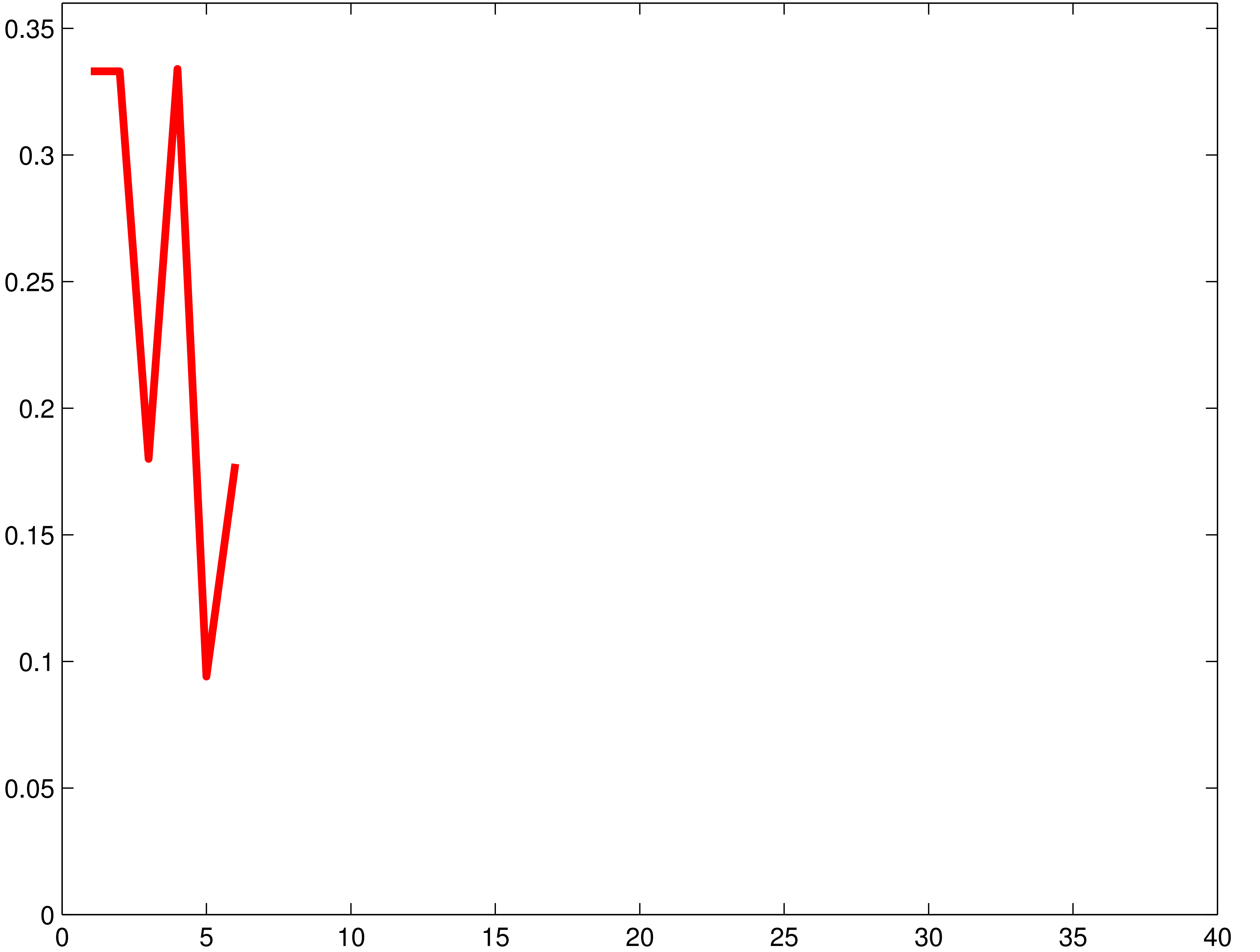

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40



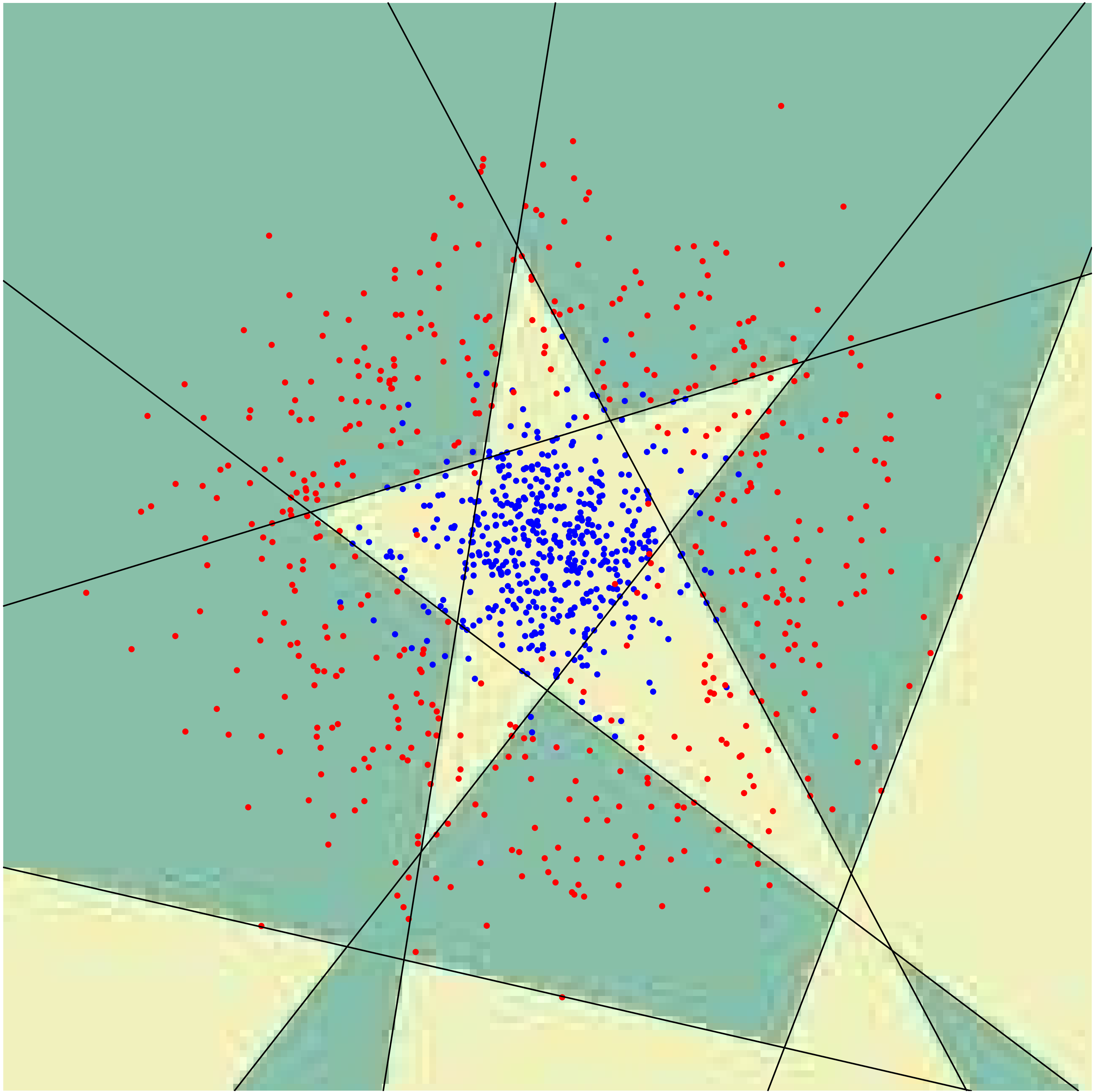


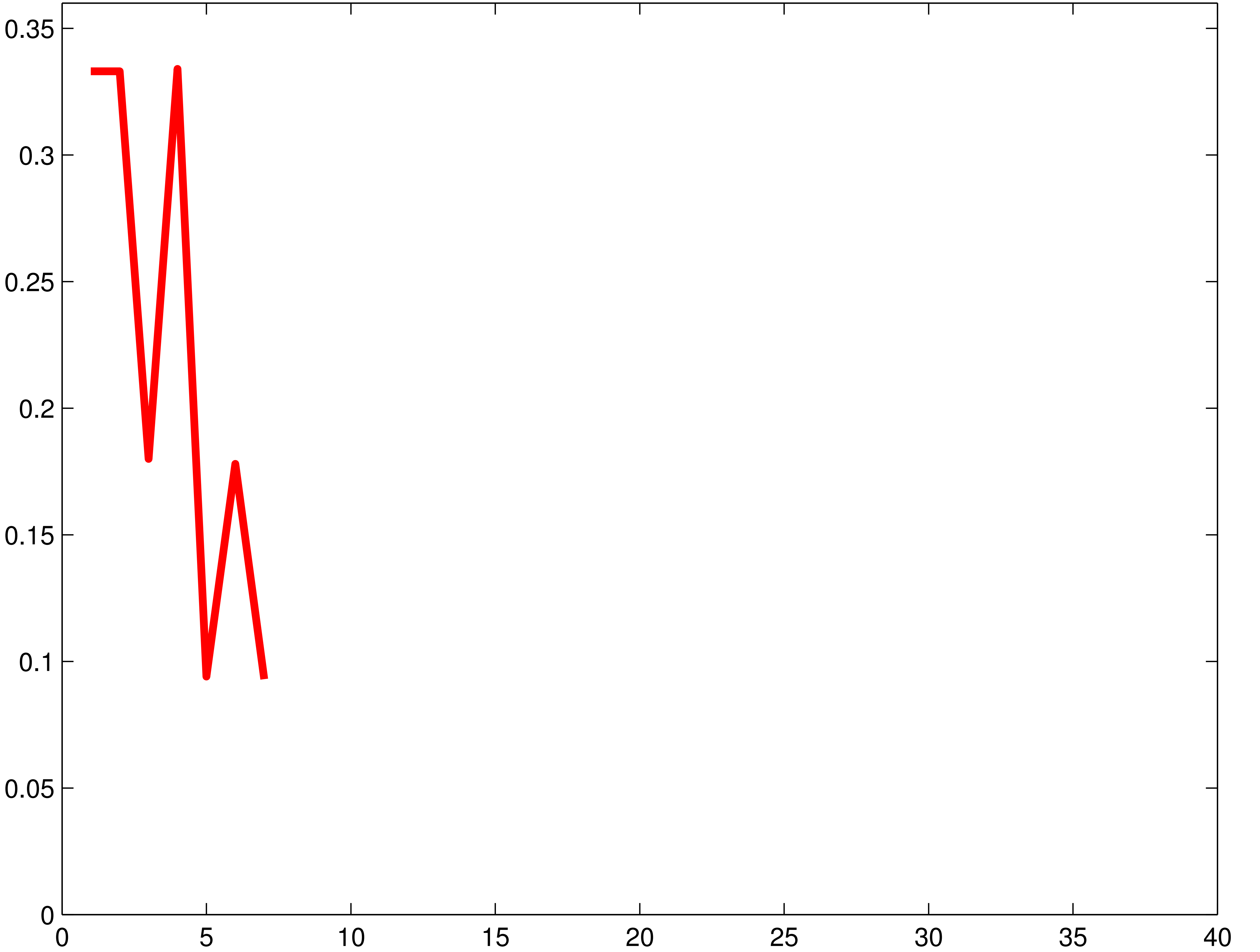
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

