Komiteemaschinen: Boosting und Bagging

Volker Tresp

Boosting: Motivation

- Wie mache ich aus einem schwachen Lerner (z.B. Entscheidungsbäumen) einen starken Lerner?
- Boosting: Verbesserung der Prognosegenauigkeit von instabilen Vorhersagesystemen
- Die Vorhersagegenauigkeit kann entscheidend verbessert werden durch Boosting (Hastie: “one of the most powerful learning ideas introduced in the last ten years”)

Beispiel: gewichtete Basisfunktionen können universelle Lernmaschinen sein

- Die gewichtete Glaussglocke $\alpha \exp(-\frac{1}{2\sigma^2}|\mathbf{x}-\mathbf{c}|^2)$ kann nur glaussglockenähnliche Funktionen gut approximieren
- Eine lineare Kombination von Gaussglocken

$$f(\mathbf{x}) = \sum_l \alpha_l \exp\left(-\frac{1}{2\sigma^2}|\mathbf{x}-\mathbf{c}_l|^2\right)$$

  wird zum universellen Approximator
- Wesentliche Frage: wie finde ich eine kleine Anzahl optimaler Basis-Lerner (Gaussglocken, Neuronen, Entscheidungsbäume, ...)

Boosting: Repräsentation

- Sei $f(\mathbf{x})$ der Ausgang eines Klassifikationssystems; betrachten wir binäre Klassifikation mit Klassen ± 1. Die Klassifikationsentscheidung erfolgt durch eine gewichtete Mittelung der Form

$$f(\mathbf{x}) = \text{sign}\left[\sum_{l=1}^{L} \alpha_l f_l(\mathbf{x})\right]$$

- $f_l(\mathbf{x})$ bezeichnet die Vorhersage des l-ten Basis-Klassifikators; $\alpha_l \geq 0$ ist das Gewicht des l-ten Basis-Klassifikators

- Im Folgenden betrachten wir als Basislerner Entscheidungsbäume

Boosting: Grundidee

- $f_1(\mathbf{x})$ ist das System, welches auf den originalen Trainingsdaten trainiert wurde
- Im Folgenden werden Daten höher gewichtet für $f_l(\mathbf{x})$, mit denen das gegenwärtige System (oder der Vorgänger $f_{l-1}(\mathbf{x})$) Probleme hatte
- Klassifikatoren $f_l(\mathbf{x})$, die eine bessere Performanz auf den Trainingsdaten erzielen, werden durch ein größeres α_l höher gewichtet

Boosting: AdaBoost.M1

- Initialisiere Gewichte: $w_j = 1/N : j = 1, \ldots, N$
- Für $l = 1, \ldots, L$
  - Trainiere einen Klassifikator $f_l(\mathbf{x})$ mit Datengewichten w_j durch Minimierung der gewichteten Missklassifikationsrate $\sum_{j \in E_l} w_j$, wobei $E_l = \{j : f_l(\mathbf{x_j}) \neq y_j\}$
  - Berechne den normalisierten Fehler

