732G57 Maskininlärning för statistiker

Föreläsning 6

Josef Wilzén
IDA, Linköping University, Sweden

Dagens föreläsning: Ensamblemetoder

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

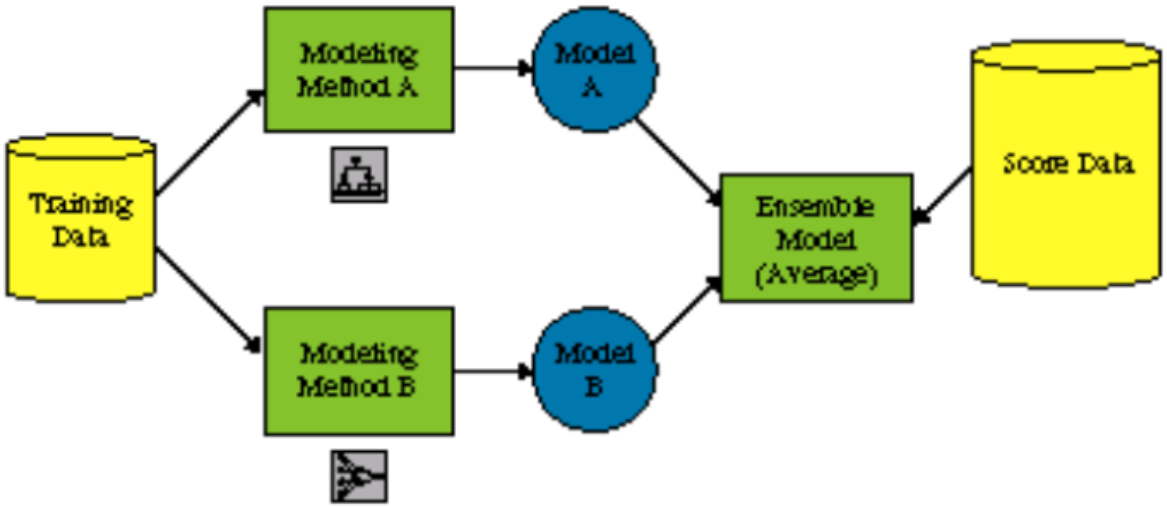
Ensamblemetoder

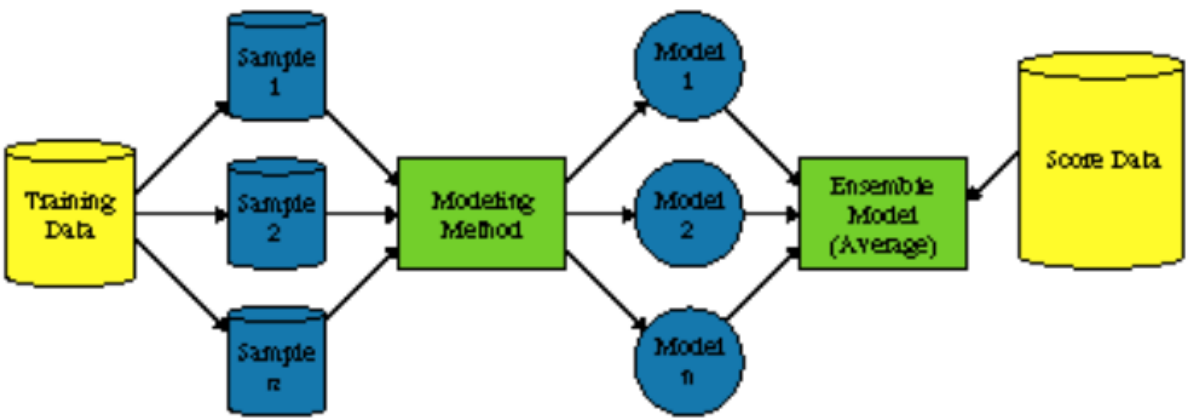
Grundidén är att skatta många modeller på träningsdata och sen kombinera dessa för att göra prediktioner.

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.

Ensamblemetoder





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 $k = 1, \ldots, B$.

Beräkna $\mathbb{V}(\mathbf{T})$.