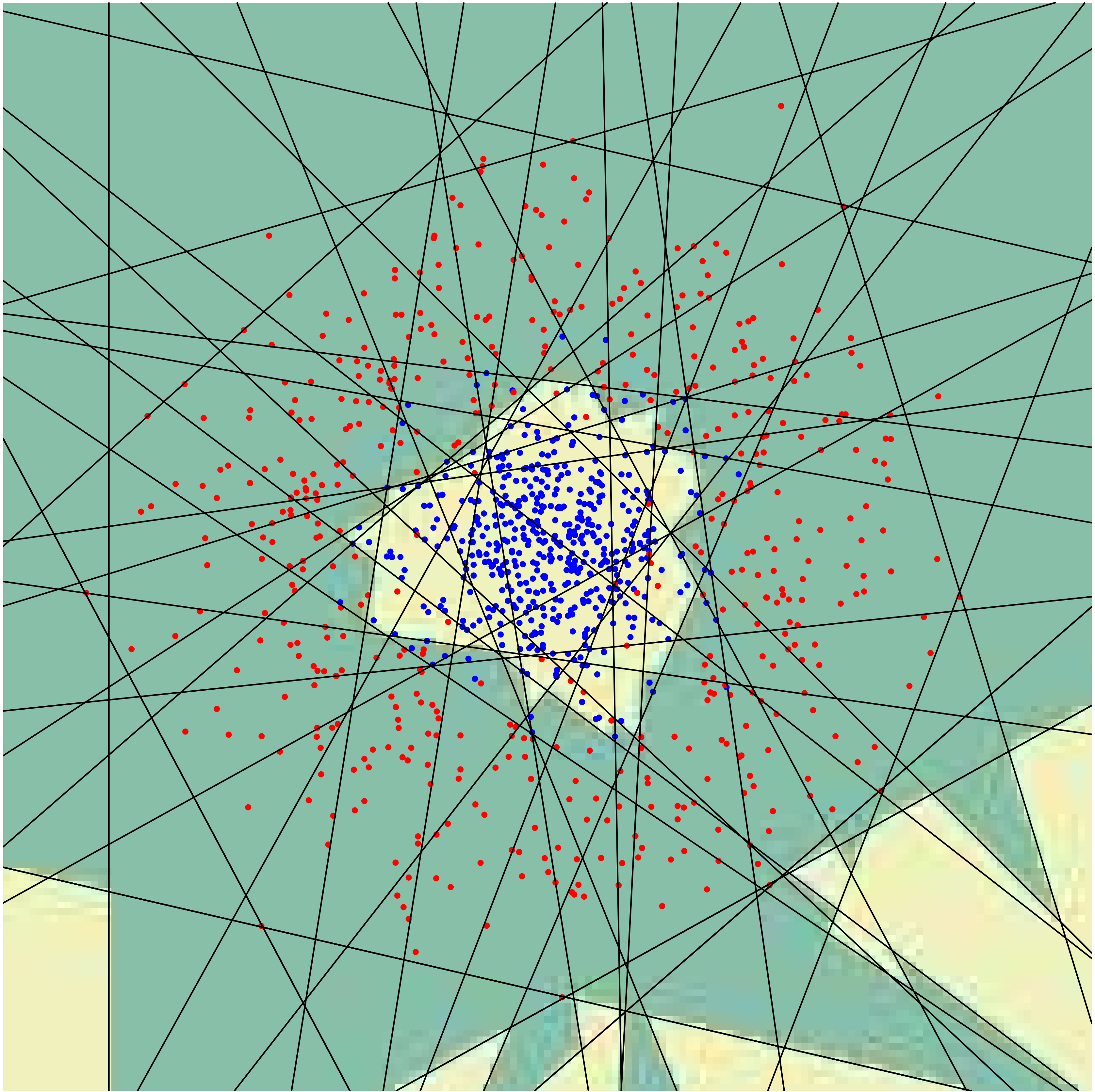


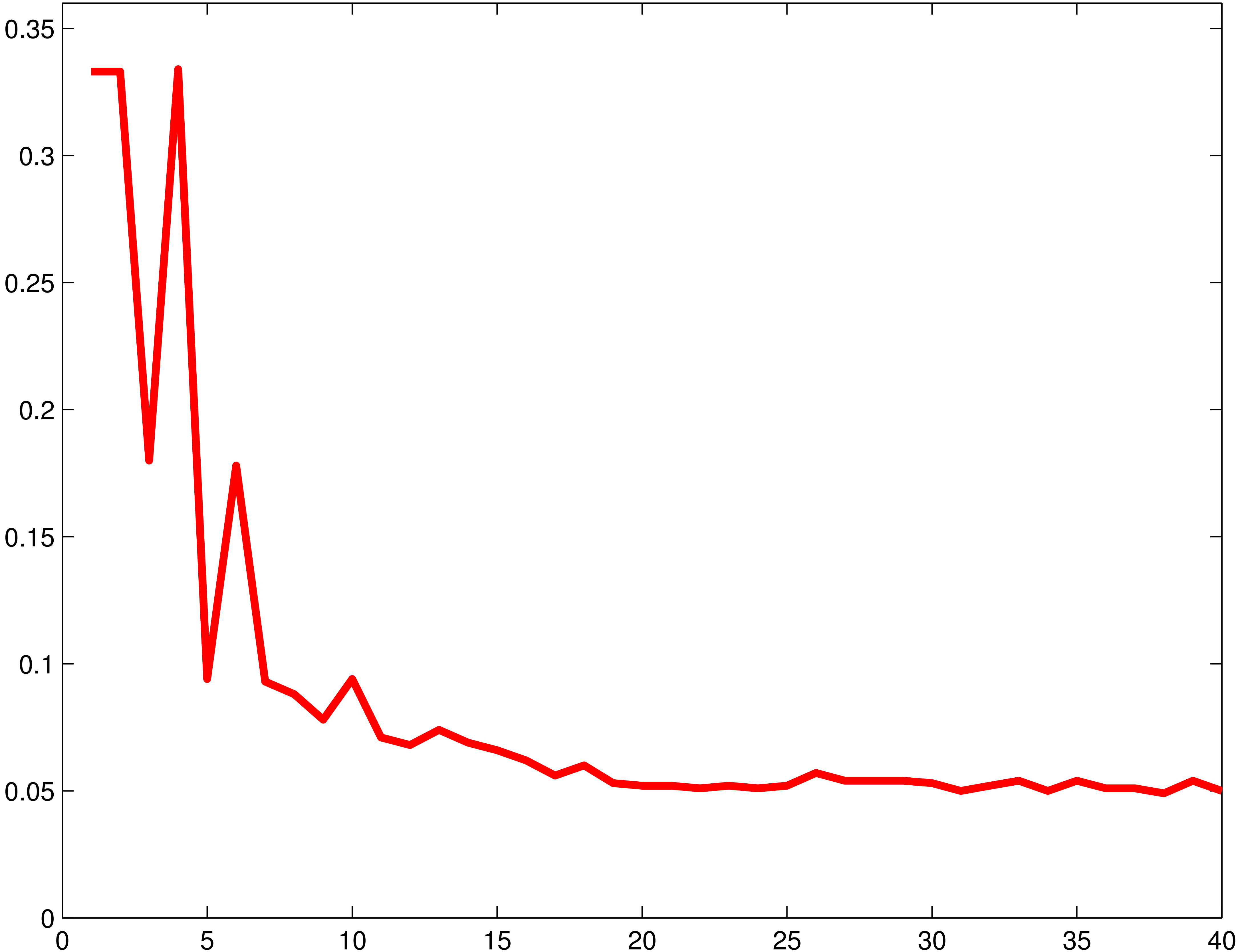


0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40




0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40





0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40

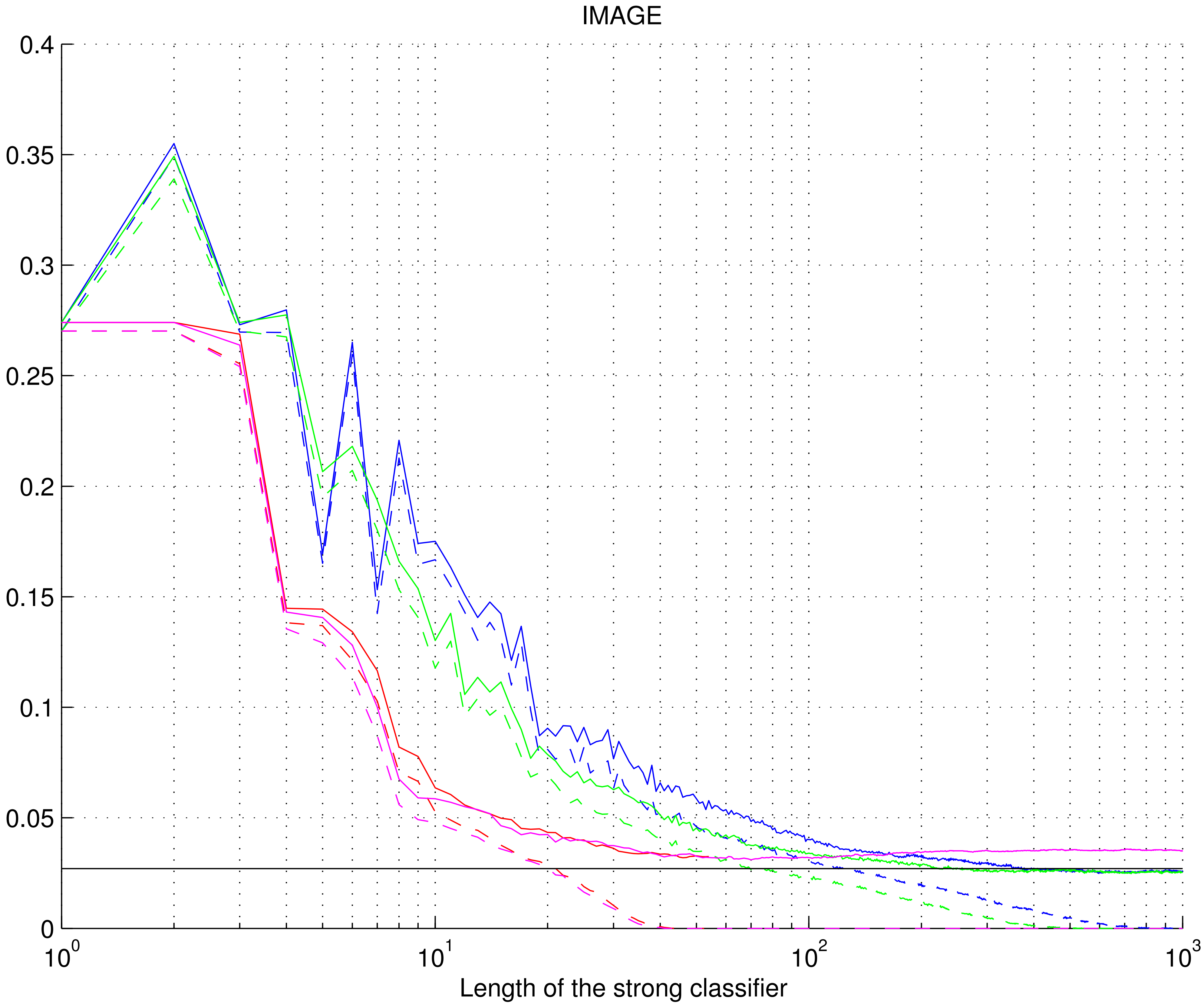