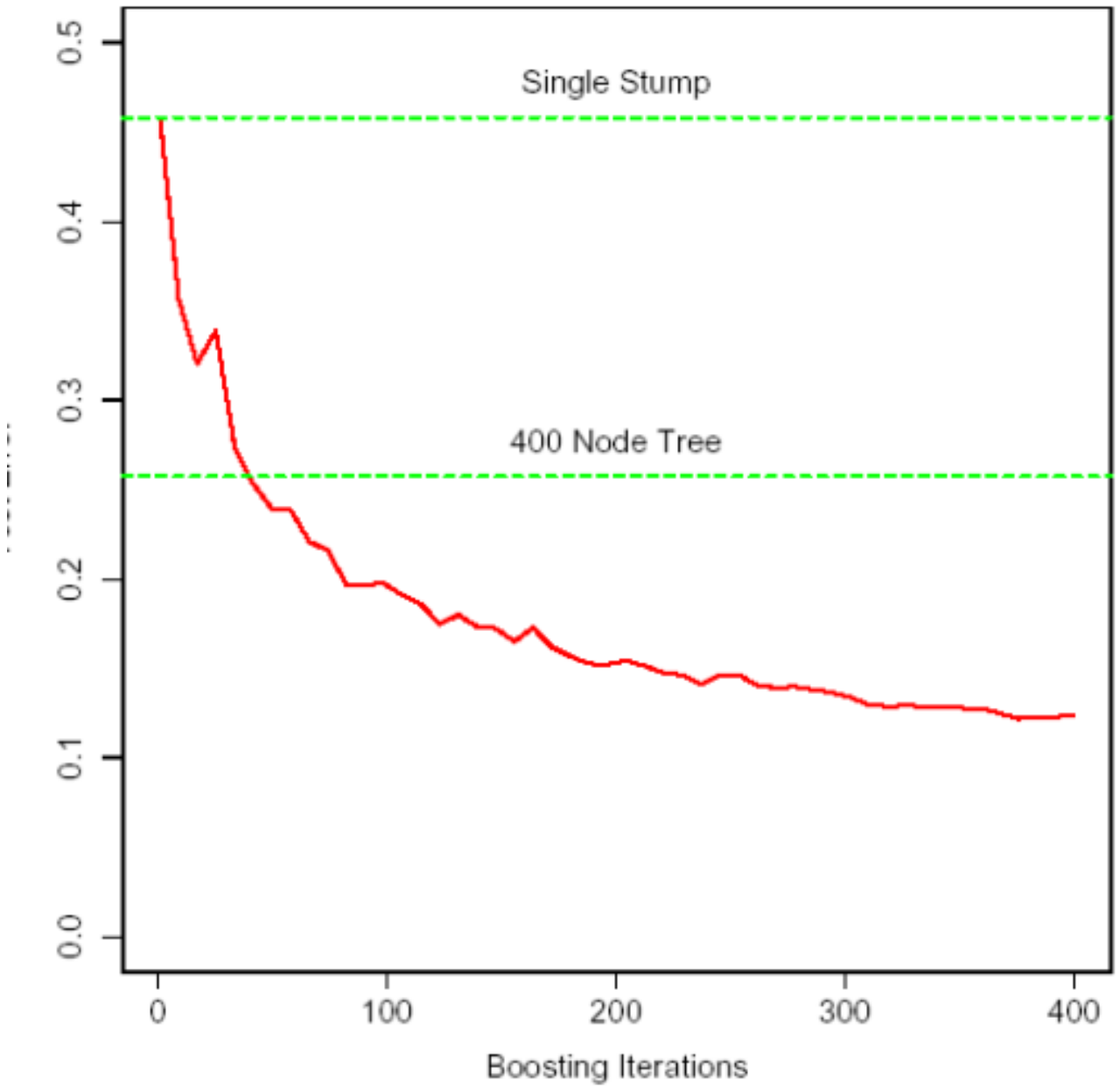
$$\mathrm{err}_l = \frac{\sum_{j \in E_l} w_j}{\sum_{j=1}^{N} w_j}$$

  - Berechne $\alpha_l = \log \frac{1-\mathrm{err}_l}{\mathrm{err}_l}$ (Beachte: $\mathrm{err}_l \leq 0.5$)
  - Berechne neue Datengewichte für alle $\mathbf{x}_j$ mit $j \in E_l$: $w_j \leftarrow w_j e^{\alpha_l}$

Abschließend:

$$f(\mathbf{x}) = \mathrm{sign}\left[\sum_{l=1}^{L} \alpha_l f_l(\mathbf{x})\right]$$

Resultate auf einem künstlichen Datensatz (Boosting mit Stumps)


Single Stump
400 Node Tree
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400
Boosting Iterations

Boosting CART

- Beste "off-the-shelf" Prozedur für Datamining
- Erhält die meisten der positiven Eigenschaften von CART
- Allerdings: langsamer, geringere Interpretierbarkeit
- Man arbeitet oft mit minimalen Bäumen: Baumstümpfe (Stumps) mit nur einem Split!
- Spam Daten: 4.0 % Klassifikationsfehler

Boosting CART: Vorgehensweise

- CART
  - Zum explorativen Datamining
  - Zur Kommunikation und zur Analyse des Lernergebnisses
- Boosting CART
  - Zur Optimierung der Prognosegenauigkeit
  - Zum operativen Einsatz