Bagging

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 slutgiltliga modellen får vi genom att ta medelvärdet av alla dessa modeller:

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

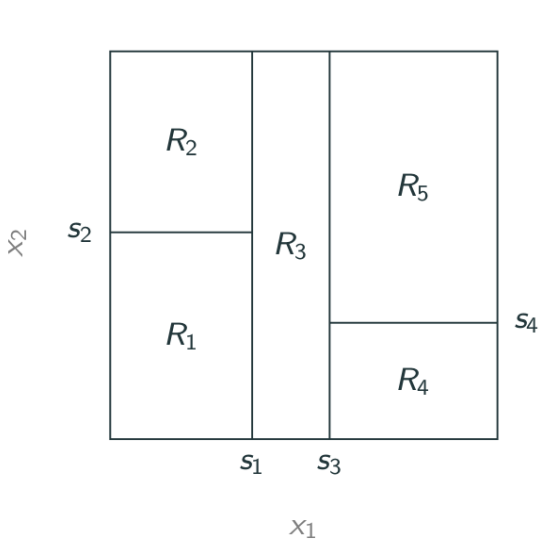
- Sänker variansen av den anpassade funktionen.
- Påverkas *mycket* av kvalitén av modellen. En bra modell blir bättre, en dålig blir sämre.
- För klassificering, använd majoritetsröstning.

Classification and Regression Trees (CART)

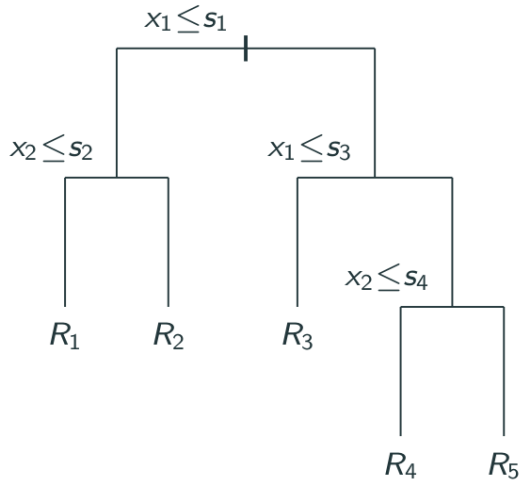
Vi delar upp variabelrummet genom att rekrusivt göra binära uppdelningar.

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

Partitioning of input space



Tree representation



Förbättre CART

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).
- Ensamblemetoder:
  - Ta ett genomsnitt över många trädmodeller.
  - Bagging
  - Random forest
  - Boosted trees

Random forest

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.

Random forest

- 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: $q = p/3$

Random forest

Slumpmässigt val av variabler leder till:

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

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

Random forest

Beräkningsmässiga fördelar:

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

Boosting

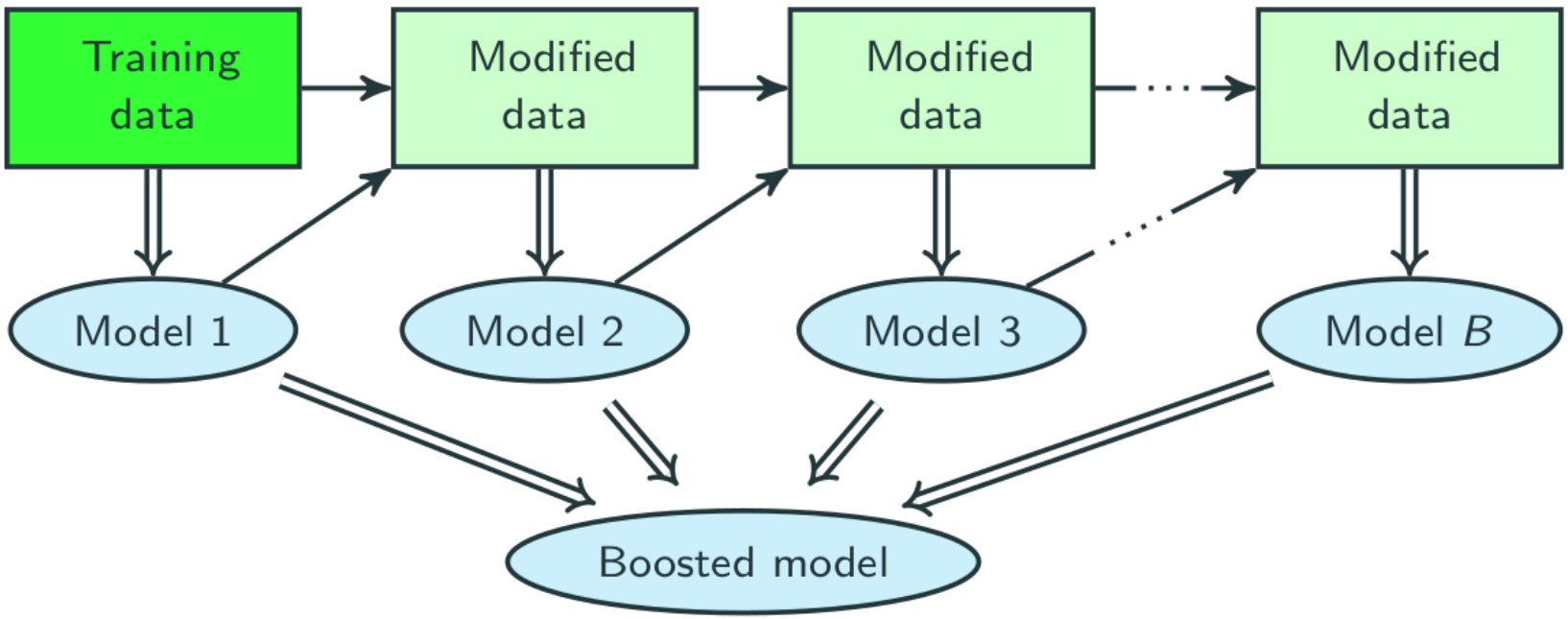
En enkel modell kan vanligtvis fånga vissa aspekter av datan.