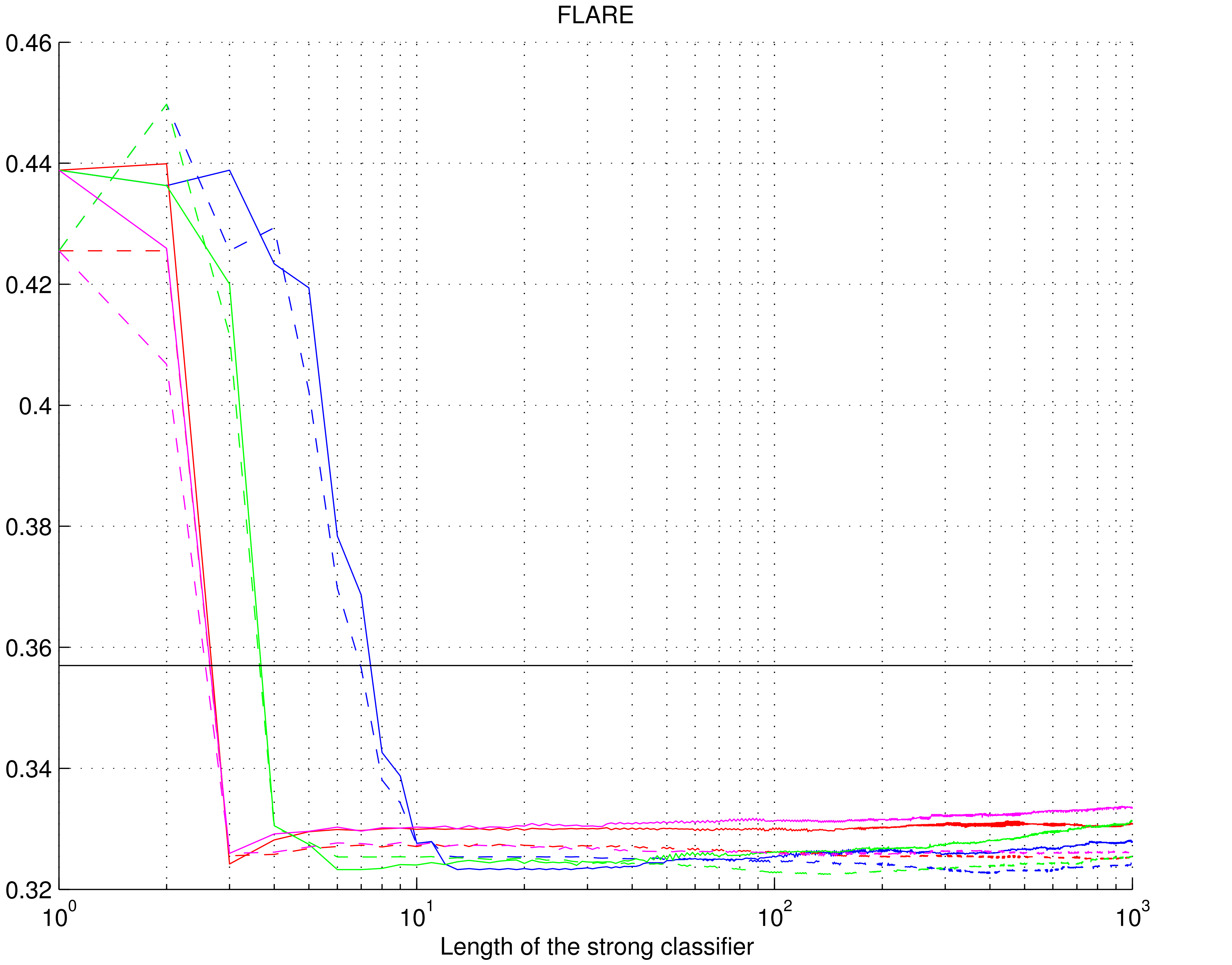
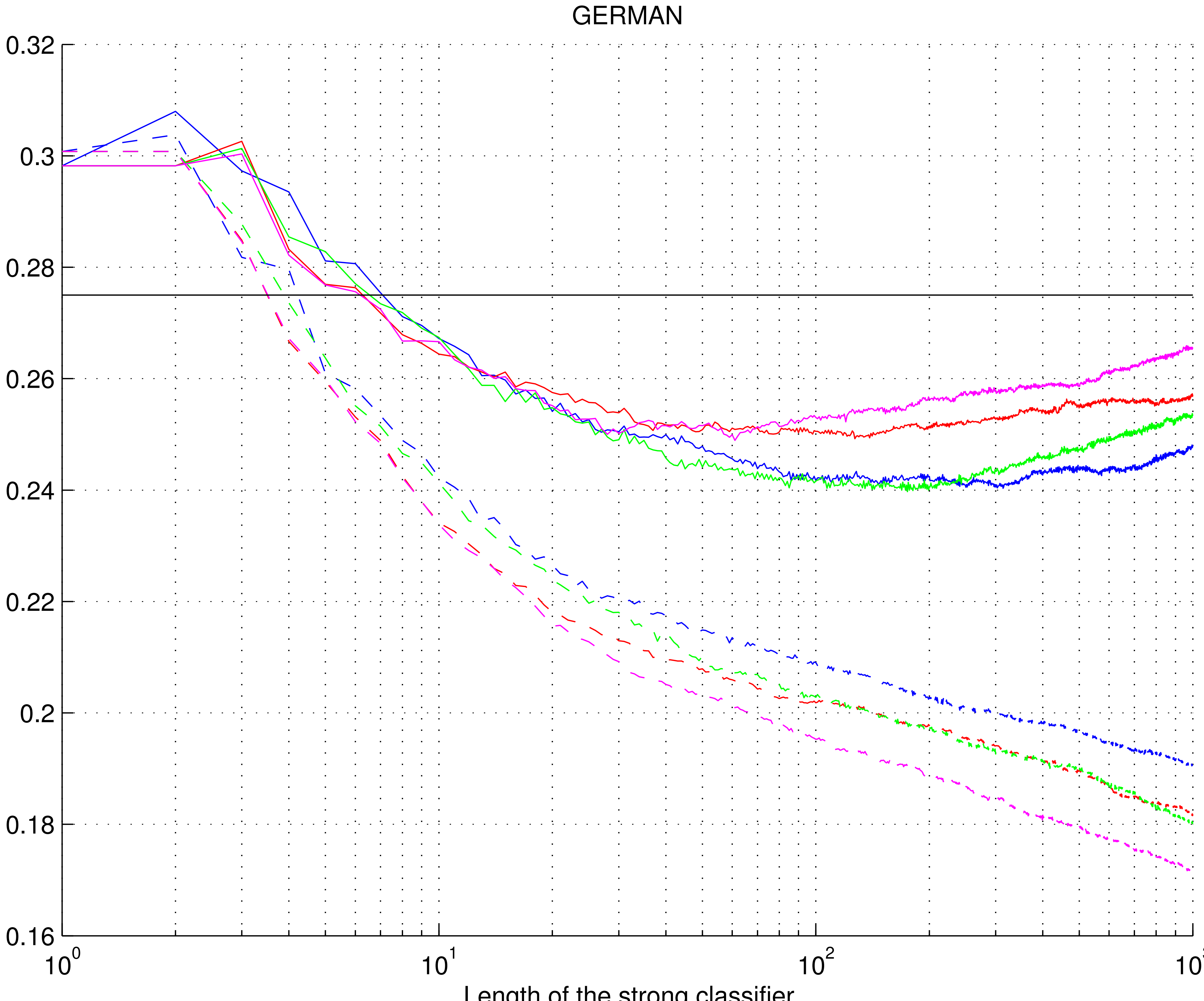
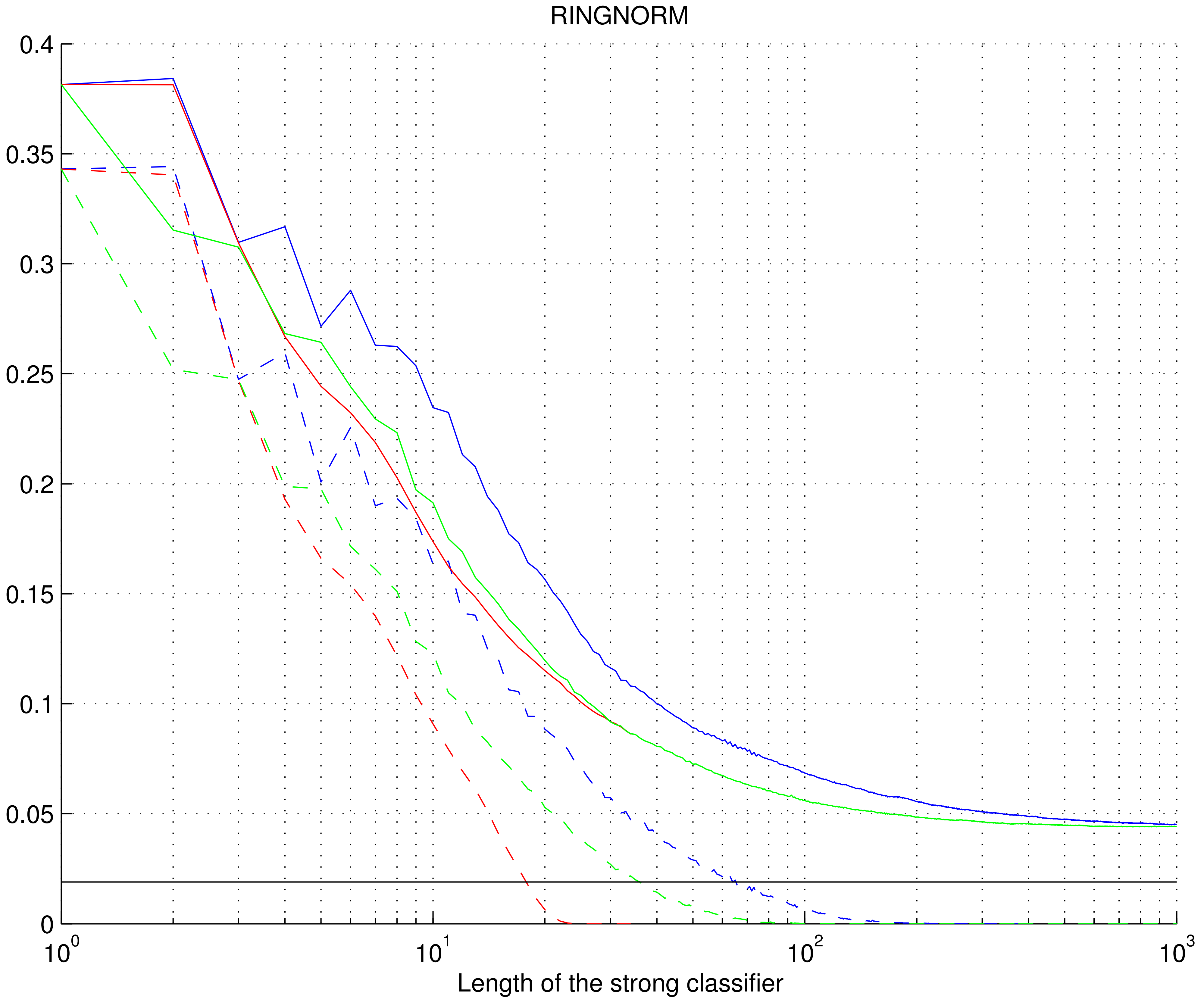


IMAGE
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier

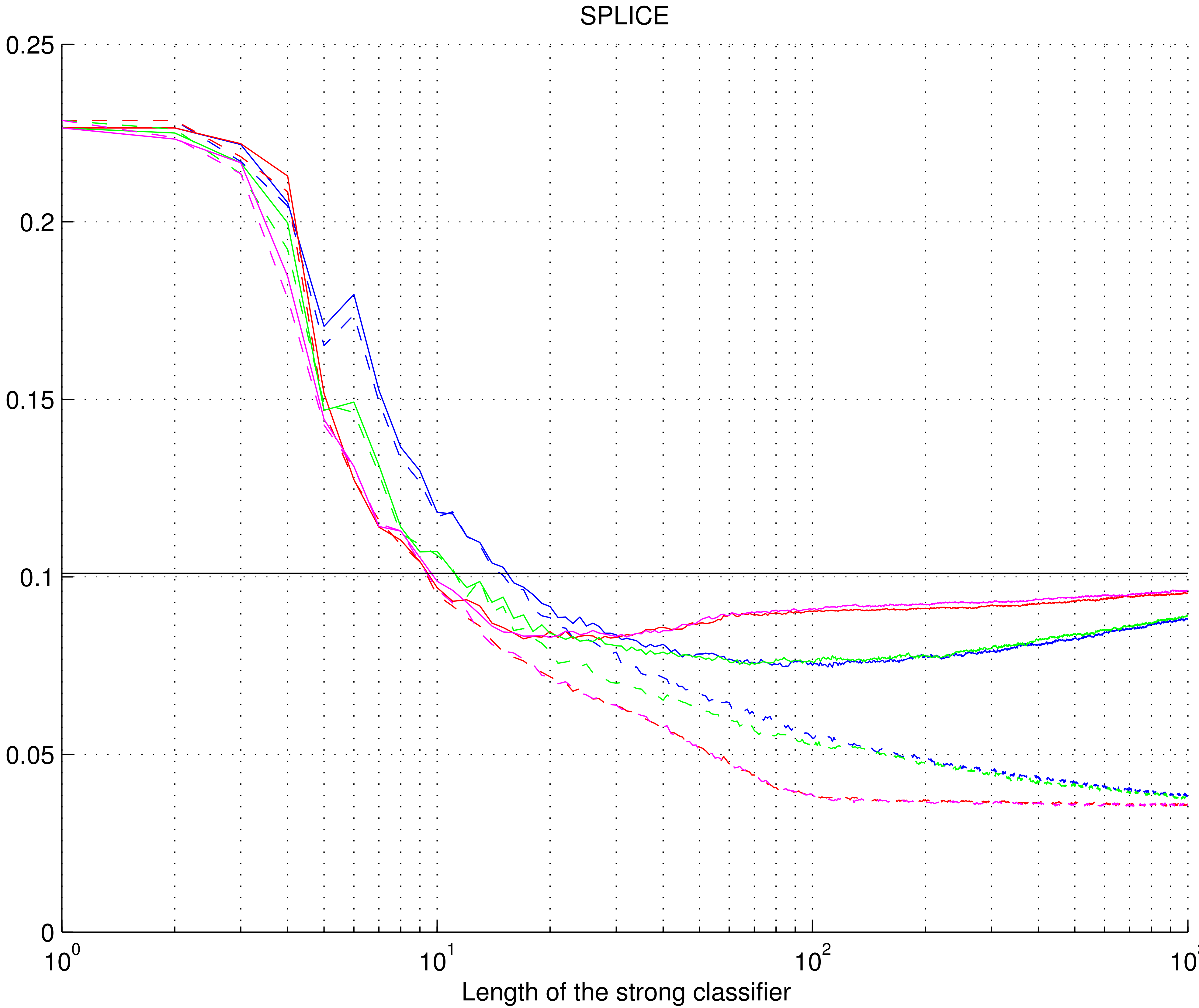

FLARE
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0.32
10^0
10^1
10^2
10^3
Length of the strong classifier