Boosting: Kostenfunktion

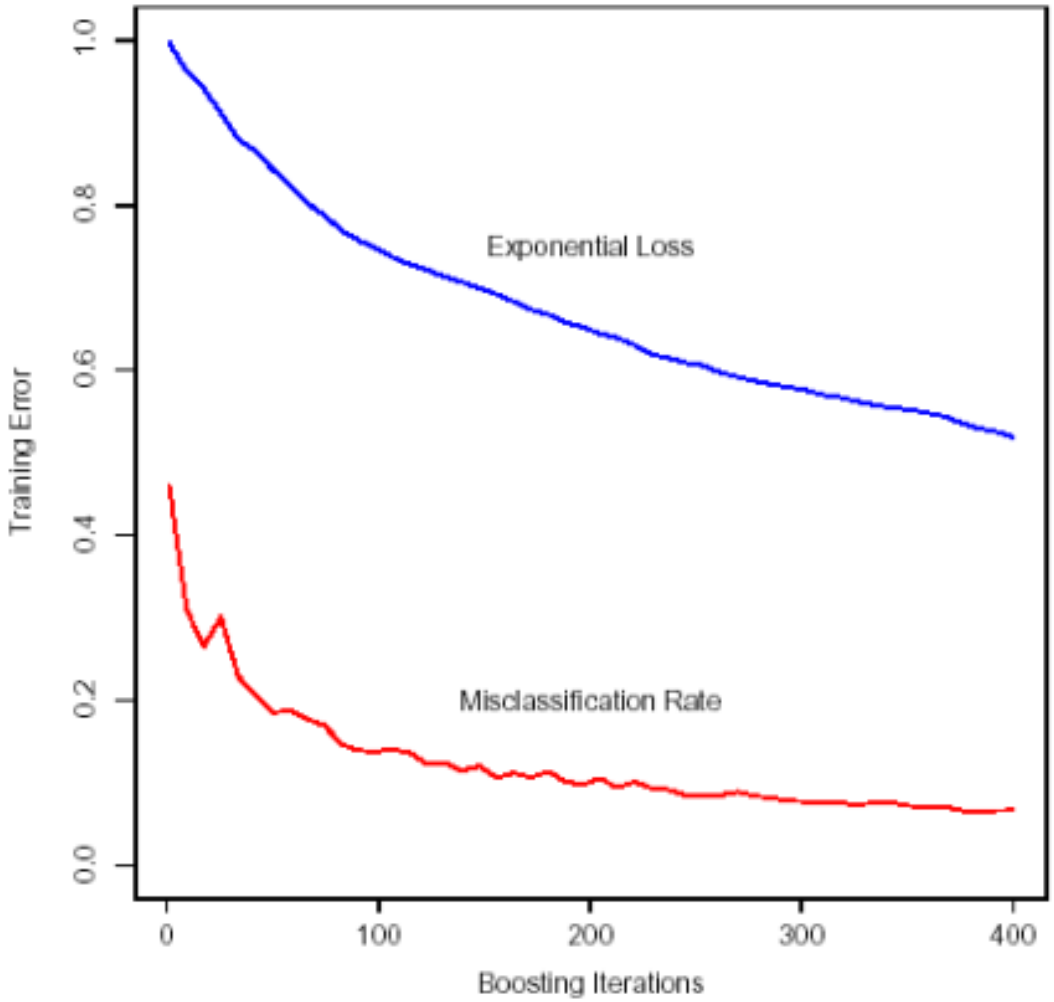
- Die Kostenfunktion, die minimiert werden soll ist