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 framgångsrik inom maskininlärning.



Binär klassificering

Vi kommer begränsa oss nu till binär klassificering.

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

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

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

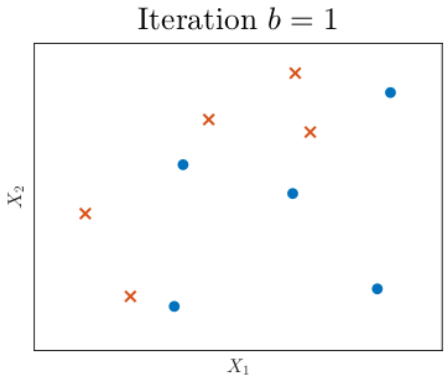
Boosting för klassificering

1. Ge varje datapunkt en vikt $w_i^1 = 1/n$.
2. För $b = 1, \ldots, B$
   a Träna en "svag" klassificerare $\hat{y}^b(\mathbf{x})$ på den **viktade träniningsdatan** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   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 klassificeradet datapunkter.

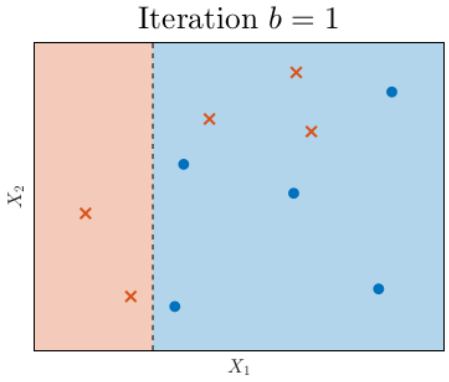
Predikationen från de B klassificerarna kombineras genom att använda en viktad majoritetsomröstning,