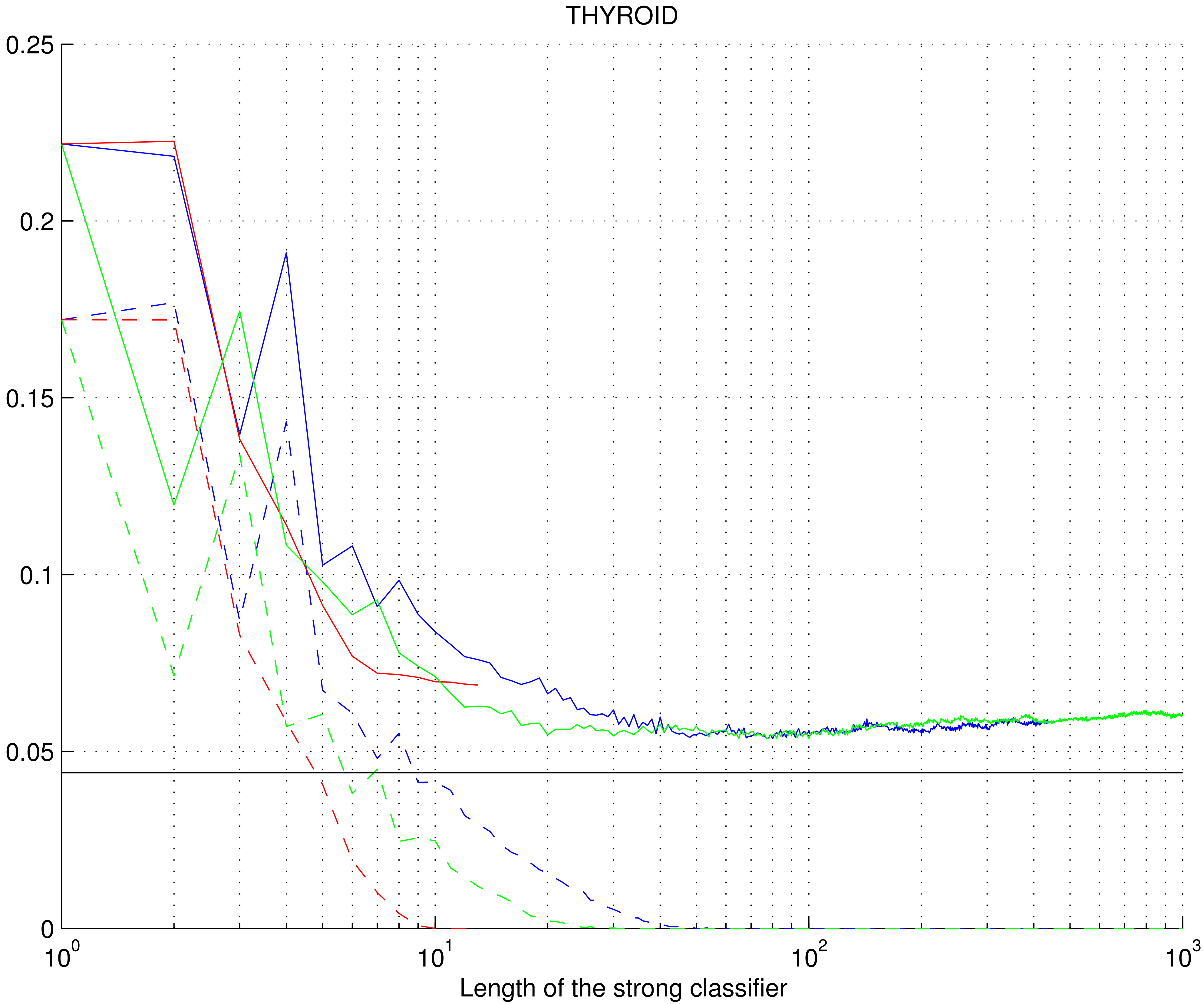


GERMAN
0.32
0.3
0.28
0.26
0.24
0.22
0.2
0.18
0.16
10^0
10^1
10^2
10^3
Length of the strong classifier

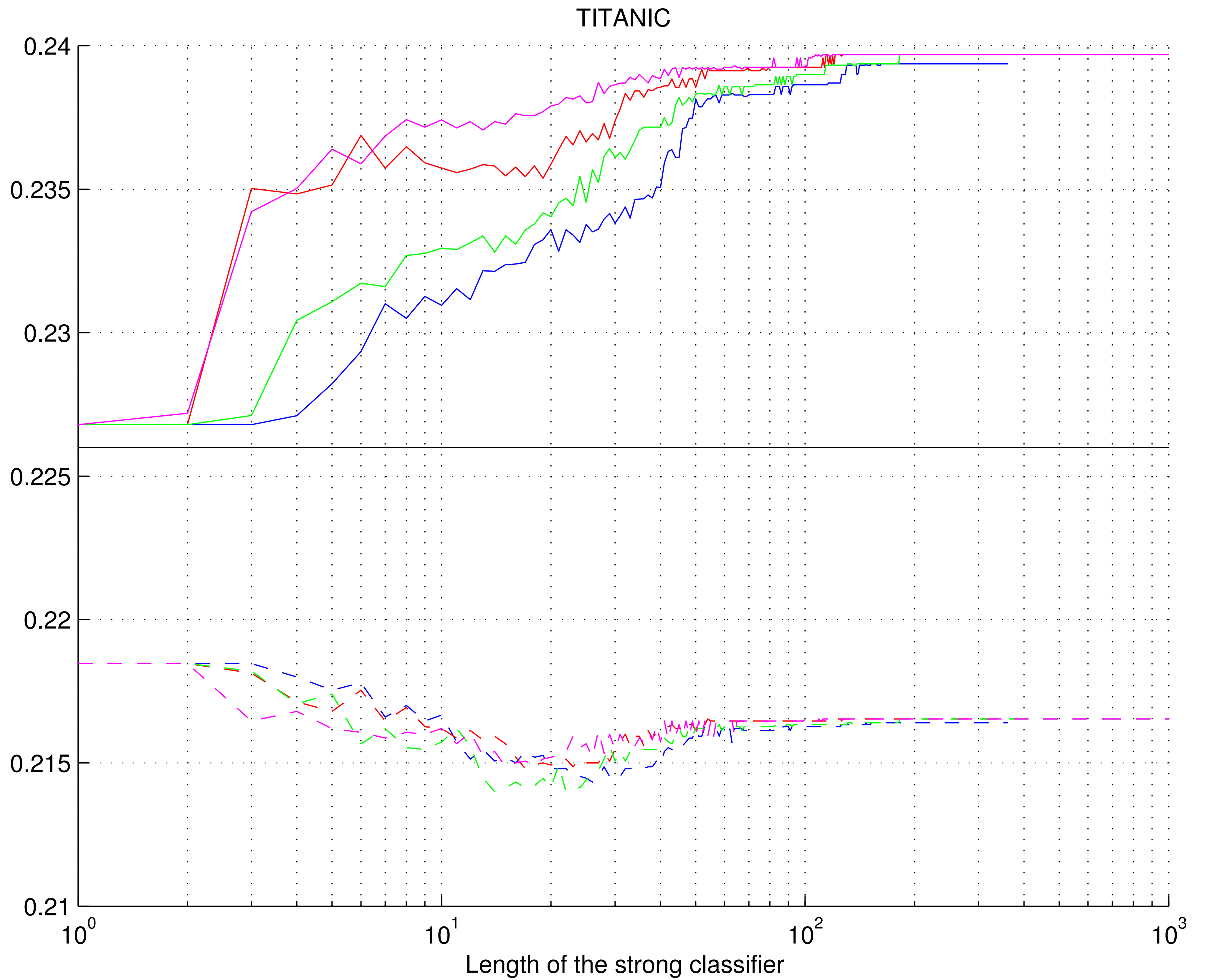


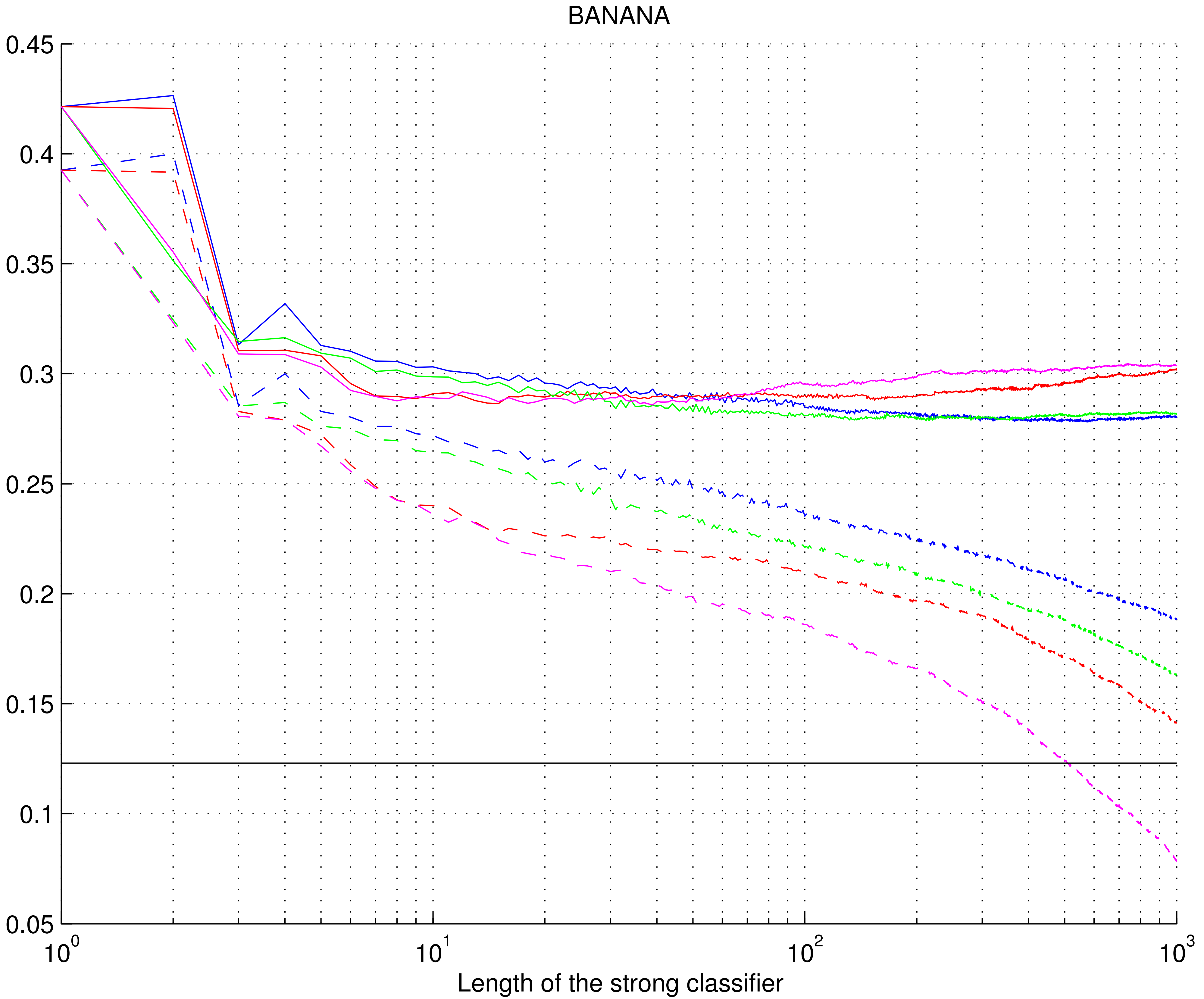
RINGNORM
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier

