

Tree model
Boosting
Gradient descent
XGBoost

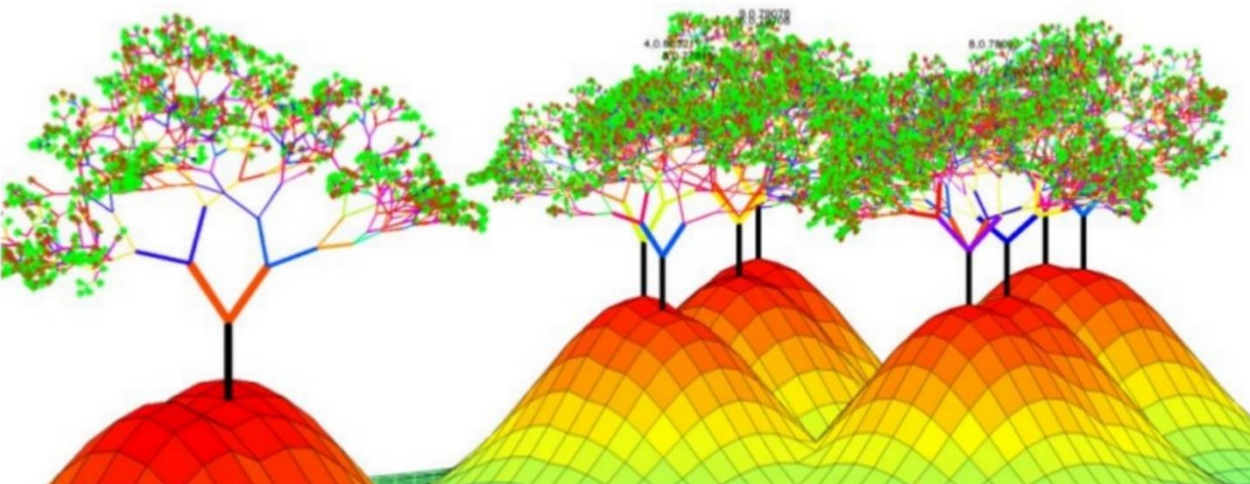
Boostede tre-modeller

Fra en enkel tre-modell til en XGBoost-modell

Martin Jullum

jullum@nr.no

SSB 9.feb 2021



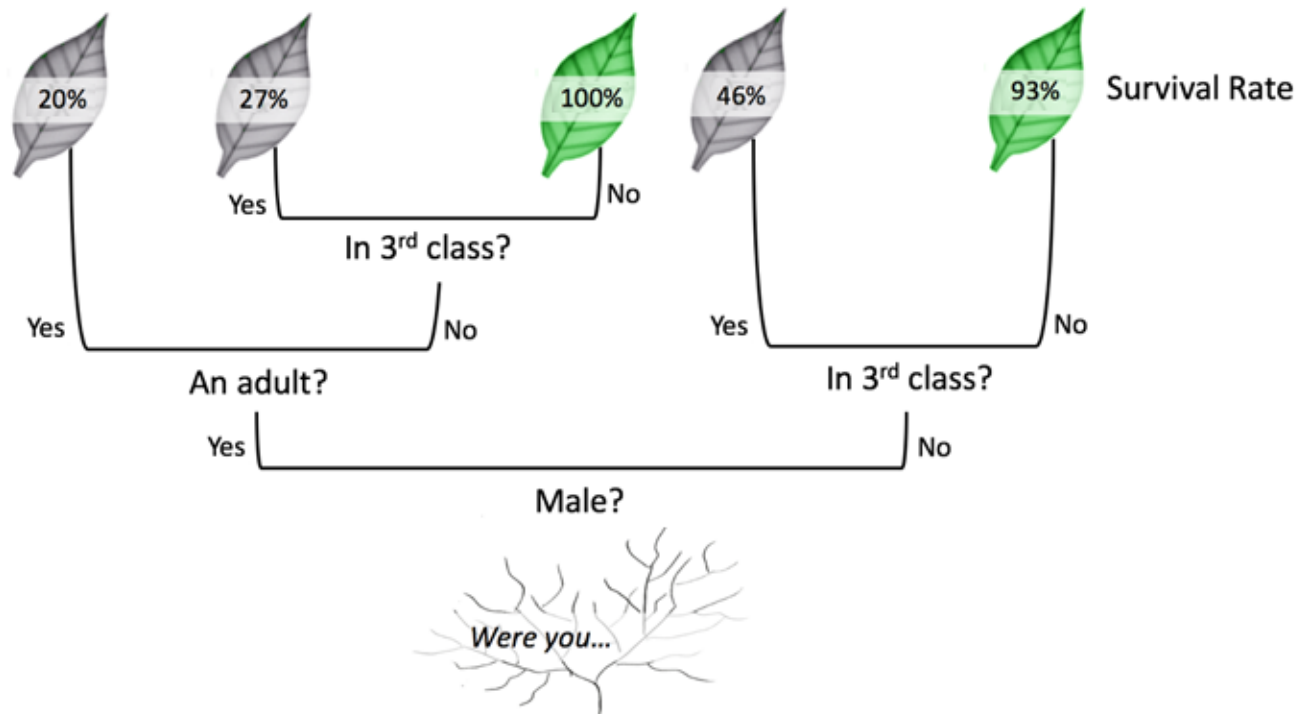
Problem

- ▶ Anta at vi har et datasett med n observasjoner