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

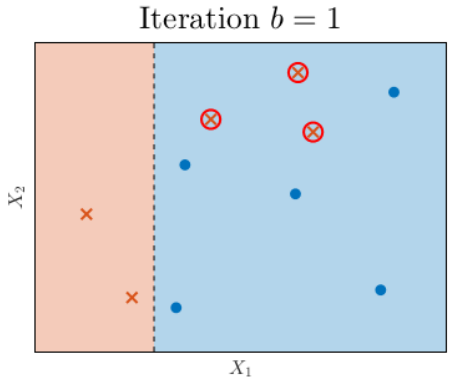
Boosting exempel



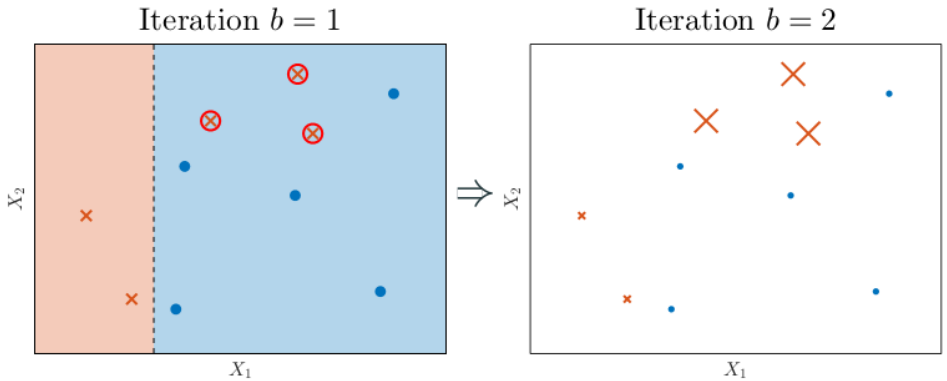
Boosting exempel



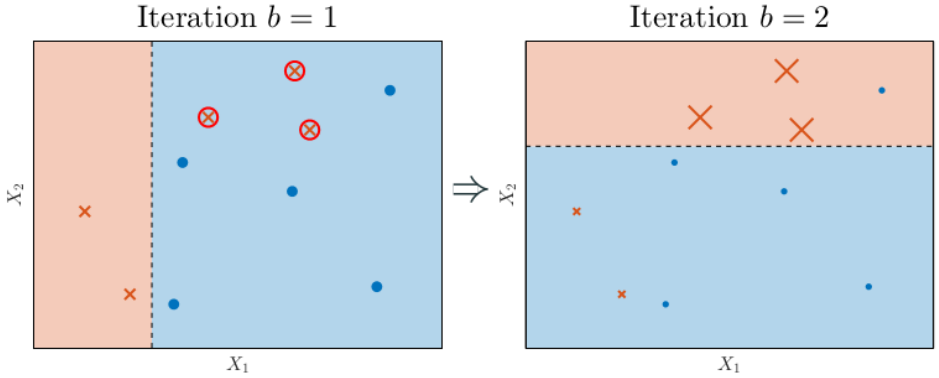