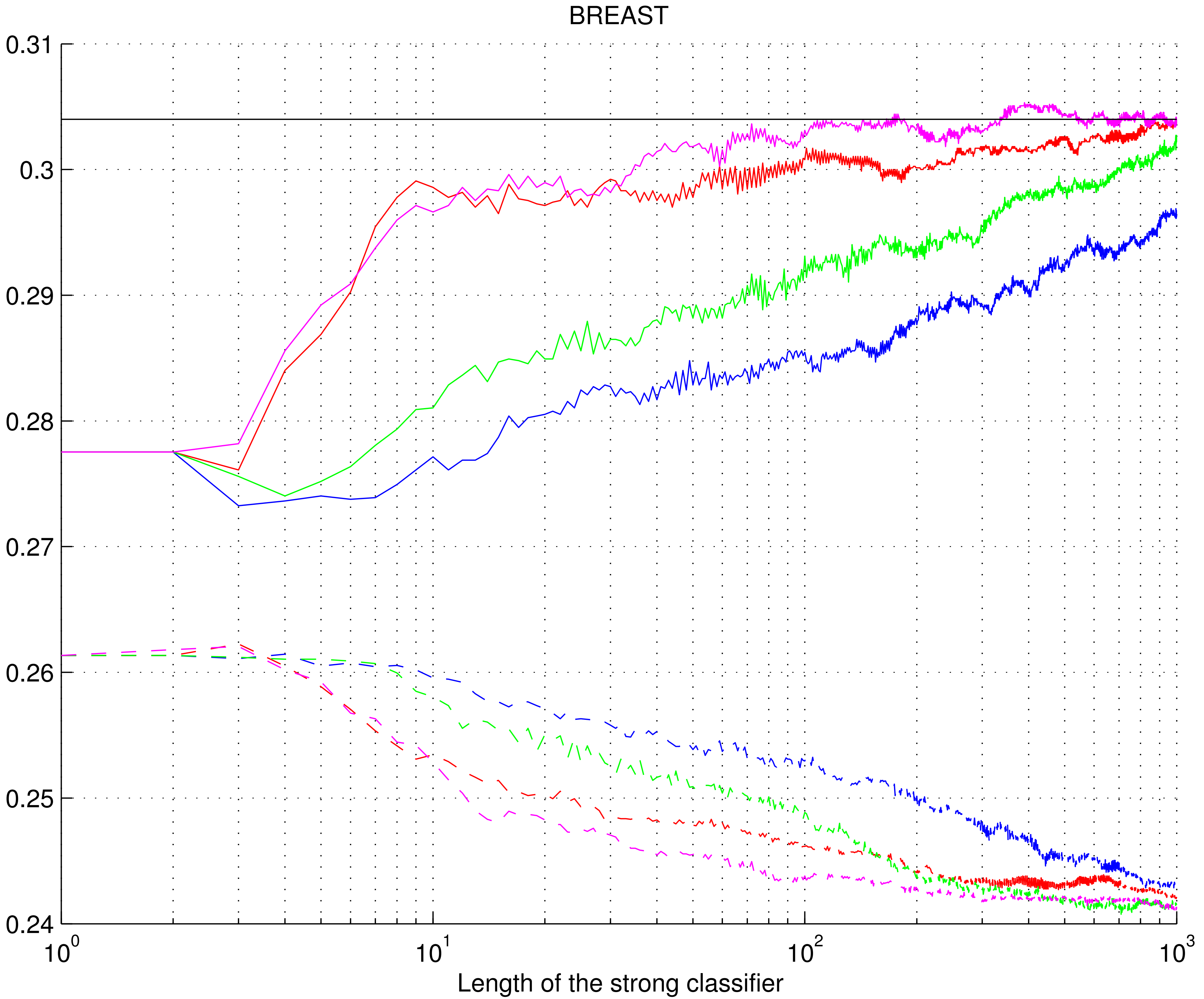

SPLICE
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier

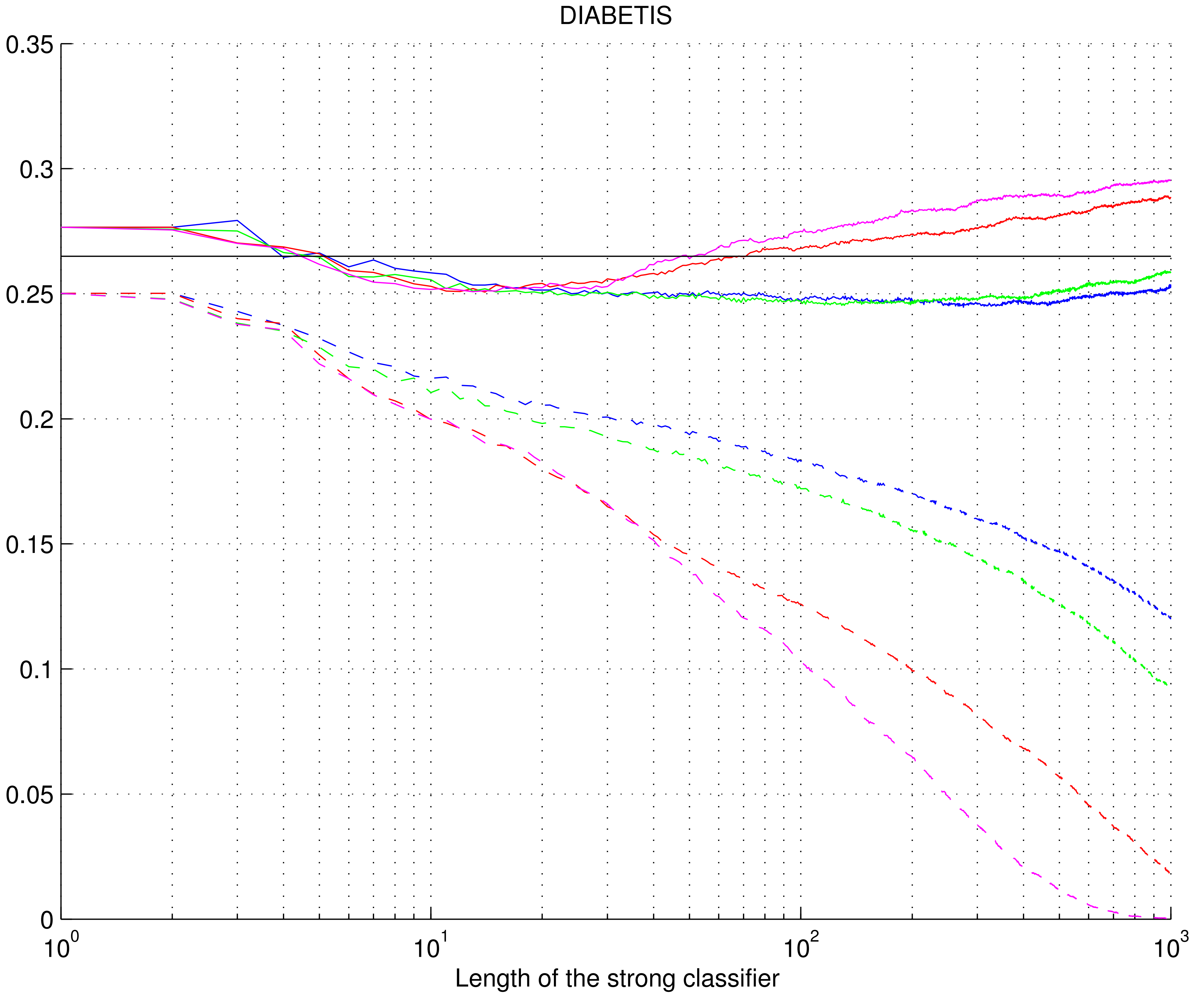


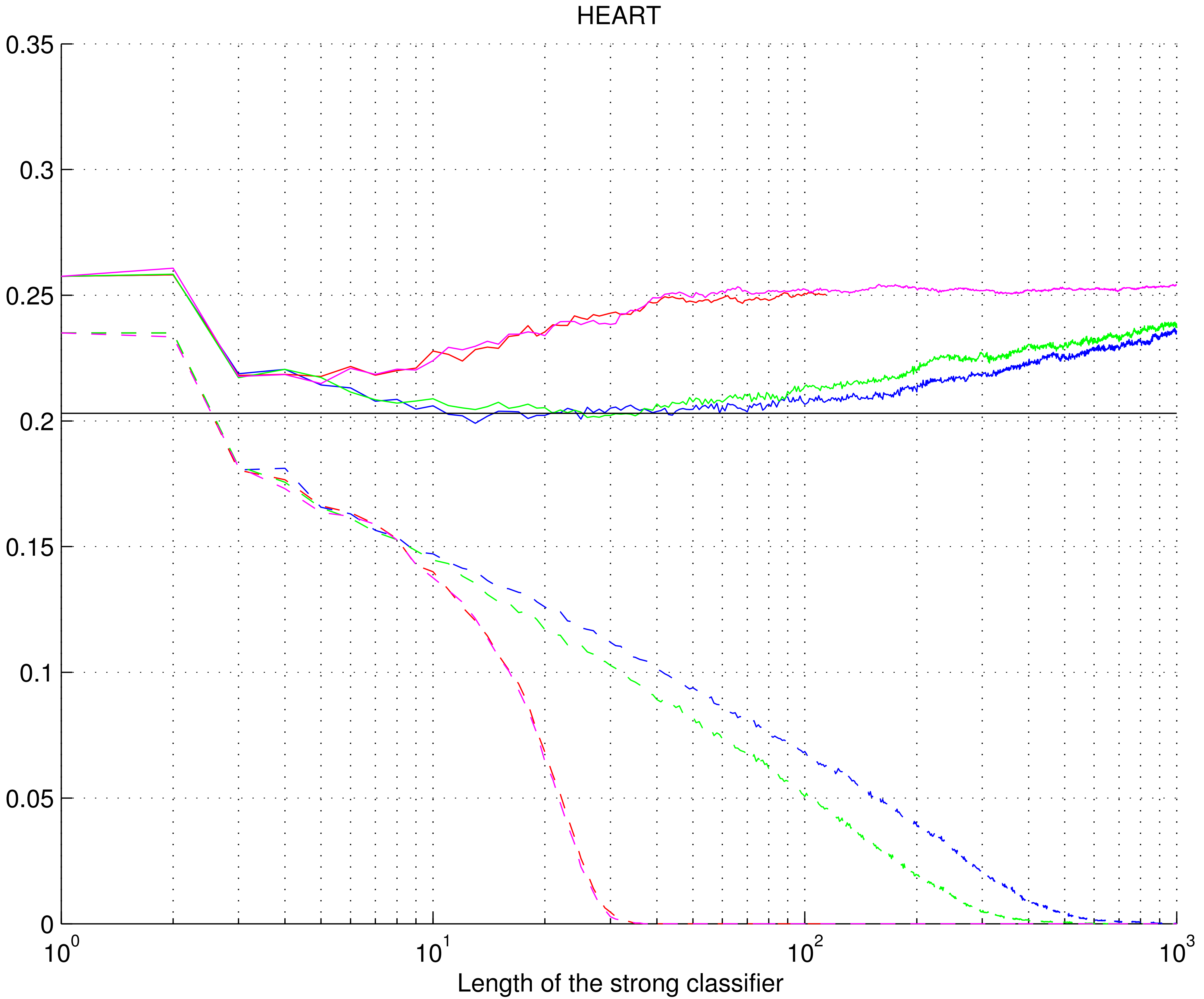
THYROID
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier


TITANIC
0.24
0.235
0.23
0.225
0.22
0.215
0.21
10^0
10^1
10^2
10^3
Length of the strong classifier


BANANA
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
10^0
10^1
10^2
10^3
Length of the strong classifier


BREAST
0.31
0.3
0.29
0.28
0.27
0.26
0.25
0.24
10^0
10^1
10^2
10^3
Length of the strong classifier


