SOS2901 Maskinlæring for samfunnsvitere

Oppgaver til hver undervisningsuke

Torbjørn Skardhamar

2026-02-23

Table of contents

Introduksjon

Dette dokumentet gir en oversikt over hva vi skal dekke i løpet av semesteret. Det lages delvis underveis, så det vil bli oppdateringer innimellom. Det vil kunne skje endringer i dokumentet underveis gjennom hele semesteret.

Hvert kapittel starter med en introduksjon til temaet og et empirisk eksempel. Datasettene vil dere finne i rommet i Canvas. Last de ned til egen maskin for å jobbe med dem.

Undervisningsvideoer

Det finnes noen undervisningsvideoer fra noen år tilbake. Disse dekker omtrent de vi går gjennom i undervisningen, men er ikke nødvendigvis helt oppdaterte. Dere har tilgang til disse videoene på denne lenken

Hva vi forutsetter av forkunnskaper

Dette kurset forutsetter at man har grunnleggende ferdigheter i kvantitative metoder og har brukt *R* før. Hvis du vet du trenger det: frisk opp litt fra tidligere kurs.

Når det er sagt, så er det begrenset hvor mye man *må* kunne fra før hvis du er motivert til å jobbe med stoffet skal du nok få til dette. Det blir mye nytt uansett.

Hvis du trenger oppfriskning av hvordan R fungerer, så er Wickham & Grolemunds bok "R for data science" et utmerket oppslagsverk. Men merk at vi i begrenset grad vil bruke "Tidyverse" til annet enn datahåndtering og noe grafikk.

R og Rstudio

Installasjon

Du må installere både R og Rstudio på din datamaskin. Hvis du har Windows-maskin trenger du også installere Rtools. Hvis du har Mac OS X kan det hende du må installere XQuartz.

Se ellers video fra SICSS https://youtu.be/ulIv0NiVTs4

Pakker

R er basert på bruk av "pakker" som må installeres for å få tilgang til funksjoner vi skal bruke. Disse installeres med bruk av kommandoen `install.packages()`. F.eks. kan man installere pakken Tidyverse med følgende: `install.packages("tidyverse")`.

For å installere flere pakker kan man kjøre `install.packages()` flere ganger, men det er enklere å liste opp alle pakkene i et objekt og så kjøre `install.packages()` på dette objektet. Noe slikt:

Vi skal i hvert fall bruke de pakkene som installeres med følgende kode:

```
pkgs <- c("tidyverse", "randomForest", "rpart", "rpart.plot",
          "rsample", "fairness", "DALEX", "fairmodels", "xgboost",
          "mgcv", "splines", "Ecdat",
          "ipred", "pROC", "caret", "forcats", "gtsummary", "janitor",
          "gbm", "dendextend", "directlabels", "GGally")
install.packages(pkgs)
```

Det kan hende vi kommer til å bruke flere etterhvert.

Men for at du skal kunne bruke pakkene må du aktivere dem i R ved `library()`. Dette må du gjøre hver gang du starter opp R. Hvert kapittel starter med en oversikt over hvilke pakker som brukes i det kapittelet.

Oversikt over pakker brukt i heftet

Tabellen under gir en oversikt over alle pakkene vi bruker, med en kort forklaring og de viktigste funksjonene.

Pakke	Beskrivelse	Viktige funksjoner	Kapittel
`tidyverse`	Samling av pakker for datahåndtering og grafikk	`mutate()`, `select()`, `filter()`, `ggplot()`, `glimpse()`	Alle
`rsample`	Splitting av data i training/testing	`initial_split()`, `training()`, `testing()`	Alle supervised-kapitler
`mgcv`	Generaliserte additive modeller (GAM)	`gam()`, `s()`	Lineær regresjon
`splines`	Spline-funksjoner for fleksibel regresjon	`bs()`	Lineær regresjon
`Ecdat`	Datasett for økonometri (Clothing-data)	`data(Clothing)`	Lineær regresjon
`pROC`	ROC-kurver og AUC-beregning	`roc()`, `auc()`	Logistisk regresjon
`caret`	Confusion matrix og modellevaluering	`confusionMatrix()`	Logistisk regresjon, CART, Bagging, Random forest, Boosting
`rpart`	Klassifikasjonstrær (CART)	`rpart()`, `prune()`	CART, Bagging
`rpart.plot`	Plotting av klassifikasjonstrær	`rpart.plot()`, `rpart.rules()`	CART, Bagging
`ipred`	Bagging av klassifikasjonstrær	`bagging()`	Bagging
`forcats`	Rekoding og håndtering av factor-variable	`fct_lump()`, `fct_recode()`	Bagging
`randomForest`	Random forest-algoritmen	`randomForest()`, `varImpPlot()`, `partialPlot()`	Random forest
`fairness`	Mål på rettferdighet/bias	`acc_parity()`, `fnr_parity()`, `fpr_parity()`	Boosting
`gbm`	Gradient boosting	`gbm()`, `gbm.perf()`	Boosting

Pakke	Beskrivelse	Viktige funksjoner	Kapittel
`DALEX`	Forklarbare modell-objekter (explainers)	`explain()`	Fairness
`fairmodels`	Fairness-sjekk og visualisering	`fairness_check()`, `fairness_heatmap()`	Fairness
`gtsummary`	Pene sammendragstabeller	`tbl_cross()`, `tbl_summary()`	Boosting
`janitor`	Datarensing	`clean_names()`	Diverse
`dendextend`	Dendrogram-visualisering		Unsupervised
`directlabels`	Direkte labels i plot		Unsupervised
`GGally`	Utvidelser til ggplot2, scatterplotmatriser	`ggpairs()`	Unsupervised

1 Introduksjon til maskinlæring

Temaet er maskinlæring generelt og datadrevne beslutninger. Vi kommer inn på flere av temaene som vil behandles grundigere gjennom kurset.

1.1 Noen innledende metodiske begrep

I standard samfunnsvitenskapelige metodekurs lærer man først og fremst teknikker for å beskrive data og statistisk inferens for å beskrive usikkerheten rundt estimatene. Avhengig av studiets øvrige design kan resultatene tolkes kausalt og/eller generalisere til en nærmere veldefinert populasjon (Berk 2016).

Usikkerhet beskrives typisk ved hjelp standardfeil, p-verdier og konfidensintervall tilhørende spesifikke statistiske tester. Dette innebærer at man bruker statistiske *modeller* for hvordan resultatene ville sett ut under spesifikke forutsetninger. Samplingfordelinger som normalfordelingen og en del andre tilsvarende fordelinger er derfor sentralt. De fleste teknikkene vi skal bruke i dette kurset er *ikke* statistiske modeller i samme forstand og det er ingen antakelser om samplingfordelinger. Standardfeil og konfidensintervall kan derfor ikke regnes ut. Usikkerhet og hvem resultatene gjelder for er også relevante for maskinlæring, men ikke helt på samme måte.

1.2 Forklaringer og prediksjoner

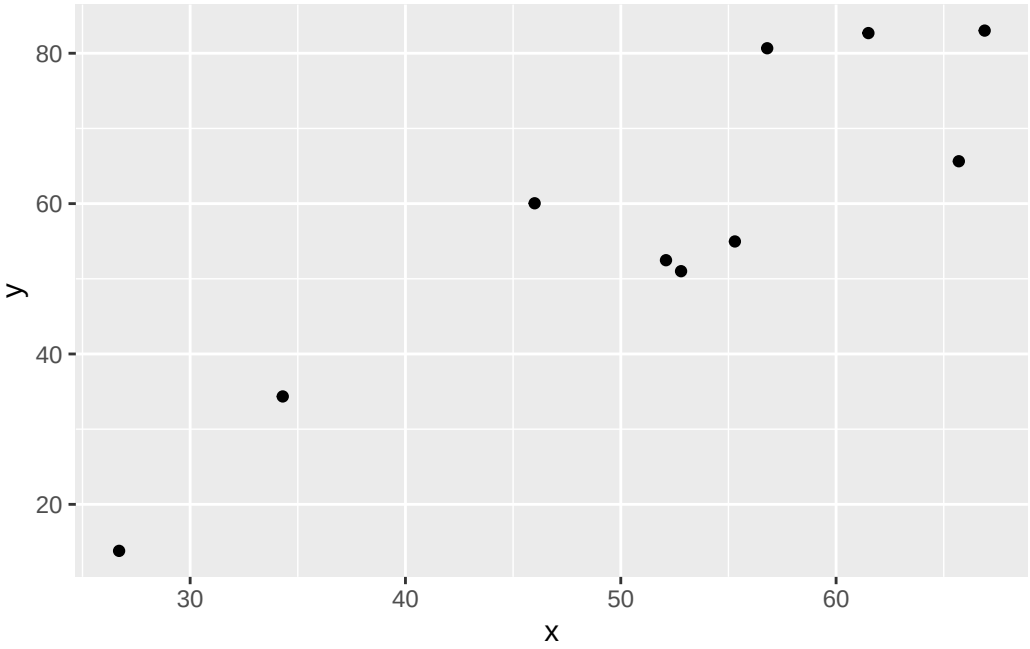
Vi er i liten grad interessert i regresjonskoeffisienter, β, og tolkning av denne. Derimot er vi interessert i det predikerte utfallet $\hat{y}_i$.

1.3 Overfitting: training og testing dataset

Når man tilpasser en statistisk modell eller algoritme til data så er det lett å tenke at modellen bør gjenspeile dataene så godt som mulig. Samtidig sies det ofte at modellene skal være så enkle som tilrådelig. En mer komplisert modell vil jo være i stand til å tilpasses dataene i større grad, så hvordan avveie dette?

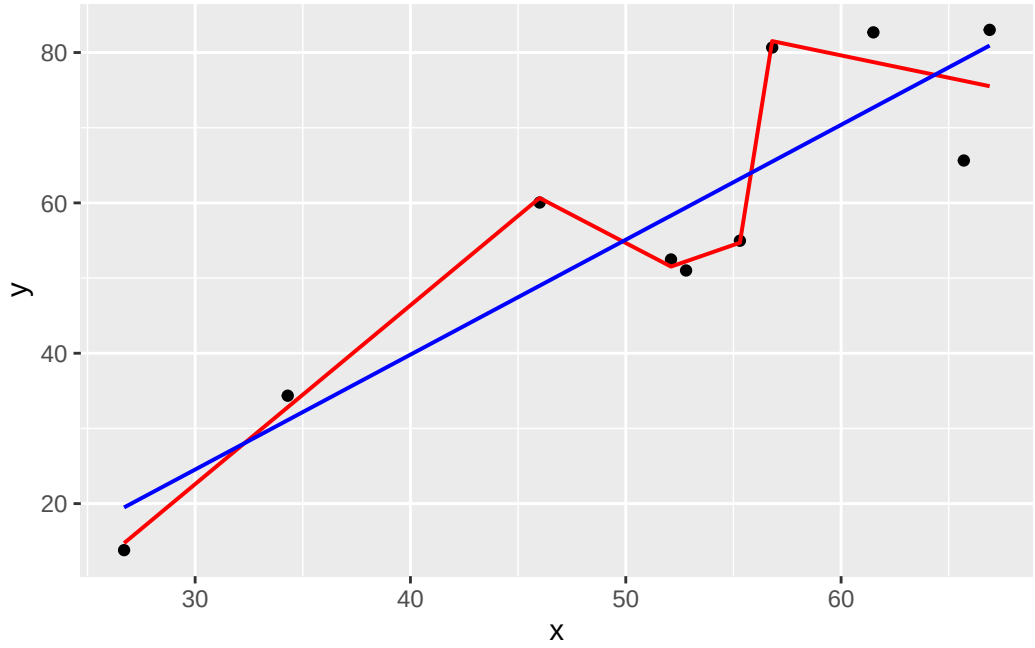
Spørsmålet nå er ikke hvor godt modellen passer til disse dataene, men hvordan den passer til *nye* data! Altså *fremtidige* data eller fremtidig situasjon. La oss si at vi har et datasett som kan plottes om følger:

```
library(tidyverse)
n <- 10
beta <- 1
set.seed(42)
x <- round(20 + runif(n)*50, digits = 1)
y <- 1 + beta*x + rnorm(n)*10

df <- data.frame(x = x, y = y) %>%
  mutate(d = case_when(x < 50 ~ 0,
                       x < 56 ~ 1,
                       TRUE ~2) %>% as_factor())


g1 <- ggplot(df, aes(x = x, y = y)) +
  geom_point()
g1
```



Vi kunne her tilpasse en enkel lineær regresjonsmodell eller en mer komplisert modell. Resultatet vises i grafen nedenfor.

```r
est1 <- lm(y ~ x + x*d, data = df)
est2 <- lm(y ~ x, data = df)

df_p1 <- df %>%
  mutate(pred = predict(est1))  %>%
  mutate(res_pred = y - pred,
         res_y = y - mean(y))


ggplot(df, aes(x = x, y = y)) +
  geom_point(col = "black") +
  geom_line(data = df_p1, aes(y = pred), col = "red", linewidth = .7) +
  stat_smooth(method='lm', formula = y ~ x, se = F, col = "blue", linewidth = .7)
```



Den kompliserte modellen gir $r^2 = 0.9567499$ mens den enkle lineære gir $r^2 = 0.8066771$.

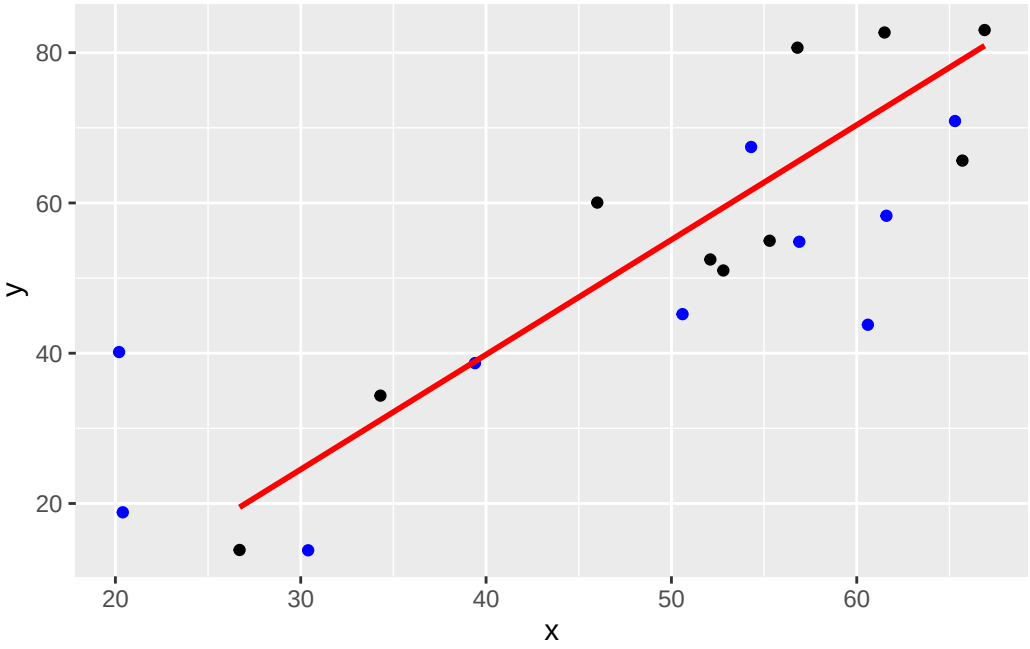
```r
summary(est1)$r.squared
```

```
[1] 0.9567499
```

```r
summary(est2)$r.squared
```

```
[1] 0.8066771
```

```r
x <- round(20 + runif(n)*50, digits = 1)
y <- 1 + beta*x + rnorm(n)*10
df2 <- data.frame(x = x, y = y)

g1 +
  geom_point(data = df2, col = "blue") +
  geom_smooth(method = lm, se = F, col = "red")
```



1.4 Klassifikasjonsusikkerhet - grunnleggende begreper

I praksis kommer vi til å fokusere mest på *klassifikasjon*, som altså er når det vi predikerer er en kategorisk variabel. For nye personer vil vi da gjerne predikere hvilken kategori vedkommende tilhører. En variant av dette er å predikere en fremtidig *handling* (slutte i jobb, ikke betale tilbake lån, begå ny kriminalitet etc).

1.4.1 Confusion matrix: riktig og feil klassifisering

Når vi predikerer et kategorisk utfall er det gjerne ett av utfallene vi primært er interessert i. Disse kalles *positive* og de andre er *negative*. Dette har ingenting å gjøre med om utfallet er bra eller dårlig å gjøre. Å predikere en sykdom vil være *positivt* og å være frisk vil være *negativt*. Å ha tilbakefall til kriminalitet vil være *positivt* og lovlydig vil være *negativt*.

En *positiv* prediksjon kan da være korrekt eller feil, og disse kalles da henholdsvis *sanne* eller *falske* positive. Tilsvarende kan en negaitv prediksjon være sann eller falsk.

		Predikert	
		Negativ	**Positiv**
Observert	**Negativ**	Sanne negative (TN)	Falske positive (FN)
	Positiv	Falske negative (FN)	Sanne positive (TP)

1.4.2 Asymetriske kostnader

Å predikere feil kan betraktes som en *kostnad*. Hvis vi skal bruke prediksjonenen til noe i praksis, så skal det jo få konsekvenser på en eller annen måte. Det kan innebære at man setter i verk tiltak som er unødvendige - eller ikke setter i verk tiltak der man burde gjort det.

Et sentralt spørsmål er derfor om begge typer feil er like viktig eller alvorlig. Noen ganger er det det, men det bør man ta stilling til helt konkret i det enkelte tilfellet. Det er ikke åpenbart hvem som har kompetanse til å vurdere dette. Det kan kreves inngående fagkunnskap for å gjøre en riktig vurdering, eller det kan være politiske prioriteringer, økonomiske forhold, rettferdighetsvurderinger osv. Det er i hvert fall ikke bare opp til forskeren eller IT-personalet å vurdere.

Dette kan koke ned til helt konkret vurdering av hvor mange falske positive er du villig til å godta per falske negative. Et konkret eksempel er studien til Berk et al (2016) av menn som er arrestert for vold i nære relasjoner. Problemstillingen er hvem skal i arrest frem til saken kommer opp og hvem skal løslates mot kausjon. Prediksjonen er da hvem som vil begå ny voldshandling mot partner. Forholdet mellom TN og FN oversettes konkret til hvor mange mistenkte skal sitte unødig i fengsel (dvs. falske positive) mot hvor mange partnere skal unødig utsettes for ny voldshendelse (dvs. falske negative)? I nevnte studie har noen "stakeholders" landet på at falske negative er vesentlig mer alvorlig enn falske positive. Prediksjonsmodellen utformes så for å reflektere akkurat det.

1.5 Rettferdighet og rimelighet

I diskusjoner av anvendelser av maskinlæring står rettferdighet helt sentralt. Men det er ikke alltid like klart hva dette egentlig betyr utover at det er forskjellsbehandling. Tross alt er hele formålet med prediksjon å nettopp forskjellsbehandle, eller *målrette* som det også kan kalles. Rettferdighet kommer inn på flere nivåer, fra det helt prinsippielle ved å la data og datamaskiner ha betydning for avgjørelser til det helt konkrete til hvordan feilratene bør se ut. Vurdering av asymetriske kostnader er åpenbart også et rettferdighetsspørsmål med etiske implikasjoner. En variant er at disse feilratene kan se forskjellige ut på tvers av undergrupper.

Hvilke konsekvenser som er akseptable er viktig. Men det er også viktig *før* man bygger modellen. Software kommer med default innstillinger, men det betyr jo ikke at de er "nøytrale". Resultatene kan til en viss grad styres, så det må man rett og slett gjøre.

1.5.1 Fundamentale skjevheter i data

Siden maskinlæring baserer seg på å lære av tilgjengelige data for å benytte det på nye tilfeller spiller det vesentlig rolle hvordan de opprinnelige dataene ble generert i utgangspunktet.

Et velkjent eksempel er hvordan Amazon besluttet å slutte å bruke en algoritme for rekruttering fordi den systematisk valgte bort kvinner. Grunnen til at algoritmen gjorde dette var så enkelt som at dataene den var trent opp på var mannsdominert. Algoritmen hadde altså primært tilgang til informasjon om hvilke egenskaper som kjennetegnet talentfulle *mannlige* kandidater, som altså kan være forskjellige fra talentfulle *kvinnelige* kandidater.

Når man skal ta en algoritme i bruk er det derfor helt avgjørende at man kan forsvare bruken av de dataene algoritmen er trent på. Kjente skjevheter kan i prinsippet motarbeides ved *tuning* (dette kommer vi tilbake til), men det er vanskelig å garantere at det er skjevheter man *ikke* har tenkt på.

2 Lineær regresjon

I dette kapittelt skal vi bruke følgende pakker:

```
invisible(Sys.setlocale(locale='no_NB.utf8'))

library(tidyverse)   # datahåndtering, grafikk og glimpse()
library(rsample)     # for å dele data i training og testing
library(splines)     # for spline-regresjon
library(Ecdat)       # inneholder datasettet Clothing
library(mgcv)        # for generaliserte additive modeller (GAM)
```

Lineær regresjon slik vi skal bruke det her er helt vanlig regresjon slik du har lært om i grunnleggende kurs i kvantitative metoder. Altså lineær funksjon estimert med minste kvadraters metode. Alt du har lært før gjelder også her. Forskjellen er at vi nå ikke er så interessert i å tolke β men i den predikerte $\hat{y}$.

2.1 Lese inn data

Vi illustrerer lineær regresjon med et empirisk eksempel. Her skal vi bruke data for norske kommuner i 2016. La oss si at vi er interessert i hvordan antall voldshendelser per 1000 innbyggere vil endre seg i en kommune. Dette kunne være relevant for langtidsplanlegging av forebygging, politibemanning, helsetjenester osv. Det kan være et område som er i stor endring slik at befolkningssammensetningen forventes å endre seg og/eller det er endrede lokale økonomiske utsikter.

Først leser vi inn dataene og tar en titt på variabellisten.

```
kommune <- readRDS( "data/kommunedata.rds")
glimpse(kommune)
```

```
Rows: 1,529
Columns: 28
$ kommune_nr             <chr> "0101", "0101", "0101", "0101", "0104", "0104",~
```

```
$ kommune                <chr> "Halden (-2019)", "Halden (-2019)", "Halden (-
2~
$ year                   <dbl> 2015, 2016, 2017, 2018, 2015, 2016, 2017, 2018,~
$ bef_18min              <int> 3556, 3503, 3505, 3544, 3594, 3652, 3704, 3655,~
$ bef_18_25              <int> 3575, 3585, 3432, 3438, 3405, 3404, 3355, 3370,~
$ bef_26_35              <int> 3728, 3804, 3985, 4035, 4057, 4071, 4124, 4110,~
$ bef_totalt             <int> 30328, 30544, 30790, 31037, 31802, 32182, 32407~
$ menn_18_25             <int> 1847, 1865, 1813, 1819, 1789, 1802, 1789, 1810,~
$ menn_26_35             <int> 1880, 1919, 2005, 2062, 2063, 2083, 2113, 2134,~
$ menn_36_67             <int> 7067, 7051, 7085, 7057, 7418, 7453, 7408, 7407,~
$ menn_67plus            <int> 2496, 2624, 2697, 2806, 2671, 2777, 2856, 2895,~
$ menn_18min             <int> 1880, 1847, 1873, 1876, 1842, 1885, 1919, 1878,~
$ kvinner_18_25          <int> 1728, 1720, 1619, 1619, 1616, 1602, 1566, 1560,~
$ kvinner_26_35          <int> 1848, 1885, 1980, 1973, 1994, 1988, 2011, 1976,~
$ kvinner_36_67          <int> 6880, 6832, 6844, 6848, 7479, 7519, 7537, 7596,~
$ kvinner_67plus         <int> 3026, 3145, 3242, 3309, 3178, 3306, 3423, 3555,~
$ kvinner_18min          <int> 1676, 1656, 1632, 1668, 1752, 1767, 1785, 1777,~
$ inntekt_totalt_median  <int> 555000, 562000, 580000, 591000, 561000, 568000,~
$ inntekt_eskatt_median  <int> 451000, 453000, 470000, 480000, 449000, 456000,~
$ ant_husholdninger      <int> 13890, 14124, 14281, 14454, 15046, 15132, 15313~
$ shj_klienter           <int> 1183, 1137, 1099, 1128, 1155, 1129, 1152, 1137,~
$ shj_unge               <int> 262, 247, 248, 242, 267, 263, 238, 222, 307, 28~
$ vinningskriminalitet   <dbl> 19.7, 18.7, 16.5, 14.5, 24.5, 21.5, 18.0, 18.0,~
$ voldskriminalitet      <dbl> 11.2, 12.6, 12.3, 11.2, 7.8, 8.3, 8.7, 9.7, 6.8~
$ nark_alko_kriminalitet <dbl> 21.0, 21.9, 21.0, 20.3, 12.0, 10.2, 10.9, 10.1,~
$ ordenslovbrudd         <dbl> 18.5, 16.5, 14.9, 13.7, 8.9, 9.0, 9.1, 9.2, 8.2~
$ trafikklovbrudd        <dbl> 15.5, 16.3, 16.7, 19.2, 7.4, 6.3, 6.9, 8.0, 9.6~
$ andre_lovbrudd         <dbl> 25.5, 26.5, 26.1, 25.2, 12.1, 12.2, 11.9, 12.4,~
```

2.1.1 Training og testing data

Vi starter med å dele datasettet i *training* og *testing*. Her kan vi bruke pakken 'rsample' og funksjonen `initial_split()` etterfulgt av funksjonene `training()` og `testing()`. Forhåndsvalget for splitten er 0.75, så dermed brukes 75% til training og resten til testing. Du kan også legge til argumentet `prop =` og sette en annen andel, f.eks. 0.7 for 70%.

Merk bruken av `set.seed()`. For å splitte genererer R tilfeldige tall, og *seed* styrer hvor den algoritmen starter. Når *seed* er satt vil du få nøyaktig samme resultatet som gjort her. Dette vil du trenge på eksamen for at sensor skal kunne sjekke resultatene dine, så begynn å bruke det med en gang. Tallet inni parentesen betyr ingenting[1] og bare sørger for reproduserbarhet der hvor det er tilfeldige tall involvert.

[1] Bortsett fra for dem som mener det er svaret på "the Ultimate Question of Life, the Universe, and Everything"

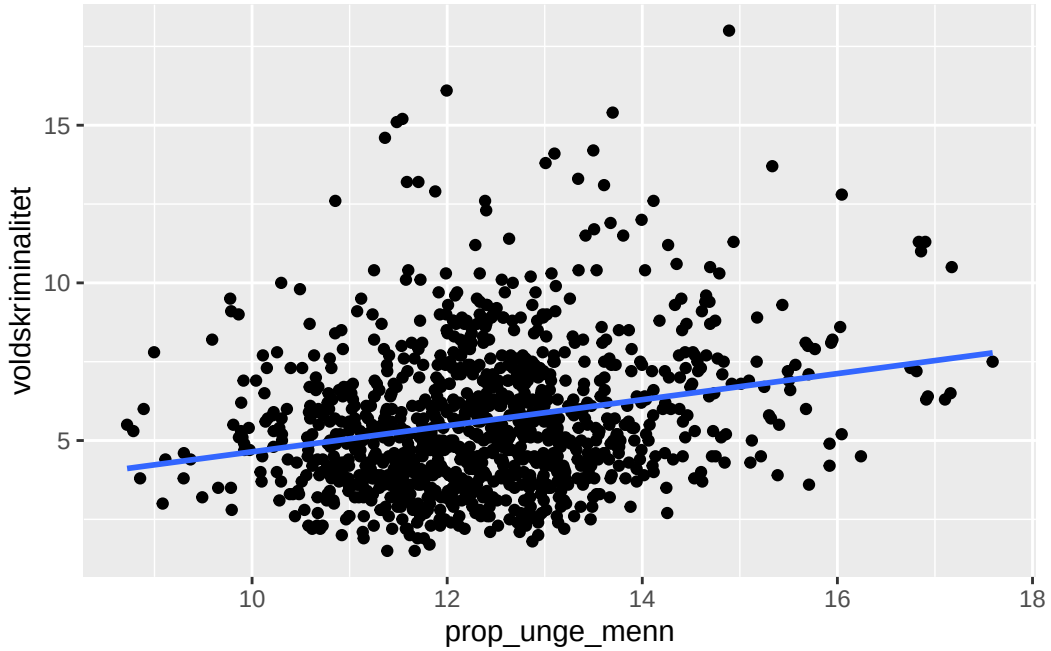
```r
set.seed(42)
kommune_split <- initial_split(kommune, prop = .7)

kommune_train <- training(kommune_split)
kommune_test <- testing(kommune_split)
```

2.1.2 Enkel lineær regresjon i R

En ganske åpenbar faktor som forklarer forekomsten av vold er andel unge menn i kommunen. Rett og slett fordi dette er den demografiske gruppen som begår mest vold - og kriminalitet generelt, faktisk. Hvis befolkningssammensetningen forventes å bli yngre vil det medføre flere unge menn, og da kan vi kanskje forvente at det blir flere voldshendelser bare av den grunn? Sammenhengen mellom unge menn og voldsrate kan estimeres med helt vanlig lineær regresjon.

En god start på de fleste empiriske analyser er å beskrive sammenhengen med et plot. Her legger vi på en lineær regresjonslinje med `geom_smooth()` der vi presiserer lineær modell med `method = "lm"` og lar være å ta med konfidensintervallet `se = FALSE`.

```r
kommune_train <- kommune_train %>%
  mutate(prop_unge_menn = (menn_18_25 + menn_26_35)/bef_totalt*100)

ggplot(kommune_train, aes(x = prop_unge_menn,
                      y = voldskriminalitet)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```



Lineær regresjon estimeres med `lm()` og du kan få en enkel output med bruk av `summary()`.