Boosting exempel



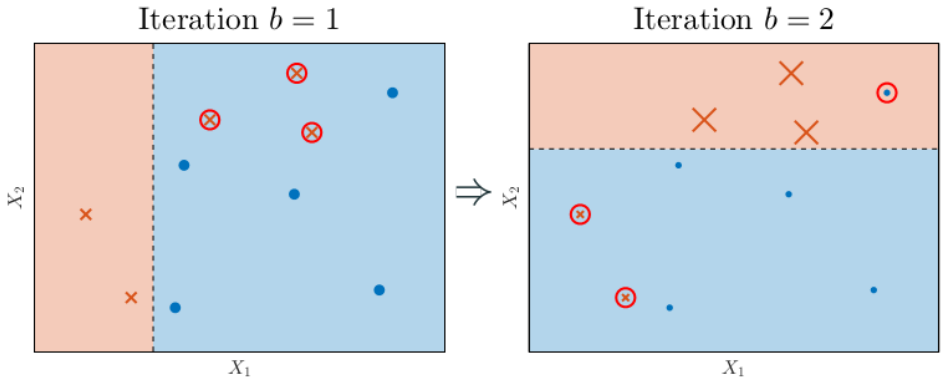
Boosting exempel



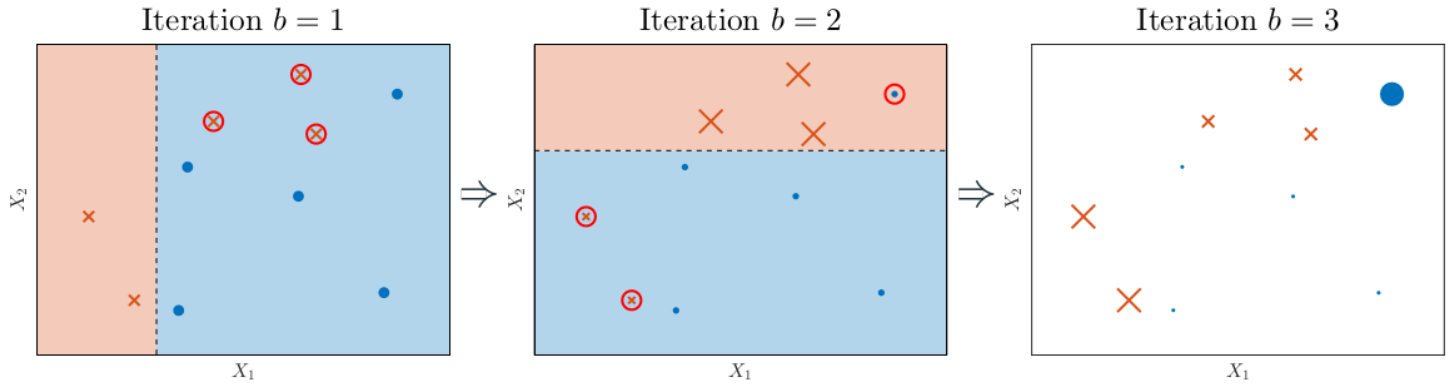
Boosting exempel



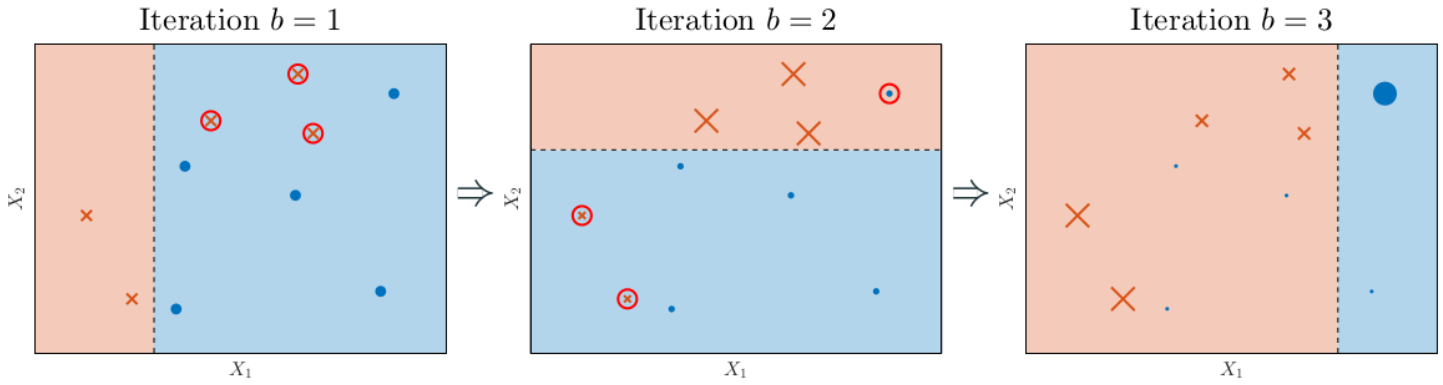
Boosting exempel



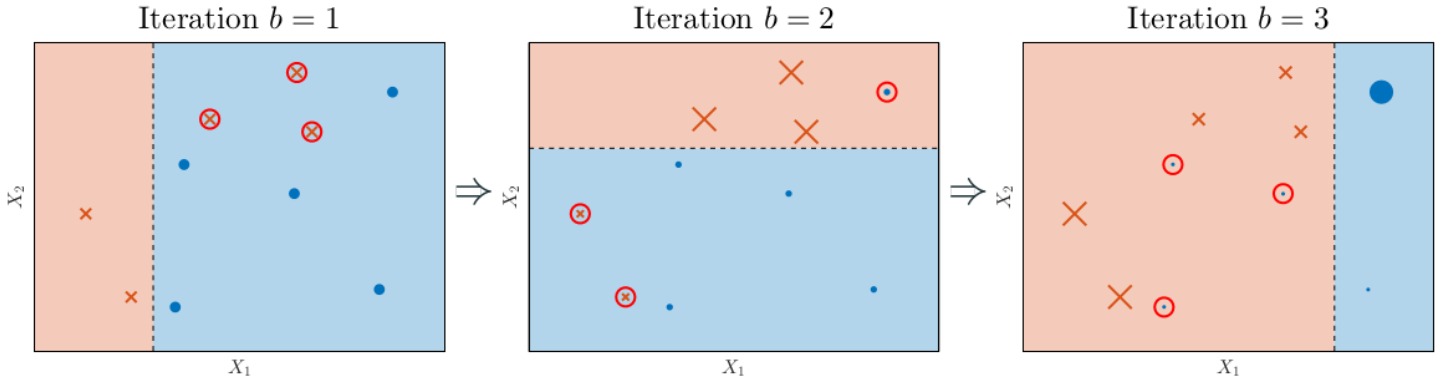