DIABETIS
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier


HEART
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier

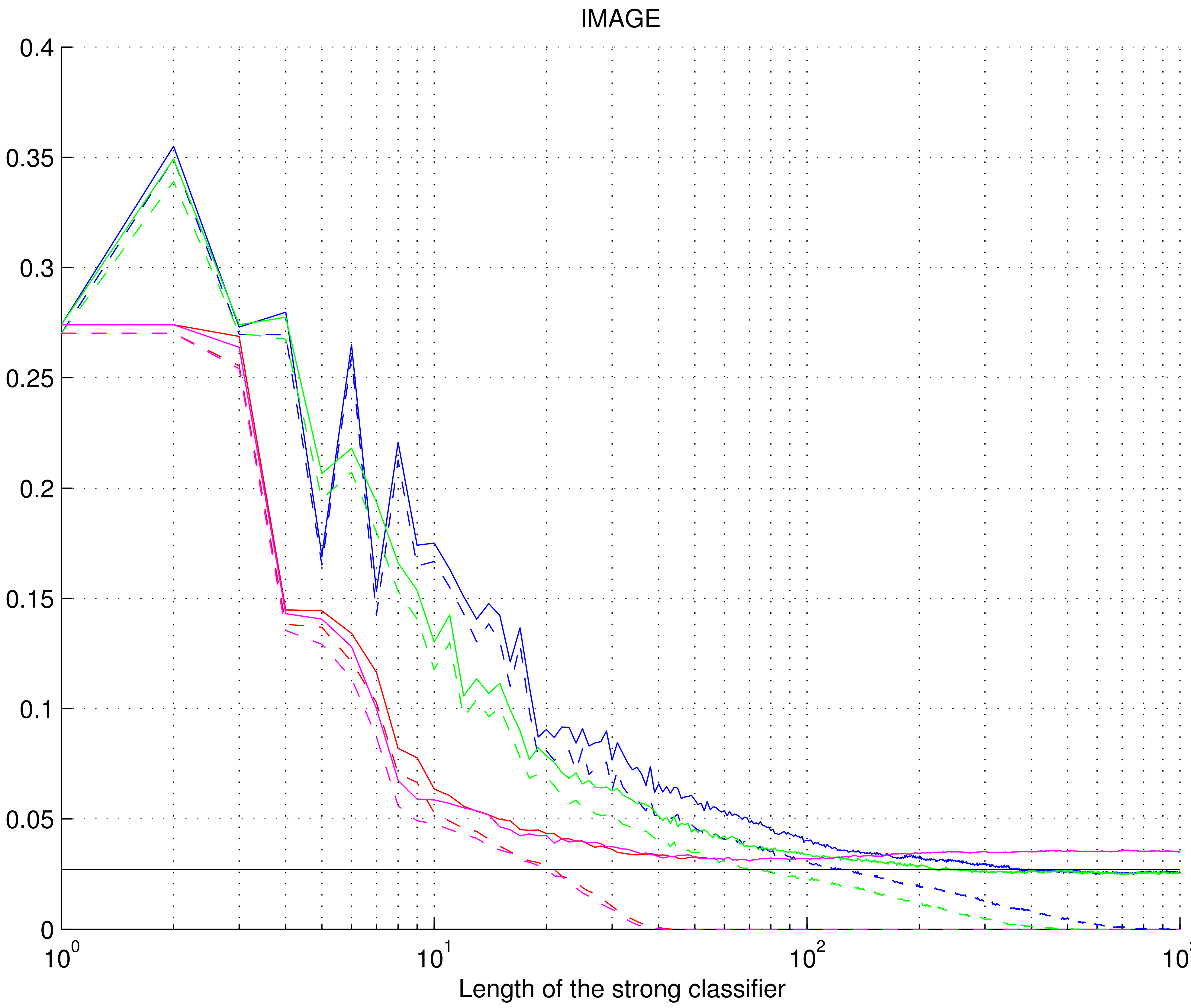
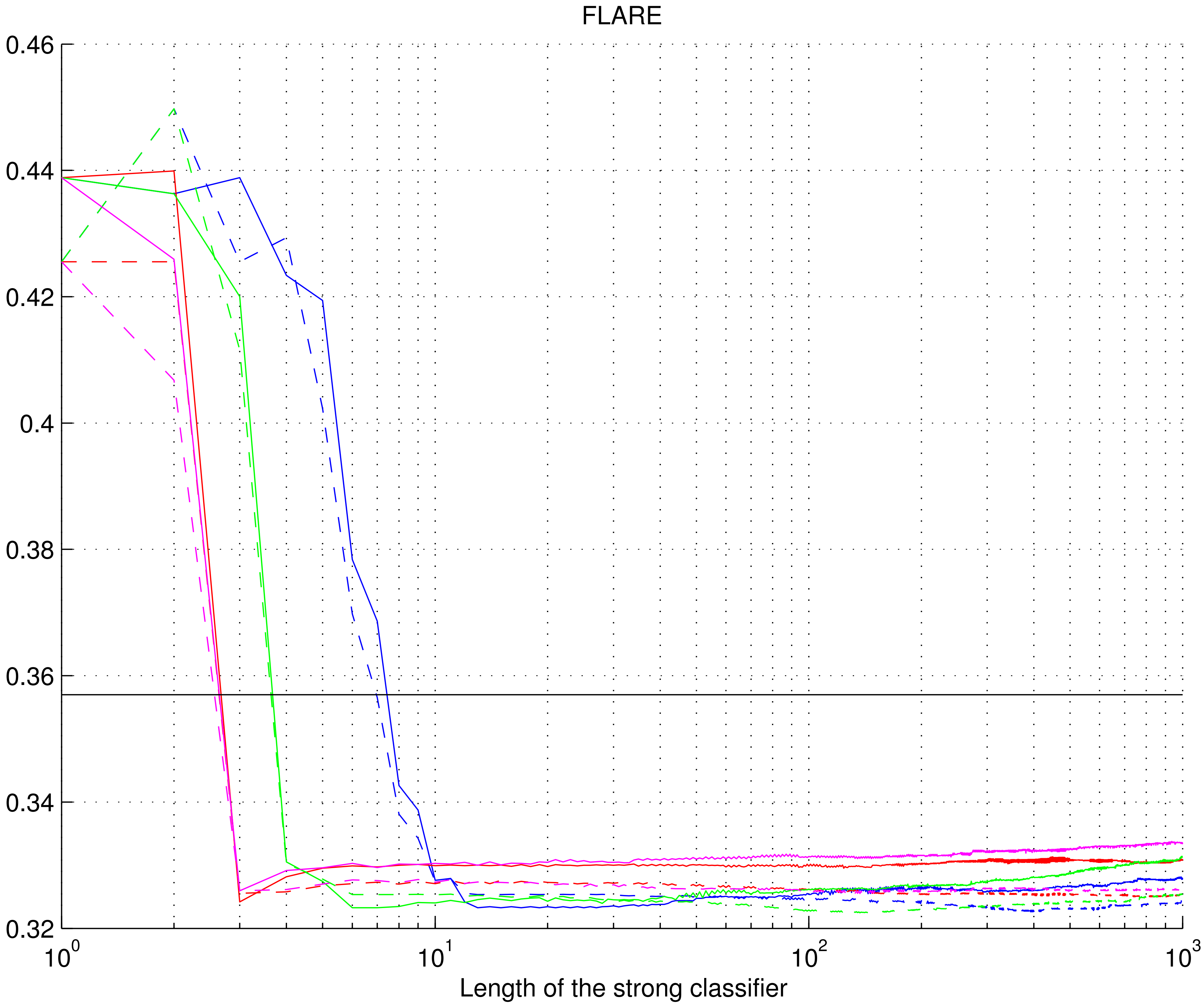
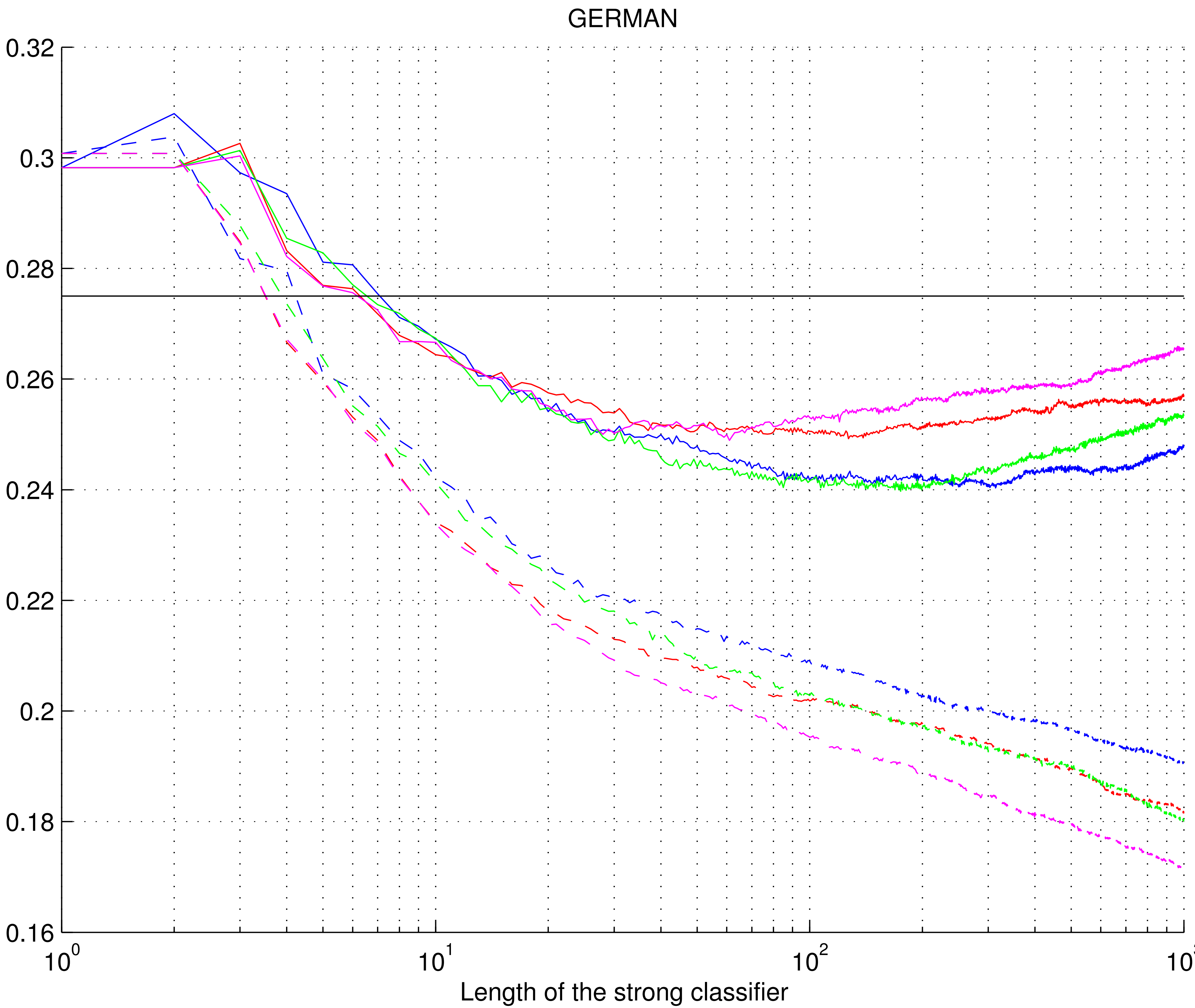
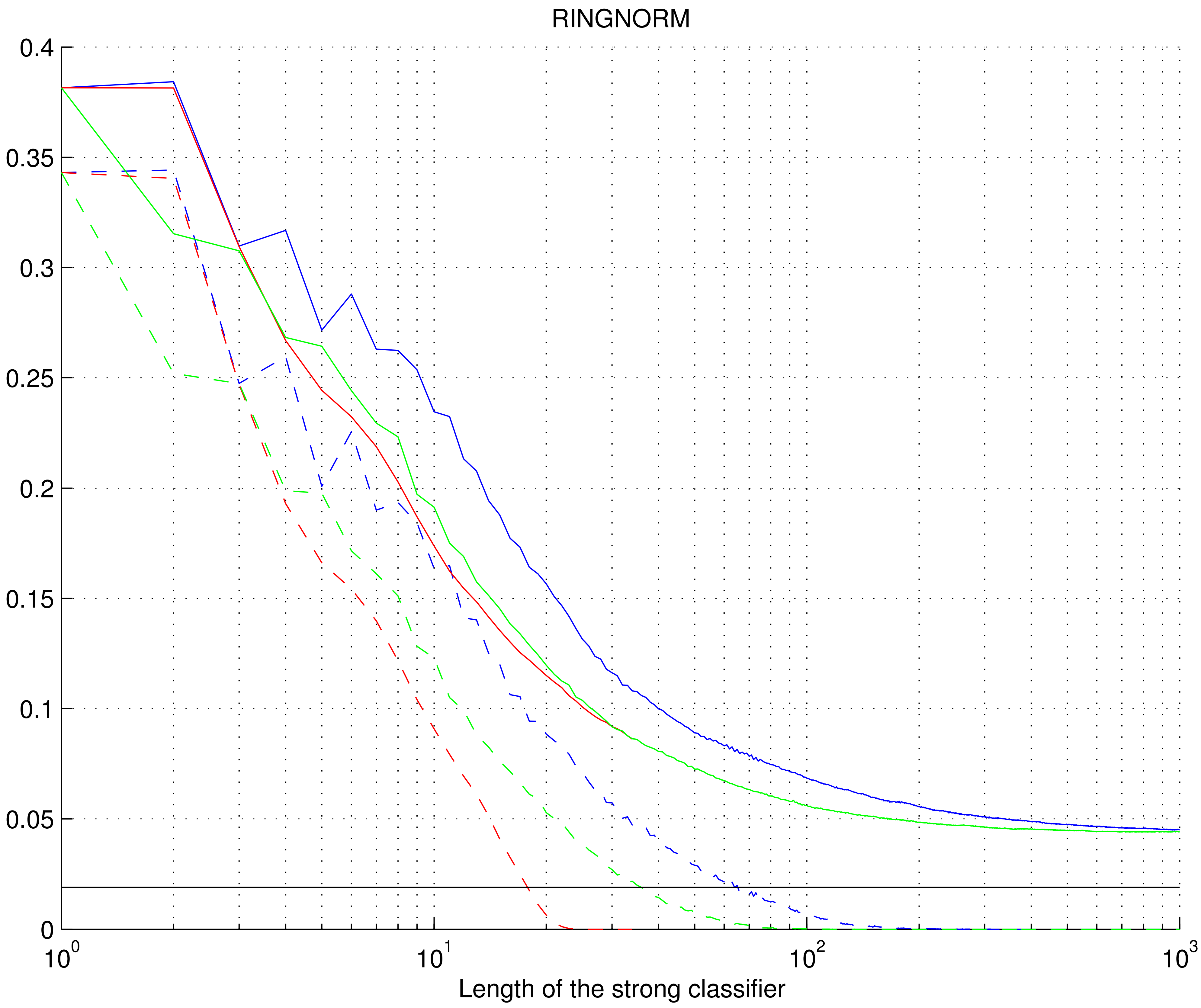