```
est <- lm(voldskriminalitet ~ prop_unge_menn, data = kommune_train)
summary(est)
```

```
Call:
lm(formula = voldskriminalitet ~ prop_unge_menn, data = kommune_train)

Residuals:
    Min      1Q  Median      3Q     Max
-4.0302 -1.5764 -0.3951  1.1826 11.3367

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     0.50954    0.63482   0.803    0.422
prop_unge_menn  0.41329    0.05097   8.109  1.4e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.248 on 1068 degrees of freedom
Multiple R-squared:  0.05799,   Adjusted R-squared:  0.05711
F-statistic: 65.75 on 1 and 1068 DF,  p-value: 1.395e-15
```

Man kan også hente ut kun r^2 med følgende kode:

```
summary(est)$r.squared
```

```
[1] 0.05799176
```

For ordens skyld: I tidligere metodekurs har du kanskje lært å få ut en penere regresjonstabell med f.eks. gtsummary-pakken. (Det finnes andre pakker også som gjør tilsvarende). Hvordan output er formatert spiller ingen rolle. I denne sammenhengen har vi lite bruk for en pen regresjonstabell da β ikke er av primær interesse.

Med andre ord kan voldsraten beskrives som:

$$\hat{\text{voldskriminalitet}} = 0.51 + 0.41(\text{prop\_unge\_menn})$$

Men vi har også sett at r^2 er ganske lav, bare 0.058. Denne koeffisienten kalles også "coefficient of determination" og sier noe om i hvor stor grad modellen fanger opp variasjoenen i dataene. En lav r^2 betyr at modellen i liten grad gjør det. Vi må altså forvente at modellen vil bomme ganske kraftig i sine prediksjoner. Vi kan velge å ta modellen seriøst likevel, men ikke ha for store forventninger for prediksjonene!

Et annet mål på hvor godt modellen treffer er "Root mean square error", RMSE. Dette kan skrives som:

$$rmse = \sqrt{\frac{\sum (O_i - P_i)^2}{N}}$$

der O er de observerte verdiene og P er de predikerte verdiene for observasjon i. Merk at $(O_i - P_i)$ er residualene. I R kan vi hente ut residualene fra regresjons-objektet med dollartegnet `...$res` etter objektnavnet. Da kan du regne ut RMSE som følger:

```
rmse <- sqrt(mean(est$res^2))
rmse
```

```
[1] 2.245685
```

RMSE sier altså omtrentlig hvor mye modellen i gjennomsnitt bommer på de observerte verdiene. [2]. Hvorvidt det er presist *nok* eller ikke vil vel strengt tatt komme an på behovet for presisjon, altså: hva man skal bruke det til.

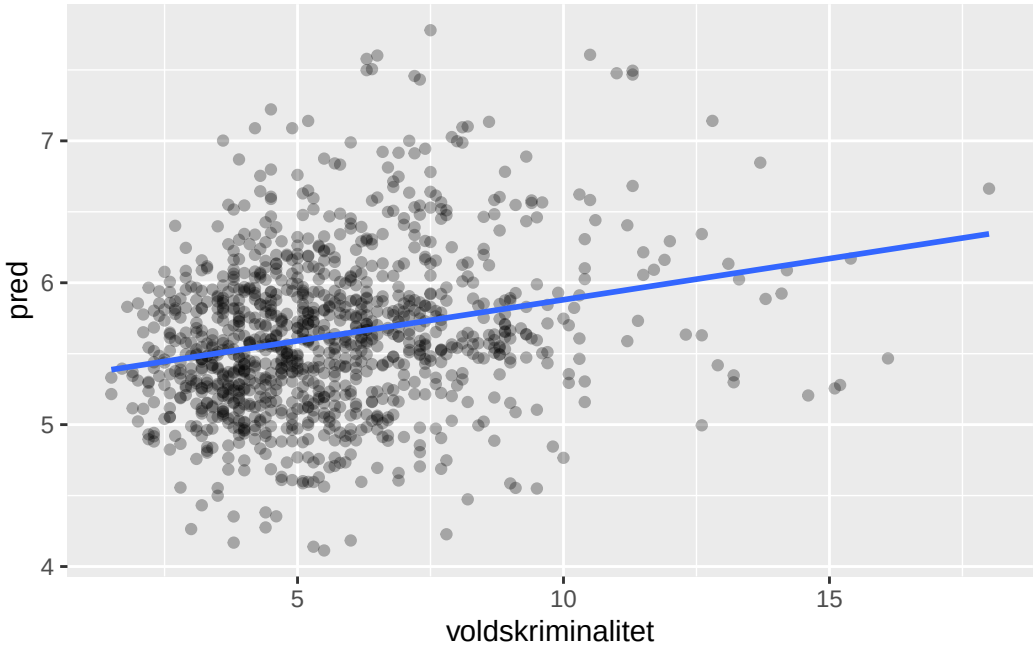
[2] Denne formuleringen er ganske omtrentlig. RMSE er egentlig kvadratroten av gjennomsnittet til de kvadrerte residualene, som er noe litt annet enn gjennomsnittet av de absolutte verdiene av residualene. Det gir bl.a. litt mer vekt til store residualer enn et vanlig gjennomsnitt.

For å få litt bedre tak på hva RMSE betyr kan vi se på et plot av de predikerte og observerte verdiene. Vi kan predikere vold for hver enkelt kommune basert på denne modellen, som altså er den forventede voldsraten *hvis modellen er sann*. Funksjonen `predict()` gir oss hva vi trenger.

```
kom <- kommune_train %>%
  mutate(pred = predict(est))
```

Merk at koden her lagde en kopi av datasettet der vi har alle de opprinnelige variablene pluss en variabel med de predikerte verdiene. Vi kan nå sammenlignet prediksjonene med de observerte utfallene.

```
ggplot(kom, aes(x = voldskriminalitet, y = pred)) +
  geom_point(alpha = .3) +
  geom_smooth(method = "lm", se = FALSE)
```



Hvis prediksjonen hadde vært perfekt ville disse punktene ligget på linja, noe den jo ikke gjør. Modellen bommer altså ganske mye.

Hva hvis vi vil vite forventet voldsrate for en kommune for en gitt andel unge menn? Løsningen er å lage et nytt datasett med de verdiene vi er interessert i og så predikere for dette datasettet med å spesifisere `newdata = dt`. Her er et eksempel der vi ønsker å vite voldsraten hvis andelen unge menn er 15%.

```r
dt <- data.frame(prop_unge_menn = 15)
p <- predict(est, newdata = dt)
p
```

```
       1
6.708919
```

I følge modellen vil altså en kommune der 15% av populasjonen er unge menn ha en 6.709 voldshendelser per 1000 innbyggere. Fra tradisjonell statistikk vet vi jo at det er usikkerhet knyttet til dette estimatet og vi kan også ta det med i beregningen her. Vanligvis vil man estimere med et *konfidensintervall*, som gjelder hvis man estimerer et gjennomsnitt i en gruppe. Her skal vi derimot predikere for en enkelt kommune, som da har større usikkerhet enn om man estimerer for en enkelt observasjon. Dette kalles prediksjonsintervall og må spesifiseres i koden. Hvis det ikke er gitt vil R gi konfidensintervallet.

```r
p_ki <- predict(est, newdata = dt, interval = "prediction")
p_ki
```

```
       fit      lwr      upr
1 6.708919 2.288514 11.12932
```

Tolkningen er ellers tilsvarende som for konfidensintervall: vi forventer med "95% sannsynlighet"[3] at voldsraten vil være mellom 2.3 og 11.1 per 1000 innbyggere.

2.1.3 Multippel regresjon

Enkel regresjon er nettopp enkel og prediksjonen blir ikke så god. Men vi kan komplisere vesentlig ved å inkludere flere variable og bruke alle triksene man evt. har lært om multippel regresjon tidligere, primært interaksjonsledd, polynomer og transformasjoner osv. (Det er ok om du ikke har lært alt dette tidligere).

I R vil vi da bare legge til flere variabelnavn i formelen. Ellers er det meste likt som for enkel lineær regresjon.

```r
est_m <- lm(voldskriminalitet ~ prop_unge_menn + inntekt_totalt_median + shj_unge +
                  ant_husholdninger ,
            data = kommune_train)
summary(est_m)
```

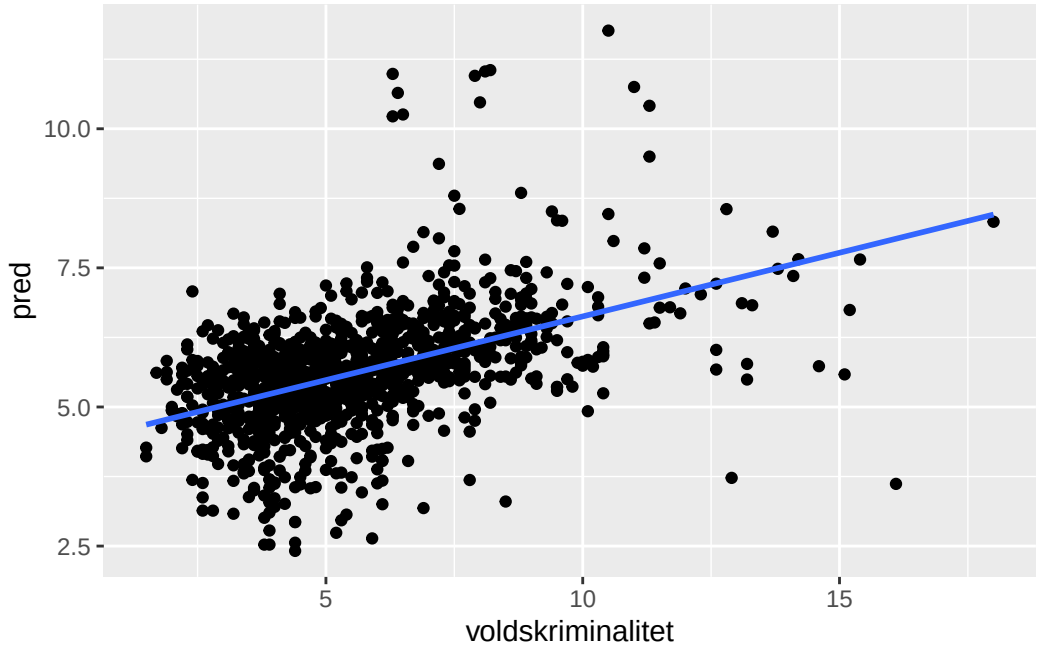
[3] Dette er en omtrentelig formulering. Alle sannsynligheter gjelder i det lange løp: altså hvis man gjør undersøkelsen veldig mange ganger.

```
Call:
lm(formula = voldskriminalitet ~ prop_unge_menn + inntekt_totalt_median +
    shj_unge + ant_husholdninger, data = kommune_train)

Residuals:
    Min      1Q  Median      3Q     Max
-4.6862 -1.3599 -0.2042  1.0689 12.4822

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)             7.421e+00  7.333e-01  10.121  < 2e-16 ***
prop_unge_menn          4.177e-01  5.328e-02   7.840 1.09e-14 ***
inntekt_totalt_median -1.099e-05  8.539e-07 -12.871  < 2e-16 ***
shj_unge                4.461e-03  1.016e-03   4.392 1.23e-05 ***
ant_husholdninger      -2.163e-05  8.361e-06  -2.587  0.00983 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.037 on 1065 degrees of freedom
Multiple R-squared:  0.2288,    Adjusted R-squared:  0.2259
F-statistic:    79 on 4 and 1065 DF,  p-value: < 2.2e-16
```

Merk at r^2 nå har gått betraktelig opp, til ca 0.23. Gitt at vi tolker dette som i hvor stor grad vi kan *predikere* utfallet fra datasettet, så er det kanskje likevel ikke imponerende høyt: vi vil fremdeles forvente mye feil prediksjon.

Her er et scatterplot av observert mot forventet voldsrater:

```r
kom_pred <- kommune_train %>%
  mutate(pred = predict(est_m))

ggplot(kom_pred, aes(x = voldskriminalitet, y = pred)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```



2.1.4 Sammenligning av modeller med ulik kompleksitet

La oss nå systematisk sammenligne flere modeller med ulik kompleksitet. Vi lager først et datasett med utvalgte variable og bruker formelen `voldskriminalitet ~ .` for å ta med alle variable.

```
kom_s <- kommune_train %>%
  select(-c(kommune, kommune_nr, ordenslovbrudd,
            nark_alko_kriminalitet, trafikklovbrudd, andre_lovbrudd,
            prop_unge_menn))

full_mod <- lm(voldskriminalitet ~ . , data = kom_s)
summary(full_mod)
```

```
Call:
lm(formula = voldskriminalitet ~ ., data = kom_s)

Residuals:
    Min      1Q  Median      3Q     Max
-3.9650 -1.2333 -0.1813  0.8919 12.2114
```

```
Coefficients: (4 not defined because of singularities)
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)           -1.198e+02  9.309e+01  -1.287  0.19851
year                   6.234e-02  4.646e-02   1.342  0.17998
bef_18min              3.156e-03  1.839e-03   1.716  0.08637 .
bef_18_25             -1.885e-05  1.513e-03  -0.012  0.99006
bef_26_35              1.180e-03  1.006e-03   1.173  0.24109
bef_totalt            -2.272e-03  7.130e-04  -3.187  0.00148 **
menn_18_25             3.698e-03  2.136e-03   1.731  0.08373 .
menn_26_35            -1.481e-03  2.013e-03  -0.736  0.46197
menn_36_67             2.710e-03  9.461e-04   2.865  0.00425 **
menn_67plus            1.654e-03  1.339e-03   1.236  0.21676
menn_18min             2.461e-03  2.875e-03   0.856  0.39223
kvinner_18_25                 NA         NA      NA       NA
kvinner_26_35                 NA         NA      NA       NA
kvinner_36_67         -1.269e-04  9.021e-04  -0.141  0.88816
kvinner_67plus                NA         NA      NA       NA
kvinner_18min                 NA         NA      NA       NA
inntekt_totalt_median -4.693e-05  7.600e-06  -6.176 9.40e-10 ***
inntekt_eskatt_median  5.580e-05  1.108e-05   5.037 5.55e-07 ***
ant_husholdninger      1.555e-03  3.169e-04   4.908 1.07e-06 ***
shj_klienter           2.170e-03  9.967e-04   2.177  0.02968 *
shj_unge               2.191e-03  3.054e-03   0.718  0.47322
vinningskriminalitet   1.066e-01  1.014e-02  10.513  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.891 on 1052 degrees of freedom
Multiple R-squared:  0.343, Adjusted R-squared:  0.3324
F-statistic:  32.3 on 17 and 1052 DF,  p-value: < 2.2e-16
```

r^2 gikk noe opp, til 0.343.

2.1.5 Eksempel på overfitting

Men vi kan gjøre modellen ekstra komplisert ved inkludere alle mulige interaksjonsledd. En åpenbar ulempe med dette er at hver enkelt koeffisent blir svært mye vanskeligere å tolke. Vi fokuserer derfor kun på r^2 og RMSE.

Her bygger vi fire modeller med økende kompleksitet og sammenligner RMSE for trainingdata:

```r
est1 <- est     # enkel modell med bare prop_unge_menn (estimert over)
est2 <- est_m   # multippel regresjon (estimert over)
est3 <- lm(voldskriminalitet ~ .^2, data = kom_s)    # alle 2-veis interaksjoner
est4 <- lm(voldskriminalitet ~ .^3, data = kom_s)    # alle 3-veis interaksjoner
```

RMSE for training-data:

```r
sqrt(mean(est1$res^2))
```

```
[1] 2.245685
```

```r
sqrt(mean(est2$res^2))
```

```
[1] 2.031873
```

```r
sqrt(mean(est3$res^2))
```

```
[1] 1.598957
```

```r
sqrt(mean(est4$res^2))
```

```
[1] 0.9963221
```

For training-data ser vi altså at RMSE synker for hver modell. Mer kompleksitet gir tilsynelatende bedre tilpassning. Men hva skjer med testing-data?

2.1.6 Predikere for nye data

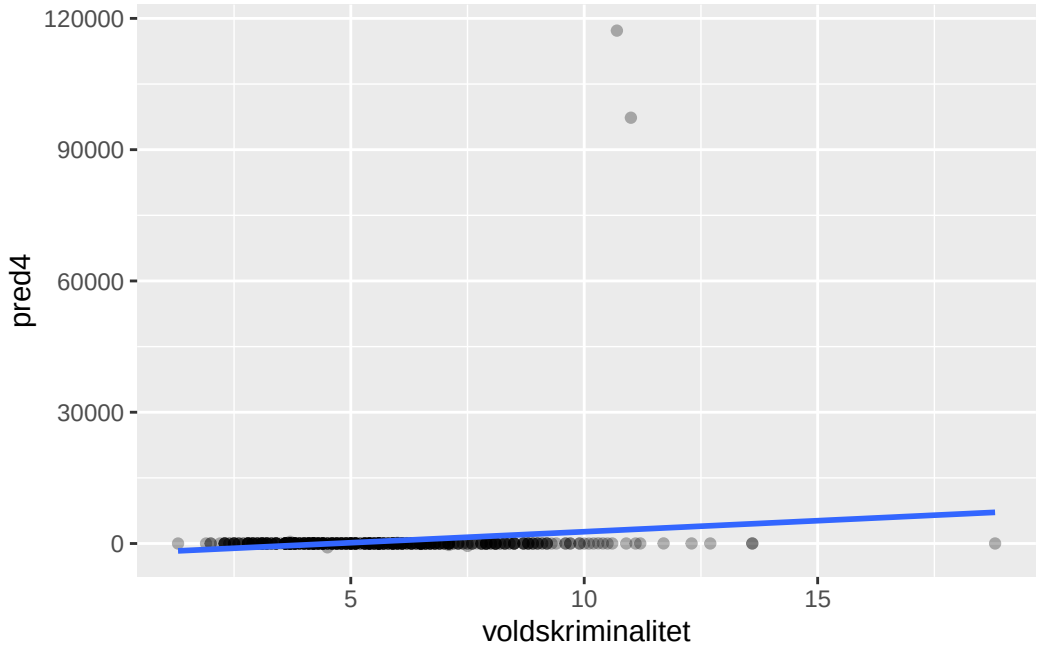
Nå har vi bare sett på hvordan prediksjonen fungerer på training-datasettet. Vi må sjekke med testing-datasettet. Det lagde vi i begynnelsen, så nå henter vi det frem. I `predict()` må vi nå angi `newdata =`. Vi predikerer fra alle fire modellene samtidig og regner ut RMSE for testing-data:

```r
kommune_test <- kommune_test %>%
  mutate(prop_unge_menn = (menn_18_25 + menn_26_35)/bef_totalt*100)

kommune_pred <- kommune_test %>%
  mutate(pred1 = predict(est1, newdata = .),
         pred2 = predict(est2, newdata = .),
         pred3 = predict(est3, newdata = .),
         pred4 = predict(est4, newdata = .))

rmse_test <- kommune_pred %>%
  mutate(res1 = pred1 - voldskriminalitet,
         res2 = pred2 - voldskriminalitet,
         res3 = pred3 - voldskriminalitet,
         res4 = pred4 - voldskriminalitet) %>%
  summarise(RMSE_enkel = sqrt(mean(res1^2)),
            RMSE_multippel = sqrt(mean(res2^2)),
            RMSE_2veis = sqrt(mean(res3^2)),
            RMSE_3veis = sqrt(mean(res4^2)))

rmse_test
```

```
  RMSE_enkel RMSE_multippel RMSE_2veis RMSE_3veis
1   2.147951       1.876014   23.41166   7109.869
```

Her ser vi at de mest kompliserte modellene gir *dårligere* prediksjon på nye data! La oss også plotte predikert mot observert for den mest kompliserte modellen:

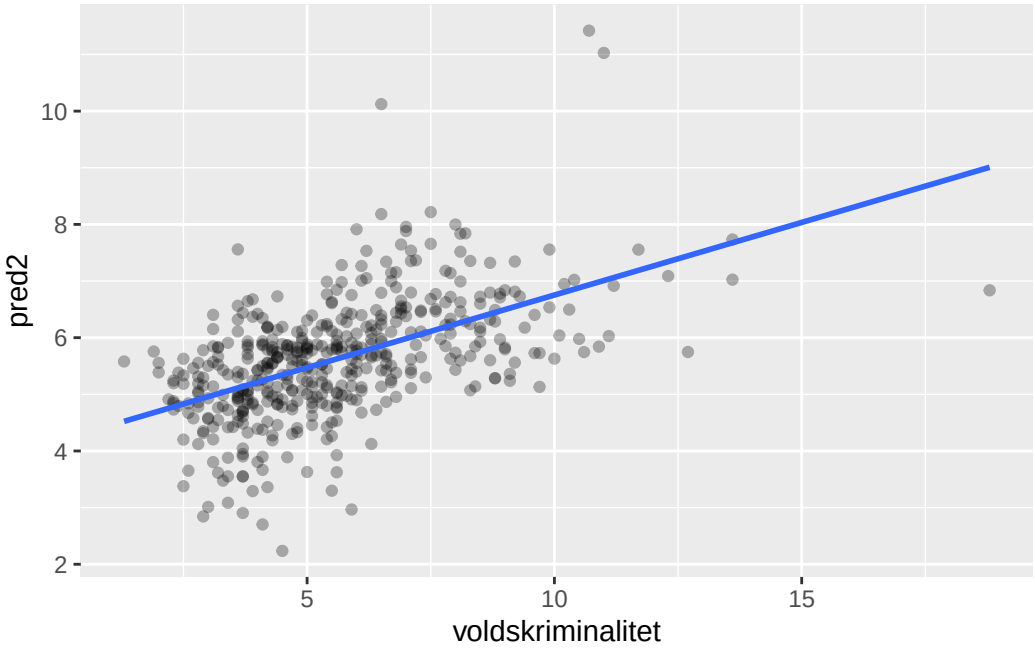
```r
ggplot(kommune_pred, aes(x = voldskriminalitet, y = pred4)) +
  geom_point(alpha = .3) +
  geom_smooth(method = "lm", se = FALSE)
```



Ser dette rart ut? Det er noen observasjoner som har predikert skyhøy voldsrate! Disse prediksjonene bommer voldsomt.

Sammenlignet er den enklere multippelregresjonsmodellen vesentlig bedre på nye data:

```
ggplot(kommune_pred, aes(x = voldskriminalitet, y = pred2)) +
  geom_point(alpha = .3) +
  geom_smooth(method = "lm", se = FALSE)
```



Dette er en ganske ekstrem variant av *overfitting*. Ved å formulere en veldig komplisert modell som passer veldig godt til training-dataene kan vi ende opp med en modell som passer veldig, veldig dårlig til nye data. Hvis man skulle utforme f.eks. politikktiltak på grunnlag av en slik prediksjon vil det kunne gå riktig så dårlig.

Det er altså slik at en enklere modell kan passe til *nye* data langt bedre enn en veldig komplisert modell. Grunnen er overfitting: modellen fanger opp vel så mye tilfeldig støy som det underliggende mønsteret. Støyen vil jo være annerledes i de nye dataene.

2.2 Splines

I modellene over har vi antatt en lineær sammenheng mellom prediktor og utfall. Men hva hvis sammenhengen *ikke* er lineær? En mulighet er å bruke *splines*, som er stykkevis definerte funksjoner som kan tilpasse mer fleksible former. Splines er nyttige for prediksjon fordi de kan fange opp ikke-lineære mønstre i dataene uten at vi trenger å spesifisere funksjonsformen på forhånd.

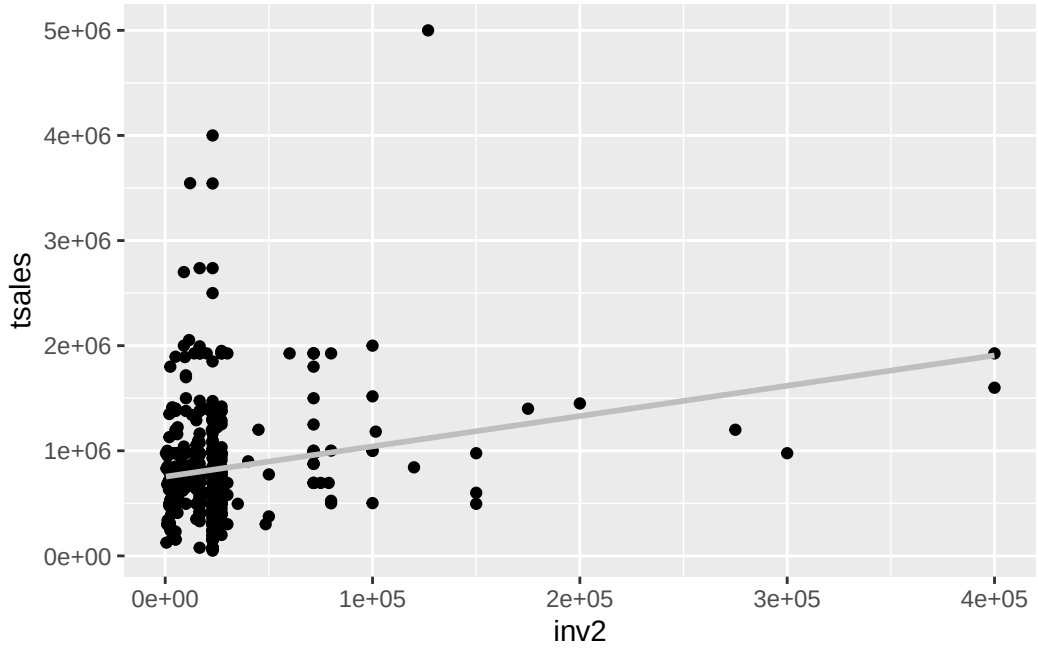
Vi bruker datasettet Clothing fra pakken `Ecdat` som eksempel. Variabelen `tsales` er totalt salg og `inv2` er investeringer.

```
data(Clothing)
glimpse(Clothing)
```

```
Rows: 400
Columns: 13
$ tsales  <int> 750000, 1926395, 1250000, 694227, 750000, 400000, 1300000, 495~
$ sales   <dbl> 4411.765, 4280.878, 4166.667, 2670.104, 15000.000, 4444.444, 3~
$ margin  <dbl> 41, 39, 40, 40, 44, 41, 39, 28, 41, 37, 32, 30, 39, 37, 34, 44~
$ nown    <dbl> 1.0000, 2.0000, 1.0000, 1.0000, 2.0000, 2.0000, 1.2228, 2.0000~
$ nfull   <dbl> 1.0000, 2.0000, 2.0000, 1.0000, 1.9556, 1.9556, 1.0000, 1.9556~
$ npart   <dbl> 1.0000, 3.0000, 2.2222, 1.2833, 1.2833, 1.2833, 3.0000, 1.2833~
$ naux    <dbl> 1.5357, 1.5357, 1.4091, 1.3673, 1.3673, 1.3673, 4.0000, 1.3673~
$ hoursw  <int> 76, 192, 114, 100, 104, 72, 161, 80, 158, 87, 156, 40, 96, 305~
$ hourspw <dbl> 16.755960, 22.493760, 17.191200, 21.502600, 15.742790, 10.8988~
$ inv1    <dbl> 17166.67, 17166.67, 292857.20, 22207.04, 22207.04, 22207.04, 2~
$ inv2    <dbl> 27177.04, 27177.04, 71570.55, 15000.00, 10000.00, 22859.85, 22~
$ ssize   <int> 170, 450, 300, 260, 50, 90, 400, 100, 450, 75, 120, 40, 90, 11~
$ start   <dbl> 41, 39, 40, 40, 44, 41, 39, 28, 41, 37, 32, 30, 39, 37, 34, 44~
```

La oss først se på sammenhengen med et scatterplot og en lineær regresjonslinje:

```
p0 <- ggplot(Clothing, aes(inv2, tsales)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, col = "gray")

p0
```



Sammenhengen ser ikke helt lineær ut. Vi kan prøve en *piecewise linear spline* der vi angir knuter (knots) der funksjonen skifter retning. Funksjonen `bs()` fra pakken `splines` brukes til dette.

