IST[A/G/T]1003: Statistisk læring og data science

Klassifikasjon

Mette Langaas IMF/NTNU

20 November, 2020

Contents

Læringsmål

Etter du har lest dette kompendiet, sett videoene som er laget og deltatt på zoom-forelesning skal du

- kunne forstå hva klassifikasjon går ut på og
- kjenne igjen situasjoner der klassifikasjon vil være en aktuell metode å bruke
- kjenne begrepene treningssett, valideringssett og testsett og forstå hvorfor vi lager dem og hva de skal brukes til
- vite hva en forvirringsmatrise er, og kjenne til begrepene feilrate og nøyaktighet (accuracy),
- forstå tankegangen bak k-nærmeste nabo-klassifikasjon, og hvordan k kan velges
- kjenne til modellen for logistisk regresjon, og kunne tolke de estimerte koeffisientene
- forstå hvordan vi utfører klassifikasjon med k-nærmeste nabo og logistisk regresjon i Python
- kunne svare godt på klassifikasjonsoppgaven i den tellende prosjektoppgaven

Introduksjon

Videoressurser

Videoer som diskuterer innholdet av dette kompendiet:

- Klassifikasjon - introduksjon og k-nærmeste-nabo-klassifikasjon (10:58 min)
- Klassifikasjon - logistisk regresjon (14:17 min)

Målet med klassifikasjon

Målet med klassifikasjon er å tilordne en ny observasjon til en av flere kjente klasser. Tilordningsregelen lager vi basert på data fra mange observasjoner, der vi kjenner den sanne klassen til hver observasjon og har med såkalte forklaringsvariabler - som er kjente egenskaper til observasjonen.

Klassifikasjon er veiledet læring.

På samme måte som for regresjon har vi observasjon av forklaringsvariabler $\mathbf{x}$ sammen med respons y. Det som skiller klassifikasjon fra regresjon er at nå er ikke responsen kontinuerlig - men heller *kategorisk*.

En kategorisk variabel kan ta verdi fra ulike kategorier, vi skal se på vindusglass av type float eller ikke-float (to kategorier). Andre eksempler på en kategorisk variabel er blodtype og sivilstand.

Vi skal se på k-nærmeste-nabo-klassifikasjon, en ikke-parametrisk metode (det er ikke noen parametere å estimere) og på logistisk regresjon, som er en såkalt parametrisk metode (vi skal estimere ukjente parametere som inngår i en matematisk modell).

Eksempler på klassifikasjonsproblemer

- Er en epost en søppelepost (spam) eller ikke? Data: ord i eposten, avsender.
- Hvilke kunder kommer ikke til å betale tilbake banklånet sitt? (default=misligholde) Data: inntekt, kontobeholdning, alder, utdanning, ...
- Hva er risikofaktorer for at en pasient får en spesiell sykdom? Data: alder, kjønn, røyk, alkohol, overvekt, andre sykdommer, ...
- Er det noen tilstede i et rom? Data fra sensorer som registrerer CO_2, fuktighet, lysforhold, ... https://github.com/LuisM78/Occupancy-detection-data
- Hva skiller tre pingvin-raser? https://education.rstudio.com/blog/2020/07/palmerpenguins-cran/
- Kan man ut fra kjemiske bestanddeler finne ut hvilken type glass som er funnet på et åsted? Tenk CSI!
- Hvem vant tennismatchen? Prosjektoppgaven 2020 oppgave 2.

Eksempel: Glass på åstedet

Vi skal se på en variant av et datasett som inneholder målinger av vektandeler av oksider og brytningsindeks for ulike typer glass. Studiet av glass er motivert av kriminologiske undersøker. Hvis man på et åsted finner glasskår kan de brukes som bevis - gitt at glasstypen kan identifiseres korrekt.

Ved kjemiske analyser har man for hver observasjon i datasettet målt

- vektprosent aluminimumoksid (Al)
- brytningsindeks (RI).

Brytningsindeks er et mål på et materials evne til å lede lys.

