

# 732G12 Data Mining

## Föreläsning 6

---

Josef Wilzén

IDA, Linköping University, Sweden

- Bagging
- Boosting
- Random forest
- Gradient Boosting
- XGBoost

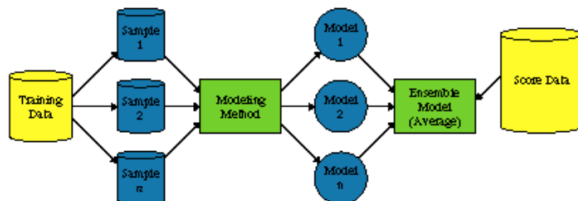
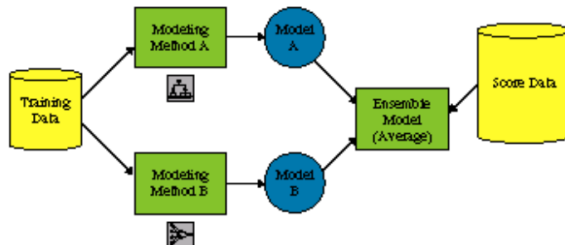
# Ensemblemetoder

Grundidén är att skatta många modeller på träningsdata och sen kombinera dessa för att göra prediktioner.

Finns många olika varainter:

- Göra slumpmässiga urval från träningsdata och skatta modeller på dessa urval (Bootstraping/Bagging)
- Skapa en mängd dataset där observationerna har olika vikter i modellanpassningen i olika dataset (Boosting)
- Skatta ett antal olika modeller (kan vara av helt olika sort) och sen kombinera dessa vid prediktioner
- Vissa modeller har slumpmässiga element vid optimieringen (tänk neurala nätverk): optimera samma modell flera gånger men med olika seed, vilket ger olika modeller/parameterar. Kombinera dessa sedan vid prediktionen.

# Ensemblemetoder



# Bootstraping

Idé: Skapa  $B$  stickprov av datan genom att **med återläggning** välja nya datapunkter. Använd dessa stickprov för att skatta modell eller funktioner.

Exempel:

Vi vill skatta  $\mathbb{V}(e^{\bar{X}})$ .

Skapa  $B$  stickprov.

Skatta  $T_k = e^{\bar{Z}_k}$  för  $k = 1, \dots, B$ .

Beräkna  $\mathbb{V}(\mathbf{T})$ .

Idé: Om man tar medelvärdet av oberoende observationer (modeller) så minskar variansen.

**Bagging (Bootstrap aggregating):** Använd Bootstrap för att skapa  $B$  träningsdataset och skatta en modell  $\hat{f}_b$  för varje av dessa set. Den slutgiltiga modellen får vi genom att ta medelvärdet av alla dessa modeller:

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(\mathbf{x}).$$

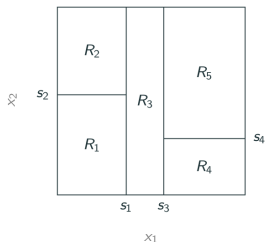
- Sänker variansen av den anpassade funktionen.
- Påverkas *mycket* av kvalitén av modellen. En bra modell blir bättre, en dålig blir sämre.
- För klassificering, använd majoritetsröstning.

# Classification and Regression Trees (CART)

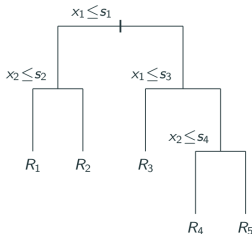
Vi delar upp variabelrummet genom att rekursivt göra binära uppdelningar.

För klassificering används vanligaste klassen, för regression medelvärdet inom regionen.

Partitioning of input space



Tree representation



Flexibilitet/komplexitet för trädmodeller beror på djupet.

Ett djupt träd ger litet bias, men mycket varians.

Förbättringar:

- Efterbeskärning (post-pruning):
  - Skapa ett djupt träd och beskär det till ett mindre (minska variansen).
- Ensamblemeter:
  - Ta ett genomsnitt över många trädmodeller.
  - Bagging
  - Random forest
  - Boosted trees



Bagging kan ge stora förbättringar för trädmodeller. Men det finns vissa problem:

- De  $B$  bootstrap-urvalen är korrelerade.
- Reduktionen i varians blir liten när vi tar medelvärde över korrelerade variabler.