2.2.1 Piecewise linear spline

Med `degree = 1` får vi en stykkevis lineær funksjon, altså rette linjer mellom knutene:

```
model1 <- lm(tsales ~ bs(inv2, knots = c(12000, 60000, 150000), degree = 1), data = Clothing)
summary(model1)
```

```
Call:
lm(formula = tsales ~ bs(inv2, knots = c(12000, 60000, 150000),
    degree = 1), data = Clothing)

Residuals:
    Min      1Q  Median      3Q     Max
-983725 -333443  -97927  178664 3670029

Coefficients:
                                                    Estimate Std. Error
(Intercept)                                           717496      84585
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)1    78041     107382
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)2   183187     134824
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)3   761569     242818
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)4   804300     365549
                                                    t value Pr(>|t|)
(Intercept)                                           8.483 4.47e-16 ***
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)1   0.727  0.46780
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)2   1.359  0.17501
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)3   3.136  0.00184 **
bs(inv2, knots = c(12000, 60000, 150000), degree = 1)4   2.200  0.02837 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 572000 on 395 degrees of freedom
Multiple R-squared:  0.0486,    Adjusted R-squared:  0.03897
F-statistic: 5.044 on 4 and 395 DF,  p-value: 0.0005649
```

2.2.2 B-spline (kubisk)

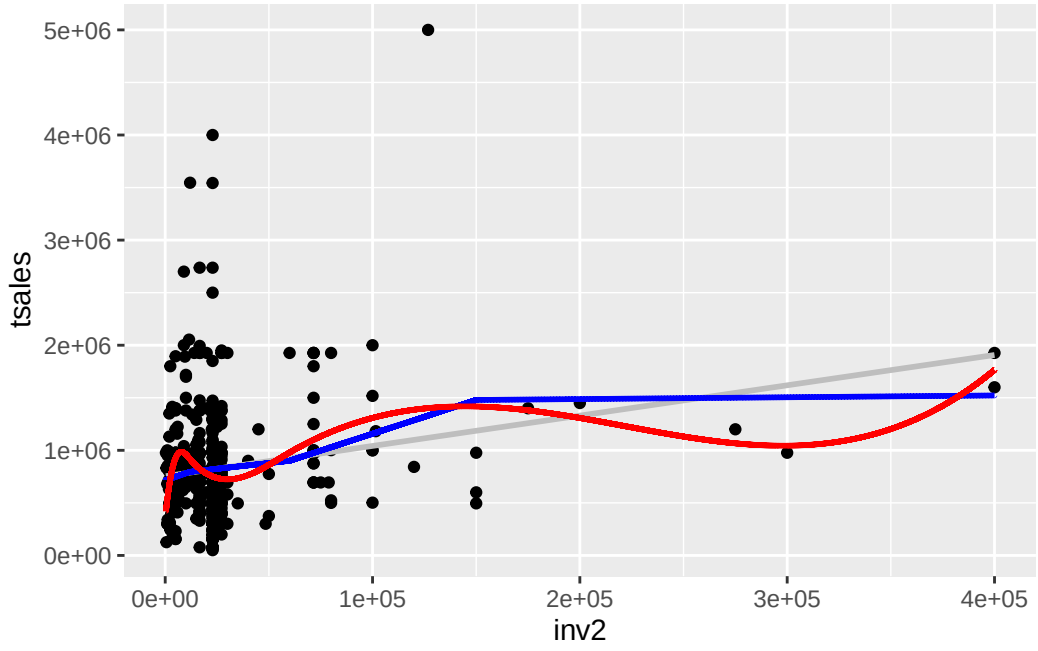
Med `degree = 3` får vi en glattere, kubisk funksjon:

```
model2 <- lm(tsales ~ bs(inv2, knots = c(12000, 60000, 150000), degree = 3), data = Clothing)
summary(model2)
```

```
Call:
lm(formula = tsales ~ bs(inv2, knots = c(12000, 60000, 150000),
    degree = 3), data = Clothing)

Residuals:
    Min      1Q  Median      3Q     Max
-921708 -329164 -125551  223018 3598276

Coefficients:
                                                    Estimate Std. Error
(Intercept)                                           419589     136370
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)1   712180     213810
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)2    63428     140939
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)3   847253     269728
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)4  1308842     707178
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)5   -14067     996832
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)6  1345263     419450
                                                    t value Pr(>|t|)
(Intercept)                                           3.077 0.002238 **
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)1   3.331 0.000948 ***
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)2   0.450 0.652929
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)3   3.141 0.001810 **
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)4   1.851 0.064949 .
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)5  -0.014 0.988748
bs(inv2, knots = c(12000, 60000, 150000), degree = 3)6   3.207 0.001450 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 562100 on 393 degrees of freedom
Multiple R-squared:  0.08582,   Adjusted R-squared:  0.07186
F-statistic: 6.149 on 6 and 393 DF,  p-value: 3.54e-06
```

For å visualisere forskjellen lager vi et datasett med predikerte verdier for begge modellene:

```r
inv2.grid <- data.frame(inv2 = seq(min(Clothing$inv2), max(Clothing$inv2)))

pred1 <- predict(model1, newdata = inv2.grid, se = TRUE, interval = "prediction")$fit %>%
  bind_cols(inv2.grid)

pred2 <- predict(model2, newdata = inv2.grid, se = TRUE, interval = "prediction")$fit %>%
  bind_cols(inv2.grid)

ggplot(Clothing, aes(inv2, tsales)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, col = "gray") +
  geom_line(data = pred1, aes(y = fit), linewidth = 1, col = "blue") +
  geom_line(data = pred2, aes(y = fit), linewidth = 1, col = "red")
```



Den blå linjen er den stykkevis lineære spline og den røde er den kubiske. Begge tilpasser dataene bedre enn den rette linjen.

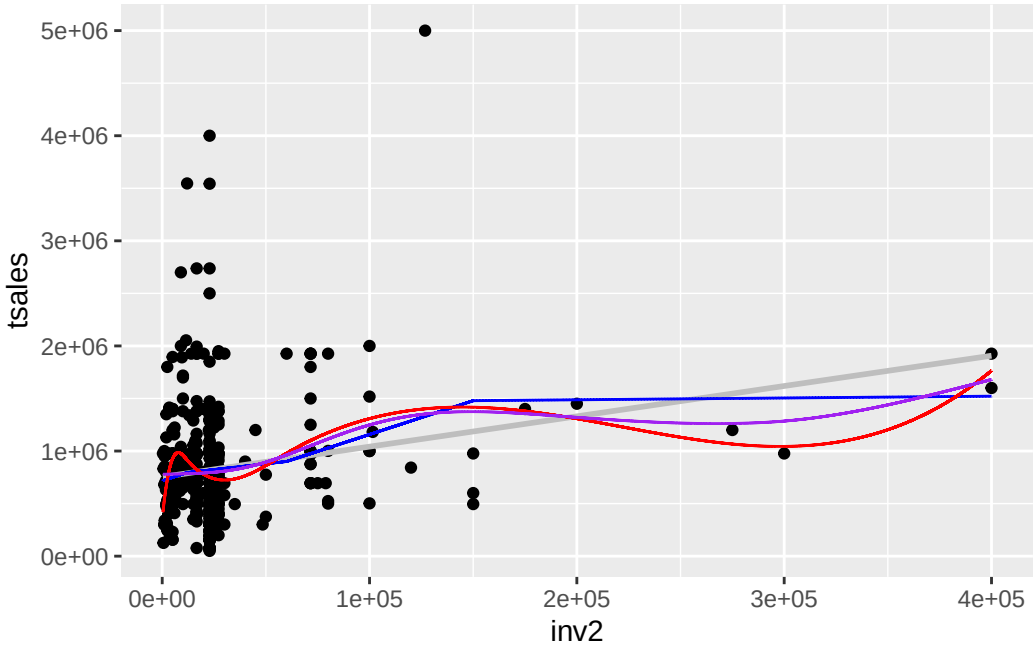
2.2.3 GAM med smoothing spline

En enda enklere tilnærming er å bruke `gam()` fra `mgcv`-pakken med `s()` som automatisk velger en god smoothing:

```r
model3 <- gam(tsales ~ s(inv2), data = Clothing)
summary(model3)
```

```
Family: gaussian
Link function: identity

Formula:
tsales ~ s(inv2)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   833584      28433   29.32   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
          edf Ref.df     F  p-value
s(inv2) 3.903  4.748 4.844 0.000409 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.0501   Deviance explained = 5.94%
GCV = 3.2739e+11  Scale est. = 3.2338e+11  n = 400
```

```r
pred3 <- inv2.grid %>%
  mutate(fit = predict(model3, newdata = inv2.grid))

ggplot(Clothing, aes(inv2, tsales)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, col = "gray") +
  geom_line(data = pred1, aes(y = fit), col = "blue") +
  geom_line(data = pred2, aes(y = fit), col = "red") +
  geom_line(data = pred3, aes(y = fit), col = "purple")
```



Den lilla linjen er GAM-modellen. Fordelen med GAM er at du ikke trenger å velge knuter selv – funksjonen finner en god tilpassning automatisk.

2.2.4 GAM med kommunedata

Generaliserte additive modeller er i utgangspunktet som lineær regresjon, men inkluderer en eller flere *smoothing*-parametre. Disse smoothing-parametrene "lærer" funksjonsformen fra dataene i stedet for å anta en lineær sammenheng. Funksjonen `gam()` fra pakken `mgcv` brukes for dette, og `s()` rundt en variabel angir at den skal ha en fleksibel form.

Her er et eksempel der vi bruker GAM med noen utvalgte variable fra kommunedataene:

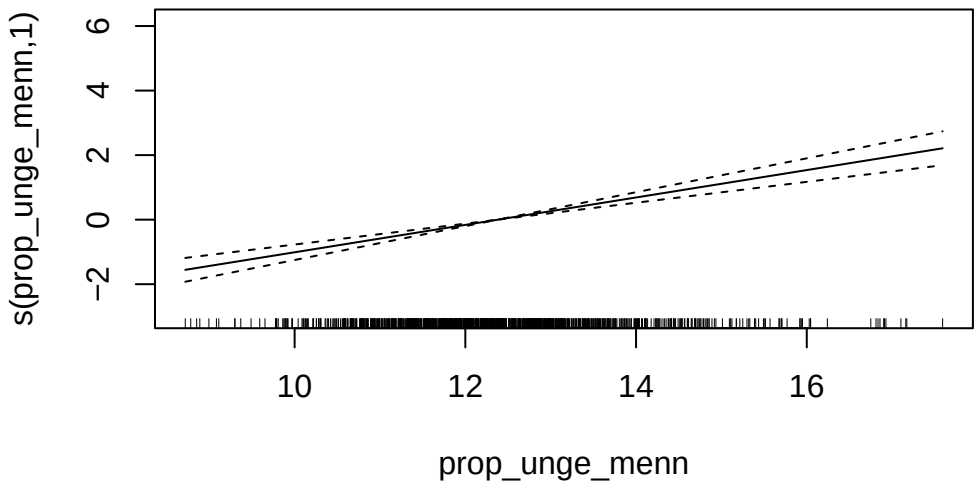
```
gam_mod <- gam(voldskriminalitet ~ s(prop_unge_menn) + s(inntekt_totalt_median) + s(shj_unge
               data = kommune_train)
summary(gam_mod)
```

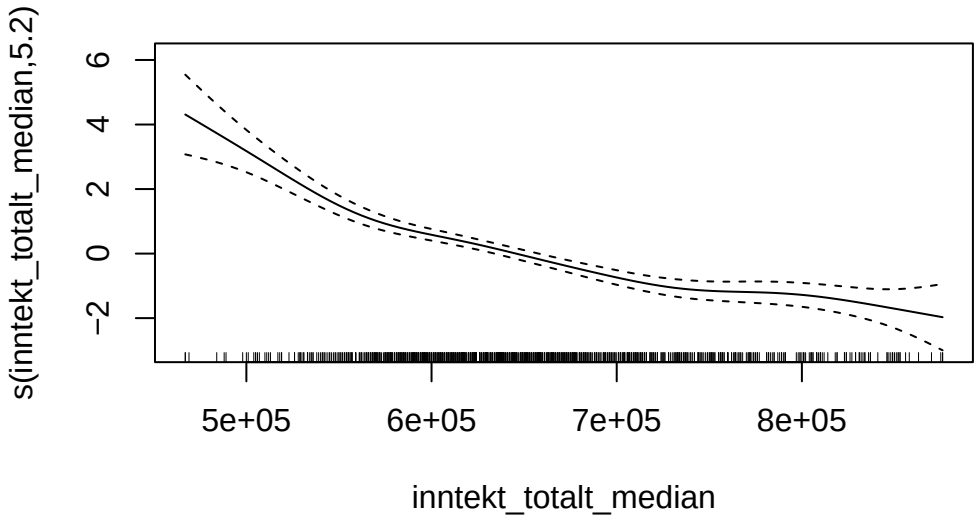
```
Family: gaussian
Link function: identity

Formula:
voldskriminalitet ~ s(prop_unge_menn) + s(inntekt_totalt_median) +
```

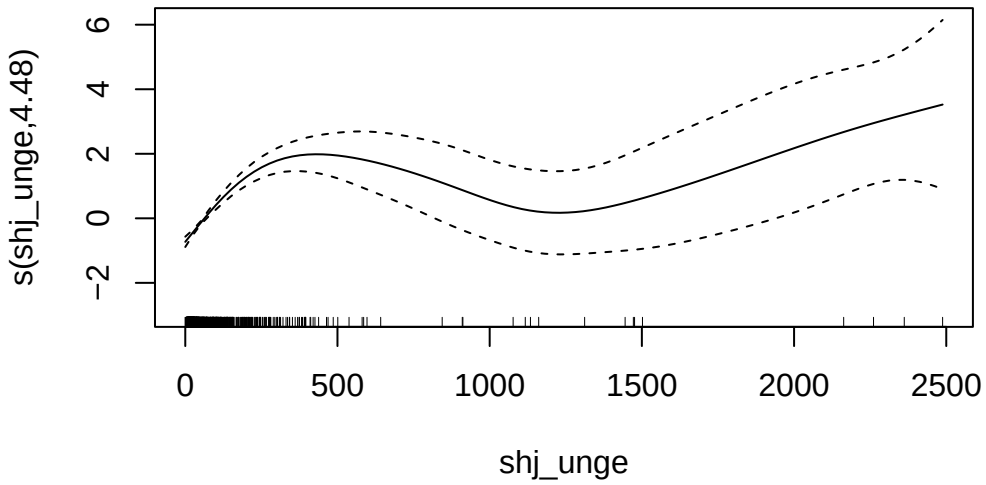
```
    s(shj_unge)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.62673    0.05937   94.77   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                           edf Ref.df     F p-value
s(prop_unge_menn)        1.000  1.000 68.21  <2e-16 ***
s(inntekt_totalt_median) 5.197  6.339 37.48  <2e-16 ***
s(shj_unge)              4.478  5.462 22.78  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.296   Deviance explained = 30.3%
GCV = 3.8136  Scale est. = 3.772     n = 1070
```

Vi kan plotte de estimerte smoothing-funksjonene for å se hvordan sammenhengen ser ut:

```
plot(gam_mod)
```







Merk at `s()` tilpasser en fleksibel kurve til dataene. Fordelen med GAM er at vi ikke trenger å anta en lineær sammenheng. Ulempen er at det er vanskeligere å tolke koeffisientene direkte, men for *prediksjon* er dette ofte ikke et problem.

2.2.5 Oppsummerende kommentar

Det gjelder generelt at mer kompliserte regresjonsmodeller vil gi tilsynelatende bedre tilpassning til data, enten man måler med r^2 eller RMSE. Man kan riktignok bruke mål på tilpassning som justerer for antall parametre etc (f.eks. justert r^2, AIC eller BIC), som kanskje kan bedre dette noe, men det vil ofte lede til overfitting likevel.

Det er derfor veldig viktig å teste modellen mot *nye* data – i praksis testing-dataene man lagde til å begynne med. Splines og GAM gir mer fleksible modeller enn vanlig lineær regresjon, men de er langt fra så risikable som f.eks. 3-veis interaksjoner. Man kan godt bruke interaksjonsledd, polynomer, splines og andre verktøy. Men med måte.

3 Logistisk regresjon

I dette kapittelt skal vi bruke følgende pakker:

```
library(tidyverse)   # datahåndtering, grafikk og glimpse()
library(rsample)     # for å dele data i training og testing
library(pROC)        # Beregne ROC-curve
library(caret)       # Funksjonen confusionMatrix()
```

3.1 Empirisk eksempel: Kategorisk utfall

Som eksempel bruker vi et datasettet Attrition. Dette er et datasett over arbeidstakere i en bedrift der utfallsvariabelen er om arbeidstakeren slutter i jobben eller ikke.

For arbeidsgivere kan det være kostbart med endringer i staben. Arbeidstakere som slutter tar med seg erfaring og kompetanse, og nye arbeidstakere må læres opp. Arbeidsgiver bør derfor generelt legge til rette for at arbeidstakere ønsker å bli værende, men det kan også være aktuelt med mer målrettede tiltak. Når en arbeidstaker har fått et nytt jobbtilbud kan det være for sent. Hvis man derimot kan komme i forkjøpet kan man kanskje gjøre noe *før* vedkommende går til det skrittet å søke ny jobb. Hvis man kunne predikere hvem som kommer til å slutte kunne man altså gjort tiltak i forkant.[1]

Først leser vi inn datasettet. Deretter kan vi se på innholdet med `glimpse()`:

```
attrition <- readRDS("data/attrition.rds")
glimpse(attrition)
```

```
Rows: 1,470
Columns: 32
$ Age                      <int> 41, 49, 37, 33, 27, 32, 59, 30, 38, 36, 35, 2~
$ Attrition                <fct> Yes, No, Yes, No, No, No, No, No, No, No, No,~
```

[1] Her kunne man jo også tenke seg at den gode lederen har en dialog med de ansatte og fanger opp deres frustrasjoner og behov slik at maskinell prediksjon ikke trengs. Det er jo også en form for prediksjon med kvalitative data! Så her er vi i en setting der dette ikke fungerer eller det er så store forhold at en kvalitativ tilnærming ikke er praktisk mulig eller noe sånt.

```
$ BusinessTravel           <fct> Travel_Rarely, Travel_Frequently, Travel_Rare~
$ DailyRate                <int> 1102, 279, 1373, 1392, 591, 1005, 1324, 1358,~
$ Department               <fct> Sales, Research & Development, Research & Dev~
$ DistanceFromHome         <int> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, 15, 26, ~
$ Education                <int> 2, 1, 2, 4, 1, 2, 3, 1, 3, 3, 3, 2, 1, 2, 3, ~
$ EducationField           <fct> Life Sciences, Life Sciences, Other, Life Sci~
$ EmployeeNumber           <int> 1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15, 16,~
$ EnvironmentSatisfaction  <int> 2, 3, 4, 4, 1, 4, 3, 4, 4, 3, 1, 4, 1, 2, 3, ~
$ Gender                   <fct> Female, Male, Male, Female, Male, Male, Femal~
$ HourlyRate               <int> 94, 61, 92, 56, 40, 79, 81, 67, 44, 94, 84, 4~
$ JobInvolvement           <int> 3, 2, 2, 3, 3, 3, 4, 3, 2, 3, 4, 2, 3, 3, 2, ~
$ JobLevel                 <int> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1, 1, 1, ~
$ JobRole                  <fct> Sales Executive, Research Scientist, Laborato~
$ JobSatisfaction          <int> 4, 2, 3, 3, 2, 4, 1, 3, 3, 3, 2, 3, 3, 4, 3, ~
$ MaritalStatus            <fct> Single, Married, Single, Married, Married, Si~
$ MonthlyIncome            <int> 5993, 5130, 2090, 2909, 3468, 3068, 2670, 269~
$ MonthlyRate              <int> 19479, 24907, 2396, 23159, 16632, 11864, 9964~
$ NumCompaniesWorked       <int> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1, 0, 5, ~
$ OverTime                 <fct> Yes, No, Yes, Yes, No, No, Yes, No, No, No, N~
$ PercentSalaryHike        <int> 11, 23, 15, 11, 12, 13, 20, 22, 21, 13, 13, 1~
$ PerformanceRating        <int> 3, 4, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 3, ~
$ RelationshipSatisfaction <int> 1, 4, 2, 3, 4, 3, 1, 2, 2, 2, 3, 4, 4, 3, 2, ~
$ StockOptionLevel         <int> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1, 1, 0, ~
$ TotalWorkingYears        <int> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, 10, 5, 3~
$ TrainingTimesLastYear    <int> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1, 2, 4, ~
$ WorkLifeBalance          <int> 1, 3, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 2, 3, 3, ~
$ YearsAtCompany           <int> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, 5, 2, 4,~
$ YearsInCurrentRole       <int> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2, 2, 2, ~
$ YearsSinceLastPromotion  <int> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4, 1, 0, ~
$ YearsWithCurrManager     <int> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3, 2, 3, ~
```

Merk at det er en variabel vi helt sikkert ikke trenger, så vi sletter denne like gjerne med en gang: *EmployeeNumber* er et løpenummer for person. Siden det er et 1:1 forhold mellom dette og utfallsvariabelen, så bør den tas ut. Vi lager også en numerisk versjon av utfallsvariabelen som vi trenger for plotting.

```
attrition <- attrition %>%
  select(- EmployeeNumber) %>%
  mutate(Attrition.num = as.numeric(Attrition == "Yes"))
glimpse(attrition)
```

```
Rows: 1,470
```

```
Columns: 32
$ Age                       <int> 41, 49, 37, 33, 27, 32, 59, 30, 38, 36, 35, 2~
$ Attrition                 <fct> Yes, No, Yes, No, No, No, No, No, No, No, No,~
$ BusinessTravel            <fct> Travel_Rarely, Travel_Frequently, Travel_Rare~
$ DailyRate                 <int> 1102, 279, 1373, 1392, 591, 1005, 1324, 1358,~
$ Department                <fct> Sales, Research & Development, Research & Dev~
$ DistanceFromHome          <int> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, 15, 26, ~
$ Education                 <int> 2, 1, 2, 4, 1, 2, 3, 1, 3, 3, 3, 2, 1, 2, 3, ~
$ EducationField            <fct> Life Sciences, Life Sciences, Other, Life Sci~
$ EnvironmentSatisfaction   <int> 2, 3, 4, 4, 1, 4, 3, 4, 4, 3, 1, 4, 1, 2, 3, ~
$ Gender                    <fct> Female, Male, Male, Female, Male, Male, Femal~
$ HourlyRate                <int> 94, 61, 92, 56, 40, 79, 81, 67, 44, 94, 84, 4~
$ JobInvolvement            <int> 3, 2, 2, 3, 3, 3, 4, 3, 2, 3, 4, 2, 3, 3, 2, ~
$ JobLevel                  <int> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1, 1, 1, ~
$ JobRole                   <fct> Sales Executive, Research Scientist, Laborato~
$ JobSatisfaction           <int> 4, 2, 3, 3, 2, 4, 1, 3, 3, 3, 2, 3, 3, 4, 3, ~
$ MaritalStatus             <fct> Single, Married, Single, Married, Married, Si~
$ MonthlyIncome             <int> 5993, 5130, 2090, 2909, 3468, 3068, 2670, 269~
$ MonthlyRate               <int> 19479, 24907, 2396, 23159, 16632, 11864, 9964~
$ NumCompaniesWorked        <int> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1, 0, 5, ~
$ OverTime                  <fct> Yes, No, Yes, Yes, No, No, Yes, No, No, No, N~
$ PercentSalaryHike         <int> 11, 23, 15, 11, 12, 13, 20, 22, 21, 13, 13, 1~
$ PerformanceRating         <int> 3, 4, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 3, ~
$ RelationshipSatisfaction <int> 1, 4, 2, 3, 4, 3, 1, 2, 2, 2, 3, 4, 4, 3, 2, ~
$ StockOptionLevel          <int> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1, 1, 0, ~
$ TotalWorkingYears         <int> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, 10, 5, 3~
$ TrainingTimesLastYear     <int> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1, 2, 4, ~
$ WorkLifeBalance           <int> 1, 3, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 2, 3, 3, ~
$ YearsAtCompany            <int> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, 5, 2, 4,~
$ YearsInCurrentRole        <int> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2, 2, 2, ~
$ YearsSinceLastPromotion   <int> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4, 1, 0, ~
$ YearsWithCurrManager      <int> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3, 2, 3, ~
$ Attrition.num             <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, ~
```

Del datasettet i to deler. Vi trekker tilfeldig 70% og legger dette i datasettet training. Resten legges i testing.

```
set.seed(426)
attrition_split <- initial_split(attrition)
training <- training(attrition_split)
testing  <- testing(attrition_split)
```

Sjekk at antallet i hvert datasett summeres til totalen

```
nrow(attrition)
```

```
[1] 1470
```

```
nrow(training)
```

```
[1] 1102
```

```
nrow(testing)
```

```
[1] 368
```

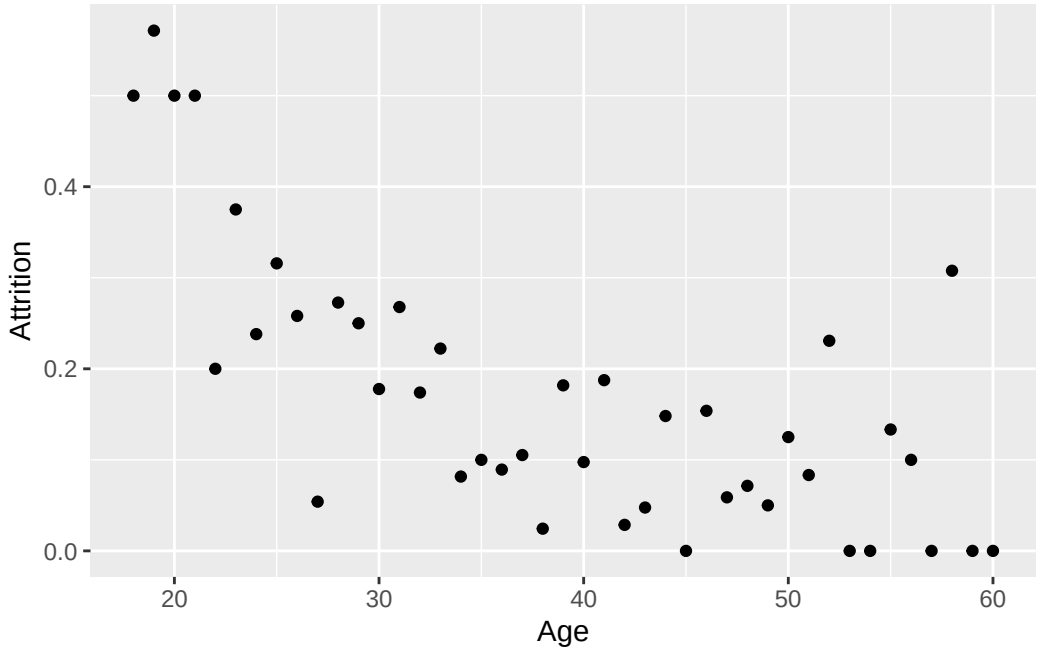
Andelen som slutter kan vi få med `mean()`:

```
mean(training$Attrition.num)
```

```
[1] 0.1533575
```

Vi kan vise hvordan det å slutte i jobben varierer med f.eks. alder ved å beregne andel per verdi av alder.

```
training_p <- training %>%
  group_by(Age) %>%
  summarise(Attrition = mean(Attrition.num))
ggplot(training_p, aes(x=Age, y=Attrition))+
  geom_point()
```



3.2 Estimere en sannsynlighet

Når utfallsvariabelen er binær (to verdier) kan man likevel bruke lineær regresjon. Det kalles da gjerne en lineær sannsynlighetsmodell.

```
lm_est <- lm(Attrition.num ~ Age , data = training)
summary(lm_est)
```

```
Call:
lm(formula = Attrition.num ~ Age, data = training)

Residuals:
     Min       1Q   Median       3Q      Max
-0.28444 -0.18737 -0.14576 -0.06256  0.99292

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.409254   0.044335   9.231  < 2e-16 ***
Age         -0.006934   0.001166  -5.948 3.65e-09 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.355 on 1100 degrees of freedom
Multiple R-squared:  0.03116,   Adjusted R-squared:  0.03027
F-statistic: 35.37 on 1 and 1100 DF,  p-value: 3.652e-09
```

Den viktigste ulempen med lineære sannsynlighetsmodeller er at modellen da kan predikere sannsynligheter lavere enn 0 og høyere enn 1. Når man er mest interessert i β er det ikke sikkert det er så nøye. Men når vi er interessert i $\hat{y}$ kan det derimot være viktig.

3.3 Logistisk regresjon i R

Logistisk regresjon har det til felles med lineær regresjon at utfallet er en lineær spesifikasjon.

$$log(\frac{\pi}{1-\pi}) = \alpha + \beta X$$

Venstresiden av ligningen kalles en *logit*, der π er en sannsynlighet. Uttrykket $\frac{\pi}{(1-\pi)}$ er en *odds*, som er et forholdstall mellom sannsynligheten for at utfallet skjer mot sannsynligheten for det motsatte. Tolkningen av β er da en endring av *odds* på logaritisk skala. Hvis man eksponensierer β er den da tolkbar som en *oddsrate*.

Man kan regne om til sannsynligheter som er vesentlig enklere å forstå. Ligningen kan skrives om slik:

$$\hat{\pi} = \frac{e^{\alpha+\beta X}}{1+e^{\alpha+\beta X}}$$

En relativt enkel omregning av regresjonsresultatet gir altså en *sannynlighet*. Denne sannsynligheten kan vi da bruke til *klassifikasjon* hvis det er formålet med analysen. Hvis utfallsvariabelen har to kategorier, så er en nærliggende mulighet å klassifisere til den gruppen hver person mest sannsynlig tilhører. Altså: de som har $\hat{\pi} = P(y = 1) > 0.5$ tilhører den ene gruppen og resten i den andre gruppen.