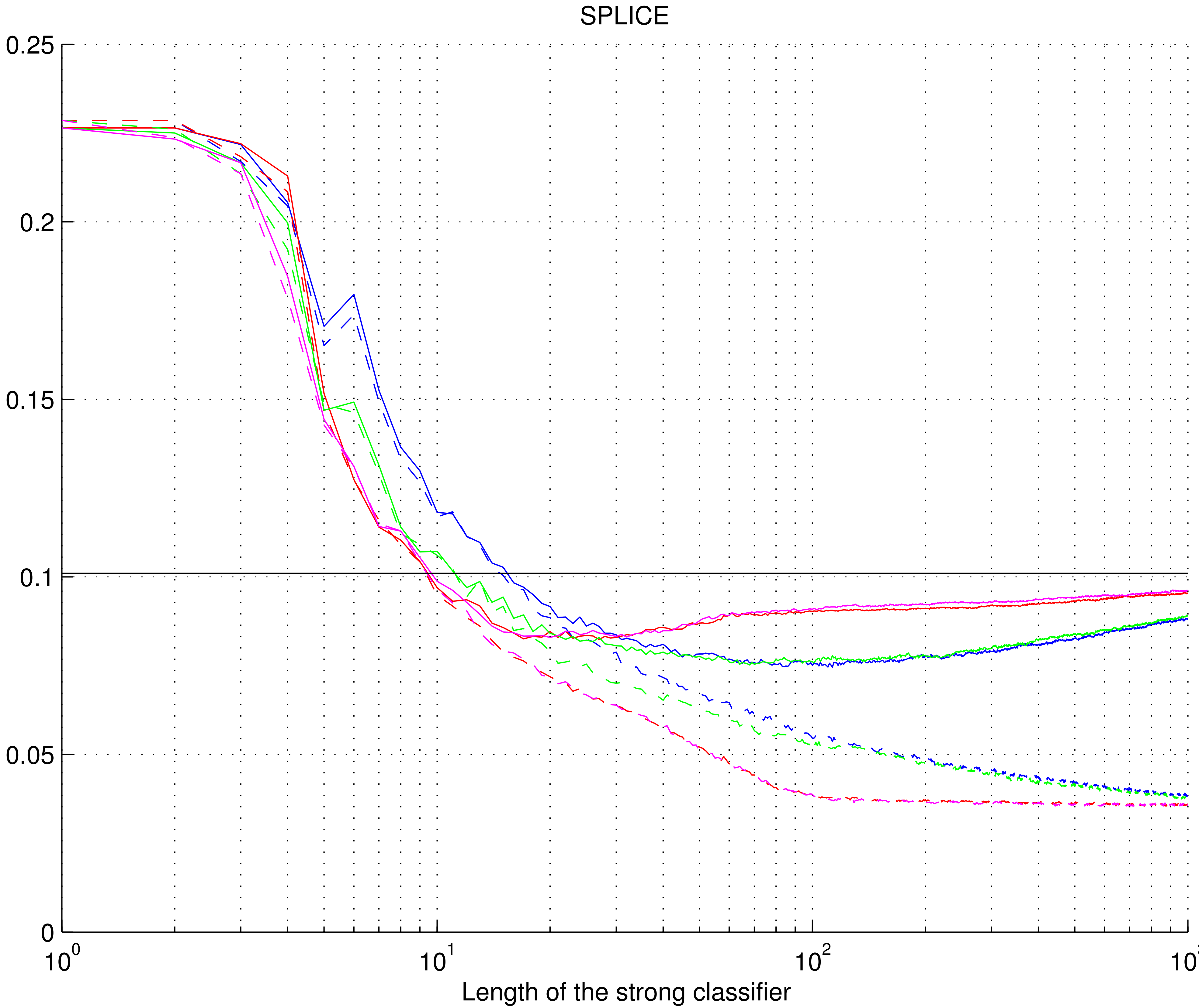
IMAGE
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier

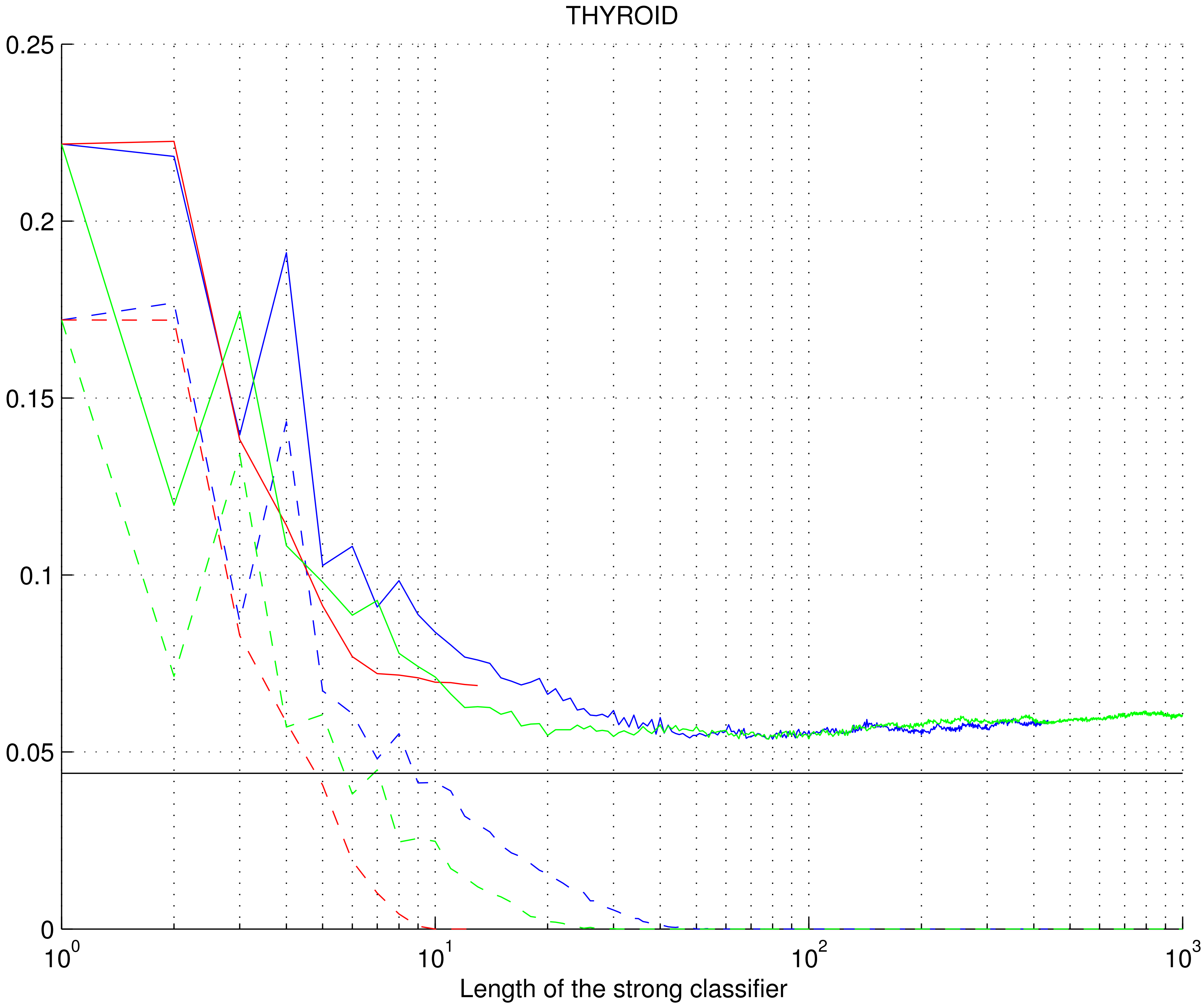

FLARE
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0.32
10^0
10^1
10^2
10^3
Length of the strong classifier

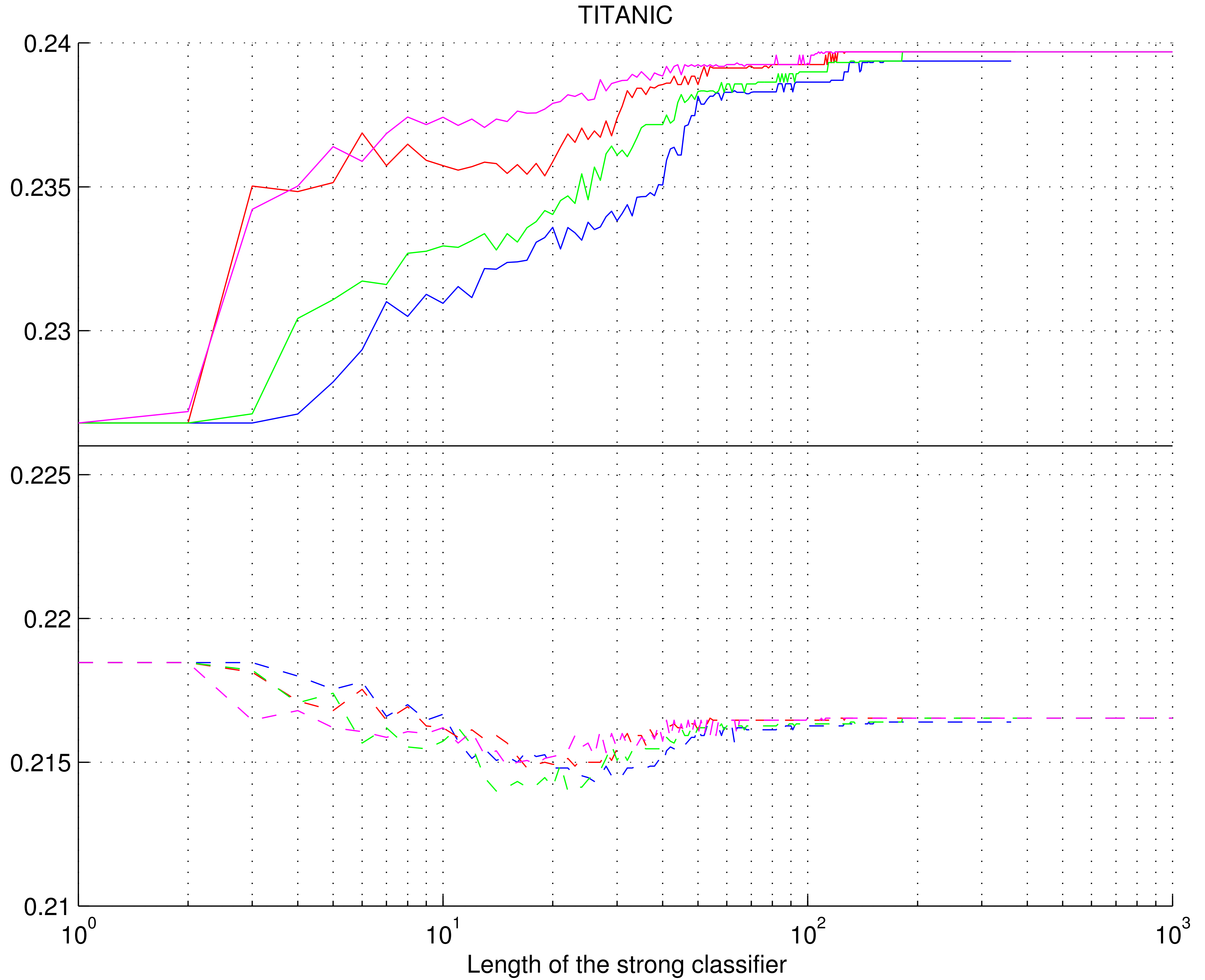

GERMAN
0.32
0.3
0.28
0.26
0.24
0.22
0.2
0.18
0.16
10^0
10^1
10^2
10^3
Length of the strong classifier