Boosting exempel



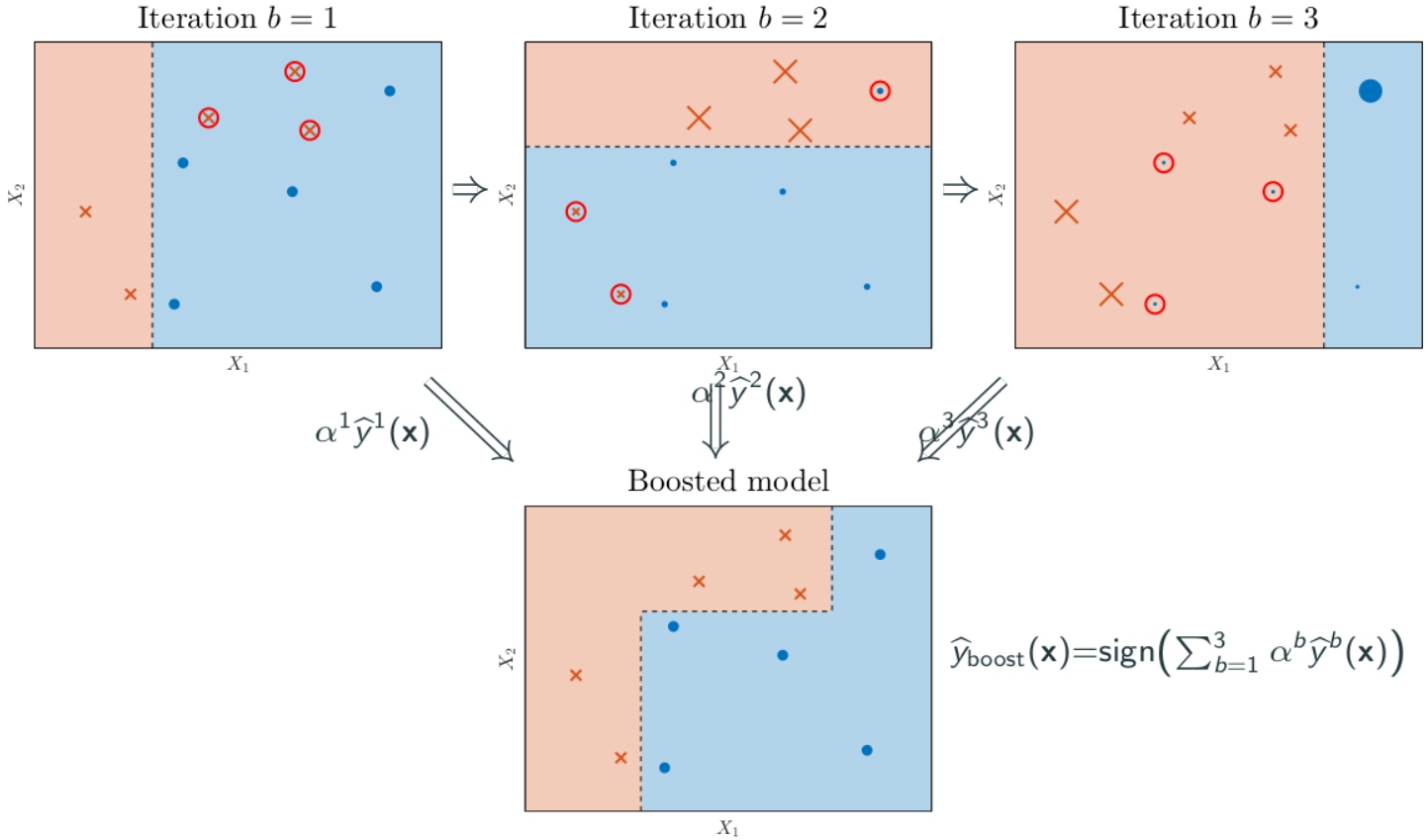
Boosting exempel



Boosting exempel



Boosting exempel



Boosting - Detaljer

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 α^b?