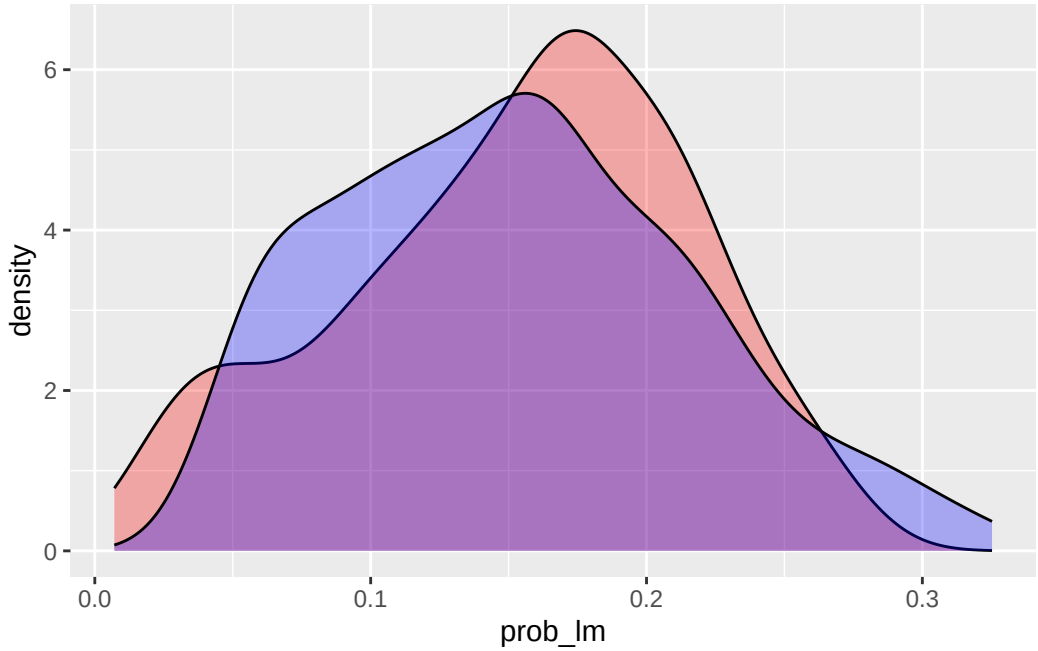
Her er et eksempel på hvordan estimere enkel logistisk regresjon i R:

```r
est_logit <- glm(Attrition ~ Age, data = training, family = "binomial")
summary(est_logit)
```

```
Call:
glm(formula = Attrition ~ Age, family = "binomial", data = training)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.41516    0.36402   1.140    0.254
Age         -0.06026    0.01050  -5.741 9.43e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 944.39  on 1101  degrees of freedom
Residual deviance: 907.48  on 1100  degrees of freedom
AIC: 911.48

Number of Fisher Scoring iterations: 5
```

Vi kan sammenligne lineær og logistisk modell for å se forskjellen i predikert sannsynlighet. Den røde kurven er den lineære modellen og den blå er den logistiske:

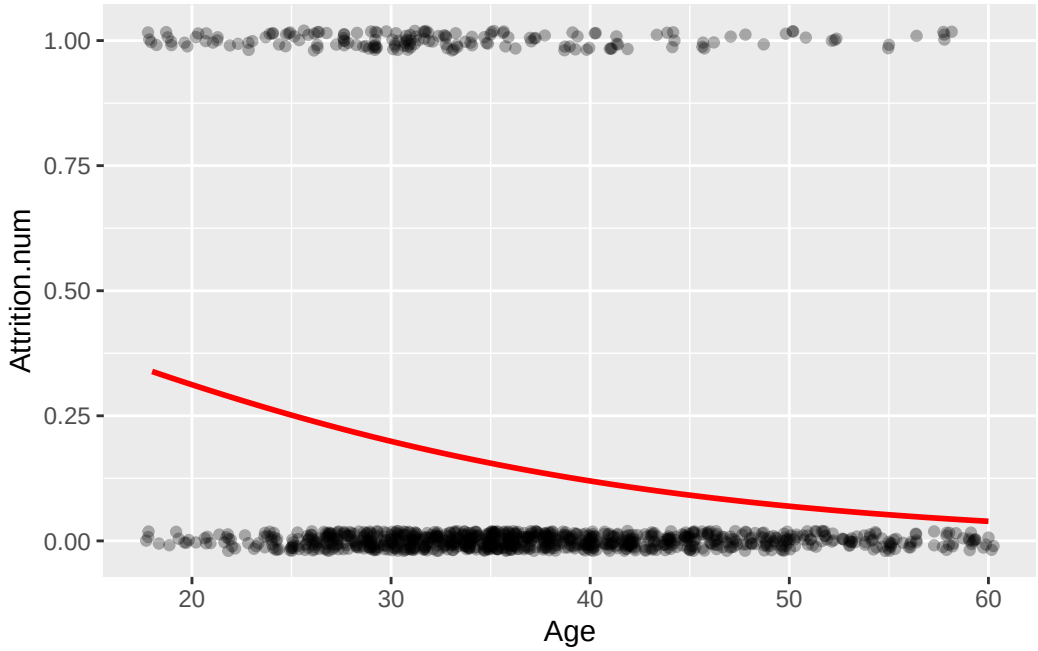
```
testing <- testing %>%
  mutate(prob_lm = predict(lm_est, newdata = testing, type = "response"),
         prob_glm = predict(est_logit, newdata = testing, type = "response"))

ggplot(testing, aes(x = prob_lm)) +
  geom_density(fill = "red", alpha = .3) +
  geom_density(aes(x = prob_glm), fill = "blue", alpha = .3)
```



Forskjellen er som regel ikke så stor i midten av fordelingen, men den lineære modellen kan gi verdier utenfor 0-1-intervallet. For klassifikasjon spiller det sjelden noen stor rolle, men for å være korrekt bruker vi logistisk regresjon når utfallet er binært.

Hvordan plotte slike data? Bruk `geom_jitter` eller `geom_point`.

```
ggplot(training, aes(x=Age, y=Attrition.num))+
  geom_jitter(height = .02, alpha=.3)+
  stat_smooth(method="glm", method.args=list(family="binomial"), se=FALSE, col="red")
```



3.4 Prediksjon

Vi kan predikere med bruk av `predict()` som tidligere. Nå er det viktig å presisere `type = "response"` for å få sannsynligheter i stedet for log-odds.

```
attrition_pred <- training %>%
  mutate(prob = predict(est_logit, type = "response"))
```

3.4.1 ROC og AUC

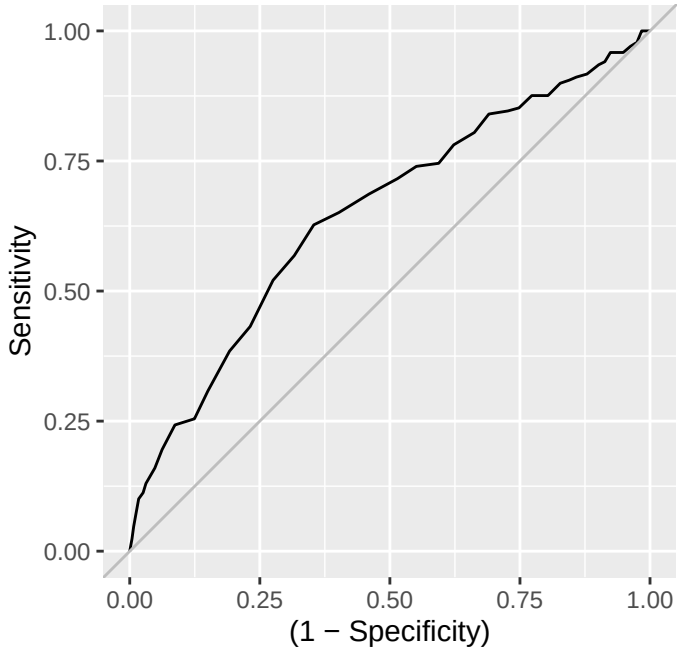
For å vurdere hvor god prediksjonen er *uavhengig* av valg av cut-off bruker vi ROC-kurven (Receiver Operating Characteristic). ROC-kurven viser sammenhengen mellom sensitivity (andelen sanne positive) og 1-specificity (andelen falske positive) for alle mulige verdier av cut-off. En modell som predikerer perfekt vil gi en kurve som går rett opp og deretter bort til høyre. En modell som gjetter tilfeldig vil ligge langs den diagonale grå linjen.

```
ROC <- roc( attrition_pred$Attrition, attrition_pred$prob )
```

```
Setting levels: control = No, case = Yes
```

```
Setting direction: controls < cases
```

```
df <- data.frame(Sensitivity = ROC$sensitivities,
                 Specificity = ROC$specificities)

ggplot(df, aes(y = Sensitivity, x= (1-Specificity))) +
  geom_line() +
  geom_abline(intercept = 0, slope = 1, col = "gray")+
  coord_equal()
```



Arealet under ROC-kurven (AUC) gir et enkelt tall som oppsummerer modellens prediksjonsevne. AUC varierer mellom 0.5 (tilfeldig gjetting) og 1.0 (perfekt prediksjon). For den enkle modellen er AUC 0.65.

3.5 Multippel logistisk regresjon

Vi kan estimere en multippel regresjon på tilsvarende måte som for lineær regresjon ved å legge til flere variable eller angi å bruke samtlige variable i datasettet med `Attrition ~ .`.

```
est_multlogit <- glm(Attrition ~ ., data = training[, !names(training) %in% "Attrition.num"]
summary(est_multlogit)
```

```
Call:
glm(formula = Attrition ~ ., family = "binomial", data = training[,
    !names(training) %in% "Attrition.num"])

Coefficients:
                                 Estimate Std. Error z value Pr(>|z|)
(Intercept)                   -1.035e+01  4.542e+02  -0.023 0.981824
Age                           -3.565e-02  1.628e-02  -2.190 0.028495 *
BusinessTravelTravel_Frequently  1.628e+00  4.675e-01   3.482 0.000498 ***
BusinessTravelTravel_Rarely   8.988e-01  4.267e-01   2.106 0.035192 *
DailyRate                     -3.117e-04  2.648e-04  -1.177 0.239034
DepartmentResearch & Development  1.203e+01  4.542e+02   0.026 0.978861
DepartmentSales                1.239e+01  4.542e+02   0.027 0.978244
DistanceFromHome               4.145e-02  1.300e-02   3.189 0.001428 **
Education                     -1.376e-02  1.039e-01  -0.132 0.894627
EducationFieldLife Sciences   -8.013e-01  1.027e+00  -0.781 0.435083
EducationFieldMarketing       -2.129e-01  1.076e+00  -0.198 0.843199
EducationFieldMedical         -8.617e-01  1.028e+00  -0.838 0.401983
EducationFieldOther           -7.977e-01  1.098e+00  -0.727 0.467345
EducationFieldTechnical Degree  1.370e-01  1.048e+00   0.131 0.896014
EnvironmentSatisfaction       -4.957e-01  9.942e-02  -4.986 6.18e-07 ***
GenderMale                     2.156e-01  2.149e-01   1.003 0.315665
HourlyRate                    -3.532e-03  5.385e-03  -0.656 0.511937
JobInvolvement                -5.422e-01  1.452e-01  -3.734 0.000189 ***
JobLevel                       1.864e-02  3.817e-01   0.049 0.961047
JobRoleHuman Resources         1.308e+01  4.542e+02   0.029 0.977028
JobRoleLaboratory Technician   1.531e+00  5.543e-01   2.762 0.005746 **
JobRoleManager                -1.451e+00  1.269e+00  -1.144 0.252636
JobRoleManufacturing Director -3.131e-01  6.086e-01  -0.514 0.606923
JobRoleResearch Director      -2.440e+00  1.244e+00  -1.961 0.049843 *
JobRoleResearch Scientist      3.417e-01  5.694e-01   0.600 0.548406
JobRoleSales Executive         3.595e-01  1.603e+00   0.224 0.822589
JobRoleSales Representative    1.518e+00  1.650e+00   0.920 0.357555
JobSatisfaction               -3.169e-01  9.701e-02  -3.267 0.001087 **
MaritalStatusMarried           3.453e-01  3.135e-01   1.101 0.270697
MaritalStatusSingle            8.690e-01  4.056e-01   2.143 0.032140 *
MonthlyIncome                  8.613e-05  9.875e-05   0.872 0.383105
MonthlyRate                    8.922e-06  1.482e-05   0.602 0.547277
NumCompaniesWorked             1.721e-01  4.623e-02   3.722 0.000198 ***
OverTimeYes                    1.877e+00  2.296e-01   8.174 2.98e-16 ***
PercentSalaryHike             -6.102e-02  4.687e-02  -1.302 0.192953
PerformanceRating              6.920e-01  4.794e-01   1.443 0.148883
```

```
RelationshipSatisfaction         -3.738e-01  9.827e-02  -3.804 0.000142 ***
StockOptionLevel                 -3.059e-01  1.810e-01  -1.690 0.091064 .
TotalWorkingYears                -6.605e-02  3.580e-02  -1.845 0.065052 .
TrainingTimesLastYear            -2.129e-01  8.533e-02  -2.495 0.012582 *
WorkLifeBalance                  -2.150e-01  1.456e-01  -1.476 0.139869
YearsAtCompany                    6.067e-02  4.877e-02   1.244 0.213537
YearsInCurrentRole               -9.948e-02  5.569e-02  -1.786 0.074038 .
YearsSinceLastPromotion           2.052e-01  5.182e-02   3.960 7.50e-05 ***
YearsWithCurrManager             -1.866e-01  6.072e-02  -3.073 0.002122 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 944.39  on 1101  degrees of freedom
Residual deviance: 619.99  on 1057  degrees of freedom
AIC: 709.99

Number of Fisher Scoring iterations: 14
```

3.5.1 ROC og AUC

For å beregne ROC og AUC gjør vi tilsvarende som over med predict og angi type respons.

```
attrition_pred <- training %>%
  mutate(prob = predict(est_multlogit, type = "response"))
```

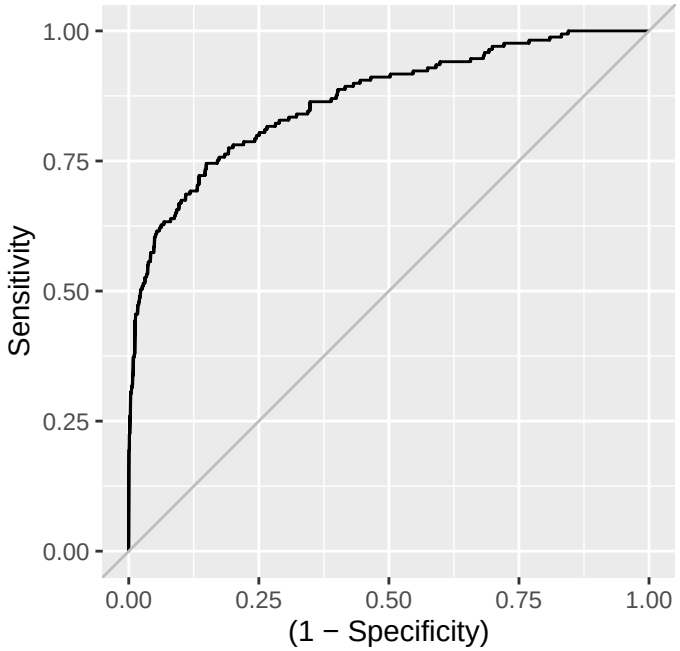
Funksjonen `roc()` gjør utregningene som trengs for ROC-kurven basert på observert utfall og predikerte sannsynligheter (Hsieh 2008).

OBS! Man må man angi data som *første* argument i funksjonen `roc()`, deretter observerte utfall og til sist predikert sannsynlighet. Rekkefølgen er viktig!

```
ROC <- roc(attrition_pred, Attrition, prob)
```

```
Setting levels: control = No, case = Yes

Setting direction: controls < cases
```

```r
df <- data.frame(Sensitivity = ROC$sensitivities,
                 Specificity = ROC$specificities)

ggplot(df, aes(y = Sensitivity, x= (1-Specificity))) +
  geom_line() +
  geom_abline(intercept = 0, slope = 1, col = "gray")+
  coord_equal()
```



Vi kan da få rapportert arealet under kurven med `auc()` slik:

```r
auc(ROC)
```

```
Area under the curve: 0.8675
```

Når arealet under kurven (AUC) er 0.867 er det vesentlig bedre prediksjon enn den enkle modellen.

3.6 Testing-data

Ovenfor er øvelsen gjort på training-data, men vi må sjekke på testing-dataene.

For å beregne ROC og AUC gjør vi tilsvarende som over med predict, men nå er det viktig å angi `newdata = ...` slik at prediksjonen gjøres på riktig datasett.

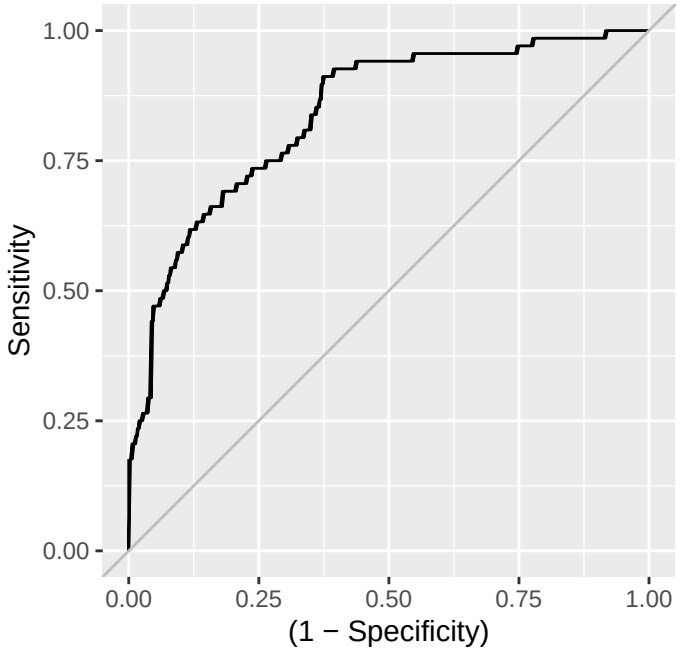
```
attrition_test <- testing %>%
  mutate(prob = predict(est_multlogit, newdata = testing, type = "response"))
```

```
ROC_test <- roc(attrition_test, Attrition, prob)
```

```
Setting levels: control = No, case = Yes
```

```
Setting direction: controls < cases
```

```
df <- data.frame(Sensitivity = ROC_test$sensitivities,
                 Specificity = ROC_test$specificities)

ggplot(df, aes(y = Sensitivity, x= (1-Specificity))) +
  geom_line() +
  geom_abline(intercept = 0, slope = 1, col = "gray")+
  coord_equal()
```



Vi kan da få rapportert arealet under kurven med `auc()` slik:

```
auc(ROC_test)
```

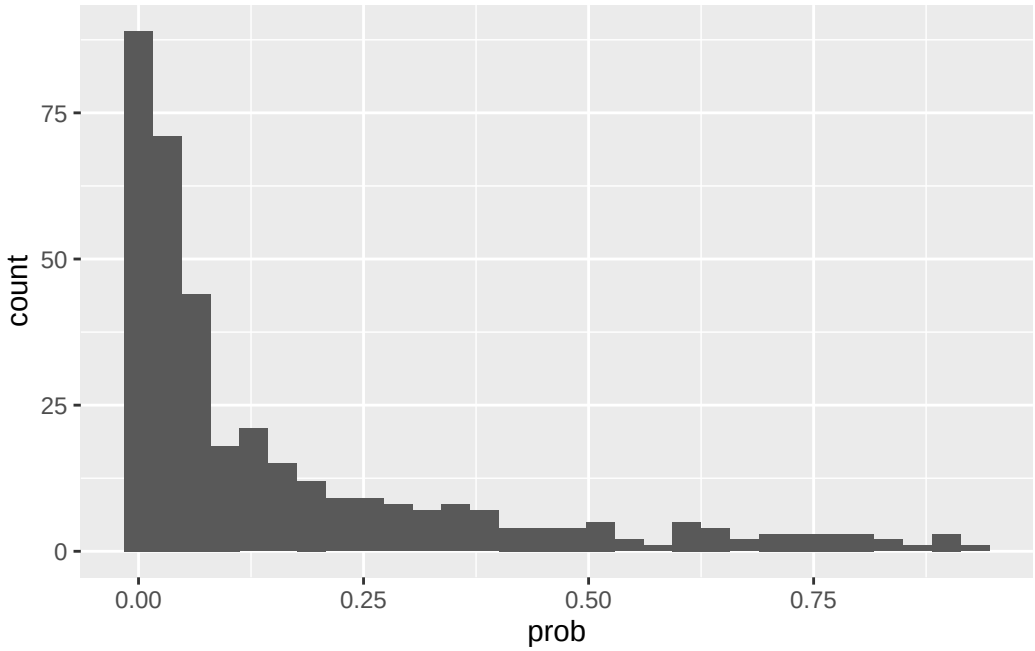
```
Area under the curve: 0.8397
```

Når arealet under kurven (AUC) er 0.84. Kanskje litt overraskende, men dette like godt som for på training dataene. AUC er altså *arealet under kurven*. Litt ulik form kan i prinsippet ha samme areal.

3.7 Klassifikasjon

Men for et handlingsvalg må vi gjøre faktisk klassifisering. Det vi har estimert så langt er bare en sannsynlighet. Selve klassifiseringen krever at man tar et aktivt valg om en cut-off for hvem man tror faktisk slutter. La oss først se på fordelingen av predikerte sannsynligheter:

```
ggplot(attrition_test, aes(x = prob)) +
  geom_histogram()
```



De aller fleste har lav predikert sannsynlighet for å slutte. Det er som forventet siden de fleste faktisk *ikke* slutter. Spørsmålet er nå: ved hvilken sannsynlighet skal vi si at noen er i faresonen?

Vi kan bestemme oss for at et rimelig cut-off er 50/50, altså med sannsynlighet 0.5. Her gjøres en klassifisering for testing-datasettet, og lager en krysstabell med den klassifiserte etter prediksjon mot observert utfall. En slik krysstabell kalles altså en *confusion matrix*.

```
attrition_test <- attrition_test %>%
  mutate(attrition_class = as.factor(ifelse(prob < .5, "No", "Yes")))

tab <- attrition_test %>%
  select(attrition_class, Attrition) %>%
  table()

tab
```

```
               Attrition
attrition_class  No Yes
            No  287  45
            Yes  13  23
```

3.7.1 Confusion matrix

Fra pakken `caret` er det en funksjon for confusion matrix som regner ut masse greier for oss. Merk at `positive = "Yes"` angir hva som er utfallsverdi av interesse.

```
confusionMatrix(reference = attrition_test$Attrition, attrition_test$attrition_class, positiv
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  287  45
       Yes  13  23

               Accuracy : 0.8424
                 95% CI : (0.8011, 0.8781)
    No Information Rate : 0.8152
    P-Value [Acc > NIR] : 0.09925

                  Kappa : 0.3605

 Mcnemar's Test P-Value : 4.691e-05
```

```
            Sensitivity : 0.33824
            Specificity : 0.95667
         Pos Pred Value : 0.63889
         Neg Pred Value : 0.86446
             Prevalence : 0.18478
         Detection Rate : 0.06250
   Detection Prevalence : 0.09783
      Balanced Accuracy : 0.64745

       'Positive' Class : Yes
```

3.8 Hvor feil kan man ta?

I klassifiseringen over er det gjort et klart valg for hvem man tror faktisk vil slutte i jobben eller ikke. Det er selvsagt slik at noen er mer sannsynlige vil slutte enn andre, men ingen har 0 sannsynlighet. Ingen har 1 heller, for den saks skyld. Det er altså usikkerhet. Men hvis vi skal gjøre et *tiltak*, så må vi ta det valget!

Hvis vi gjør klassifiseringen på 0.5 som over, så betyr jo det at vi synes begge feil er like viktige: Falske positive eller falske negative. Hvis det er et langt større problem at folk slutter enn at noen f.eks. får tilbud om goder eller ekstra oppfølging etc, så kan det hende cut-off skal settes lavere? Da får man flere sanne positive, men også flere feil. Det kan det jo være verd, men kommer jo an på hva konsekvensene. Vi kommer tilbake til dette, men test gjerne ut selv med ulik cut-off og se hvordan resultatene endrer seg.

4 Klassifikasjonstrær

I dette kapittelt skal vi bruke følgende pakker:

```
library(tidyverse)   # datahåndtering, grafikk og glimpse()
library(rsample)     # for å dele data i training og testing
library(rpart)       # funksjoner for CART
library(rpart.plot)  # funksjon for å plotte CART
library(caret)       # inneholder funksjon for confusion matrix
```

Klassifikasjonstrær (CART: Classification and Regression Trees) er en helt annen tilnærming enn regresjon. I stedet for å estimere koeffisienter i en ligning, deler algoritmen dataene i stadig mindre grupper basert på verdier av prediktorene. Resultatet er et "tre" som kan leses som en serie if-else-regler. Fordelen er at trær er enkle å tolke og kan fange opp ikke-lineære sammenhenger og interaksjoner automatisk. Ulempen er at enkelttrær kan være ustabile og har en tendens til overfitting.

Vi skal her bruke Attrition-datasettet fra kapittelet om logistisk regresjon. Utfallsvariabelen er `Attrition` som angir om en arbeidstaker slutter i jobben ("Yes") eller ikke ("No"). Vi starter med å lage en logistisk regresjonsmodell som sammenligningsgrunnlag, og deretter bygger vi klassifikasjonstrær.

4.1 Lese inn data

```
attrition <- readRDS("data/attrition.rds") %>%
  select(-EmployeeNumber)
glimpse(attrition)
```

```
Rows: 1,470
Columns: 31
$ Age                      <int> 41, 49, 37, 33, 27, 32, 59, 30, 38, 36, 35, 2~
$ Attrition                <fct> Yes, No, Yes, No, No, No, No, No, No, No, No,~
$ BusinessTravel           <fct> Travel_Rarely, Travel_Frequently, Travel_Rare~
$ DailyRate                <int> 1102, 279, 1373, 1392, 591, 1005, 1324, 1358,~
```

```
$ Department                <fct> Sales, Research & Development, Research & Dev~
$ DistanceFromHome          <int> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, 15, 26, ~
$ Education                 <int> 2, 1, 2, 4, 1, 2, 3, 1, 3, 3, 3, 2, 1, 2, 3, ~
$ EducationField            <fct> Life Sciences, Life Sciences, Other, Life Sci~
$ EnvironmentSatisfaction   <int> 2, 3, 4, 4, 1, 4, 3, 4, 4, 3, 1, 4, 1, 2, 3, ~
$ Gender                    <fct> Female, Male, Male, Female, Male, Male, Femal~
$ HourlyRate                <int> 94, 61, 92, 56, 40, 79, 81, 67, 44, 94, 84, 4~
$ JobInvolvement            <int> 3, 2, 2, 3, 3, 3, 4, 3, 2, 3, 4, 2, 3, 3, 2, ~
$ JobLevel                  <int> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1, 1, 1, ~
$ JobRole                   <fct> Sales Executive, Research Scientist, Laborato~
$ JobSatisfaction           <int> 4, 2, 3, 3, 2, 4, 1, 3, 3, 3, 2, 3, 3, 4, 3, ~
$ MaritalStatus             <fct> Single, Married, Single, Married, Married, Si~
$ MonthlyIncome             <int> 5993, 5130, 2090, 2909, 3468, 3068, 2670, 269~
$ MonthlyRate               <int> 19479, 24907, 2396, 23159, 16632, 11864, 9964~
$ NumCompaniesWorked        <int> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1, 0, 5, ~
$ OverTime                  <fct> Yes, No, Yes, Yes, No, No, Yes, No, No, No, N~
$ PercentSalaryHike         <int> 11, 23, 15, 11, 12, 13, 20, 22, 21, 13, 13, 1~
$ PerformanceRating         <int> 3, 4, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 3, ~
$ RelationshipSatisfaction <int> 1, 4, 2, 3, 4, 3, 1, 2, 2, 2, 3, 4, 4, 3, 2, ~
$ StockOptionLevel          <int> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1, 1, 0, ~
$ TotalWorkingYears         <int> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, 10, 5, 3~
$ TrainingTimesLastYear     <int> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1, 2, 4, ~
$ WorkLifeBalance           <int> 1, 3, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 2, 3, 3, ~
$ YearsAtCompany            <int> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, 5, 2, 4,~
$ YearsInCurrentRole        <int> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2, 2, 2, ~
$ YearsSinceLastPromotion   <int> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4, 1, 0, ~
$ YearsWithCurrManager      <int> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3, 2, 3, ~
```

Vi splitter datasettet i training og testing:

```
set.seed(426)
attrition_split <- initial_split(attrition, prop = .7)
training <- training(attrition_split)
testing  <- testing(attrition_split)
```

4.2 Logistisk regresjon som baseline

Før vi lager klassifikasjonstrær er det nyttig å ha et sammenligningsgrunnlag. Vi bruker logistisk regresjon med alle variable:

```r
est.glm <- glm(Attrition ~ ., data = training, family = "binomial")
summary(est.glm)
```

```
Call:
glm(formula = Attrition ~ ., family = "binomial", data = training)

Coefficients:
                                 Estimate Std. Error z value Pr(>|z|)
(Intercept)                   -1.171e+01  7.504e+02  -0.016 0.987554
Age                           -3.647e-02  1.688e-02  -2.161 0.030667 *
BusinessTravelTravel_Frequently  1.648e+00  4.983e-01   3.308 0.000940 ***
BusinessTravelTravel_Rarely    9.891e-01  4.541e-01   2.178 0.029387 *
DailyRate                     -3.576e-04  2.753e-04  -1.299 0.193941
DepartmentResearch & Development  1.298e+01  7.504e+02   0.017 0.986196
DepartmentSales                1.357e+01  7.504e+02   0.018 0.985571
DistanceFromHome               4.746e-02  1.361e-02   3.488 0.000486 ***
Education                     -1.807e-02  1.084e-01  -0.167 0.867654
EducationFieldLife Sciences   -3.399e-01  1.129e+00  -0.301 0.763441
EducationFieldMarketing        3.270e-01  1.179e+00   0.277 0.781525
EducationFieldMedical         -5.647e-01  1.133e+00  -0.498 0.618260
EducationFieldOther           -5.878e-01  1.207e+00  -0.487 0.626217
EducationFieldTechnical Degree  5.805e-01  1.154e+00   0.503 0.615076
EnvironmentSatisfaction       -5.414e-01  1.053e-01  -5.140 2.74e-07 ***
GenderMale                     2.498e-01  2.252e-01   1.109 0.267218
HourlyRate                    -2.457e-03  5.598e-03  -0.439 0.660717
JobInvolvement                -5.900e-01  1.538e-01  -3.835 0.000126 ***
JobLevel                       1.022e-01  3.942e-01   0.259 0.795375
JobRoleHuman Resources         1.399e+01  7.504e+02   0.019 0.985126
JobRoleLaboratory Technician   1.562e+00  5.660e-01   2.760 0.005777 **
JobRoleManager                -1.454e+00  1.269e+00  -1.145 0.252018
JobRoleManufacturing Director -3.797e-01  6.172e-01  -0.615 0.538366
JobRoleResearch Director      -2.476e+00  1.256e+00  -1.971 0.048692 *
JobRoleResearch Scientist      2.632e-01  5.858e-01   0.449 0.653273
JobRoleSales Executive        -1.863e-01  1.594e+00  -0.117 0.906944
JobRoleSales Representative    1.378e+00  1.648e+00   0.836 0.403020
JobSatisfaction               -3.369e-01  1.022e-01  -3.298 0.000975 ***
MaritalStatusMarried           2.514e-01  3.242e-01   0.775 0.438093
MaritalStatusSingle            9.638e-01  4.225e-01   2.281 0.022524 *
MonthlyIncome                  8.287e-05  1.030e-04   0.804 0.421245
MonthlyRate                    6.922e-06  1.558e-05   0.444 0.656792
NumCompaniesWorked             1.638e-01  4.800e-02   3.413 0.000642 ***
```

```
OverTimeYes                  2.007e+00  2.437e-01   8.236  < 2e-16 ***
PercentSalaryHike           -6.251e-02  4.976e-02  -1.256 0.209020
PerformanceRating            6.650e-01  5.094e-01   1.305 0.191744
RelationshipSatisfaction    -3.787e-01  1.031e-01  -3.674 0.000239 ***
StockOptionLevel            -2.105e-01  1.875e-01  -1.123 0.261579
TotalWorkingYears           -5.930e-02  3.680e-02  -1.611 0.107130
TrainingTimesLastYear       -1.856e-01  9.002e-02  -2.061 0.039275 *
WorkLifeBalance             -1.886e-01  1.525e-01  -1.236 0.216394
YearsAtCompany               4.602e-02  5.114e-02   0.900 0.368242
YearsInCurrentRole          -1.046e-01  5.771e-02  -1.812 0.069940 .
YearsSinceLastPromotion      2.017e-01  5.441e-02   3.707 0.000210 ***
YearsWithCurrManager        -1.745e-01  6.343e-02  -2.752 0.005927 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 875.64  on 1028  degrees of freedom
Residual deviance: 566.77  on  984  degrees of freedom
AIC: 656.77

Number of Fisher Scoring iterations: 15
```

Vi predikerer på testing-data og gjør en klassifisering med cut-off på 0.5:

```
testing_glm <- testing %>%
  mutate(prob_glm = predict(est.glm, newdata = testing, type = "response")) %>%
  mutate(klassifiser = factor(ifelse(prob_glm < .5, "No", "Yes")))

confusionMatrix(reference = testing_glm$Attrition,
                testing_glm$klassifiser,
                positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  345  52
       Yes  15  29

               Accuracy : 0.8481
                 95% CI : (0.8111, 0.8803)
```

```
    No Information Rate : 0.8163
    P-Value [Acc > NIR] : 0.04596

                  Kappa : 0.3844

 Mcnemar's Test P-Value : 1.092e-05

            Sensitivity : 0.35802
            Specificity : 0.95833
         Pos Pred Value : 0.65909
         Neg Pred Value : 0.86902
             Prevalence : 0.18367
         Detection Rate : 0.06576
   Detection Prevalence : 0.09977
      Balanced Accuracy : 0.65818

       'Positive' Class : Yes
```

Denne confusion matrix gir oss et referansepunkt for å vurdere klassifikasjonstrærne.