 - Respons: y_i
 - Kovariater: $x_i = (x_{i1}, \dots, x_{ip})^T, i = 1, \dots, n$
- ▶ Ønsker å tilpasse en modell f til disse dataene slik at $f(x_i)$ approksimerer y_i så godt som mulig (på et separat datasett) i form av en tapsfunksjon $L(y_i, f(x_i))$

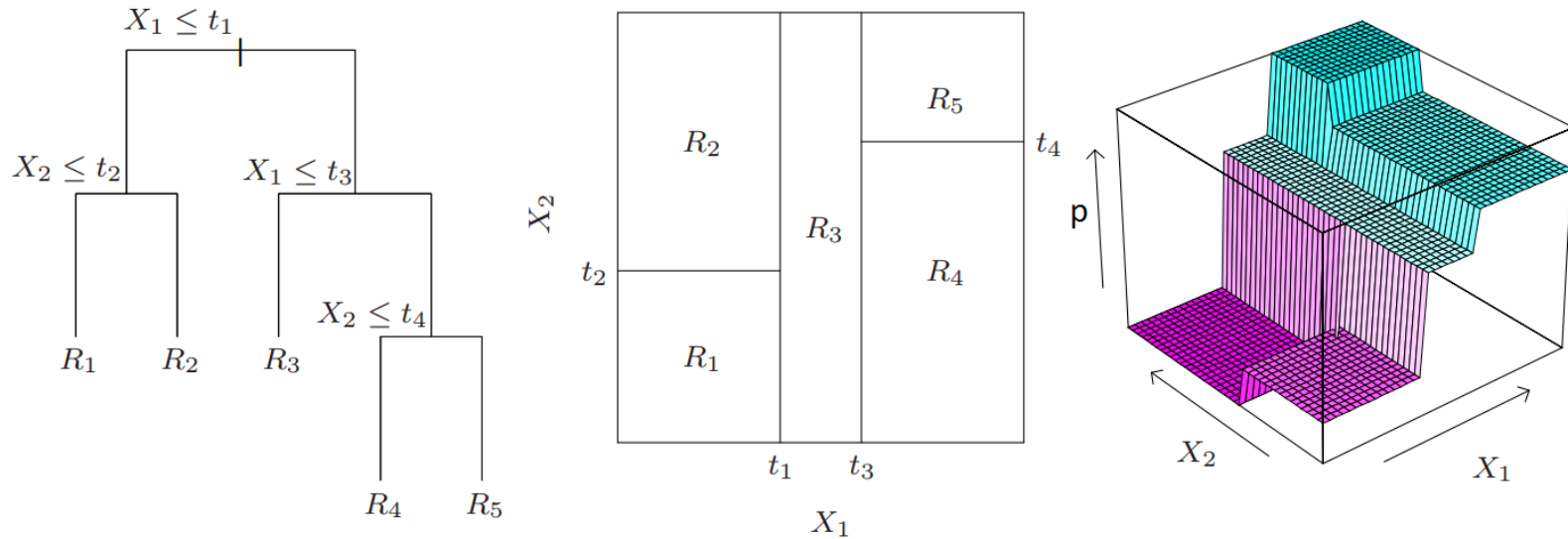


Tre-modeller (I)