Idé: Avkorrelera de  $B$  trädmodellerna genom att göra slumpmässiga ändringar av modellerna.

- Använd bagging för att skatta  $B$  träd,
  - Vid varje uppdelning/regel använd endast en slumpmässig delmängd  $q \leq p$  av de förklarande variablerna.
- Tumregel: (Förslag från Leo Breiman)
  - Klassificering:  $q = \sqrt{p}$
  - Regression:  $q = p/3$

Slumpmässigt val av variabler leder till:

- Minskar bias, men ofta mycket långsamt.
- Läger till varians till varje träd.
- + Avkorrelerar träden.

Ofta dominerar den avkorrelerade effekten vilket leder till att MSE minskar på testdata.

Beräkningsmässiga fördelar:

- Lätt att parallellisera.
- $q < p$  minskar kostanden för vaje regel/uppdelning.
- Inte så många hyperparametrar.

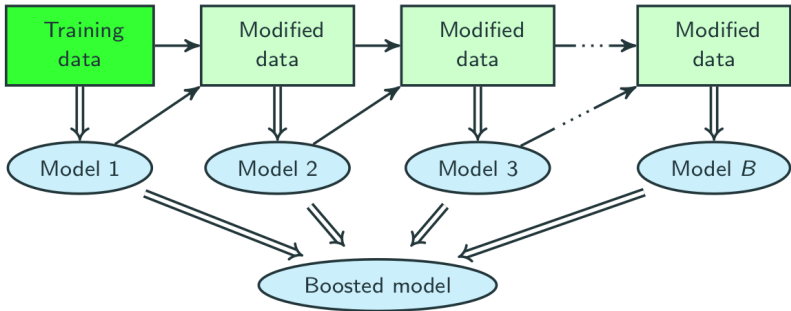
En enkel modell kan vanligtvis fånga vissa aspekter av datan.

Kan vi lära oss en stor mängd enkla modeller som var och en lär sig en liten del av datarelationen och sen kombinera dessa "dåliga" modeller till en stark modell.

Hur skulle vi göra detta?

# Boosting

- Lär sig sekventiellt en ensamble av "svaga" modeller.
- Kombinerar dessa till en "stark" modell.
- Generell metod som kan användas till all form av övervakad inlärning.
- Mycket framgångsrikt inom maskininlärning.



Vi kommer begränsa oss nu till binär klassificering.

Vi låter klasserna vara  $-1$  och  $1$  (möjliga  $y$  värden).

Använder vi detta kan vi skriva majoritetsröstning av  $B$  klassificerare  $\hat{y}^b(\mathbf{x})$  som

$$\text{sign} \left( \sum_{b=1}^B \hat{y}^b(\mathbf{x}) \right).$$

# Boosting för klassificering

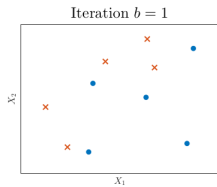
1. Ge varje datapunkt en vikt  $w_i^1 = 1/n$ .
2. För  $b = 1, \dots, B$ 
  - a Träna en "svag" klassificerare  $\hat{y}^b(\mathbf{x})$  på den **viktade träningsdatan**  $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$ .
  - b Uppdatera vikterna  $\{w_i^{b+1}\}_{i=1}^n$  från  $\{w_i^b\}_{i=1}^n$ 
    - i Öka vikterna för missklassificerade datapunkter.
    - ii Minska vikterna för korrekt klassificerade datapunkter.