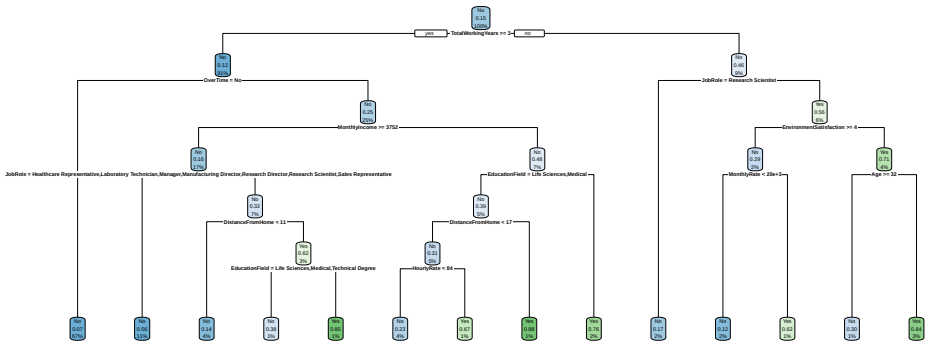
4.3 Klassifikasjonstre

Utfallsvariabel og prediktorer spesifiseres som en formel på samme måte som for regresjon. Siden vi her har en klassifikasjon må vi spesifisere `method = "class"`. Hvis ikke vil `rpart()` gjette hva slags modell (som kanskje er riktig), så du kan få andre resultater enn du forventet.

```
klass_tre <- rpart(Attrition ~ .,
                   data = training, method = "class")
```

Resultatet kan fremstilles grafisk med funksjonen `rpart.plot()` slik:

```
rpart.plot(klass_tre)
```



Treet leses fra toppen og nedover. I hver node vises den predikerte klassen, andelen som tilhører denne klassen, og andelen av totalen som havner i denne noden. Hver split angir et kriterium: observasjoner som oppfyller kriteriet går til venstre, resten til høyre. Bladnodene (nederst) gir den endelige klassifiseringen.

Vi kan også få printet ut resultatet som regler i en tabell med `rpart.rules()`:

```
rpart.rules(klass_tre, extra = 4)
```

```
 Attrition   No Yes
        No [.94 .06] when TotalWorkingYears >= 3 & OverTime is Yes & JobRole is Healthcare Re
        No [.93 .07] when TotalWorkingYears >= 3 & OverTime is  No
        No [.88 .12] when TotalWorkingYears <  3                  & JobRole is
        No [.86 .14] when TotalWorkingYears >= 3 & OverTime is Yes & JobRole is
        No [.83 .17] when TotalWorkingYears <  3                  & JobRole is
        No [.77 .23] when TotalWorkingYears >= 3 & OverTime is Yes
        No [.70 .30] when TotalWorkingYears <  3                  & JobRole is
        No [.62 .38] when TotalWorkingYears >= 3 & OverTime is Yes & JobRole is
       Yes [.38 .62] when TotalWorkingYears <  3                  & JobRole is
       Yes [.33 .67] when TotalWorkingYears >= 3 & OverTime is Yes
       Yes [.24 .76] when TotalWorkingYears >= 3 & OverTime is Yes
       Yes [.16 .84] when TotalWorkingYears <  3                  & JobRole is
       Yes [.15 .85] when TotalWorkingYears >= 3 & OverTime is Yes & JobRole is
       Yes [.12 .88] when TotalWorkingYears >= 3 & OverTime is Yes
```

Da kan vi sammenligne prediksjoner med observert utfall for testingdataene. I `predict()` må det angis `type = "class"` for å spesifisere at det skal være klassifikasjon.

```
testing_pred <- testing %>%
  mutate(Attrition_pred = predict(klass_tre, newdata = testing, type = "class"))


confusionMatrix(reference = testing_pred$Attrition, testing_pred$Attrition_pred, positive = "
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  342  67
       Yes  18  14

               Accuracy : 0.8073
                 95% CI : (0.7673, 0.843)
    No Information Rate : 0.8163
    P-Value [Acc > NIR] : 0.7131

                  Kappa : 0.1605

 Mcnemar's Test P-Value : 1.926e-07

            Sensitivity : 0.17284
            Specificity : 0.95000
         Pos Pred Value : 0.43750
         Neg Pred Value : 0.83619
             Prevalence : 0.18367
         Detection Rate : 0.03175
   Detection Prevalence : 0.07256
      Balanced Accuracy : 0.56142

       'Positive' Class : Yes
```

Sammenlign dette resultatet med logistisk regresjon-baseline ovenfor. Spesielt interessant er accuracy, sensitivity (andelen av de som faktisk slutter som vi fanger opp) og specificity (andelen av de som blir som vi riktig klassifiserer). Et enkelt tre med forvalgte parametre gjør det ofte ganske greit, men vi kan forsøke å forbedre det med tuning.

4.4 Tuning/pruning

Vi har vært inne på tidligere at vi kan styre hvordan algoritmen fungerer. Det er noen parametres om styrer prosessen, og disse kan vi justere. Her tar vi for oss de viktigste.

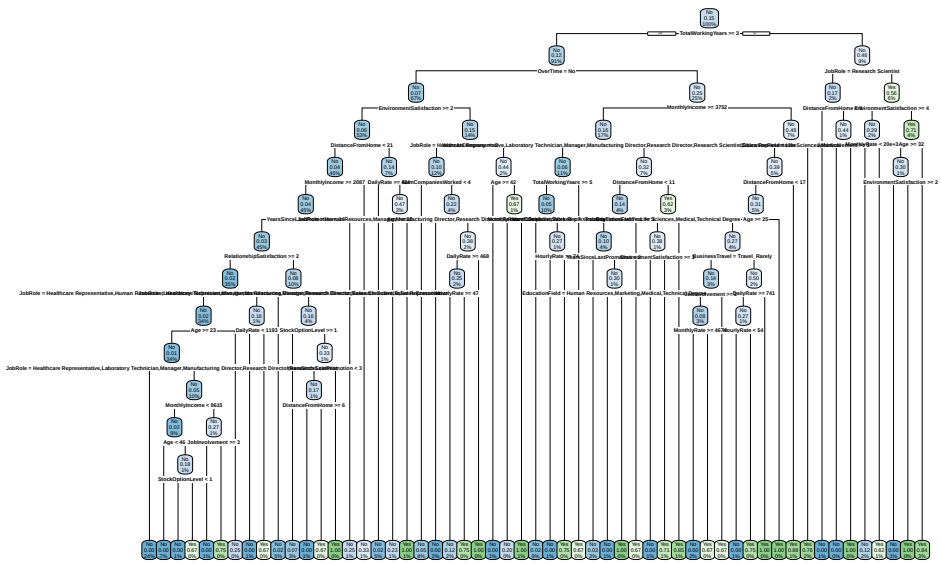
Kort fortalt styrer disse parametrene hvor komplekse trærne kan bli. Husk nå fra tidligere kapittel: mer kompleks modell gir bedre tilpassning til trainingdata - men kan gi dårligere tilpassning til testingdata. Målet er altså å finne en slags balansert kompleksitet.

Nå er arbeidsflyten slik at man skrur litt på disse parametrene, sjekker resultatet og justerer på nytt og sjekker… osv. Da er det viktig at bruker trainingdata! Ikke bruke testingdata før du er rimelig fornøyd med resultatet.

4.4.1 Bruk av `cp = ...`

`cp` er complexity parameter som setter et krav på hvor mye hver split skal bidra til modellens tilpassning til data. Forvalget er 0.01. Med en lavere verdi tillates mer komplekse trær:

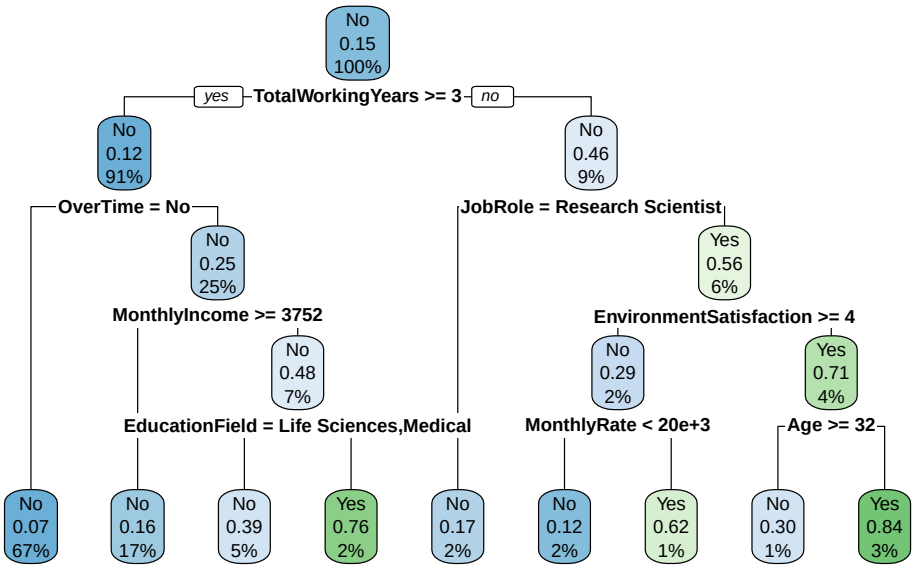
```
klass_tre2 <- rpart(Attrition ~ .,
                    data = training, method = "class",
                    cp = 0.0000001, maxdepth = 20, minbucket = 3)

rpart.plot(klass_tre2)
```



4.4.2 Bruk av `maxdepth = ...`

Parameteren `maxdepth` setter en grense for hvor mange splitter det kan gjøres i hver forgrening:

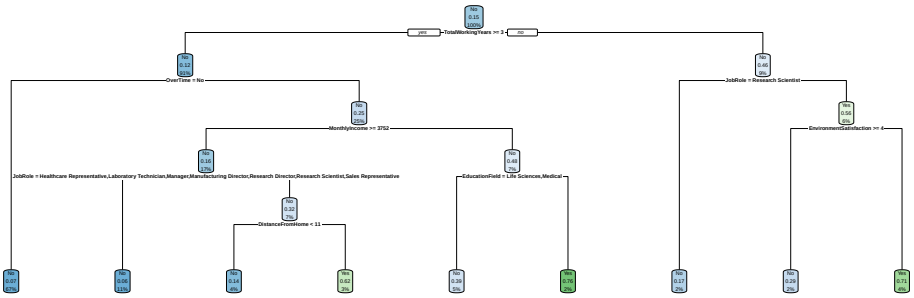
```
klass_tre3 <- rpart(Attrition ~ .,
                    data = training, method = "class",
                    maxdepth = 4)

rpart.plot(klass_tre3)
```



4.4.3 Bruk av `minbucket = ...`

Parameterne `minbucket = ...` styrer hvor mange observasjoner det minst må være i den siste noden. Forvalget er en 1/3 av `minsplit`, altså 7 hvis man ikke har endret på `minsplit`.

```
klass_tre4 <- rpart(Attrition ~ .,
                    data = training, method = "class",
                    minbucket = 15)

rpart.plot(klass_tre4)
```

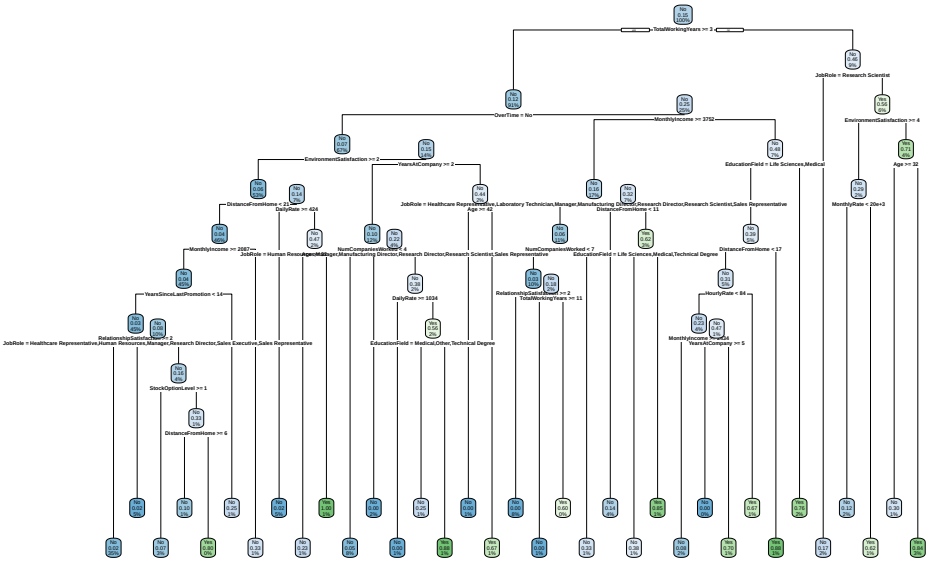


4.4.4 Sette delene sammen

Du kan kombinere disse parametrene. Alle har forvalgte verdier, så du bruker dem uansett – forskjellen er bare om du har tatt et eksplisitt valg eller overlater det hele til softwaren.

Her er et eksempel der vi bruker et ganske komplekst tre med justerte parametre:

```
klass_tre5 <- rpart(Attrition ~ .,
                    data = training, method = "class",
                    cp = 0, maxdepth = 25, minbucket = 5)

rpart.plot(klass_tre5)
```



```
rpart.rules(klass_tre5)
```

```
 Attrition
      0.00 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
      0.00 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
      0.00 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
      0.00 when TotalWorkingYears >=       3 & OverTime is Yes
      0.00 when TotalWorkingYears >=      11 & OverTime is Yes
      0.00 when TotalWorkingYears >=       3 & OverTime is Yes
      0.02 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.02 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.02 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.05 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
      0.07 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.08 when TotalWorkingYears >=       3 & OverTime is Yes
      0.10 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.12 when TotalWorkingYears <  3                         & EnvironmentSatisfaction >= 4
      0.14 when TotalWorkingYears >=       3 & OverTime is Yes
      0.17 when TotalWorkingYears <  3
      0.23 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.25 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.25 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
```

```
      0.30 when TotalWorkingYears <  3                        & EnvironmentSatisfaction <  4
      0.33 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.33 when TotalWorkingYears >=       3 & OverTime is Yes
      0.38 when TotalWorkingYears >=       3 & OverTime is Yes
      0.60 when TotalWorkingYears is 3 to 11 & OverTime is Yes
      0.62 when TotalWorkingYears <  3                        & EnvironmentSatisfaction >= 4
      0.67 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
      0.67 when TotalWorkingYears >=       3 & OverTime is Yes
      0.70 when TotalWorkingYears >=       3 & OverTime is Yes
      0.76 when TotalWorkingYears >=       3 & OverTime is Yes
      0.80 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
      0.84 when TotalWorkingYears <  3                        & EnvironmentSatisfaction <  4
      0.85 when TotalWorkingYears >=       3 & OverTime is Yes
      0.88 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction <  1
      0.88 when TotalWorkingYears >=       3 & OverTime is Yes
      1.00 when TotalWorkingYears >=       3 & OverTime is  No & EnvironmentSatisfaction >= 1
```

Vi kan sjekke resultatet med confusion matrix på testing-data:

```
testing_pred <- testing %>%
  mutate(Attrition_pred = predict(klass_tre5, newdata = testing, type = "class"))

confusionMatrix(reference = testing_pred$Attrition,
                testing_pred$Attrition_pred,
                positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  328  64
       Yes  32  17

               Accuracy : 0.7823
                 95% CI : (0.7408, 0.82)
    No Information Rate : 0.8163
    P-Value [Acc > NIR] : 0.969618

                  Kappa : 0.1429

 Mcnemar's Test P-Value : 0.001557
```

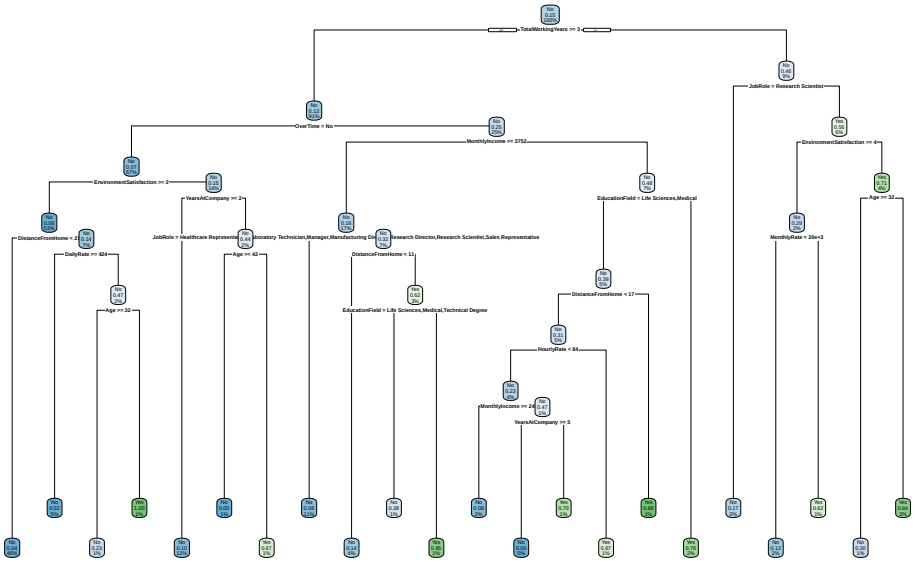
```
            Sensitivity : 0.20988
            Specificity : 0.91111
         Pos Pred Value : 0.34694
         Neg Pred Value : 0.83673
             Prevalence : 0.18367
         Detection Rate : 0.03855
   Detection Prevalence : 0.11111
      Balanced Accuracy : 0.56049

       'Positive' Class : Yes
```

Legg merke til at et veldig komplekst tre ikke nødvendigvis gir bedre resultater på testing-data. Treet kan ha blitt for tilpasset training-dataene (overfitting). En måte å motvirke dette på er å *beskjære* treet i etterkant.

4.4.5 Pruning

En relatert teknikk er å beskjære treet basert på `cp`. Altså, når du har bygget et tre som du tenker er for komplekst, så kan du beskjære grenene slik at de minst viktige grenene kuttes. Funksjonen `prune()` gjør jobben:

```
pruned_tre <- prune(klass_tre5, cp = .01)
rpart.plot(pruned_tre)
```



4.5 Asymetriske kostnader med loss matrix (Ekstramateriale)

Loss matrix gjør det mulig å vekte ulike typer feil forskjellig. Utgangspunktet er følgende matrise:

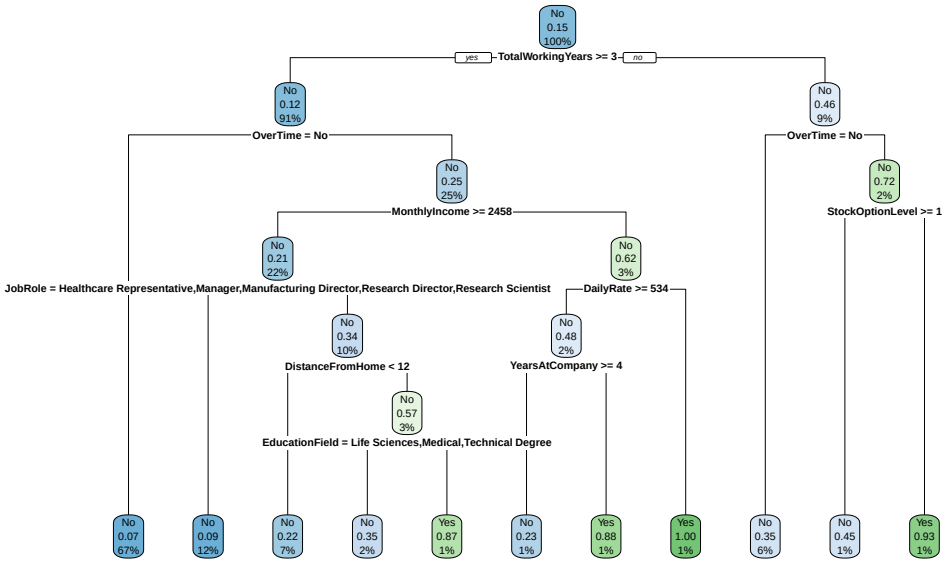
$$loss = \begin{bmatrix} TN & FN \\ FP & TP \end{bmatrix}$$

Vi setter alltid vektingen av sanne positive og sanne negative til 0. Feilene vektes. Her er et eksempel der falske negative (at vi ikke fanger opp noen som faktisk slutter) veier 4 ganger tyngre enn falske positive:

```
lossm <- matrix(c(0, 1, 4, 0), ncol=2)
lossm
```

```
     [,1] [,2]
[1,]    0    4
[2,]    1    0
```

```
rpart_loss <- rpart(Attrition ~ . ,
                    data = training,
                    parms = list(loss = lossm),
                    method = "class")

rpart.plot(rpart_loss)
```



Sjekk mot testingdata:

```
testing_pred <- testing %>%
  mutate(Attrition_pred = predict(rpart_loss, newdata = testing, type = "class"))

tab <- testing_pred %>%
  select(Attrition_pred, Attrition) %>%
  table()
confusionMatrix(tab, positive = "Yes")
```

```
Confusion Matrix and Statistics

              Attrition
Attrition_pred  No Yes
           No  356  68
```

```
         Yes   4  13

               Accuracy : 0.8367
                 95% CI : (0.7989, 0.87)
    No Information Rate : 0.8163
    P-Value [Acc > NIR] : 0.1476

                  Kappa : 0.2153

 Mcnemar's Test P-Value : 1.131e-13

            Sensitivity : 0.16049
            Specificity : 0.98889
         Pos Pred Value : 0.76471
         Neg Pred Value : 0.83962
             Prevalence : 0.18367
         Detection Rate : 0.02948
   Detection Prevalence : 0.03855
      Balanced Accuracy : 0.57469

       'Positive' Class : Yes
```

Sammenlign sensitivity og specificity med den opprinnelige modellen uten loss matrix. Ved å vekte falske negative tyngre tvinger vi modellen til å fange opp flere av de som faktisk slutter – men på bekostning av flere falske positive. Denne avveiningen mellom ulike typer feil er sentral i resten av kurset.

5 Bagging

I dette kapittelt skal vi bruke følgende pakker:

```
library(tidyverse)   # datahåndtering, grafikk og glimpse()
library(rsample)     # for å dele data i training og testing
library(rpart)       # for fitting decision trees
library(rpart.plot)  # for å plotte trær
library(ipred)       # for fitting bagged decision trees
library(caret)       # for general model fitting og confusionMatrix()
library(forcats)     # for rekoding av factor-variable
```

6 Introduksjon til bagging

Merk at (Berk 2016) gjør et poeng av at med *bagging* gjør vi et større steg vekk fra mer vanlige statistiske prosedyrer. Regresjon og klassifikasjonstrær (og det meste annet vi er mer vant med) gir *ett* sett av resultater. Med bagging får vi en hel haug av resultater. Prinsippet er rett og slett at man trekker et tilfeldig utvalg av dataene, bygger et klassifikasjonstre og gjør klassifikasjonen. Så gjentar man dette med et nytt tilfeldig utvalg fra det opprinnelige datasettet.[1] Så gjentar man dette mange ganger.[2]

Hver observasjon i det opprinnelige datasettet har da blitt klassifisert mange ganger, men ikke nødvendigvis likt i hvert tre. (Enkle klassifikasjonstrær er nemlig litt ustabile greier). Hvis man har gjort prosedyren 100 ganger, så vil hver observasjon bli klassifisert 100 ganger - en gang i hvert tre. For prediksjon på trainingdata brukes rett og slett en opptelling over alle trærne for hver observasjon. Når (Berk 2016) snakker om "votes", så er dette som menes: hvert tre har en stemme og så stemmes det over hva som blir utfallet for den enkelte.

Dette betyr at antall trær kan ha noe å si. Forvalget for funksjonen `bagging()` er 25. Det er en god start, men flere trær gir jo mer stabile resultater.

Husk at klassifikasjonstrær er litt ustabile greier, og små tilfeldige variasjoner kan påvirke en enkelt split. Ved å bruke bagging jevnes de tilfeldige feilene ut slik at man totalt sett skal få et mer stabilt resultat. Altså: at faren for overfitting reduseres. Hvor mange trær som trengs er litt verre å si noe om.

6.1 Empirisk eksempel: Syntetiske data

Vi bruker de syntetiske fødselskohort-dataene. Disse er generert for å etterligne en kohort født i 1996, der utfallsvariabelen `felonies` angir om personen har begått alvorlige lovbrudd (1) eller ikke (0). Vi fjerner variablene `drugs` og `violence` fordi disse er delkomponenter av utfallet. Vi demonstrerer også med Attrition-dataene.

[1] Dette er altså det vi ofte kaller tilfeldig utvalg *med* tilbakelegging.

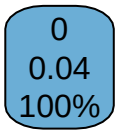
[2] Bagging ligner altså veldig på det vi i vanlig statistikk kaller for bootstrapping. Men i bootstrapping er ofte formålet å korrigere standardfeilene eller noe slikt og ikke så mye punktestimatet. Skjønt, det går an å bruke bagging på regresjonsmodeller også, og da tror jeg forskjellen mot bootstrapping er mest semantisk.

```r
load("data/synthetic_birthCohort96.RData")

synth <- synthetic_birthCohort %>%
  select(-drugs, -violence) %>%
  mutate(felonies = factor(felonies))

set.seed(426)
synth_split <- initial_split(synth)
training <- training(synth_split)
testing  <- testing(synth_split)
```

6.1.1 Klassifikasjonstre som utgangspunkt

Først lager vi et klassifikasjonstre for sammenligning:

```r
klass_tre <- rpart(felonies ~ ., data = training, method = "class")
rpart.plot(klass_tre)
```



6.1.2 Bagging

Funksjonen `bagging()` har tilsvarende oppbygning som tidligere modeller. Her angir vi 100 trær med `nbagg = 100` og bruker OOB med `coob = TRUE`:

```
set.seed(42)
bag <- bagging(felonies ~ ., data = training,
               nbagg = 100, coob = TRUE)

bag
```

```
Bagging classification trees with 100 bootstrap replications

Call: bagging.data.frame(formula = felonies ~ ., data = training, nbagg = 100,
    coob = TRUE)

Out-of-bag estimate of misclassification error:  0.0411
```

Vi kan legge til justeringer for de enkelte trærne med `control = rpart.control()`:

```
set.seed(42)
bag2 <- bagging(felonies ~ ., data = training,
                nbagg = 100, coob = TRUE,
                control = rpart.control(cp = 0.0000001, maxdepth = 20, minbucket = 3))

bag2
```

```
Bagging classification trees with 100 bootstrap replications

Call: bagging.data.frame(formula = felonies ~ ., data = training, nbagg = 100,
    coob = TRUE, control = rpart.control(cp = 1e-07, maxdepth = 20,
        minbucket = 3))

Out-of-bag estimate of misclassification error:  0.0403
```

Sammenlign med testing-data. Her lager vi prediksjoner fra både det enkle klassifikasjonstreet og bagging-modellen, slik at vi kan sammenligne dem direkte:

```r
testing_pred <- testing %>%
  mutate(pred_tre = predict(klass_tre, newdata = testing, type = "class"),
         pred_bag = predict(bag, newdata = testing))

confusionMatrix(reference = testing_pred$felonies, testing_pred$pred_tre, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7193  307
         1    0    0

               Accuracy : 0.9591
                 95% CI : (0.9543, 0.9634)
    No Information Rate : 0.9591
    P-Value [Acc > NIR] : 0.5152

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.00000
            Specificity : 1.00000
         Pos Pred Value :     NaN
         Neg Pred Value : 0.95907
             Prevalence : 0.04093
         Detection Rate : 0.00000
   Detection Prevalence : 0.00000
      Balanced Accuracy : 0.50000

       'Positive' Class : 1
```

```r
confusionMatrix(reference = testing_pred$felonies, testing_pred$pred_bag, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7183  302
```

```
         1   10    5

               Accuracy : 0.9584
                 95% CI : (0.9536, 0.9628)
    No Information Rate : 0.9591
    P-Value [Acc > NIR] : 0.6287

                  Kappa : 0.0273

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.0162866
            Specificity : 0.9986098
         Pos Pred Value : 0.3333333
         Neg Pred Value : 0.9596526
             Prevalence : 0.0409333
         Detection Rate : 0.0006667
   Detection Prevalence : 0.0020000
      Balanced Accuracy : 0.5074482

       'Positive' Class : 1
```

Sammenlign accuracy, sensitivity og specificity for de to modellene. Bagging gir typisk bedre og mer stabile resultater enn et enkelt tre fordi de tilfeldige feilene jevner seg ut over mange trær.