- ▶ Verdens enkleste nyttige statistiske modell!
 - Hver forgrening er en IF-THEN regel
 - Fungerer både for kontinuerlig og binær respons, samt klassifisering



Tre-modeller (II)



3 ulike visualiseringer av samme tre-modell

- Kan skrives som en vektet sum av indikatorvariable over regionene:

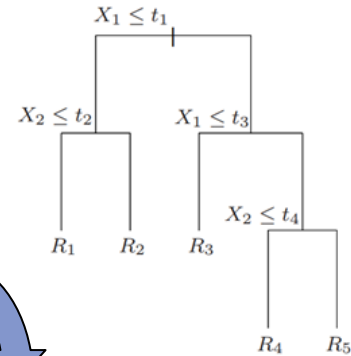
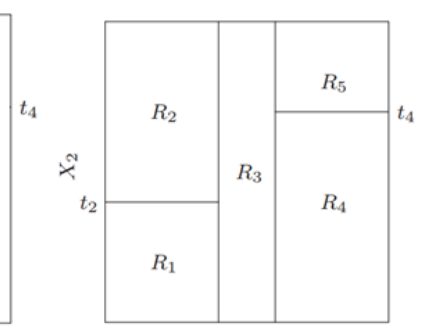
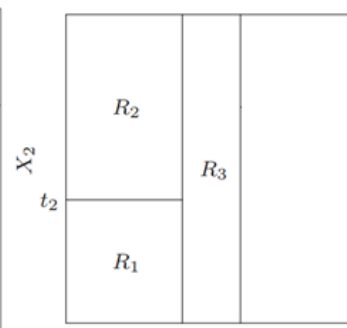
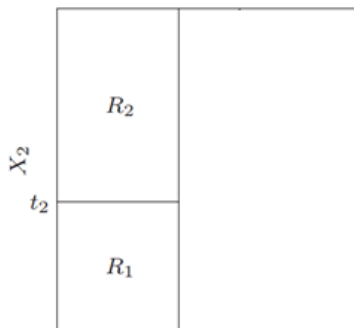
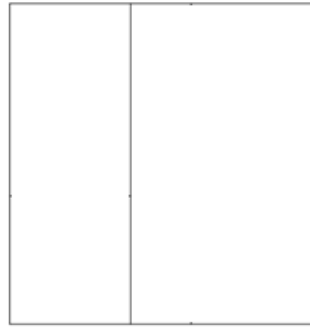
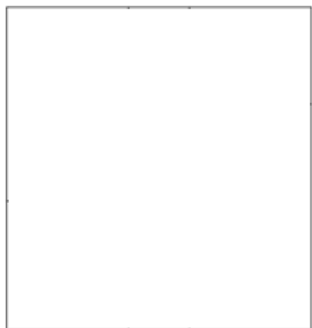
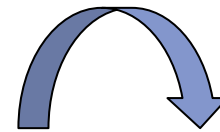
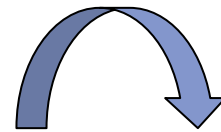
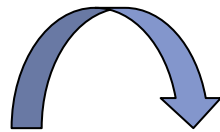
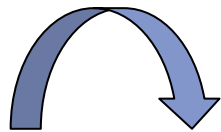
$$f(x) = \sum_{j=1}^T \theta_j 1_{\{x \in R_j\}}$$

Trening av tre-modeller

$$\sum_{i \in R_j} [L(y_i, \hat{y}_{R_{1j}}) + L(y_i, \hat{y}_{R_{2j}})]$$

where $\hat{y}_{R_{kj}} = \operatorname{argmin}_c \sum_{i \in R_{kj}} L(y_i, c)$

- Beregningsmessig svært tungt å finne optimal regionsoppdeling
- Bruker en grådige algoritme i stedet (start med roten som den eneste (blad-)noden)
 1. For hver blad-node og kovariate x_j , finn det splittpunktet som gir lavest tap når man deler regionen i to deler
 2. Velg blad-node, kovariat og splittpunkt med størst tapsreduksjon
 3. Gjennomfør splitt med mindre stoppekriterium inntreffer
 4. Gjenta punkt 1-3