$$C_L = \sum_{j=1}^{N} \exp\left(-y_j F_L(\mathbf{x}_j)\right)$$

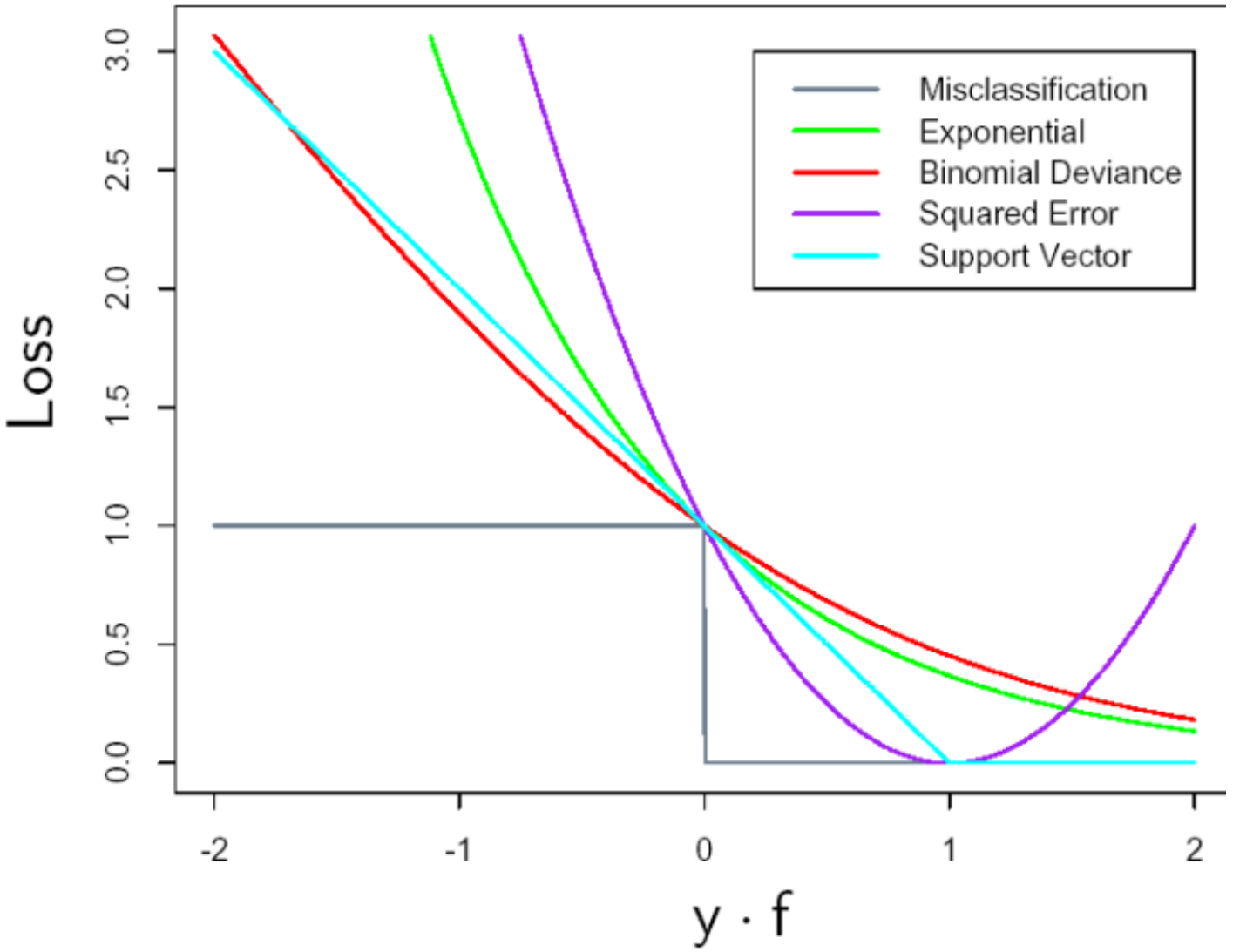
- wobei $F_l(\mathbf{x}) = \sum_{i=1}^{l} \beta_i f_i(\mathbf{x})$

Boosting: wesentliche Vorteile

- Schon eingestellte Basislerner und deren Gewichtungen ändern sich nicht im Späteren
- Boosting führt zu exzellenten Resultaten
- Boosting maximiert den *margin* $yf(\mathbf{x})$; Schwergewicht liegt auf falsch prognostierten Datenpunkten (daher gelegentlich Probleme bei Ausreißern)
- Bei separierbaren Aufgaben ist Ziel die Maximierung des *margins*



igure 10.3: *Simulated data, boosting with stumps: isclassification error rate on the training set, and av-rage exponential loss:* $(1/N)\sum_{i=1}^{N}\exp(-y_i f(x_i))$


Loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
-2
-1
0
1
2
y · f
Misclassification
Exponential
Binomial Deviance
Squared Error
Support Vector

Varianten

- Andere Kostenfunktionen: z.b.: *binomial deviance*

$$\log[1 + \exp(-2yf(x))]$$

- Erweiterung für Regression
- MART: boosting mit Bäumen für eine beliebige Kostenfunktion
- Regularisierung der Basislerner