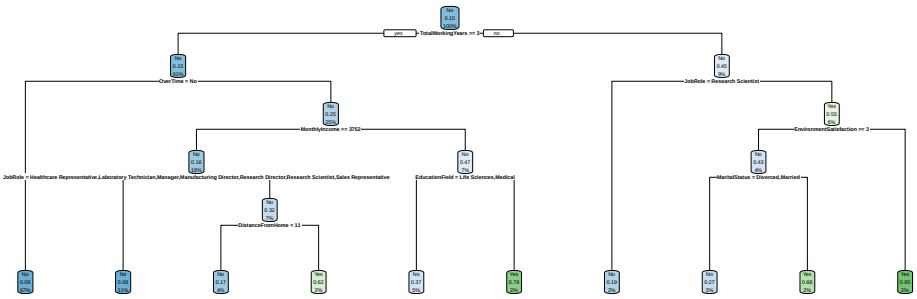
6.2 Eksempel med Attrition-data

Vi bruker også Attrition-dataene for å vise at bagging fungerer på tvers av datasett:

```
attrition <- readRDS("data/Attrition.rds") %>%
  select(-EmployeeNumber)

set.seed(426)
attrition_split <- initial_split(attrition)
attr_train <- training(attrition_split)
attr_test  <- testing(attrition_split)
```

Klassifikasjonstre, bagging og confusion matrix:

```
attr_tre <- rpart(Attrition ~ ., data = attr_train, cp = 0.00001, maxdepth = 5,
                  minbucket = 15, method = "class")
rpart.plot(attr_tre)
```




```
set.seed(42)
attr_bag <- bagging(Attrition ~ ., data = attr_train,
                    nbagg = 500, coob = TRUE)
attr_bag
```

```
Bagging classification trees with 500 bootstrap replications

Call: bagging.data.frame(formula = Attrition ~ ., data = attr_train,
    nbagg = 500, coob = TRUE)

Out-of-bag estimate of misclassification error:  0.1325
```

```
attr_test_pred <- attr_test %>%
  mutate(pred_tre = predict(attr_tre, newdata = attr_test, type = "class"),
         pred_bag = predict(attr_bag, newdata = attr_test, type = "class"))

confusionMatrix(reference = attr_test_pred$Attrition, attr_test_pred$pred_tre, positive = "Ye
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  288  57
       Yes  12  11

               Accuracy : 0.8125
                 95% CI : (0.7688, 0.8511)
    No Information Rate : 0.8152
    P-Value [Acc > NIR] : 0.5851

                  Kappa : 0.1636

 Mcnemar's Test P-Value : 1.177e-07

            Sensitivity : 0.16176
            Specificity : 0.96000
         Pos Pred Value : 0.47826
         Neg Pred Value : 0.83478
             Prevalence : 0.18478
         Detection Rate : 0.02989
   Detection Prevalence : 0.06250
      Balanced Accuracy : 0.56088

       'Positive' Class : Yes
```

```
confusionMatrix(reference = attr_test_pred$Attrition, attr_test_pred$pred_bag, positive = "Ye
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  296  55
       Yes   4  13

               Accuracy : 0.8397
                 95% CI : (0.7981, 0.8757)
    No Information Rate : 0.8152
    P-Value [Acc > NIR] : 0.1258
```

```
                 Kappa : 0.2505

 Mcnemar's Test P-Value : 7.543e-11

            Sensitivity : 0.19118
            Specificity : 0.98667
         Pos Pred Value : 0.76471
         Neg Pred Value : 0.84330
             Prevalence : 0.18478
         Detection Rate : 0.03533
   Detection Prevalence : 0.04620
      Balanced Accuracy : 0.58892

       'Positive' Class : Yes
```

Igjen ser vi at bagging typisk gir et resultat som er noe mer stabilt enn et enkelt tre. Merk at forbedringen ikke alltid er like dramatisk – det kommer an på dataene og det underliggende mønsteret.

6.3 Out-of-bag data (OOB)

Bagging kan brukes på training data og testing data som vi har gjort før. Men en ulempe med dette er jo at man får et litt mindre datasett å tilpasse modellene med. Merk at bagging i utgangspunktet baserer seg på å trekke et tilfeldig *utvalg* fra de opprinnelige dataene. Så for hvert tre er det en del observasjoner som ikke blir brukt til å bygge treet. Altså har vi et testing-datasett umiddelbart tilgjengelig!

Altså: hvis man bruker 70% av dataene til å bygge hvert enkelt tre, så har man også 30% testingdata tilgjengelig for å gjøre klassifiseringen på det enkelte treet. Dette testing-datasettet kalles *out-of-bag* (OOB) data, men er altså i prinsippet det samme som et testingdatasett.

Forvalget i `bagging()` er imidlertid å *ikke* bruke OOB. For å gjøre dette legges det til argumentet `coob = TRUE` og man får utregnet klassifikasjonsfeilen i output.

6.4 Mer tuning

Man kan si at bagging består av to deler: 1) en grunnleggende klassifikasjonsteknikk (noen ganger omtalt som "base learner"), og 2) en bootstrapping med gjentatte estimeringer på utvalg av data.

Forvalget i `bagging()` er å bruke klassifikasjonstrær med `rpart()`. Det betyr at tuning som kan gjøres med bruk av `rpart()` også kan gjøres med `bagging()`. Dette gjøres ved å legge til argumentet `control = rpart.control()`:

```
set.seed(42)
bag_tuned <- bagging(felonies ~ ., data = training,
                     nbagg = 500, coob = TRUE,
                     control = rpart.control(maxdepth = 6, cp = 0.0001))
bag_tuned
```

```
Bagging classification trees with 500 bootstrap replications

Call: bagging.data.frame(formula = felonies ~ ., data = training, nbagg = 500,
    coob = TRUE, control = rpart.control(maxdepth = 6, cp = 1e-04))

Out-of-bag estimate of misclassification error:  0.0403
```

Det viktige poenget akkurat nå er egentlig ikke all tuning som er mulig å gjøre i bagging, men at det er bygget på en annen underliggende algoritme. Vi bruker i praksis ikke bagging noe særlig alene, men derimot er random forest en variant av bagging med litt ekstra saker. Slik sett er bagging å regne som en byggesten til random forest sammen med klassifikasjonstrær. Så derfor skal vi ikke bruke så mye tid på bagging, men gå raskt videre til random forest.

7 Random forest

I dette kapittelt skal vi bruke følgende pakker:

```
library(tidyverse)   # datahåndtering, grafikk og glimpse()
library(rsample)     # for å dele data i training og testing
library(randomForest)
library(caret)
```

Random forest bruker klassifikasjonstrær og bagging som byggestener. I prinsippet er det "bagged trees", men i stedet for å bagge samme type trær, så gjøres det en endring i hvert tre: I hver split i hvert enkelt tre trekkes det bare et fåtall variable for å bestemme hver split. Det brukes med andre ord *mindre* informasjon i hvert enkelt tre! Man skulle intuitivt tro at dette vil gi dårligere prediksjon, og det stemmer forsåvidt i hvert enkelt tre. Men på den annen side lages det mange trær og det er det aggregerte resultatet som blir prediksjonen. På samme måte som i bagging, så avgjøres klassifikasjonen ved majoritetsstemme over alle trærne.

Husk nå at random forest er en variant av bagging, så det som i forrige kapittel ble sagt om OOB gjelder også her. Med andre ord: det training og testing datasett benyttes internt i algoritmen. Slik sett trenger man ikke splitte datasettet i forkant. Resultatene og feilratene er beregnet med OOB-dataene. I utgangspunktet trenger man altså ikke å splitte datasettet.

Men: Det går an å bruke gjøre en split likevel. Da er det mer å regne som et *valideringsdatasett*. Hvis man tuner modellen og sjekker OOB-resultatene, så står man i fare for å overfitte mot OOB.[1] Med andre ord: Det kan være en god ide å splitte dataene først likevel.

7.1 Eksempel med COMPAS-data

COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) er et risikoverktøy brukt av politi og rettsvesen i flere amerikanske stater. Datasettet inneholder informasjon om personer som har vært gjennom rettssystemet, der utfallsvariabelen `Two_yr_Recidivism` angir om personen begikk ny kriminalitet innen to år. Vi bruker dette datasettet gjennom flere kapitler i heftet.

[1] Jeg har forstått det slik at de lærde strides noe om dette.

```
compas <- readRDS("data/compas.rds")
glimpse(compas)
```

```
Rows: 6,172
Columns: 7
$ Two_yr_Recidivism    <fct> 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1~
$ Number_of_Priors     <int> 0, 0, 4, 0, 14, 3, 0, 0, 3, 0, 0, 1, 7, 0, 3, 6, ~
$ Age_Above_FourtyFive <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
$ Age_Below_TwentyFive <fct> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
$ Misdemeanor          <fct> 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0~
$ Ethnicity            <fct> Other, African_American, African_American, Other,~
$ Sex                  <fct> Male, Male, Male, Male, Male, Male, Female, Male,~
```

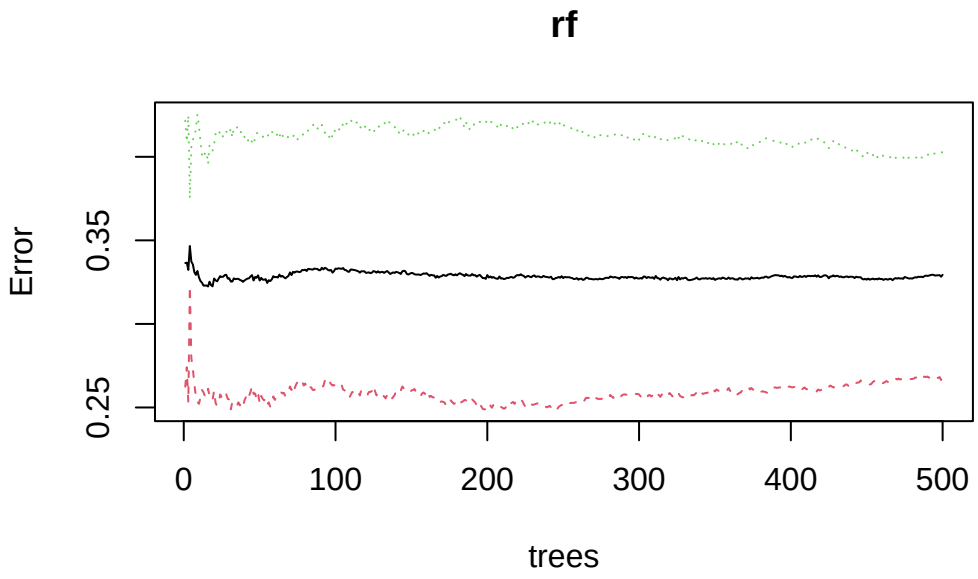
Estimerer random forest med alle variable.

```
set.seed(4356)
rf <- randomForest(Two_yr_Recidivism ~ . ,
                   data = compas)
rf
```

```
Call:
 randomForest(formula = Two_yr_Recidivism ~ ., data = compas)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 32.94%
Confusion matrix:
     0    1 class.error
0 2462  901   0.2679156
1 1132 1677   0.4029904
```

Følgende plot gir en oversikt over feilrater for random forest etter hvor mange trær. Den svarte linjen er den totale feilraten, den grønne er falske positive, og den røde er falske negative. I utgangspunktet bruker random forest 500 trær. Dette plottet viser når resultatene stabiliserer seg.

```
plot(rf)
```



Prediksjon fungerer på tilsvarende måte som vi har gjort tidligere.

```
compas_p <- compas %>%
  mutate(pred_rf = predict(rf))
```

Confusion matrix:

```
confusionMatrix(reference = compas_p$Two_yr_Recidivism, compas_p$pred_rf, positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2462 1132
         1  901 1677

               Accuracy : 0.6706
                 95% CI : (0.6587, 0.6823)
    No Information Rate : 0.5449
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3313
```

```
 Mcnemar's Test P-Value : 3.378e-07

            Sensitivity : 0.5970
            Specificity : 0.7321
         Pos Pred Value : 0.6505
         Neg Pred Value : 0.6850
             Prevalence : 0.4551
         Detection Rate : 0.2717
   Detection Prevalence : 0.4177
      Balanced Accuracy : 0.6645

       'Positive' Class : 1
```

Vi kan justere resultatet med å endre antall variable som tas med i hver split (i hvert tre). Parameteren `mtry` styrer dette:

```
set.seed(4356)
rf2 <- randomForest(Two_yr_Recidivism ~ . ,
                    mtry=4,
                    data = compas)
rf2
```

```
Call:
 randomForest(formula = Two_yr_Recidivism ~ ., data = compas,      mtry = 4)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 34.36%
Confusion matrix:
     0    1 class.error
0 2608  755   0.2245019
1 1366 1443   0.4862941
```

7.1.1 Variable importance

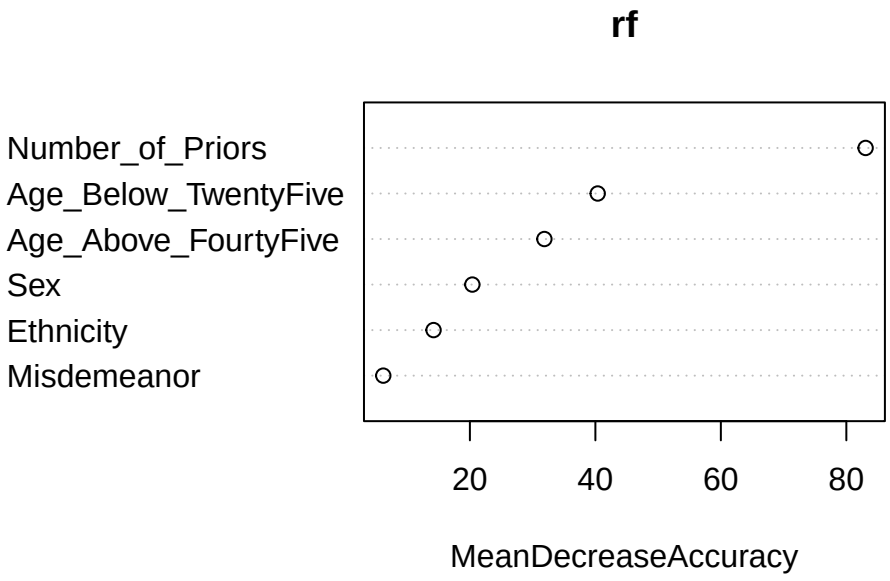
For å få ut variable importance må dette settes i estimeringen med `importance = TRUE`. Det tar nå litt lengre tid å estimere, så med store datasett bør du vente med dette til du ellers er fornøyd med modellen.

```r
set.seed(4356)
rf <- randomForest(Two_yr_Recidivism ~ . ,
                   importance = TRUE,
                    data = compas)
rf
```

```
Call:
 randomForest(formula = Two_yr_Recidivism ~ ., data = compas,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 32.42%
Confusion matrix:
     0    1 class.error
0 2538  825   0.2453167
1 1176 1633   0.4186543
```

Vi kan da plotte variable importance plot. Set `type = 1` for at det skal vise gjennomsnittlig reduksjon i accuracy. Plottet viser hvor mye dårligere modellens prediksjon blir hvis en variabel permuteres (stokkes om). Variable som er høyt oppe i plottet er mest viktige for prediksjonen.

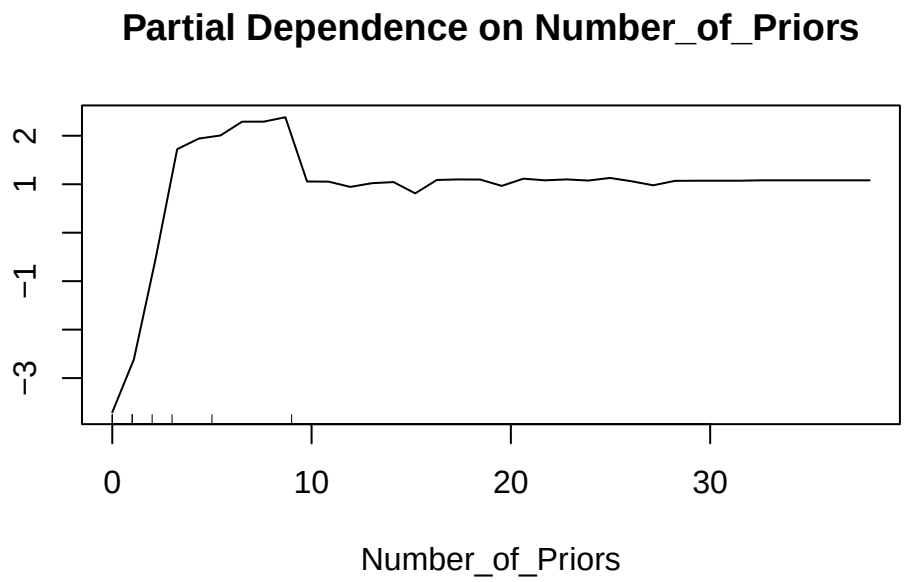
```r
varImpPlot(rf, type = 1)
```



7.1.2 Partial dependence

Variable importance forteller oss *hvilke* variable som er viktige, men ikke *hvordan* de påvirker prediksjonen. Partial dependence plot viser sammenhengen mellom en enkelt variabel og den predikerte sannsynligheten, justert for alle andre variable. Her må du velge hvilken variabel du ønsker å se på, og `which.class` angir hvilken klasse du vil se sannsynligheten for.

```
partialPlot(rf, pred.data = compas,
            x.var = Number_of_Priors,
            which.class = "1")
```



Plottet viser at sannsynligheten for tilbakefall øker med antall tidligere lovbrudd, noe som er intuitivt rimelig.

7.2 Tuning av random forest med sampsize

Det er mest vanlig å justere sampling-skjemaet i baggingen: altså hvor mange observasjoner som trekkes til å bygge hvert tre. Ved å bruke argumentet `sampsize = ...` kan vi angi antallet som trekkes fra hver kategori i utfallsvariabelen.

Hensikten med å gjøre dette er at hvis det er et mindretall som har det ene utfallet, så blir hvert tre bygget med mer informasjon om majoritetsgruppen. Hvis vi vekter opp minoritetsgruppen, så får disse større inflytelse på hvert tre.

7.2.1 Eksempel med COMPAS-data

Her setter vi `sampsize = c(1900, 1900)`, altså like mange fra hver kategori av utfallsvariabelen. Det første tallet er antallet som trekkes fra kategori 0 (ingen tilbakefall) og det andre fra kategori 1 (tilbakefall):

```r
set.seed(4356)
rf3 <- randomForest(Two_yr_Recidivism ~ . ,
                    sampsize = c(1900, 1900),
                    data = compas)
rf3
```

```
Call:
 randomForest(formula = Two_yr_Recidivism ~ ., data = compas,      sampsize = c(1900, 1900))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 32.73%
Confusion matrix:
     0    1 class.error
0 2321 1042   0.3098424
1  978 1831   0.3481666
```

Her er et eksempel der gruppene veies ulikt – vi trekker færre fra kategori 0 enn fra kategori 1, noe som gir mer vekt til tilbakefalls-gruppen:

```r
set.seed(4356)
rf4 <- randomForest(Two_yr_Recidivism ~ . ,
                    sampsize = c(1000, 1900),
                   data = compas)
rf4
```

```
Call:
 randomForest(formula = Two_yr_Recidivism ~ ., data = compas,      sampsize = c(1000, 1900))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 41.15%
Confusion matrix:
     0    1 class.error
0 1170 2193   0.6520963
1  347 2462   0.1235315
```

Det viktige nå er at feilratene for falske positive og falske negative blir vesentlig forskjellig! Det betyr at ved hvordan vi estimerer modellen kan vi legge sterke føringer på resultatet. Vi bør derfor ta stilling til *på forhånd* hvilke feilrater vi er villig til å akseptere.

7.2.2 Eksempel med syntetiske data

Vi kan gjøre tilsvarende med de syntetiske fødselskohort-dataene. Her håndterer vi først eventuelle manglende verdier med `replace_na()`. Merk at `randomForest()` ikke håndterer manglende verdier (NA), så disse må erstattes før estimeringen. En enkel løsning er å sette NA til 0, men i praksis bør man vurdere om dette er faglig rimelig.

```
load("data/synthetic_birthCohort96.RData")

synth <- synthetic_birthCohort %>%
  select(-drugs, -violence) %>%
  mutate(felonies = factor(felonies)) %>%
  mutate(across(everything(), ~replace_na(.x, 0)))

set.seed(426)
synth_split <- initial_split(synth)
synth_train <- training(synth_split)
synth_test  <- testing(synth_split)

table(synth_train$felonies)
```

```
    0     1
21579   921
```

Vi sammenligner flere ulike konfigurasjoner av `sampsize`. Først kjører vi med forvalget, uten å angi `sampsize`. Da trekker `randomForest()` like mange observasjoner som det er i datasettet, men med tilbakelegging:

```
# Default (ingen sampsize angitt)
rf_s1 <- randomForest(felonies ~ ., data = synth_train)

synth_test1 <- synth_test %>%
  mutate(pred = predict(rf_s1, newdata = ., type = "response"))

confusionMatrix(reference = synth_test1$felonies, synth_test1$pred, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7193  306
         1    0    1

               Accuracy : 0.9592
                 95% CI : (0.9545, 0.9636)
    No Information Rate : 0.9591
    P-Value [Acc > NIR] : 0.4919

                  Kappa : 0.0062

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.0032573
            Specificity : 1.0000000
         Pos Pred Value : 1.0000000
         Neg Pred Value : 0.9591946
             Prevalence : 0.0409333
         Detection Rate : 0.0001333
   Detection Prevalence : 0.0001333
      Balanced Accuracy : 0.5016287

       'Positive' Class : 1
```

Deretter prøver vi en balansert sampsize der vi trekker like mange fra hver kategori:

```r
# Balansert sampsize
rf_s2 <- randomForest(felonies ~ .,
                      sampsize = c(600, 600),
                      data = synth_train)

synth_test2 <- synth_test %>%
  mutate(pred = predict(rf_s2, newdata = ., type = "response"))

confusionMatrix(reference = synth_test2$felonies, synth_test2$pred, positive = "1")
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction    0    1
         0 5516  145
         1 1677  162

               Accuracy : 0.7571
                 95% CI : (0.7472, 0.7667)
    No Information Rate : 0.9591
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0869

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.52769
            Specificity : 0.76686
         Pos Pred Value : 0.08809
         Neg Pred Value : 0.97439
             Prevalence : 0.04093
         Detection Rate : 0.02160
   Detection Prevalence : 0.24520
      Balanced Accuracy : 0.64727

       'Positive' Class : 1
```

Vi kan også prøve en ubalansert sampsize der vi trekker litt flere fra minoritetsgruppen:

```r
# Ubalansert sampsize
rf_s3 <- randomForest(felonies ~ .,
                      sampsize = c(500, 600),
                      data = synth_train)

synth_test3 <- synth_test %>%
  mutate(pred = predict(rf_s3, newdata = ., type = "response"))

confusionMatrix(reference = synth_test3$felonies, synth_test3$pred, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
```

```
         0 4922  106
         1 2271  201

               Accuracy : 0.6831
                 95% CI : (0.6724, 0.6936)
    No Information Rate : 0.9591
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0775

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.65472
            Specificity : 0.68428
         Pos Pred Value : 0.08131
         Neg Pred Value : 0.97892
             Prevalence : 0.04093
         Detection Rate : 0.02680
   Detection Prevalence : 0.32960
      Balanced Accuracy : 0.66950

       'Positive' Class : 1
```

Til sist prøver vi en konfigurasjon der vi trekker langt flere fra kategori 0 enn fra kategori 1:

```r
# Større sampsize
rf_s4 <- randomForest(felonies ~ .,
                      sampsize = c(800, 400),
                      data = synth_train)

synth_test4 <- synth_test %>%
  mutate(pred = predict(rf_s4, newdata = ., type = "response"))

confusionMatrix(reference = synth_test4$felonies, synth_test4$pred, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 6861  245
         1  332   62
```

```
            Accuracy : 0.9231
              95% CI : (0.9168, 0.929)
   No Information Rate : 0.9591
   P-Value [Acc > NIR] : 1.0000000

                 Kappa : 0.1372

 Mcnemar's Test P-Value : 0.0003433

           Sensitivity : 0.201954
           Specificity : 0.953844
        Pos Pred Value : 0.157360
        Neg Pred Value : 0.965522
            Prevalence : 0.040933
        Detection Rate : 0.008267
  Detection Prevalence : 0.052533
     Balanced Accuracy : 0.577899

      'Positive' Class : 1
```

Merk at de ulike konfigurasjonene gir tydelig forskjellige feilrater. Det er dette Berk (2016) kaller *asymetriske kostnader* og må vurderes i henhold til konsekvenser av hva prediksjonen skal brukes til.

8 Boosting

I dette kapittelt skal vi bruke følgende pakker:

```
library(tidyverse)   # datahåndtering, grafikk og glimpse()
library(rsample)     # for å dele data i training og testing
library(gbm)         # Funksjoner for boosting
library(caret)       # For funksjonen confusionMatrix()
library(fairness)    # For fairnessmetrics
library(gtsummary)   # For å lage litt penere tabeller
```

Det er flere typer boosting-algoritmer. Men vi skal fokusere på stochastic gradient boosting. Adaptive boosting er presentert i læreboka, men først og fremst som et illustrativt eksempel eller en pedagogisk introduksjon. Andre boosting-algoritmer er varianter av gradient boosting.

Det er også et poeng at softwaren for adaptive boosting er mer begrenset enn for gradient boosting. Vi bruker 'gbm' pakken som gir grunnleggende funksjonalitet.

Det sentrale punktet for boosting er at modellen bygges steg for steg med en gradvis forbedring fra forrige runde slik at modellen blir gradvis bedre og bedre. Dette til kontrast til *bagging* der styrken ligger i avstemning på tvers av mange klassifikasjonstrær. I boosting er det altså samme modell som forbedres i hvert steg. Dette gjøres på en måte der feilklassifiseringen vektes opp i hver iterasjon. I adaptive boosting lages den en vekt litt mekanisk, mens i gradient boosting baseres det på en gradient fra velkjente statistiske fordelinger, f.eks. binomisk fordeling for kategorisk utfall.

Vi bruker compas-dataene igjen. Men nå må vi omkode utfallsvariabelen til en numerisk variabel fordi funksjonen `gbm()` krever numerisk utfallsvariabel. Vi bruker `na.omit()` for å fjerne eventuelle rader med manglende verdier.

```
compas <- readRDS("data/compas.rds") %>%
  mutate(Two_yr_Recidivism = ifelse(Two_yr_Recidivism == "1", 1, 0)) %>%
  na.omit()
glimpse(compas)
```

```
Rows: 6,172
Columns: 7
$ Two_yr_Recidivism    <dbl> 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1~
$ Number_of_Priors     <int> 0, 0, 4, 0, 14, 3, 0, 0, 3, 0, 0, 1, 7, 0, 3, 6, ~
$ Age_Above_FourtyFive <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
$ Age_Below_TwentyFive <fct> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
$ Misdemeanor          <fct> 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0~
$ Ethnicity            <fct> Other, African_American, African_American, Other,~
$ Sex                  <fct> Male, Male, Male, Male, Male, Male, Female, Male,~
```

Vi lager en split som tidligere

```
set.seed(42)
training_init <- initial_split(compas)

training <- training(training_init)
testing  <- testing(training_init)
```

8.1 Stochastic gradient boosting

Gradient boosting for klassifisering bruker ikke vekter på samme måte som adaboost, men bruker residualene. For et binomisk utfall (to kategorier kodet 0 eller 1) er residualen, r_i, er definert som

$$r_i = y_i - \frac{1}{e^{-f(x_i)}}$$

Litt forenklet kan vi si at dette er det observert utfallet minus det predikerte utfallet fra logistisk regresjon. "Gradienten" er da $f(x)$.

Gradient boosting fungerer med flere andre fordelinger, men hvis vi begrenser oss til klassifikasjon kan vi angi `distribution = "bernoulli"`. Merk at for `gbm` må utfallsvariabelen være numerisk, ikke factor.