Egenskaper med tre-modeller

► Fordeler

- Enkle å tolke
- Enkle å trene
- Invariant til monotone transformasjoner av kovariatene
- Håndterer naturlig kontinuerlige og kategoriske data
- Kan håndtere missing data
- Modellerer ikke-lineariteter og interaksjoner direkte
- Skalerer godt til store datamengder

► Ulemper

- Fort gjort å overtilpasse
- Diskrete prediksjoner
- Begrenset prediksjonskraft

Boosting: Prinsippet

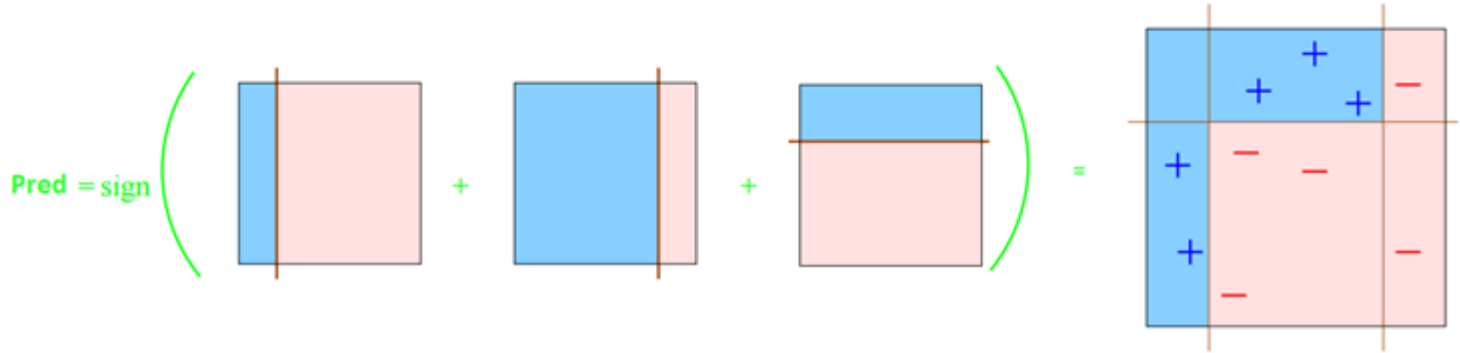
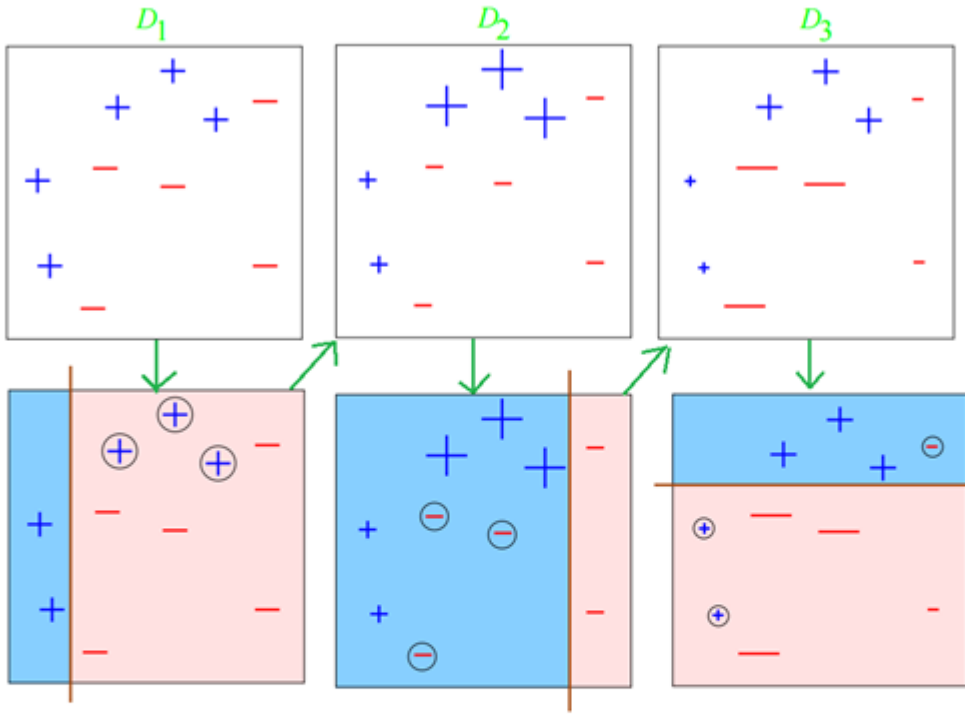
- ▶ Modell-ensemble teknikk som slår sammen mange enkle «basismodeller» $f_m(x)$, $m = 1, \dots, M$ (weak learners) til en avansert (strong learner) $f_{final}(x)$
- ▶ Trener iterativt en og en basismodell, hver og en med mål om å reparere feilene til tidligere trente modeller (og minimere empirisk tap)
- ▶ Endelig prediksjon = Sum av prediksjoner fra alle basismodellene