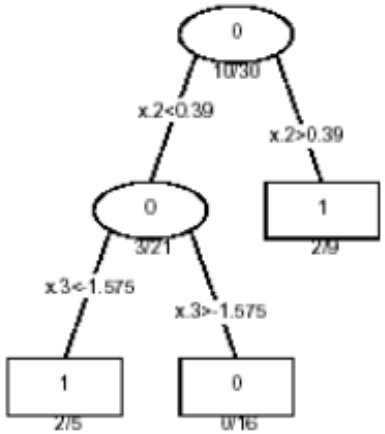
Bagging

- **B**ootstrap **Agg**regation
- Regression:

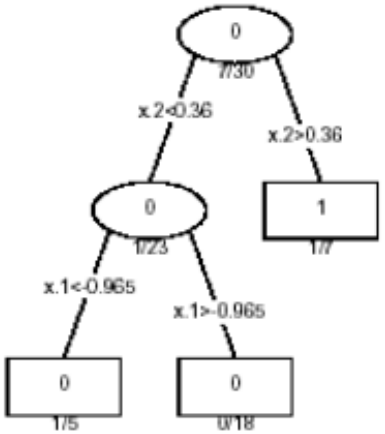
$$f_{bagg}(\mathbf{x}) = \frac{1}{L}\sum_{l=1}^{L} f_l^{boot}(\mathbf{x})$$

- $f_l^{boot}(\mathbf{x})$ wir mit einem *bootstrap sample* der Trainingsdaten trainiert
- Bei einem Bootstrap sample zieht man N mal zufällige Trainingsbeispiele, wobei der gezogene Datenpunkt wieder zurückgelegt wird: *sampling with replacement*
- Klassifikation: Mehrheitsentscheidung der Basisklassifikatoren