Predikationen från de  $B$  klassificerarna kombineras genom att använda en viktad majoritetsomröstning,

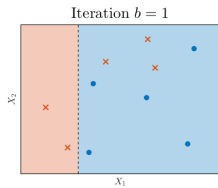
$$\hat{y}_{\text{boost}}^B(\mathbf{x}) = \text{sign} \left( \sum_{b=1}^B \alpha^b \hat{y}^b(\mathbf{x}) \right).$$



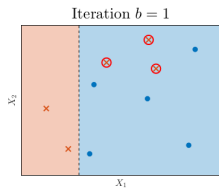
# Boosting exempel



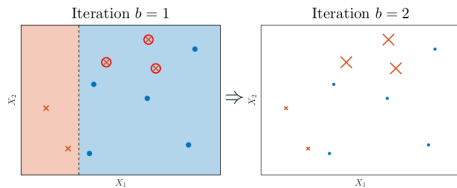
# Boosting exempel



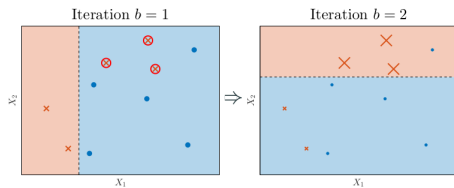
# Boosting exempel



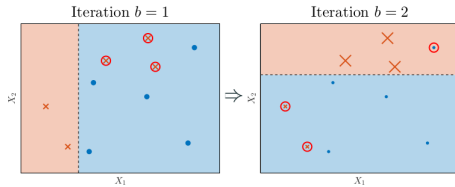
# Boosting exempel



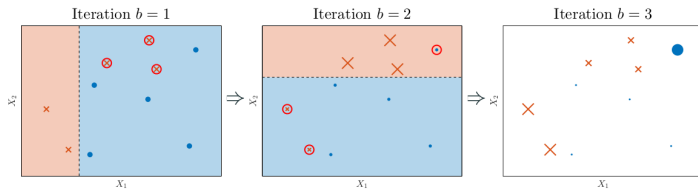
# Boosting exempel



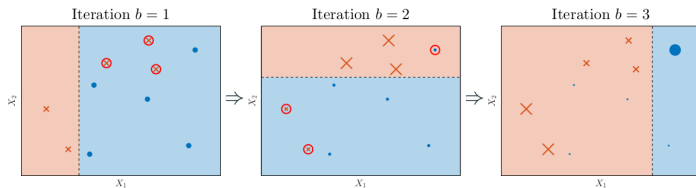
# Boosting exempel



# Boosting exempel

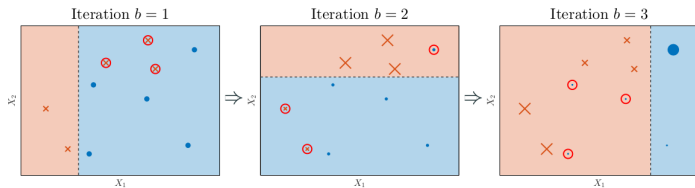


# Boosting exempel

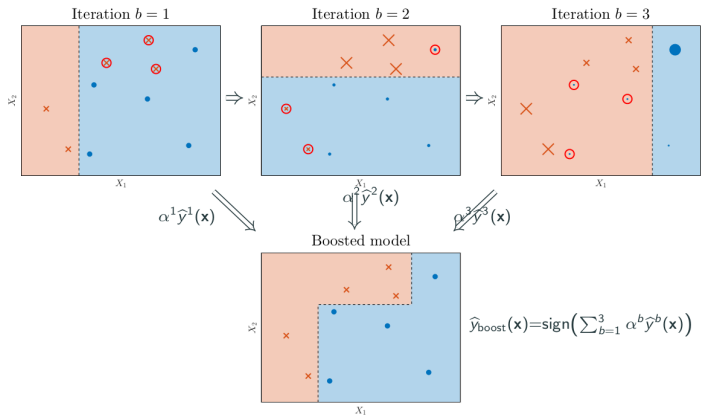




# Boosting exempel



# Boosting exempel



Boosting fungerar bra, men vi har lite detaljer vi måste reda ut först.

1. Hur ska vi vikta om data?
2. Hur ska vi vikta koefficienterna  $\alpha^b$ ?

Olika boostingalgoritmer svarar olika på dessa frågor.

Den första praktiska algoritmen AdaBoost, svarade på dessa frågor genom att minimera exponentialförlust.