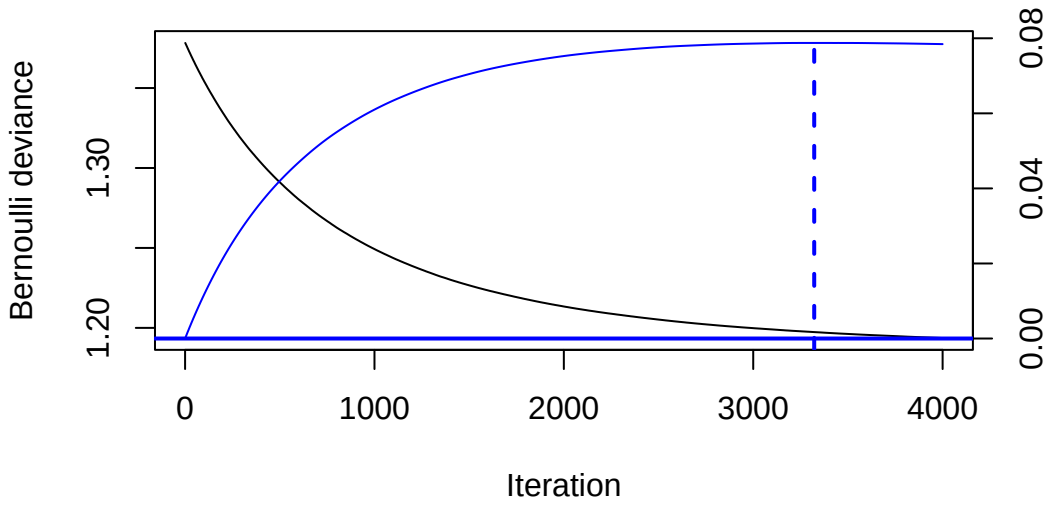
```
set.seed(542)
gradboost1 <- gbm(formula = Two_yr_Recidivism ~ .,
                  data = training,
                  distribution = "bernoulli",
                  n.trees = 4000,
                  interaction.depth = 3,
                  n.minobsinnode = 1,
```

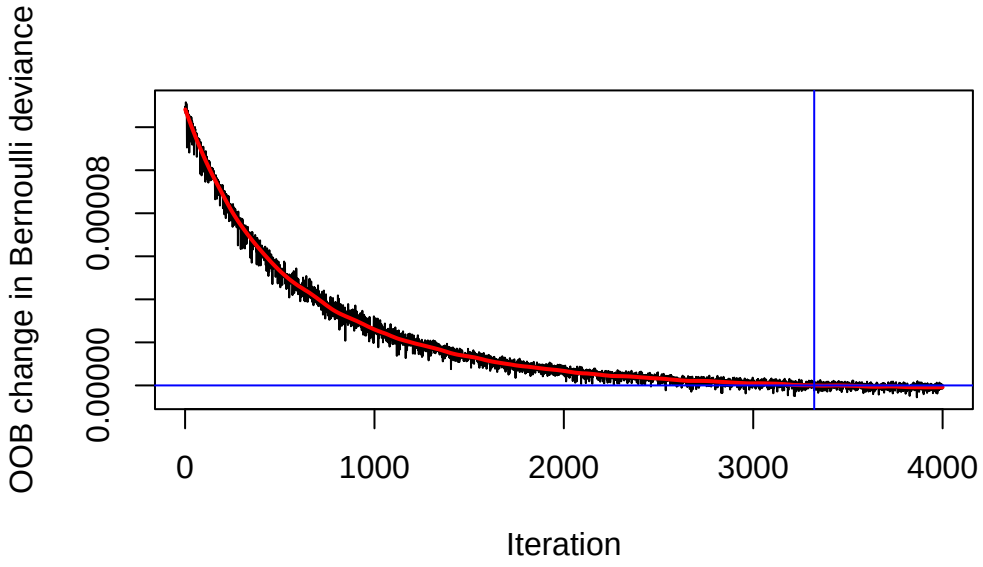
```
                shrinkage = 0.001,
                bag.fraction = 0.5)
```

Merk angir argumentet `bag.fraction = 0.5` i modellen (som forøvrig også er forvalget, så man behøver ikke skrive det, egentlig). I hver split brukes altså halve utvalget til å bestemme neste split slik at andre halvdel er out-of-bag data. Hvis dette argumentet settes til 1 brukes hele datasettet i hver split.

Standard plot av gis ved `gbm.perf()` som nedenfor, og viser forbedring for hver iterasjon. Ytterligere forbedring stopper opp ved nærmere 2500 iterasjoner, markert ved den vertikale blå linjen.

```
gbm.perf(gradboost1, oobag.curve = TRUE, method = "OOB",
            plot.it=T, overlay = T)
```






```
[1] 3322
attr(,"smoother")
Call:
loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,
    length(x)/10), 50))

Number of Observations: 4000
Equivalent Number of Parameters: 39.99
Residual Standard Error: 2.048e-06
```

Denne estimeringen kan justeres ved å endre tuningparametrene.

- `n.trees` er antall iterasjoner, dvs. antall klassifikasjonstrær. Forvalget er 100, som er åpenbart altfor lavt, så sett noen tusen.
- `interaction.depth` er antall split i hvert tre. Forvalget er 1, men Berk sier at 1-3 er ok.
- `n.minobsinnode` er minste antall observasjoner i siste node. Forvalg er 1, men Berk anslår at 5-15 fungerer bra. (Hvis modellen har bare en split er det ikke sikkert dette er så viktig)
- `shrinkage` er hvor store skritt langs gradienten som testes i hver iterasjon, og dette skal være lavt. Forvalget er 0.001. Kostnaden er at det tar lengre tid enn ved et høyere tall (opp til 10 kan testes).

- `bag.fraction` er hvor stor andel av data som brukes i hver iterasjon. Dette hjelper for å redusere overfitting. Forvalget er 0.5, som innebærer at andre del av data kan fungere som OOB. Merk at Berk understreker at OOB bare kan brukes til å vurdere tilpassingen slik som i figuren over.

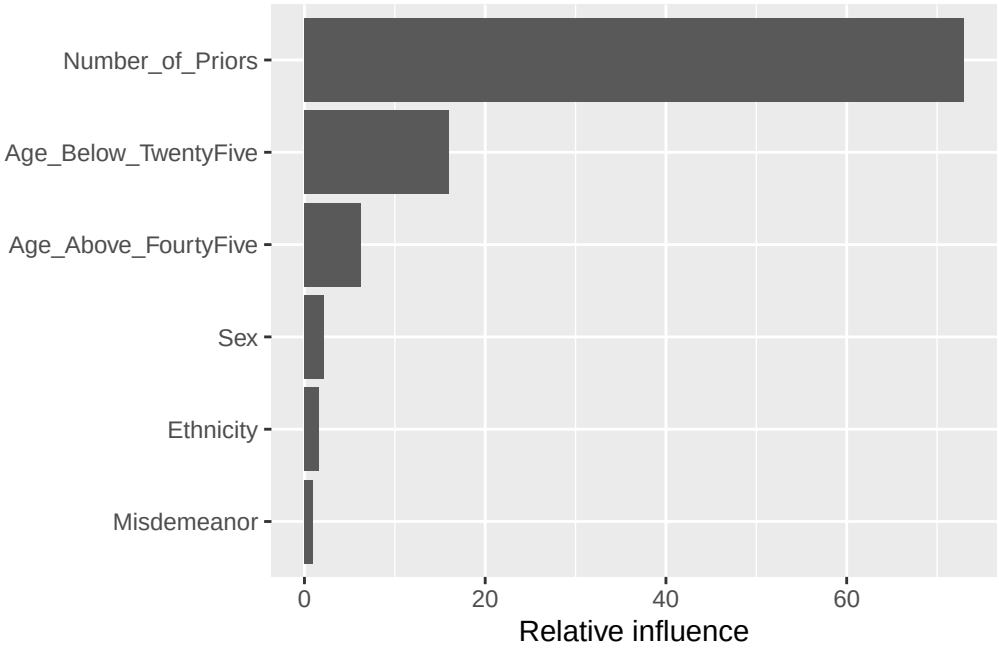
8.2 Tolkbarhet

Som i random forest kan vi undersøke hvilke variable som er mest betydningsfulle i prediksjonen. I utgangspunktet gir `summary()` mot et gbm-objekt et plot, men dette er stygt og kan være vanskelig å lese skikkelig. Derfor inneholder koden nedenfor argumentet `plotit=FALSE`, så lages det et bedre plot med `ggplot()` etterpå.[1]

Resultatene er tolkbare tilsvarende som relative importance som vi så i random forest.

```
sumboost <- summary(gradboost1, method = permutation.test.gbm, normalize = T,
                    plotit = F)

ggplot(sumboost, aes(x = reorder(var, rel.inf), y = rel.inf)) +
  geom_col()+
  ylab("Relative influence")+
  xlab("")+
  coord_flip()
```

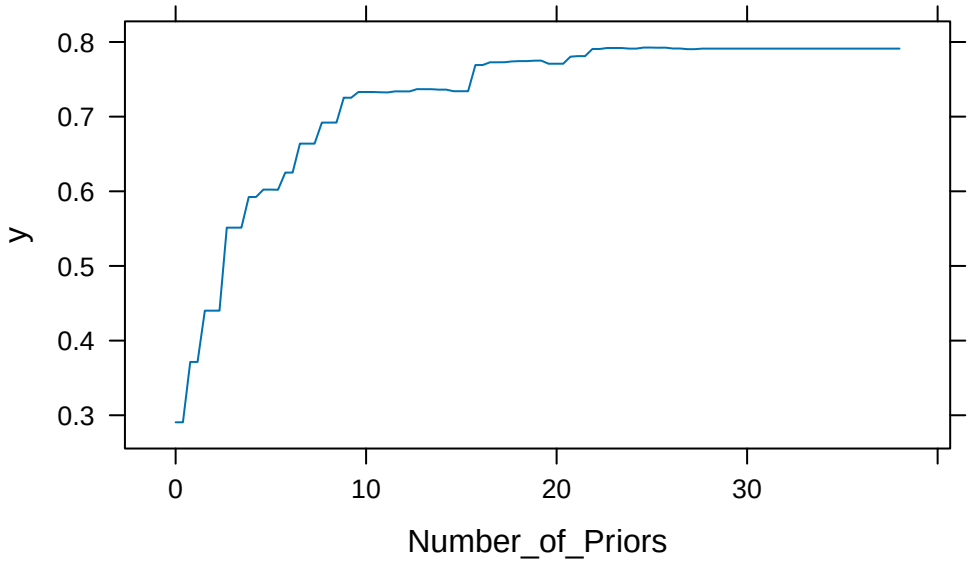
[1]Estetikken er imidlertid underordet her. Sett gjerne `plotit = TRUE` for et mer quisk-and-dirty plot.



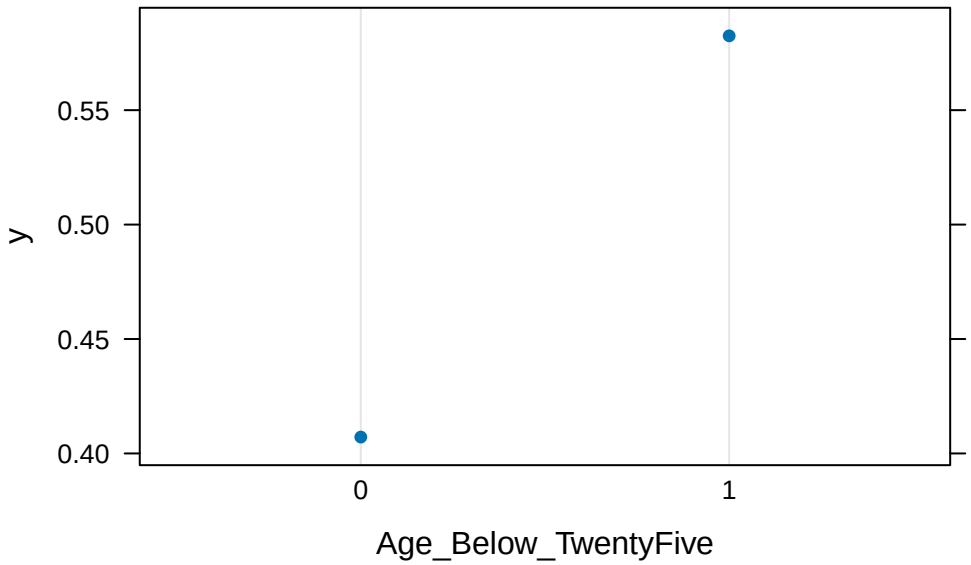
Merk at resultatet fra `summary()` her returnerer en data.frame som kan plottes direkte med `ggplot`. Eneste mystiske som skjer er at `x` er angitt som sortert etter verdier på `y`-variabelen. Så er plottet snudd om med `coord_flip` til slutt.

Man kan også plotte hvordan resultatet endres med ulike verdier på prediktorene. Dette tilsvarer *partial dependence*. Her er det mest hensiktsmessig å bruke den innebygde funksjonen `plot()` som kaller en underliggende plot-funksjon for gbm-objekter.

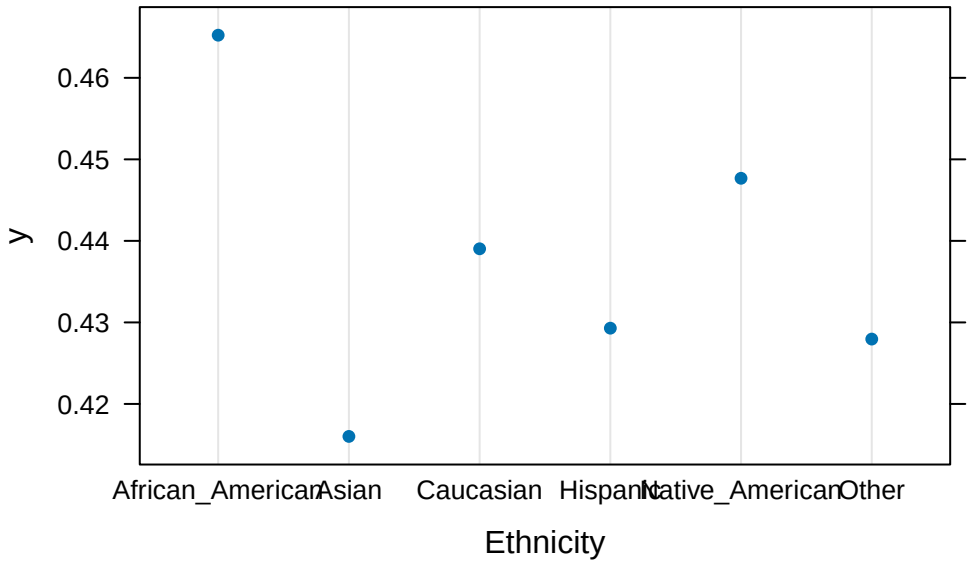
```
plot(gradboost1, "Number_of_Priors", type = "response")
```



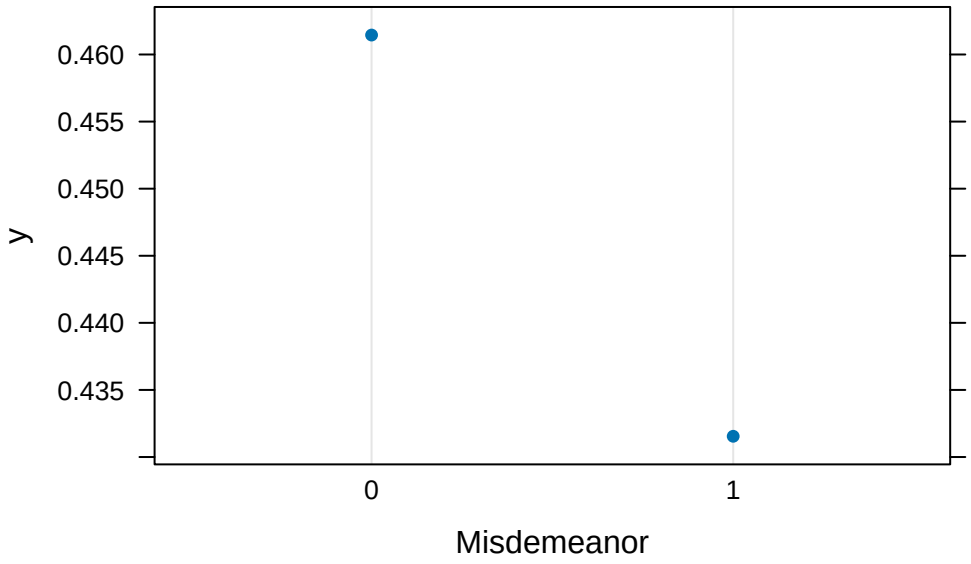

```
plot(gradboost1, "Age_Below_TwentyFive", type = "response")
```



```
plot(gradboost1, "Ethnicity", type = "response")
```




```
plot(gradboost1, "Misdemeanor", type = "response")
```



8.3 Prediksjon og confusion matrix

Vi gjør prediksjon på tilsvarende måte som før, men det er et par viktige detaljer. Forvalget for `predict()` for gbm-objekter er $f(x)$ som her er på log odds skalaen, men hvis vi setter `type = "response"` så får vi ut en sannsynlighet (dvs. et tall mellom 0 og 1). Da kan vi klassifisere etter hvilken gruppe som er mest sannsynlig, dvs. hvis høyere enn 0.5

```
compas_p <- training %>%
  mutate(pred = predict(gradboost1, type = "response"),
         p_klass = ifelse(pred > 0.5, 1, 0))
```

Lager enkel krysstabell med predikert mot observert (dvs confusion matrix)

```
tab <- table(compas_p$p_klass, training$Two_yr_Recidivism)
tab
```

```
   0    1
0 1936  850
1  583 1260
```

Lager bedre confusion matrix med alle øvrige utregninger. NB! Husk å presisere hva som er positiv verdi for at tallene skal blir riktig vei.

```
confusionMatrix(tab, positive="1")
```

```
Confusion Matrix and Statistics


   0    1
0 1936  850
1  583 1260

               Accuracy : 0.6904
                 95% CI : (0.6769, 0.7037)
    No Information Rate : 0.5442
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3695
```

```
 Mcnemar's Test P-Value : 2.113e-12

            Sensitivity : 0.5972
            Specificity : 0.7686
         Pos Pred Value : 0.6837
         Neg Pred Value : 0.6949
             Prevalence : 0.4558
         Detection Rate : 0.2722
   Detection Prevalence : 0.3981
      Balanced Accuracy : 0.6829

       'Positive' Class : 1
```

8.4 Asymetriske kostnader

Vi har det vanlige problemet med å vurdere om falske positive og falske negative er like problematisk. Igjen er det slik at den vurderingen krever en vurdering av utfallet man ønsker gjøre noe med og konsekvensene av tiltaket som skal iverksettes. For credit-dataene kan det jo være at noen ikke vurderes som kredittverdige.[2]

For å håndtere asymetriske kostnader kan vi vekte utfallene forskjellig og angi denne vekten i prosedyren. Argumentet `weights = ...` tar en vektor med verdier (ét tall for hver observasjon i dataene) og skal ikke være en del av datasettet.

Her er et eksempel der *negative* tillegges 2 ganger så mye vekt som *positive* i modellen. Når utfallene vektes ulikt i modellen vil det dermed slå ut på feilratene slik at det blir ulikt forhold for falske postive og falske negative (se mer i Berk kap. 6.4 og eksempel s. 277).

Vær obs på at det er ikke så lett å vite helt hvordan slik vekting slår ut på resultatene. Det er ikke et 1-til-1 forhold mellom vekting og feilrater. Som i annen justering kan dette gå ut over accuracy og andre mål, så det er vanligvis en trade-off her. Du må derfor sjekke resultatene og så evt. gå tilbake og justere vektene hvis det ikke ble slik du ønsket.

```
wts <- training %>%
  mutate(wts = ifelse(Two_yr_Recidivism == 1, 1, 2)) %>%   # se begrunnelse s. 274 i Berk
  pull(wts)

set.seed(542)
```

[2] Det er forresten også en implisitt vurdering her om hvem sitt problem dette er: banken behøver ikke synes det er problematisk å si nei til et boliglån, mens det fra samfunnets synsvinkel kan være uheldig at noen grupper sperres ute av boligmarkedet. Hvem som er "stakeholder" her er altså også et poeng, som vi lar ligge i denne sammenheng.

```
gradboost2 <- gbm(formula = Two_yr_Recidivism ~ .,
                  data = training,
                  weights = wts,
                  distribution = "bernoulli",
                  n.trees = 4000,
                  interaction.depth = 3,
                  n.minobsinnode = 1,
                  shrinkage = 0.001,
                  bag.fraction = 0.5)
```

Så kan vi gjøre prediksjon og confusion matrix på nytt.

```
compas_p <- training %>%
  mutate(pred = predict(gradboost2, type = "response"),
         p_klass = ifelse(pred > 0.5, 1, 0))
```

Så kan vi legge tabellen inn i funksjonen `confusionMatrix()` for å få diverse utregninger. (Husk å presisere hva som er positiv verdi for at tallene skal blir riktig vei.) Dette er altså akkurat samme prosedyre som vi har brukt ved tidligere prediksjoner.

En endring her er at `confusionMatrix()` vil at variablene skal være factor-variable, mens boosting-algoritmen bruker numeriske variable selv om verdiene bare er 0 eller 1. Her er en løsning å legge til `factor()` som følger. Alternativt kan man gjøre om variablene til factor i steget ovenfor i det nye datasettet `compas_p`.

```
confusionMatrix(reference = factor(compas_p$Two_yr_Recidivism),
                factor(compas_p$p_klass), positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2329 1447
         1  190  663

               Accuracy : 0.6464
                 95% CI : (0.6324, 0.6601)
    No Information Rate : 0.5442
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2509
```

```
Mcnemar's Test P-Value : < 2.2e-16

           Sensitivity : 0.3142
           Specificity : 0.9246
        Pos Pred Value : 0.7773
        Neg Pred Value : 0.6168
            Prevalence : 0.4558
        Detection Rate : 0.1432
  Detection Prevalence : 0.1843
     Balanced Accuracy : 0.6194

      'Positive' Class : 1
```

9 Fairness i maskinlæring

I dette kapittelet skal vi bruke følgende pakker:

```
library(tidyverse)      # datahåndtering og grafikk
library(randomForest)   # random forest-modeller
library(gbm)            # gradient boosting
library(caret)          # confusionMatrix()
library(DALEX)          # lage explainer-objekter for modeller
library(fairmodels)     # fairness-sjekk og visualisering
```

Nå som vi har sett på klassifikasjonstrær, bagging, random forest og boosting er det på tide å gjøre noen rettferdighetsbetraktninger. I maskinlæring er *fairness* et sentralt tema: modeller kan systematisk behandle ulike grupper ulikt, noe som kan ha alvorlige konsekvenser avhengig av hva modellen brukes til.

9.1 Hva slags rettferdighet?

I denne settingen kan rettferdighet komme i betraktning på flere måter, herunder følgende:

- I hvilken grad maskiner vs mennesker tar avgjørelser, og herunder mulighet til å bli hørt og legge frem sin sak
- I hvilken grad *dataene* algoritmen er trent opp på inneholder skjevheter i utgangspunktet som så reproduseres i videre implementering
- I hvilken grad sluttresultatet har rimelig presisjon og akseptable feilrater, herunder vurdering av *asymetriske feilrater*
- I hvilken grad forrige punkt er avpasset mot hvilke *tiltak* man så setter i verk
- I hvilken grad feilrater og presisjon varierer systematisk med undergrupper i populasjonen

Det er nok av ting å ta tak i her, men vi skal her fokusere på det som kan tallfestes gitt den modellen man har. Men for all del: Er datakvaliteten dårlig er det begrenset hvor bra det kan bli uansett. Selv om kjente skjevheter i dataene i prinsippet kan motarbeides, er det vel i praksis slik at en skjevhet sjelden kommer alene.

9.2 COMPAS-datasettet

COMPAS er et risikoverktøy brukt av amerikansk politi i flere stater som benyttes på individnivå til å vurdere risikoen for at en person begår ny kriminalitet. Bruken av dette verktøyet har vært kontroversielt i flere år og kraftig kritisert. En viktig grunn er at prediksjonene slår forskjellig ut for ulike grupper og er slik sett "biased" mot bl.a. svarte borgere. Resultatet er at de blir mer utsatt for politiets oppmerksomhet enn andre.[1] Et datasett er gjort tilgjengelig av Propublica her som vi skal bruke.

```
compas <- readRDS("data/compas.rds")
glimpse(compas)
```

```
Rows: 6,172
Columns: 7
$ Two_yr_Recidivism    <fct> 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1~
$ Number_of_Priors     <int> 0, 0, 4, 0, 14, 3, 0, 0, 3, 0, 0, 1, 7, 0, 3, 6, ~
$ Age_Above_FourtyFive <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
$ Age_Below_TwentyFive <fct> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
$ Misdemeanor          <fct> 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0~
$ Ethnicity            <fct> Other, African_American, African_American, Other,~
$ Sex                  <fct> Male, Male, Male, Male, Male, Male, Female, Male,~
```

9.3 Baseline-modell

Vi tilpasser en random forest-modell på datasettet:

```
set.seed(4356)
rf <- randomForest(Two_yr_Recidivism ~ .,
                   data = compas)
```

Vi lagrer prediksjonen i en kopi av datasettet, slik at vi kan sammenligne prediksjon og observert utfall:

```
compas_p <- compas %>%
  mutate(pred_rf = predict(rf))
```

En overordnet confusion matrix viser modellens ytelse for alle observasjoner samlet:

[1] Det er verd å minne på at politi i USA i stor grad er mer hardhendte enn norsk politi. Konsekvensene er altså litt mer alvorlig enn at unødig mange føler seg unødig mistenkte.

```
confusionMatrix(compas_p$pred_rf,
                compas_p$Two_yr_Recidivism, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2462 1132
         1  901 1677

               Accuracy : 0.6706
                 95% CI : (0.6587, 0.6823)
    No Information Rate : 0.5449
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3313

 Mcnemar's Test P-Value : 3.378e-07

            Sensitivity : 0.5970
            Specificity : 0.7321
         Pos Pred Value : 0.6505
         Neg Pred Value : 0.6850
             Prevalence : 0.4551
         Detection Rate : 0.2717
   Detection Prevalence : 0.4177
      Balanced Accuracy : 0.6645

       'Positive' Class : 1
```

For å forstå *bias* kan vi dele opp confusion matrix etter kjønn. Her er modellens ytelse for menn:

```
compas_m <- compas_p %>% filter(Sex == "Male")
confusionMatrix(compas_m$pred_rf,
                compas_m$Two_yr_Recidivism, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
```

```
Prediction    0    1
         0 1807  897
         1  794 1499

               Accuracy : 0.6616
                 95% CI : (0.6483, 0.6747)
    No Information Rate : 0.5205
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.3209

 Mcnemar's Test P-Value : 0.01312

            Sensitivity : 0.6256
            Specificity : 0.6947
         Pos Pred Value : 0.6537
         Neg Pred Value : 0.6683
             Prevalence : 0.4795
         Detection Rate : 0.3000
   Detection Prevalence : 0.4589
      Balanced Accuracy : 0.6602

       'Positive' Class : 1
```

Og for kvinner:

```
compas_f <- compas_p %>% filter(Sex == "Female")
confusionMatrix(compas_f$pred_rf,
                compas_f$Two_yr_Recidivism, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 655 235
         1 107 178

               Accuracy : 0.7089
                 95% CI : (0.682, 0.7348)
    No Information Rate : 0.6485
    P-Value [Acc > NIR] : 6.251e-06
```

```
                 Kappa : 0.3128

 Mcnemar's Test P-Value : 6.539e-12

            Sensitivity : 0.4310
            Specificity : 0.8596
         Pos Pred Value : 0.6246
         Neg Pred Value : 0.7360
             Prevalence : 0.3515
         Detection Rate : 0.1515
   Detection Prevalence : 0.2426
      Balanced Accuracy : 0.6453

       'Positive' Class : 1
```

Modellen er mer presis for kvinner enn for menn — dette er et eksempel på *bias* i modellen. Forholdet mellom accuracy for menn og kvinner gir et enkelt mål på skjevheten. Men å sammenligne confusion matriser manuelt for mange grupper og mange mål er arbeidskrevende. Det er her fairness-verktøy kommer inn.

9.4 Mål på fairness med fairmodels

Pakken `fairmodels` gjør det enkelt å måle fairness på tvers av grupper for mange mål på én gang. For å bruke den trengs et *explainer*-objekt fra `DALEX`-pakken. Et explainer-objekt er en standardisert innpakning rundt en modell, slik at `fairmodels` kan bruke ulike typer modeller (random forest, boosting, etc.) med det samme grensesnittet.