Olika boostingalgoritmer svarar olika på dessa frågor.

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

AdaBoost

1. Ge varje datapunkt en vikt $w_i^1 = 1/n$.
2. För $b = 1, \ldots, B$
   a Träna en "svag" klassificerare $\hat{y}^b(\mathbf{x})$ på den **viktade träninings**data**n** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   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 $w_i^{b+1} = w_i^b \exp(-\alpha^b y_i \hat{y}^b(\mathbf{x}_i)), i = 1, \ldots, n$.
      iv Normalisera w_i^{b+1}.
3. Output $\hat{y}_{\text{boost}}^B(\mathbf{X}) = \text{sign}\left(\sum_{b=1}^B \alpha^b \hat{y}^b(\mathbf{x})\right)$.

Boosting för regressionsträd

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 $b = 1, 2, \ldots, B$, 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 shrunken version of the new tree:

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

   (c) Update the residuals,

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

3. Output the boosted model,

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

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 ν är inlärningshastigheten (learning rate).

Gradient Boosting

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 ν 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 $m = 1, \ldots, M$:
   - 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 ν 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.

Grundläggande idéer inom XGBoost

- **XGBoost** (Extreme Gradient Boosting) är en effektiv implementation av gradient boosting med fokus på prestanda och skalbarhet.
- Modellen bygger beslutsträd med hjälp av en greedy-algoritm som väljer den bästa splitpunkten i varje steg.
- Inbyggd regularisering används för att kontrollera modellens komplexitet och minska risken för överanpassning.
- Träden byggs med hjälp av en kostandsfunktion som kombinerar förlust och modellkomplexitet.
- XGBoost är optimerad för parallellisering och hantering av stora datamängder.
- Kan hantera saknade värden automatiskt

Viktiga hyperparametrar i XGBoost

- `eta` – Inlärningshastighet. Lägger mindre vikt vid varje nytt träd.
- `nrounds` – Antal boosting-rundor, dvs. antal träd som byggs.
- `lambda` – L2-regularisering av vikter. Minskar överanpassning.
- `gamma` – Minsta förbättring som krävs för att göra en split. Påverkar pruning.
- `max_depth` – Maximal djup för varje träd. Påverkar modellens komplexitet.
- `subsample` – Andel av träningsdata som används för varje träd. Minskar överanpassning.
- `colsample_bytree` – Andel features som används vid varje träd. Ökar variation mellan träden.

Kostandsfunktion och additiv modell

Additiv modell:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i), \quad f_k \in \mathcal{F}$$

Där varje f_k är ett beslutsträd och modellen byggs upp stegvis.

Kostandsfunktion:

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Där l är en förlustfunktion (t.ex. kvadratisk eller logistisk) och $\Omega(f)$ är en regulariseringsterm som straffar komplexa träd.

Regulariseringsterm:

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

Där T är antalet blad i trädet och w_j är vikten för blad j.

Taylor-expansion av en funktion

Taylor-expansion:

- En metod för att approximera en funktion $f(x)$ nära en punkt x_0 med hjälp av derivator.
- För en tillräckligt mjuk funktion kan vi skriva:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 + \cdots$$

- I XGBoost används en andra ordningens Taylor-expansion för att approximera kostandsfunktionen.
- Detta gör det möjligt att analysera hur ett nytt träd påverkar förlusten och därmed optimera modellen effektivt.

Första steg mot härledning av Gain

Approximerad kostandsfunktion:

$$\text{Obj}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t)$$

Där:

- $g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}$ är första derivatan (gradient).
- $h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$ är andra derivatan (hessian).

Skattning av lövvikter via Taylor-expansion

Utgångspunkt:

- Vi approximerar kostandsfunktionen med en andra ordningens Taylor-expansion.
- För varje löv i trädet summerar vi gradienter och hessianer för de observationer som hamnar där.

Approximerad kostandsfunktion för ett löv:

$$\text{Obj}_{\text{leaf}}(w) = Gw + \frac{1}{2}Hw^2 + \Omega(w)$$

Där:

- $G = \sum_{i \in \text{leaf}} g_i$ är summan av första derivator (gradienter).
- $H = \sum_{i \in \text{leaf}} h_i$ är summan av andra derivator (hessianer).
- w är vikten som tilldelas lövet.