1. Ge varje datapunkt en vikt  $w_i^1 = 1/n$ .
2. För  $b = 1, \dots, B$ 
  - a Träna en "svag" klassificerare  $\hat{y}^b(\mathbf{x})$  på den **viktade träningsdatan**  $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$ .
  - b Uppdatera vikterna  $\{w_i^{b+1}\}_{i=1}^n$  från  $\{w_i^b\}_{i=1}^n$ 
    - i Beräkna  $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \hat{y}^b(\mathbf{x}_i)\}$ .
    - ii Beräkna  $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$ .
    - iii Beräkna  $w_i^{b+1} = w_i^b \exp(-\alpha^b y_i \hat{y}^b(\mathbf{x}_i))$ ,  $i = 1, \dots, n$ .
    - iv Normalisera  $w_i^{b+1}$ .
3. Output  $\hat{y}_{\text{boost}}^B(\mathbf{X}) = \text{sign}\left(\sum_{b=1}^B \alpha^b \hat{y}^b(\mathbf{x})\right)$ .

---

**Algorithm 8.2** *Boosting for Regression Trees*

---

1. Set  $\hat{f}(x) = 0$  and  $r_i = y_i$  for all  $i$  in the training set.
2. For  $b = 1, 2, \dots, B$ , repeat:
  - (a) Fit a tree  $\hat{f}^b$  with  $d$  splits ( $d+1$  terminal nodes) to the training data  $(X, r)$ .
  - (b) Update  $\hat{f}$  by adding in a shrunk version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \quad (8.10)$$

- (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \quad (8.11)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x). \quad (8.12)$$

# Boosting - Sammanfattning

Finns många andra varianter:

- Gradient boosting:
  - XGboost
  - LightGBM
  - CatBoost
- Presterar bra och vinner ofta tävlingar.

Om vi jämför med baggning kan vi se:

Bagging	Boosting
Kan träna modeller parallellt	Tränar modeller sekventiellt
Använder bootstrappade dataset	Använder viktade dataset
Överanpassar inte med ökande $B$	Kan överanpassa när $B$ ökar
Minskar variansen men inte bias	Minska varians och bias.

# Gradient Boosting: En teoretisk översikt

**Gradient Boosting** är en iterativ ensemblemetod som bygger en modell  $f(x)$  som approximerar en målfunktion genom att minimera en differentiabel förlustfunktion  $L(y, f(x))$ .

**Grundidé:** Vid varje iteration  $m$ , lägg till en ny modell  $h_m(x)$  som approximerar den negativa gradienten av förlusten:

$$h_m(x) \approx - \left. \frac{\partial L(y, f(x))}{\partial f(x)} \right|_{f(x)=f_{m-1}(x)}$$

**Uppdatering:**

$$f_m(x) = f_{m-1}(x) + \nu \cdot h_m(x)$$

där  $\nu$  är inlärningshastigheten (learning rate).

## Fördelar:

- Teoretiskt välgrundad: bygger på gradient descent i funktionsrymden.
- Flexibel: fungerar med olika förlustfunktioner (t.ex. log-loss, kvadratisk förlust).
- Effektiv: varje steg fokuserar på att minska felet mest effektivt.



# Gradient för kvadratisk förlust

**Förlustfunktion:** Kvadratisk förlust (MSE) ges av

$$L(y, F(x)) = \frac{1}{2}(y - F(x))^2$$

**Gradient:** Vi beräknar derivatan med avseende på modellens prediktion  $F(x)$ :

$$\frac{\partial L(y, F(x))}{\partial F(x)} = -(y - F(x))$$

**Tolkning:** Den negativa gradienten är residualen:

$$-\frac{\partial L}{\partial F(x)} = y - F(x)$$

**I Gradient Boosting:** Vid varje iteration tränas en ny modell  $h_m(x)$  för att approximera residualerna  $y - F_{m-1}(x)$ .

**Uppdatering:**

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)$$

där  $\nu$  är inlärningshastigheten.

# Gradient för cross-entropy (binär klassificering)

**Förlustfunktion:** Cross-entropy för två klasser (etiketter  $y \in \{0, 1\}$ ) ges av:

$$L(y, F(x)) = -y \log(p(x)) - (1 - y) \log(1 - p(x))$$

där  $p(x) = \sigma(F(x)) = \frac{1}{1+e^{-F(x)}}$  är sannolikheten från modellen.