Dette er våre to *forklaringsvariabler*. (I det originale datasettet er det flere forklaringsvariabler, men vi ser bare på to for å forenkle presentasjonen.)

Vi skal se på to glasstyper:

- vindusglass av typen float (en spesiell produksjonsprosess) og
- vindusglass av typen ikke-float,

Glasstypen er vår *respons*.

Målet vårt er å lage en regel for å klassifisere et glasskår som vidusglass av type float eller ikke-float basert på vektprosent av oksider og målt brytningsindeks. Vi ønsker også at metoden skal gi oss en *sannsynlighet* for at en (fremtidig) observasjon er av type float eller ikke-float (ikke bare beste gjett på klassen). Hvis vi klarer å lage en regel som bare er basert på forkaringsvariablene, så kan den brukes på et glasskår fra et fremtidig åsted.

Trening-, validering- og testsett

Vi ønsker å lage en regel for å klassifisere en observasjon til en av to (eller flere) klasser, pluss at vi også ønsker en sannsynlighet for at observasjonen tilhører hver av klassene. Fokus er å lage en regel som skal kunne brukes i fremtiden - ikke bare på akkurat det datasettet vi nå har.

Vi har fokus på prediksjon. Det finnes mange veldig fleksible klassifikasjonsmetoder og vi er litt bekymret for at vi skal lage en regel som passer alt for godt til dataene våre - og ikke generaliserer til nye data.

For å ta vare på alt dette vil vi dele opp datasettet vårt inn i tre datasett, som brukes i ulike deler av analysen.

1) Treningssettet
2) Valideringssettet
3) Testsettet

Treningssettet bruker vi til å estimere modellparametere (for eksempel regresjonsparametere) eller det lager regelen (som for k-nærmeste nabo) - slik at modellen vår passer til data. Dette kan godt være rundt 60-80% av dataene vi har tilgjengelig.

Valideringssettet bruker vi til å gjøre modellvalg, for eksempel hvilke forklaringsvaribler skal være med i modellen. Det kan også brukes til å bestemme verdi for *hyperparametere*. Vi skal se på en metode der klassen til en ny observasjon bestemmes av klassen til observasjonene som er nærmest -vi kaller disse naboer - og da må vi bestemme hvor mange observasjoner som er naboer - det er en hyperparameter. Valideringssettet kan være på rundt 20% av dataene vi har tilgjengelig.

Hvis man har et lite datasett ønsker man å slå sammen treningssettet og valideringsettet og heller bruke noe som heter *kryssvalidering* - det gjør man for å kunne bruke data så effektivt som mulig. Vi skal ikke se på kryssvalidering her.

Testsettet bruker vi kun etter at vi har bestemt alle modellparametere og hyperparameter, og estimert (lært) en modell på treningssettet. Da tar vi frem testsettet og bruke det til å rapportere hvor god regelen vår er på helt nye data som metoden ikke har sett før. Testsettet kan være på rundt 20% av dataene vi har tilgjengelig. Det er viktig at testsettet "låses ned i en bankboks" når vi starter med å estimere (lære) modellen ved bruke av trening- og valideringssettet, og at det tas fram når trening og validering av metoden (regelen) vår er avsluttet.

Det er hvordan metoden fungerer (performance) på *nye data* som er gullstandarden! Hvordan metoden fungerer på *treningssettet* er mindre interessant. Dette er fordi at hvis man har en veldig fleksibel metode

(for eksempel hvis vi bare ser på en nabo, eller hvis vi tilpasser et niendegradspolynom til 10 observasjoner) så vil det være mulig å tilpasse modellen nesten perfekt til treningsdatane. Dette kalles *overtilpasning*. Det å velge "riktig" fleksibilitet til en metode er koblet til det som kalles "bias-variance-tradeoff".

Det er viktig at de tilgjengelig dataene deles tilfeldig inn i de tre settene, og vi passer på at andelen av de ulike klassene er de samme i alle de tre settene. Det siste kan gjøres ved såkalt *stratifisert oppdeling*.