Optimal lövvikt:

$$w^* = -\frac{G}{H + \lambda}$$

Detta är den vikt som minimerar den approximativa kostandsfunktionen

XGBoost: Gain-beräkning

Syfte: Välj den split som maximerar förbättringen i den regulariserade kostandsfunktionen.

För varje split:

- Gain ges av:

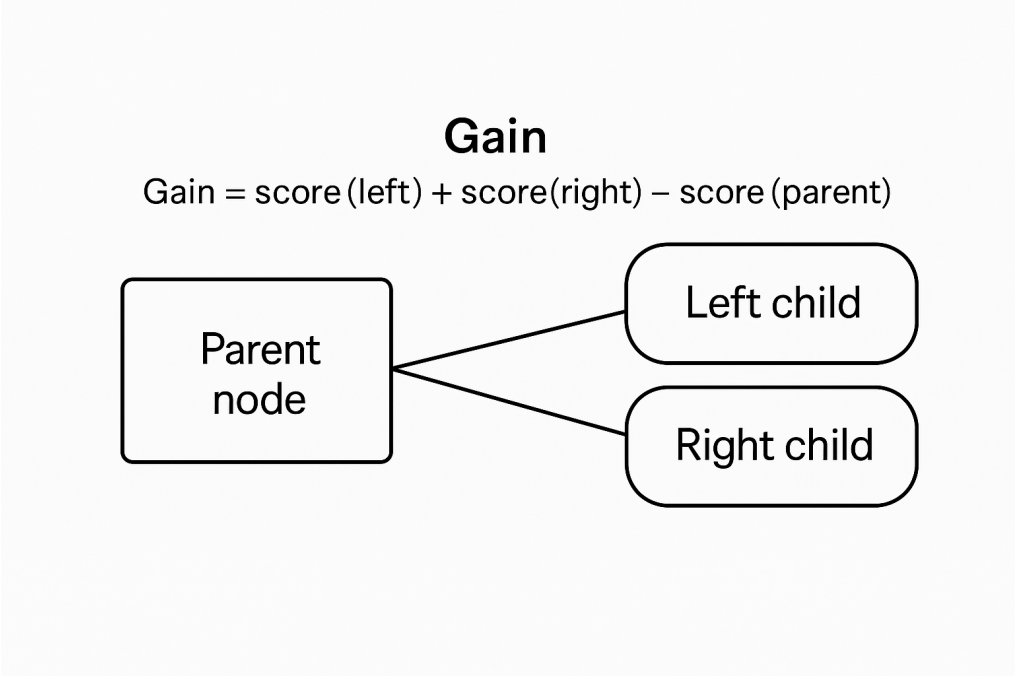
$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

  där G_L, H_L är gradient/hessian för vänster barn, G_R, H_R för höger barn, λ är L2-regularisering och γ 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 kostandsfunktionen 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.

Översikt av XGBoost-algoritmen

- Starta med en initial modell (t.ex. en konstant prediktion).
- Upprepa för varje boosting-runda:
  - Beräkna gradienter och hessianer för varje observation.
  - Bygg ett nytt beslutsträd som approximerar dessa derivator.
  - Optimera varje split med hjälp av en kostandsfunktion som inkluderar regularisering.
  - Beräkna vikter för varje löv i trädet.
- Lägg till det nya trädet i den befintliga modellen.
- Fortsätt tills ett stoppkriterium uppfylls (t.ex. antal träd eller ingen förbättring).

Regulariseringens roll i XGBoost

- Regularisering används för att kontrollera modellens komplexitet och minska risken för överanpassning.
- I XGBoost införs regularisering direkt i kostandsfunktionen, vilket påverkar hur träd byggs.
- Två typer av regularisering används:
  - **L2-straff** på lövvikterna – straffar stora värden och gör modellen mer stabil.
  - **Strukturstraff** – straffar träd med många löv för att föredra enklare modeller.
- Regularisering påverkar både hur splitar väljs och hur lövvikter beräknas.
- Genom att justera hyperparametrar som `lambda` och `gamma` kan man styra regulariseringens styrka.