Vi lager explainer-objektet for baseline random forest-modellen. Vi angir modellen, prediktordata (uten utfallsvariabelen), og den numeriske versjonen av utfallsvariabelen:

```
y_num    <- as.numeric(as.character(compas$Two_yr_Recidivism))
compas_X <- compas %>% select(-Two_yr_Recidivism)

explainer_rf <- DALEX::explain(
  model = rf,
  data  = compas_X,
  y     = y_num,
  label = "Random Forest"
)
```

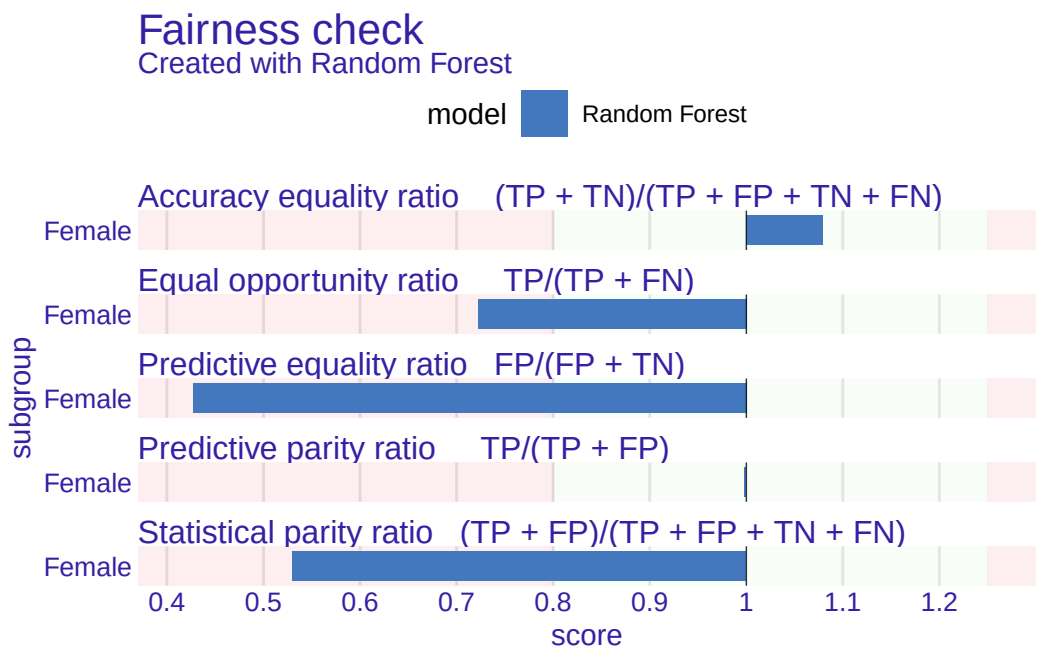
```
Preparation of a new explainer is initiated
  -> model label       :  Random Forest
  -> data              :  6172  rows  6  cols
  -> target variable   :  6172  values
  -> predict function  :  yhat.randomForest  will be used (  default  )
  -> predicted values  :  No value for predict function target column. (  default  )
  -> model_info        :  package randomForest , ver. 4.7.1.2 , task classification (  defaul
  -> predicted values  :  numerical, min =  0 , mean =  0.4037122 , max =  1
  -> residual function :  difference between y and yhat (  default  )
  -> residuals         :  numerical, min =  -1 , mean =  0.05140765 , max =  1
  A new explainer has been created!
```

Deretter bruker vi `fairness_check()` med `Sex` som beskyttet variabel og `"Male"` som referansegruppe:

```r
fcheck <- fairness_check(
  explainer_rf,
  protected  = compas$Sex,
  privileged = "Male"
)
```

```
Creating fairness classification object
-> Privileged subgroup      : character ( Ok  )
-> Protected variable       : factor ( Ok  )
-> Cutoff values for explainers : 0.5 ( for all subgroups )
-> Fairness objects     : 0 objects
-> Checking explainers      : 1 in total (  compatible  )
-> Metric calculation       : 13/13 metrics calculated for all models
 Fairness object created succesfully
```

```r
plot(fcheck)
```



Plottet viser *parity loss* for en rekke ulike fairness-mål. Den grønne sonen angir et akseptabelt område basert på 4/5-regelen: forholdet mellom gruppen og referansegruppen bør ligge mellom 0.8 og 1.25. Søyler utenfor den grønne sonen betyr at modellen er statistisk sett *unfair* på det aktuelle målet.

Tre mål er særlig sentrale:

Mål i fairmodels	Hva måles	Konsept
`PPV`	Andelen sanne positive av alle predikerte positive	Predictive rate parity
`TPR`	Andelen sanne positive av alle faktisk positive (sensitivitet)	Equalized odds
`FPR`	Andelen falske positive av alle faktisk negative	False positive rate parity

Hvilket mål som er viktigst avhenger av hva modellen skal brukes til og hva konsekvensene av feil er. Et eksempel: Hvis modellen brukes til å vurdere prøveløslatelse, vil en høy FPR for én gruppe bety at den gruppen urettmessig beholder i fengsel mer enn andre grupper.

Vi kan også sjekke fairness for etnisitet ved å bytte ut den beskyttede variabelen:

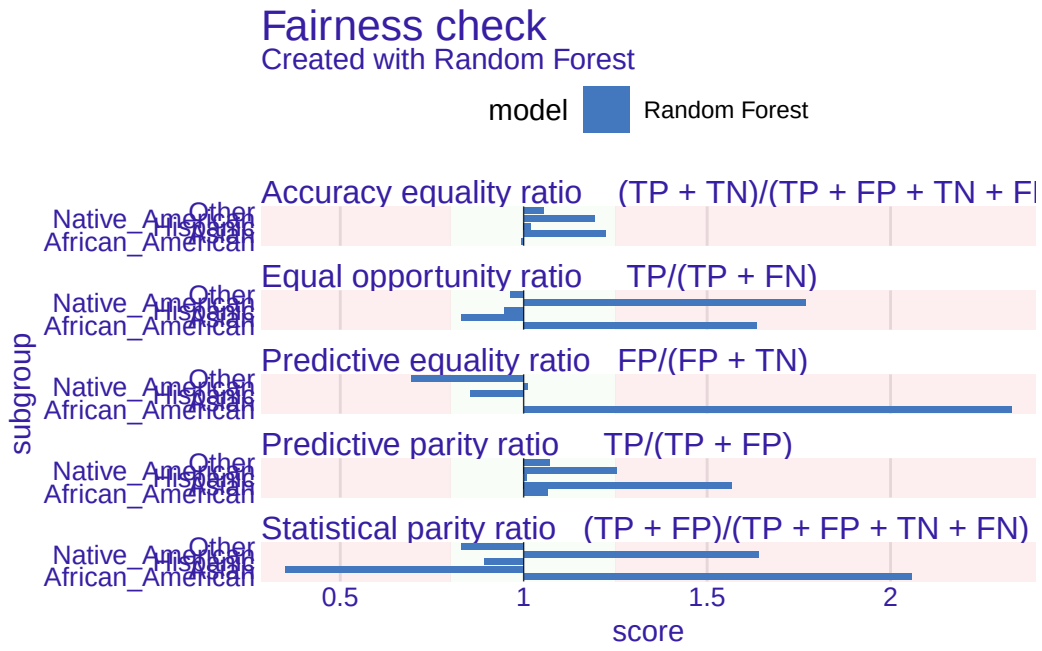
```
fcheck_etni <- fairness_check(
  explainer_rf,
  protected  = compas$Ethnicity,
  privileged = "Caucasian"
)
```

```
Creating fairness classification object
-> Privileged subgroup      : character ( Ok  )
-> Protected variable       : factor ( Ok  )
-> Cutoff values for explainers : 0.5 ( for all subgroups )
-> Fairness objects     : 0 objects
-> Checking explainers      : 1 in total (  compatible  )
-> Metric calculation       : 11/13 metrics calculated for all models ( 2 NA created )
 Fairness object created succesfully
```

```
plot(fcheck_etni)
```

```
Warning in plot.fairness_object(fcheck_etni): Omiting NA for models: Random Forest
Information about passed metrics may be inaccurate due to NA present, it is advisable to chec
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_bar()`).
```



9.5 Forbedre fairness: Stratifisering

Vi har sett at `sampsize` i random forest styrer fordelingen av treningsdata per utfallskategori. For å motvirke *bias* mellom undergrupper kan vi bruke *stratifisert sampling*: ved å trekke fra grupper i styrte proporsjoner kan vi justere modellens feilrater per gruppe.

`sampsize` i random forest styrer ikke bare fordelingen av utfall, men kan kombineres med `strata` for å stratifisere *etter en egendefinert variabel*. Stratifiseringsvektoren kombinerer kjønn og utfall til fire kategorier:

```
strat <- compas %>%
  mutate(strat = paste0(Sex, Two_yr_Recidivism)) %>%
  pull(strat)

table(strat)
```

```
strat
Female0 Female1   Male0   Male1
    762     413    2601    2396
```

Tallene i `sampsize` oppgis i samme rekkefølge som tabellen: kvinner uten tilbakefall, kvinner med tilbakefall, menn uten tilbakefall, menn med tilbakefall. Her vektes kvinner med tilbakefall opp relativt, for å øke modellens sensitivitet for denne gruppen:

```
set.seed(45)
rf_strat <- randomForest(Two_yr_Recidivism ~ .,
                         data     = compas,
                         strata   = strat,
                         sampsize = c(215, 290, 1500, 1500),
                         ntree    = 800)
```

Vi lager en explainer for den stratifiserte modellen og sammenligner med baseline ved å sende begge explainere til `fairness_check()`:

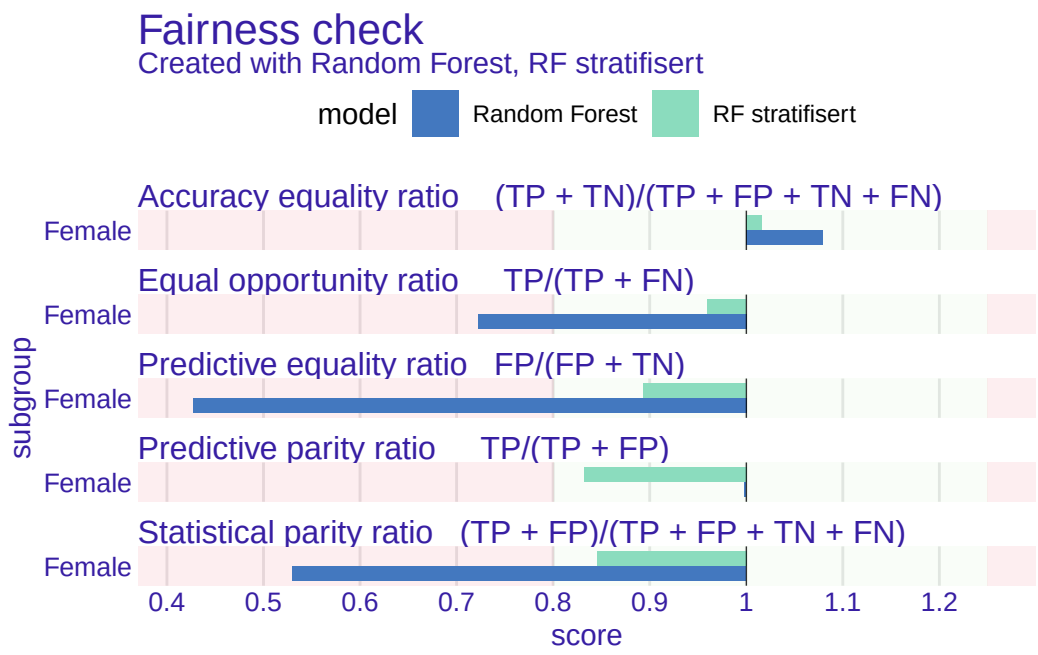
```
explainer_rf_strat <- DALEX::explain(
  model = rf_strat,
  data  = compas_X,
  y     = y_num,
  label = "RF stratifisert"
)
```

```
Preparation of a new explainer is initiated
  -> model label       :  RF stratifisert
  -> data              :  6172  rows  6  cols
  -> target variable   :  6172  values
  -> predict function  :  yhat.randomForest  will be used (  default  )
  -> predicted values  :  No value for predict function target column. (  default  )
  -> model_info        :  package randomForest , ver. 4.7.1.2 , task classification (  defaul
  -> predicted values  :  numerical, min =  0 , mean =  0.4916229 , max =  1
  -> residual function :  difference between y and yhat (  default  )
  -> residuals         :  numerical, min =  -1 , mean =  -0.03650296 , max =  1
  A new explainer has been created!
```

```r
fcheck2 <- fairness_check(
  explainer_rf,
  explainer_rf_strat,
  protected  = compas$Sex,
  privileged = "Male"
)
```

```
Creating fairness classification object
-> Privileged subgroup      : character ( Ok  )
-> Protected variable       : factor ( Ok  )
-> Cutoff values for explainers : 0.5 ( for all subgroups )
-> Fairness objects     : 0 objects
-> Checking explainers      : 2 in total (  compatible  )
-> Metric calculation       : 13/13 metrics calculated for all models
 Fairness object created succesfully
```

```r
plot(fcheck2)
```



Dette er en av de store fordelene med `fairmodels`: vi kan sammenligne flere modeller direkte i ett plot. Ble den stratifiserte modellen mer rettferdig enn baseline-modellen?

Merk at det ikke er noe åpenbart svar på hvilke verdier man skal velge for `sampsize`. Utgangspunktet er å sørge for at det trekkes omtrent like mange observasjoner fra grupper man ønsker å balansere. Det krever litt utprøving.

9.5.1 Hva med andre subgrupper?

Vi justerte for kjønn — men slo det positivt ut for etnisitet også?

```
fcheck2_etni <- fairness_check(
  explainer_rf,
  explainer_rf_strat,
  protected  = compas$Ethnicity,
  privileged = "Caucasian"
)
```

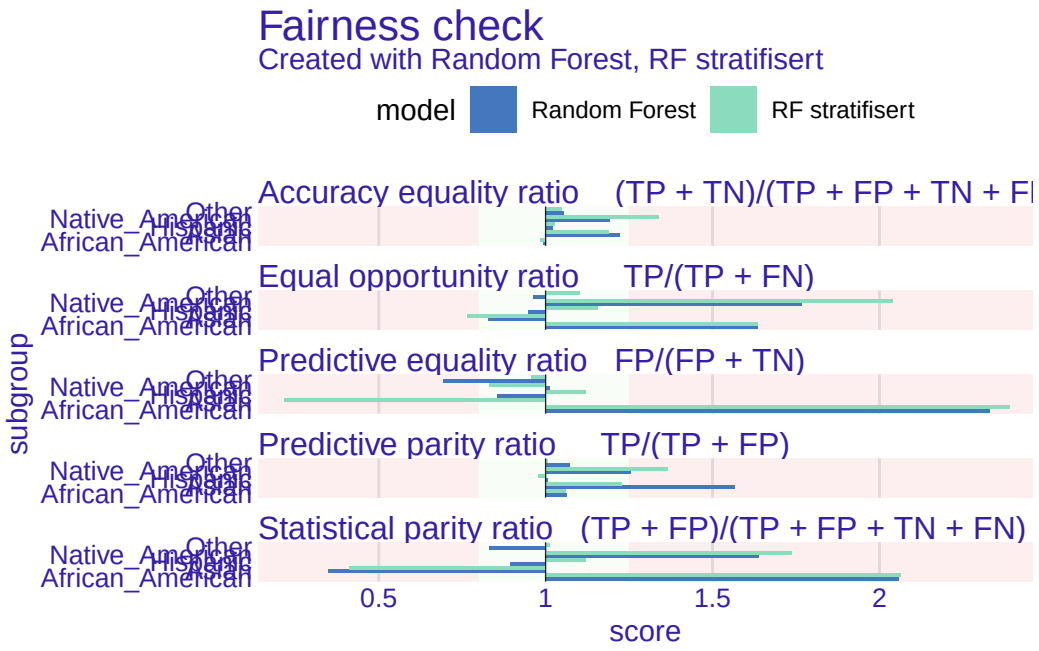
```
Creating fairness classification object
-> Privileged subgroup      : character ( Ok  )
-> Protected variable       : factor ( Ok  )
-> Cutoff values for explainers : 0.5 ( for all subgroups )
-> Fairness objects     : 0 objects
```

```
-> Checking explainers       : 2 in total (  compatible  )
-> Metric calculation        : 8/13 metrics calculated for all models ( 5 NA created )
 Fairness object created succesfully
```

```
plot(fcheck2_etni)
```

```
Warning in plot.fairness_object(fcheck2_etni): Omiting NA for models: Random Forest
Information about passed metrics may be inaccurate due to NA present, it is advisable to chec
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_bar()`).
```



Sannsynligvis ikke. Vi justerte for kjønn, ikke for etnisitet. En mulighet er å stratifisere på begge variable samtidig. Vi starter med å lage en kombinert stratifiseringsvektor:

```
strat2 <- compas %>%
  mutate(caucasian = ifelse(Ethnicity == "Caucasian", "Caucasian", "Other"),
         strat = paste0(Sex, caucasian, Two_yr_Recidivism)) %>%
  pull(strat)

table(strat2)
```

```
strat2
FemaleCaucasian0 FemaleCaucasian1     FemaleOther0     FemaleOther1
             312              170              450              243
  MaleCaucasian0   MaleCaucasian1       MaleOther0       MaleOther1
             969              652             1632             1744
```

Problemet her er at noen grupper er svært små. Å stratifisere på disse gir bare støy. Vi begrenser oss til å skille mellom "Caucasian" og "Other" fremfor alle etnisitetskategorier:

```r
set.seed(45)
rf_strat2 <- randomForest(Two_yr_Recidivism ~ .,
                          data     = compas,
                          strata   = strat2,
                          sampsize = c(120, 120,
                                       200, 200,
                                       400, 400,
                                       900, 900),
                          ntree    = 800)
```

```r
explainer_rf_strat2 <- DALEX::explain(
  model = rf_strat2,
  data  = compas_X,
  y     = y_num,
  label = "RF strat. kjønn+etni"
)
```

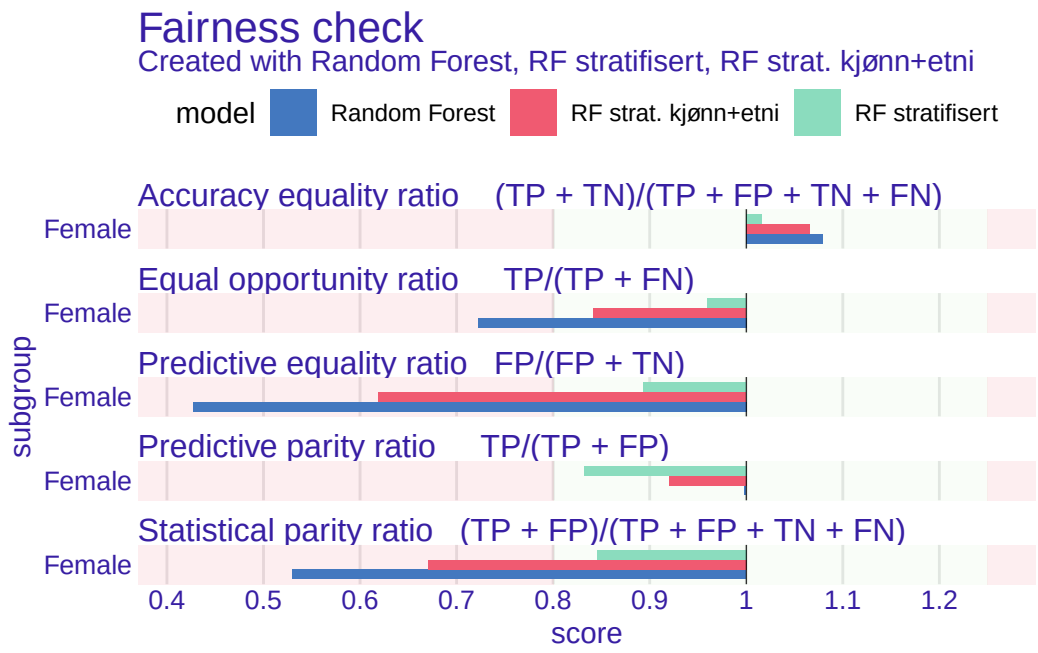
```
Preparation of a new explainer is initiated
  -> model label       :  RF strat. kjønn+etni
  -> data              :  6172  rows  6  cols
  -> target variable   :  6172  values
  -> predict function  :  yhat.randomForest  will be used (  default  )
  -> predicted values  :  No value for predict function target column. (  default  )
  -> model_info        :  package randomForest , ver. 4.7.1.2 , task classification (  defaul
  -> predicted values  :  numerical, min =  0 , mean =  0.4721089 , max =  1
  -> residual function :  difference between y and yhat (  default  )
  -> residuals         :  numerical, min =  -0.99875 , mean =  -0.01698902 , max =  1
  A new explainer has been created!
```

```r
fcheck3 <- fairness_check(
  explainer_rf,
  explainer_rf_strat,
```

```
  explainer_rf_strat2,
  protected  = compas$Sex,
  privileged = "Male"
)
```

```
Creating fairness classification object
-> Privileged subgroup      : character ( Ok  )
-> Protected variable       : factor ( Ok  )
-> Cutoff values for explainers : 0.5 ( for all subgroups )
-> Fairness objects     : 0 objects
-> Checking explainers      : 3 in total (  compatible  )
-> Metric calculation       : 13/13 metrics calculated for all models
 Fairness object created succesfully
```

```
plot(fcheck3)
```



Sammenlign de tre modellene. Man kan justere for noe, men ikke alt samtidig. Med mer data kan man justere for *mer*, men man er alltid begrenset av datamaterialet.

9.6 Justere rettferdighet med vekting

En annen strategi er å *vekte observasjoner* i treningsfasen. Dette fungerer godt i boosting, der vektene direkte påvirker hva algoritmen fokuserer på. Vektingen er en måte å fortelle modellen at noen typer feil er viktigere enn andre.

`gbm` krever at utfallsvariabelen er numerisk (0/1) med `distribution = "bernoulli"`. Vi lager en kopi av datasettet med konvertert utfall:

```
compas_gbm <- compas %>%
  mutate(Two_yr_Recidivism = as.numeric(as.character(Two_yr_Recidivism)))
```

Deretter lager vi en vektvektor der kvinner og observasjoner med tilbakefall vektes opp. Tanken er at modellen skal ta disse tilfellene mer alvorlig under trening:

```
wts <- compas_gbm %>%
  mutate(wts = case_when(
    Sex == "Male"   & Two_yr_Recidivism == 0 ~ 1,
    Sex == "Male"   & Two_yr_Recidivism == 1 ~ 2,
    Sex == "Female" & Two_yr_Recidivism == 0 ~ 2,
    Sex == "Female" & Two_yr_Recidivism == 1 ~ 4)) %>%
  pull(wts)
```

`case_when()` setter én vektverdi per observasjon basert på kombinasjonen av kjønn og utfall, og `pull()` trekker ut vektene som en vektor. Kvinner med tilbakefall vektes fire ganger så mye som menn uten tilbakefall.

```
set.seed(542)
gradboost <- gbm(formula = Two_yr_Recidivism ~ .,
                 data         = compas_gbm,
                 weights      = wts,
                 distribution = "bernoulli",
                 n.trees      = 4000,
                 interaction.depth  = 3,
                 n.minobsinnode = 1,
                 shrinkage    = 0.001,
                 bag.fraction = 0.5)
```

For å bruke `gbm` med `DALEX::explain()` må vi angi en tilpasset prediksjonsfunksjon, fordi `gbm` trenger `n.trees` eksplisitt:

```r
predict_gbm <- function(model, newdata) {
  predict(model, newdata = newdata,
          n.trees = model$n.trees, type = "response")
}

compas_gbm_X <- compas_gbm %>% select(-Two_yr_Recidivism)

explainer_gbm <- DALEX::explain(
  model            = gradboost,
  data             = compas_gbm_X,
  y                = compas_gbm$Two_yr_Recidivism,
  predict_function = predict_gbm,
  label            = "GBM med vekter"
)
```

```
Preparation of a new explainer is initiated
  -> model label       :  GBM med vekter
  -> data              :  6172  rows  6  cols
  -> target variable   :  6172  values
  -> predict function  :  predict_gbm
  -> predicted values  :  No value for predict function target column. (  default  )
  -> model_info        :  package gbm , ver. 2.2.2 , task classification (  default  )
  -> predicted values  :  numerical, min =  0.2275137 , mean =  0.599534 , max =  0.8853893
  -> residual function :  difference between y and yhat (  default  )
  -> residuals         :  numerical, min =  -0.8737405 , mean =  -0.1444141 , max =  0.74172
  A new explainer has been created!
```

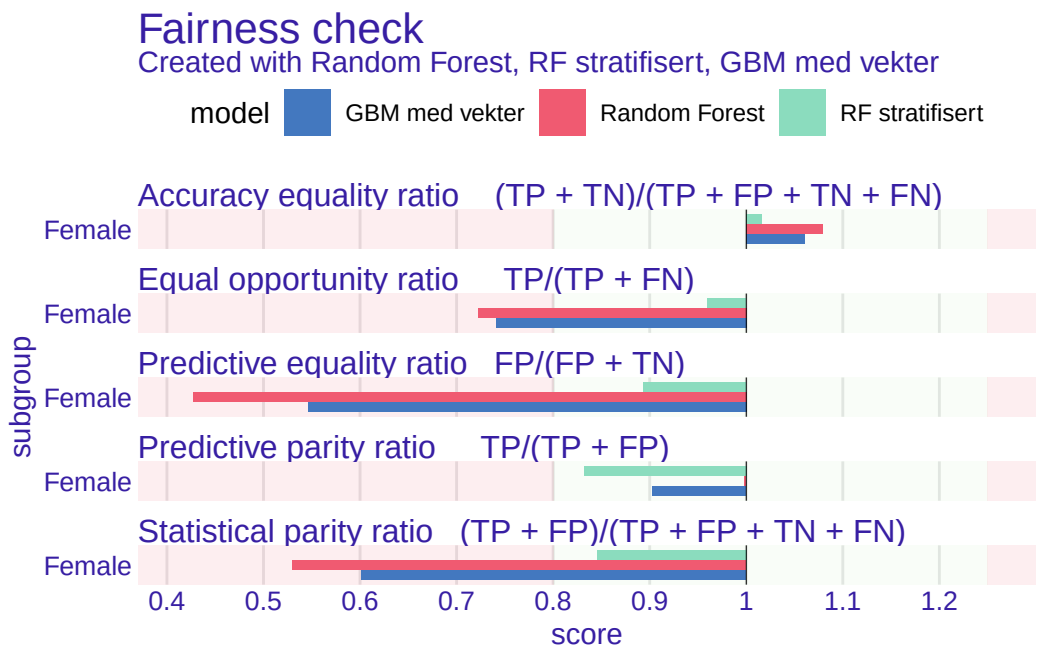
Nå kan vi sammenligne alle tre modellene i ett fairness-kall:

```r
fcheck_all <- fairness_check(
  explainer_rf,
  explainer_rf_strat,
  explainer_gbm,
  protected  = compas$Sex,
  privileged = "Male"
)
```

```
Creating fairness classification object
-> Privileged subgroup      : character ( Ok  )
-> Protected variable       : factor ( Ok  )
-> Cutoff values for explainers : 0.5 ( for all subgroups )
```

```
-> Fairness objects     : 0 objects
-> Checking explainers      : 3 in total (  compatible  )
-> Metric calculation       : 13/13 metrics calculated for all models
 Fairness object created succesfully
```

```
plot(fcheck_all)
```



For en kompakt oversikt over alle modeller og alle mål kan vi bruke `fairness_heatmap()`:

```
fairness_heatmap(fcheck_all)
```

```
heatmap data top rows:
  parity_loss_metric           model score
1                TPR   Random Forest  0.33
2                TPR RF stratifisert  0.04
3                TPR  GBM med vekter  0.30
4                TNR   Random Forest  0.21
5                TNR RF stratifisert  0.05

matrix model not scaled :
                      TPR        TNR         PPV        NPV        FNR
Random Forest   0.32585661 0.21473973 0.001931638 0.09351694 0.41239929
```

```
RF stratifisert 0.04097714 0.05222386 0.183665525 0.12662643 0.08413193
GBM med vekter  0.29995238 0.48139816 0.101917421 0.02585717 0.90244226
                      FPR         FDR        FOR        TS       STP        ACC
Random Forest   0.8504339 0.003894726 0.23922716 0.2571368 0.6344638 0.07622932
RF stratifisert 0.1125916 0.275123636 0.37186916 0.1704876 0.1678504 0.01579906
GBM med vekter  0.6048768 0.123561862 0.08350239 0.2719322 0.5085737 0.05840212
                       F1 NEW_METRIC
Random Forest   0.1796063  0.7382559
RF stratifisert 0.1165885  0.1251091
GBM med vekter  0.1866799  1.2023946
```

Sammenlign modellene. Ble den vektede boosting-modellen mer rettferdig enn random forest? Husk at det alltid er en trade-off: økt rettferdighet for én gruppe eller på ett mål kan gå på bekostning av total presisjon eller rettferdighet på et annet mål.

9.7 Avsluttende betraktninger

Hvilket fairness-mål som er viktigst er det ikke noe klart svar på. Det kommer an på hva man har tenkt å bruke prediksjonen til, hvilke konsekvenser tiltaket har, og hvilke konsekvenser det har å ikke gjøre noe. Disse konsekvensene kan vurderes forskjellig for ulike personer, grupper og situasjoner.

Problemet er at det ikke går an å justere i det uendelige for alle variable. I tillegg kan det uansett være andre egenskaper man ikke har data for som vil vise seg å gi urettferdige utslag. Man kan justere for *noe*, men ikke alt. Med nok data kan man justere for *mer* — men fremdeles ikke alt. I tillegg er justeringer som regel ikke gratis: økt rettferdighet for én gruppe kan innebære lavere presisjon totalt sett, eller redusert rettferdighet for en annen gruppe.

Så da er vi tilbake til det litt ubehagelige gjennomgangstemaet om prioriteringer og valg: Vil du ha en rettferdig modell eller en presis modell? Hvilke grupper bør den være rettferdig for — og hvilke grupper er det ikke så farlig om den er urettferdig for? Dette kan lett bli umulige valg.

Bør det nevnes at noen ganger ville man gjort noe uansett — altså gjort de samme tiltakene basert på skjønn eller andre typer vurderinger. Det vil også ha feilrater og forskjeller mellom undergrupper, selv om man ikke har et oppsett som gir testing-data å estimere dette på. Det betyr at du ikke bare må ta stilling til om algoritmen er "fair" eller ikke, men også om den er *mer eller mindre "fair"* enn den alternative fremgangsmåten.

References

Berk, Richard. 2016. *Statistical Learning from a Regression Perspective*. USA: Springer.

Hsieh, John. 2008. "Receiver Operating Characteristic (ROC) Curve." In *Encyclopedia of Epidemiology*, 895–98. Thousand Oaks, California: Sage. https://doi.org/10.4135/9781412953948.