**Gradient:** Vi beräknar derivatan med avseende på  $F(x)$ :

$$\frac{\partial L(y, F(x))}{\partial F(x)} = p(x) - y$$

**Tolkning:** Den negativa gradienten är:

$$-\frac{\partial L}{\partial F(x)} = y - p(x)$$

vilket motsvarar residualen mellan det sanna värdet och den predikterade sannolikheten.

**I Gradient Boosting:** Vid varje iteration tränas en ny modell  $h_m(x)$  för att approximera  $y - p(x)$ .

# Gradient Boosting: Algoritmen steg för steg

**Mål:** Approximation av en funktion  $F(x)$  som minimerar en differentiabel förlustfunktion  $L(y, F(x))$ .

## Algoritm:

1. Initialisera modellen med en konstant:

$$F_0(x) = \arg \min_c \sum_{i=1}^n L(y_i, c)$$

2. För varje iteration  $m = 1, \dots, M$ :

- Beräkna negativa gradienten (pseudo-residualer):

$$r_i^{(m)} = - \left. \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x)=F_{m-1}(x)}$$

- Träna en svag modell  $h_m(x)$  för att approximera  $r_i^{(m)}$ .
- Uppdatera modellen:  $F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)$   
där  $\nu$  är inlärningshastigheten.

3. Slutlig modell:  $F_M(x) = F_0(x) + \sum_{m=1}^M \nu \cdot h_m(x)$

**Kommentar:**  $h_m(x)$  är ofta ett litet beslutsträd, och gradienten beror på vald förlustfunktion.

# XGBoost: Grundläggande idéer

**XGBoost** (Extreme Gradient Boosting) är en effektiv och förbättrad implementation av gradient boosting.

## Nyckelkomponenter:

- **Regularisering:** Läger till straff för komplexa träd:

$$\text{Objektiv} = \sum_i L(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

där  $\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_j w_j^2$

- **Trädstruktur:** Varje svag modell är ett beslutsträd som byggs med hjälp av en greedy-algoritm för att maximera "gain".
- **Andra förbättringar:**
  - Hanterar saknade värden automatiskt
  - Stöd för parallellisering och cache-optimering
  - Shrinkage (learning rate) och column subsampling

**Resultat:** Hög prediktiv precision, snabb träning, och god kontroll över överanpassning.

# XGBoost: Gain-beräkning

**Syfte:** Välj den split som maximerar förbättringen i den regulariserade objektfunktionen.

**För varje split:**

- Beräkna summan av första och andra derivatan (gradient och hessian) för varje nod:  $G = \sum_{i \in \text{node}} g_i$ ,  $H = \sum_{i \in \text{node}} h_i$  där  $g_i = \frac{\partial L}{\partial F(x_i)}$ ,  $h_i = \frac{\partial^2 L}{\partial F(x_i)^2}$
- Gain ges av:

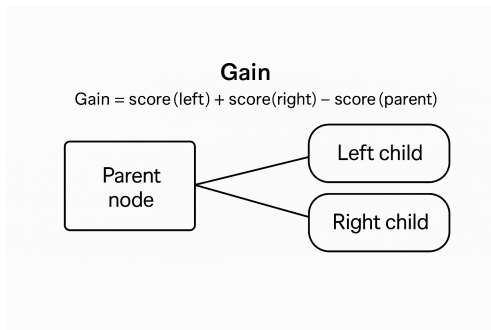
$$\text{Gain} = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

där  $G_L$ ,  $H_L$  är gradient/hessian för vänster barn,  $G_R$ ,  $H_R$  för höger barn,  $\lambda$  är L2-regularisering och  $\gamma$  är straff för att skapa en ny nod.

**Tolkning:** En split görs om gain är positiv och tillräckligt stor – annars stoppas trädet.

**Effekt:** Effektiv trädbyggnad med inbyggd regularisering mot överanpassning.

# XGBoost: Visuell förklaring av Gain



**Gain** mäts som förbättring i objektfunktionen när en nod delas:

$$\text{Gain} = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

**Split** görs om gain är positiv och tillräckligt stor.