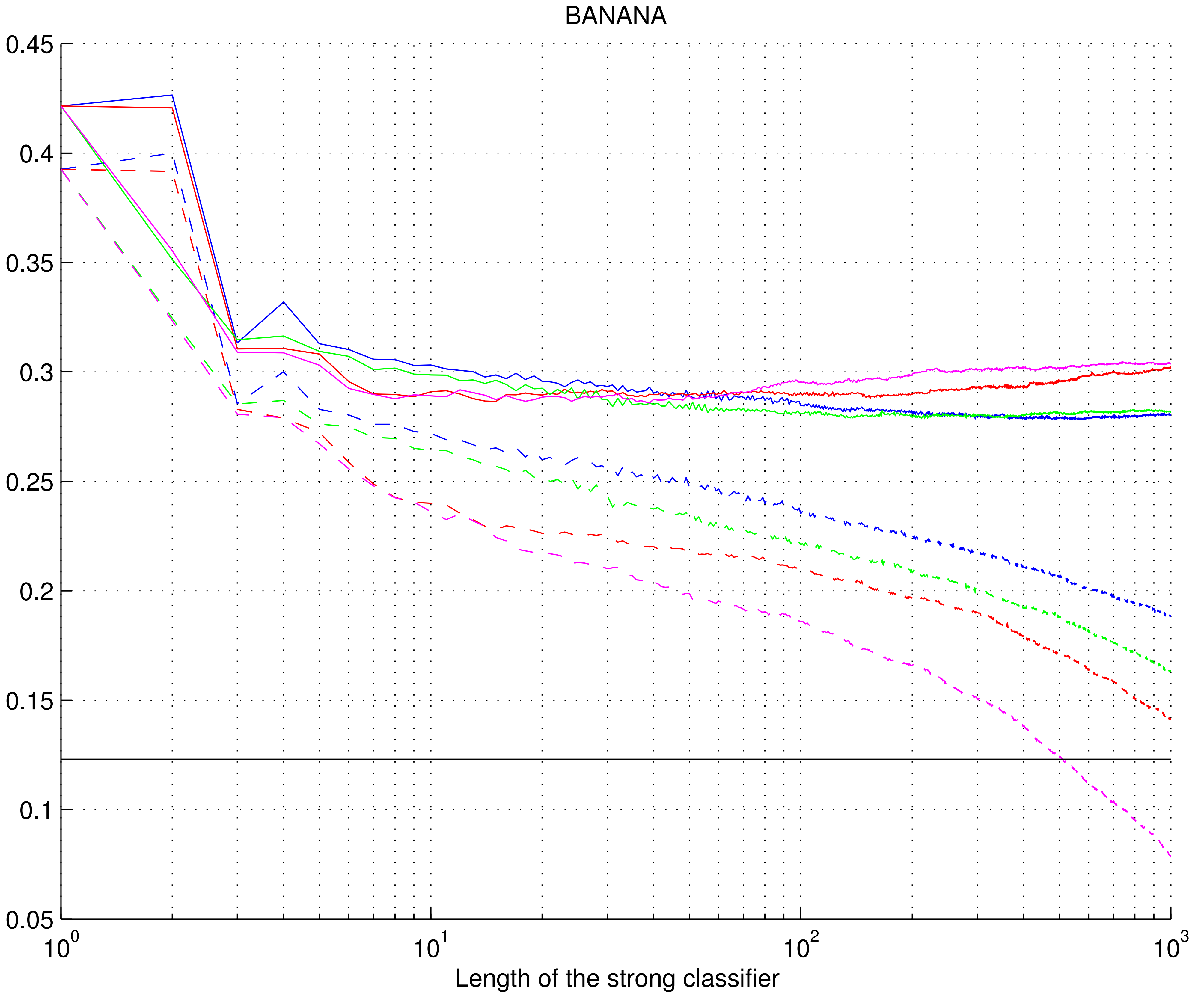


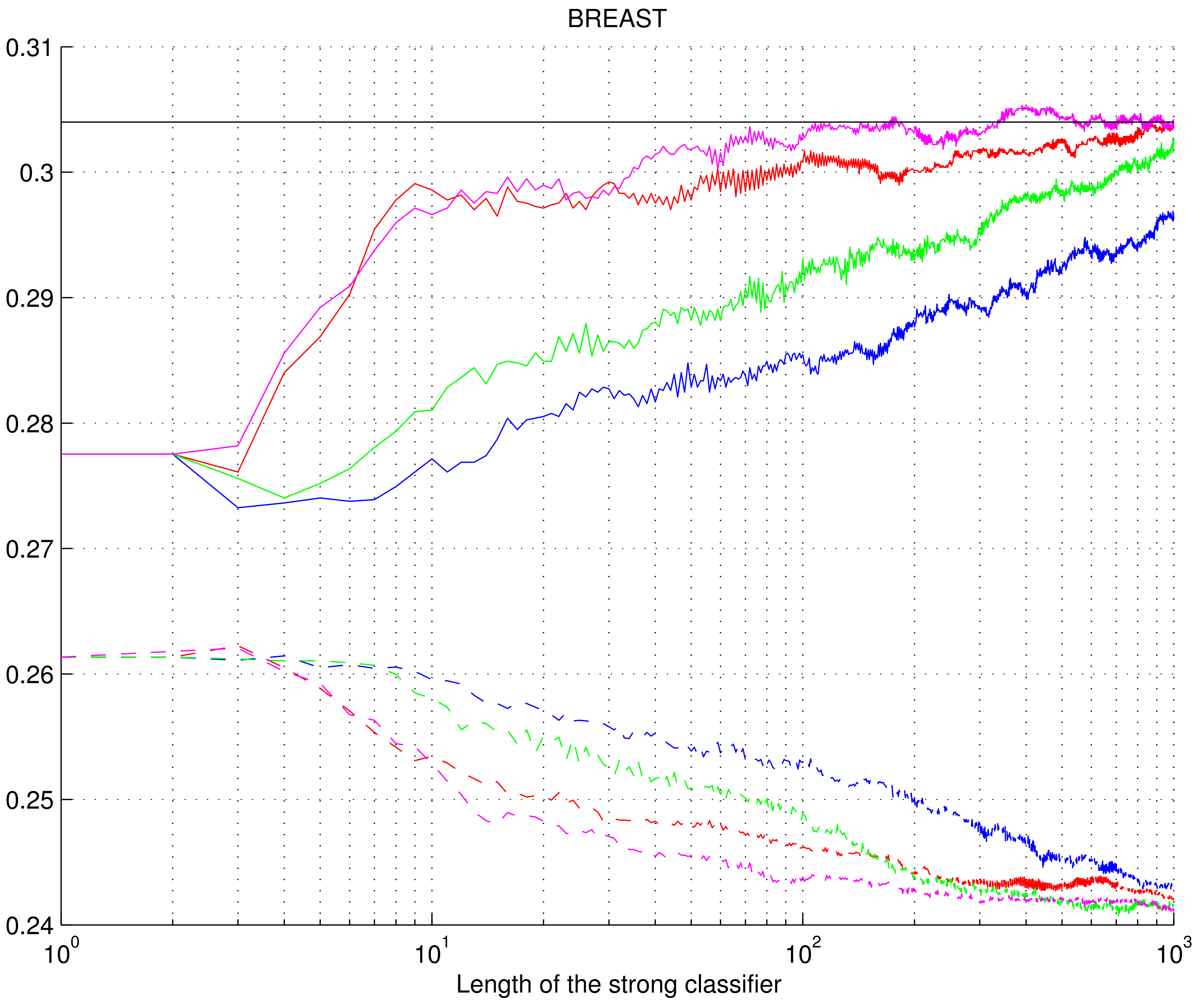
RINGNORM
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier


SPLICE
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier

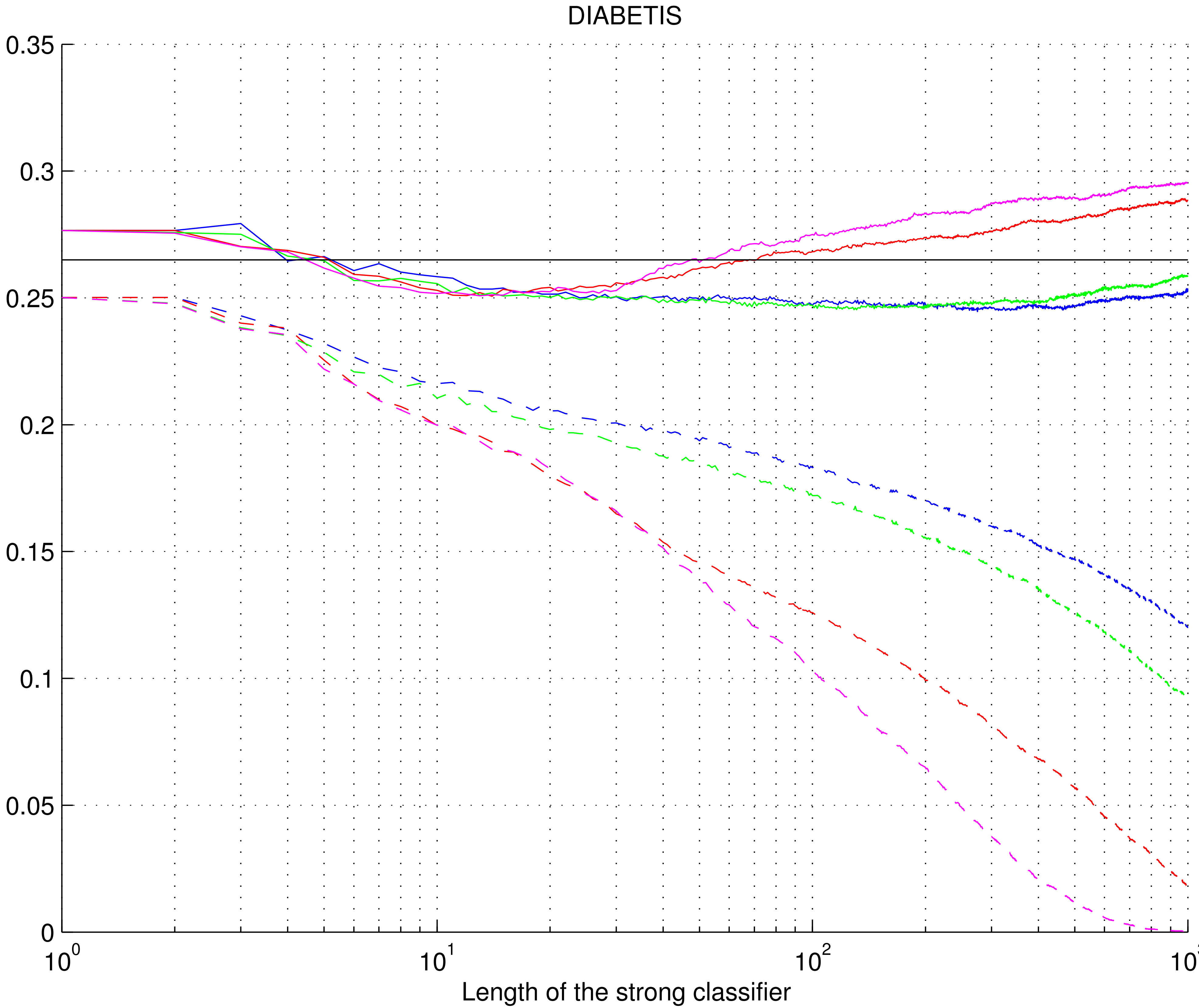

THYROID
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier


TITANIC
0.24
0.235
0.23
0.225
0.22
0.215
0.21
10^0
10^1
10^2
10^3
Length of the strong classifier

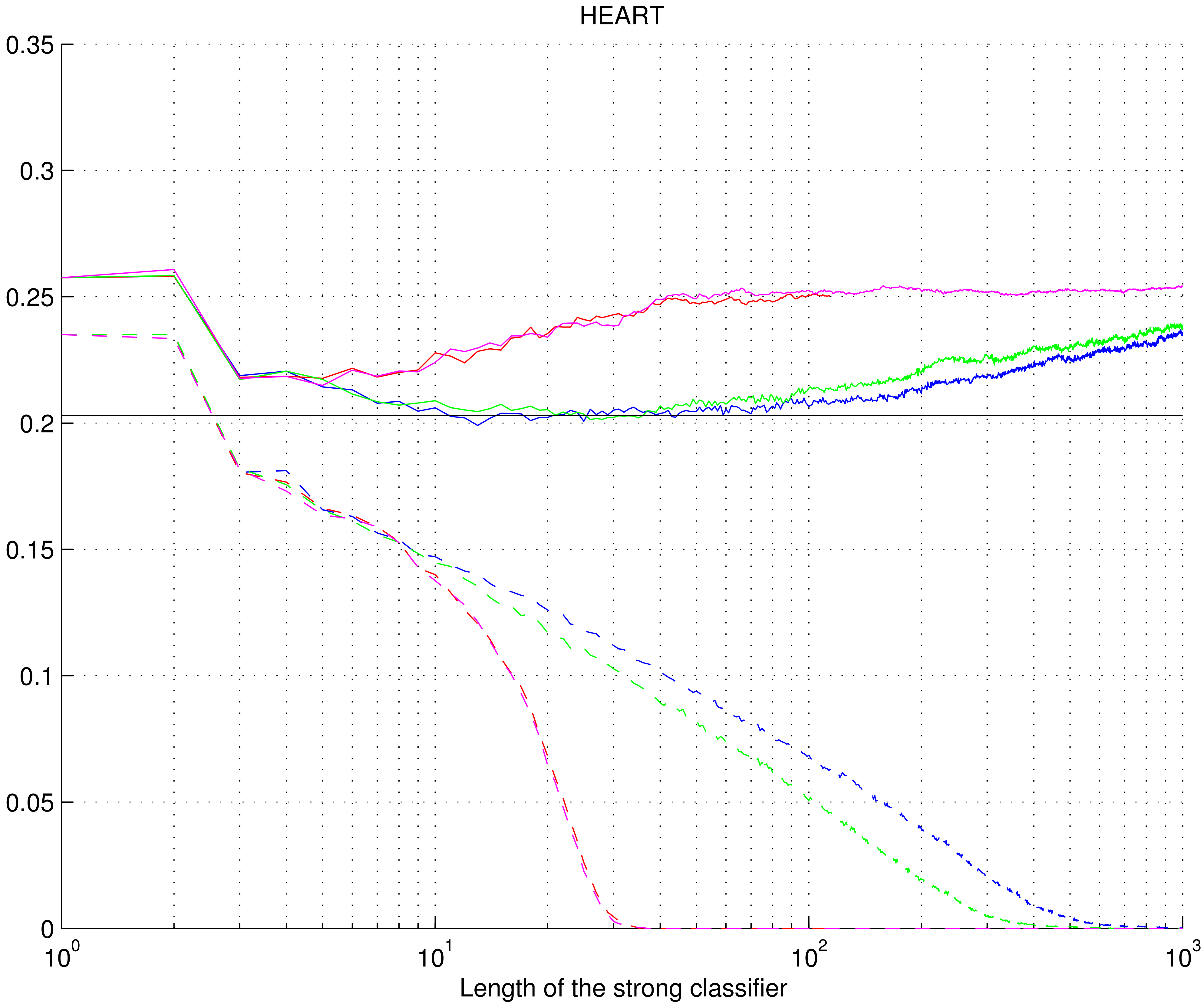

BANANA
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
10^0
10^1
10^2
10^3
Length of the strong classifier



BREAST
0.31
0.3
0.29
0.28
0.27
0.26
0.25
0.24
10^0
10^1
10^2
10^3
Length of the strong classifier



DIABETIS
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier


HEART
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2
10^3
Length of the strong classifier