$$f_{final}(x) = f^{(M)}(x) = \sum_{m=1}^M f_m(x)$$

$$f_m = \operatorname{argmin}_{h \in \Phi} \sum_{i=1}^n L(y_i, f^{(m-1)}(x_i) + h(x_i))$$

For modellklasse Φ

Eksempel boosting



Egenskaper med boosting

► Fordeler

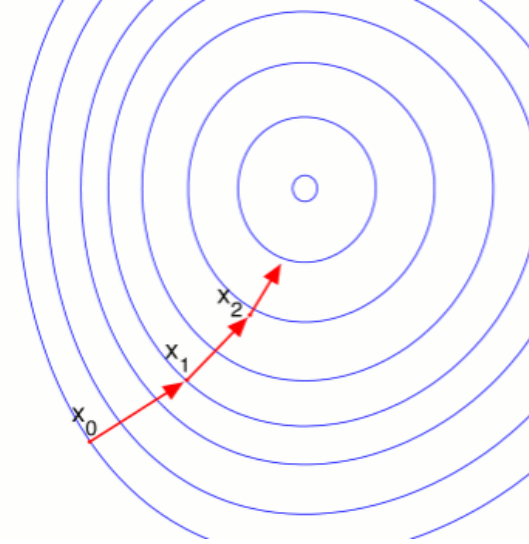
- Arver typisk alle egenskapene til basismodellene, men gir en vilkårlig god prediksjonskraft i tillegg

► Utfordringer

- Svært viktig å kontrollere overtilpasning for å få en god modell
- Boosting kan i seg selv ikke parallelliseres
- Generelt vanskelig å oppdatere med nye modeller via

$$f_m = \operatorname{argmin}_{h \in \Phi} \sum_{i=1}^n L(y_i, f^{(m-1)}(x_i) + h(x_i))$$

Gradient boosting



- ▶ Gradient descent
 - Iterativ metode for å finne minimum av multivariat funksjon $s(z)$
 - Tar steg langs den negative gradienten: $z_m = z_{m-1} - \rho_m s'(z_{m-1})$

- ▶ Vi vil minimere

$$f_m = \underset{h \in \Phi}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, f^{(m-1)}(x_i) + h(x_i))$$

- ▶ Gradient boosting = Gradient descent for funksjoner/modeller

- La $s_i(z) = L(y_i, f^{(m-1)}(x_i))$, $i = 1, \dots, n$
- Bruk gradient descent for hver s_i , kall disse for y_i^*
- Tilpass modell av klasse Φ med tapsfunksjon $(y_i^* - f(x_i))^2$
- + diverse detaljer om steglengde
- Dette er den “vanlige” boosting-metoden. Brukes f.eks. i gbm-pakka i R
- For steglengde = $\frac{1}{2}$ og original kvadratisk tap er $y_i^* =$ residualene

2. ordens approksimasjon

- ▶ Vil fortsatt minimere

$$f_m = \underset{h \in \Phi}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, f^{(m-1)}(x_i) + h(x_i))$$