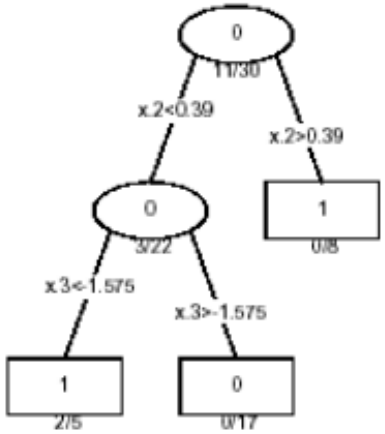
Original Tree



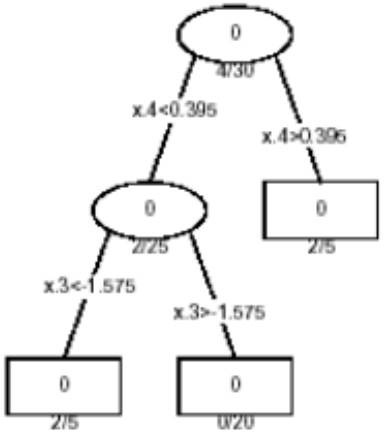
Bootstrap Tree 1

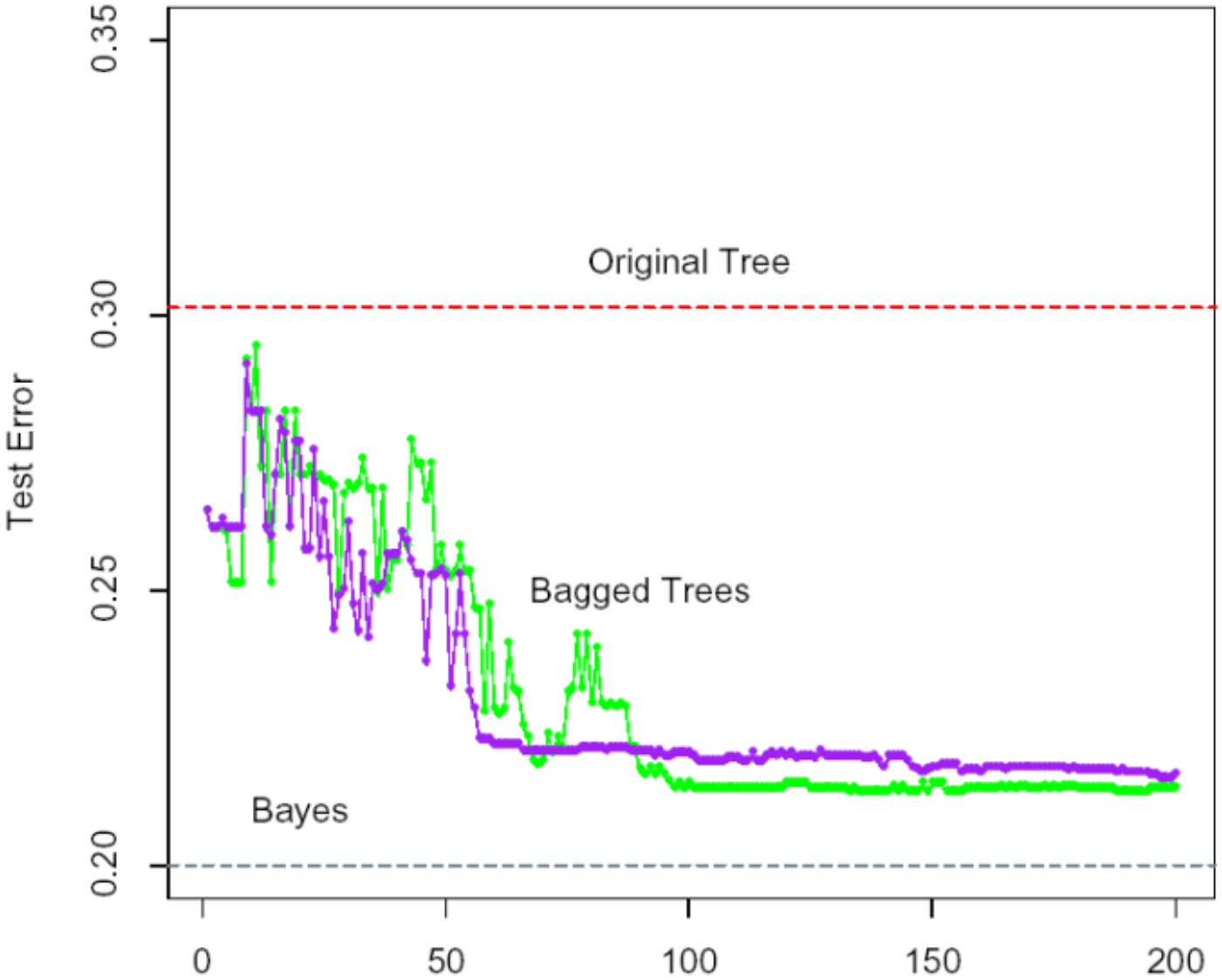


Bootstrap Tree 2

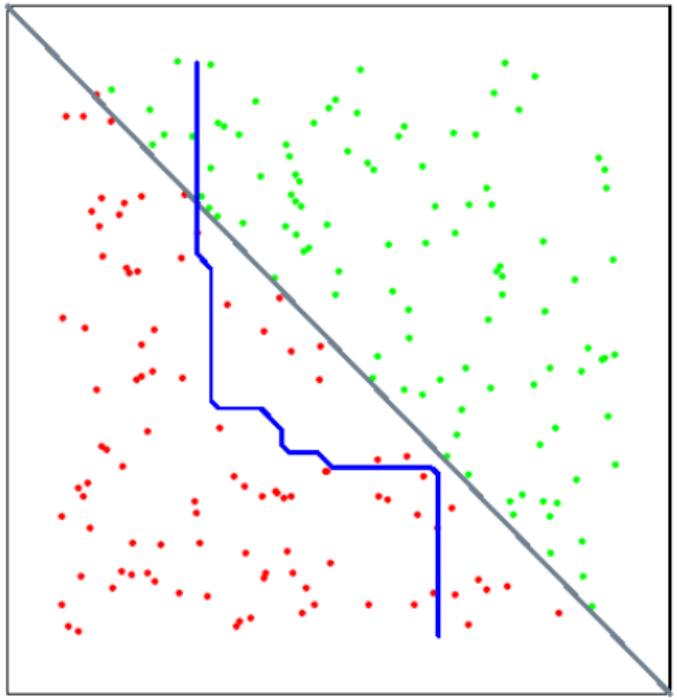


Bootstrap Tree 3




Test Error
0.35
0.30
0.25
0.20
Original Tree
Bagged Trees
Bayes
0
50
100
150
200

Bagged Decision Rule



Boosted Decision Rule