- ▶ Approksimer $s_i(z) = L(y_i, z)$, ved en 2. ordens Taylor ekspansjon:

$$\begin{aligned} & L(y_i, f^{(m-1)}(x_i) + h(x_i)) \\ & \approx s_i(f^{(m-1)}(x_i)) + s_i'(f^{(m-1)}(x_i)) h(x_i) + \frac{1}{2} s_i''(f^{(m-1)}(x_i)) h(x_i)^2 \\ & = \frac{1}{2} s_i''(f^{(m-1)}(x_i)) \left[-\frac{s_i'(f^{(m-1)}(x_i))}{s_i''(f^{(m-1)}(x_i))} - h(x_i) \right]^2 \end{aligned}$$

- ▶ Bytt ut original tapsfunksjon med kvadratisk approksimasjon og tilpass modell
- ▶ Brukes av XGBoost og andre moderne boosting-rammeverk, med tre-modeller som modellklasse Φ
- ▶ Ekvivalent med gradient boosting for kvadratisk tap

XGBoost = eXtreme Gradient Boosting

- ▶ Et open source bibliotek bygget rundt en effektiv implementering av gradient boosting med tre-modeller som basismodeller
 - Utviklet av Tianqi Chen (Uni. Washington) i 2014
- ▶ Implementasjon
 - Grensesnitt for mange språk/plattformer: C++, Python, R, Julia, Java, Apache Spark etc.
 - Parallelliserbar trening av trærne, minnegjerrig og skalerbar
 - Kjører både på CPU og GPU
- ▶ Metodiske nyvinninger
 - 2.ordens approksimasjon av tapsfunksjonen – mer presis/effektiv enn ordinær gradient boosting
 - Legger til regularisering på toppen av original tapsfunksjon
- ▶ Praktisk bruk
 - Veldig mange parametere som kan skrues på, må gjøres manuelt
 - Kan ta lang tid å optimalisere/tune, men brukbare defaultparametere
 - «The Kaggle game killer»

Funksjonalitet i XGBoost

- ▶ Håndterer både kryssvalidering og ferdigoppdelt trening/validering/testsett
- ▶ Kan definere egne tapsfunksjoner og valideringsmål (mange allerede implementert).
- ▶ Kan følge valideringsresultater mens modellen kjører (f.eks. AUC på trening, validering og testsett)
- ▶ «Early stopping» (stopper å legge til nye trær når valideringsresultater ikke forbedres lenger)
- ▶ Mange tilgjengelige måter å håndtere overtilpasning på
- ▶ Ingen pre-prosessering/skalering/standardisering nødvendig
- ▶ Håndterer manglende data automatisk (lærer default retning i hver splitt)
- ▶ Effektiviserer trening av trær ved å forhåndsdefinere en begrenset mengde splittpunkter (histogram-metoden)

XGBoost – diverse

- ▶ Konkurrenter
 - LightGBM (Microsoft)
 - Har drevet/motivert mye av utviklingen av XGBoost
 - Mye likt, men ikke like modent og mangler noe funksjonalitet
 - Fortsatt noe raskere enn XGBoost?
 - CatBoost (Yandex)
 - Lignende, men håndtere også kategoriske variable direkte
 - Var langt treigere, men har blitt vesentlig bedre
 - Begrenset dokumentasjon
- ▶ Jeg har enda til gode å se et eksempel der Random Forest gjør det bedre enn en tunet XGBoost modell!
- ▶ Hovedutfordringer:
 - Vanskelig/tidkrevende å finne optimal modell
 - Takler kun numerisk input: Ikke så god når det er mange kategoriske variable med mange klasser.

Ressurser

- ▶ Didrik Nielsen, Masteroppgave NTNU, 2016:
<https://brage.bibsys.no/xmlui/handle/11250/2433761>
- ▶ Chen & Guestrin (2016), XGBoost: A Scalable Tree Boosting System: <https://arxiv.org/abs/1603.02754>
- ▶ Hastie et al. (2009), Elements of Statistical Learning, Ch 9.2 + 10
- ▶ XGBoost GitHub: <https://github.com/dmlc/xgboost>
- ▶ XGBoost dokumentasjon: <http://xgboost.readthedocs.io>
- ▶ Slides fra foredrag med Tianqi Chen:
<http://datascience.la/xgboost-workshop-and-meetup-talk-with-tianqi-chen/>