Testsettet brukes også til å sammenligne hvor gode ulike metoder (vi skal se på logistisk regresjon og k-næremeste-nabo-klassifikasjon) er for å løse klassifikasjonsproblemet.

Eksempel: Oppdeling av glass-datasettet inn i trening, validering og testsett

```
import pandas as pd
from sklearn.model_selection import train_test_split

pydf=pd.read_csv("glass2org.csv")
df_trenval, df_test = train_test_split(pydf, test_size = 0.2,random_state=0,stratify=pydf['y'])
df_tren, df_val = train_test_split(df_trenval, test_size = 0.25,random_state=0,stratify=df_trenval['y'])
print("tren: ", df_tren.shape)
```

```
## tren:  (87, 4)
```

```
print("val: ",df_val.shape)
```

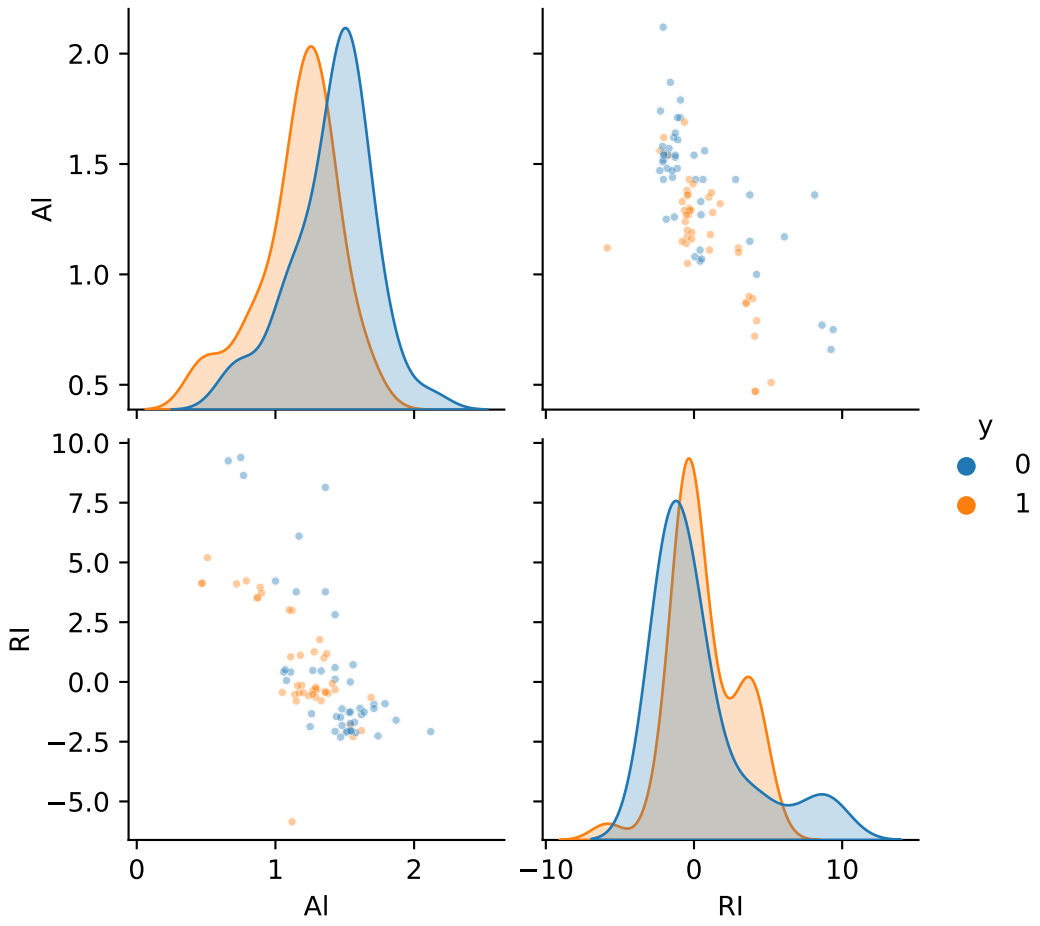
```
## val:  (29, 4)
```

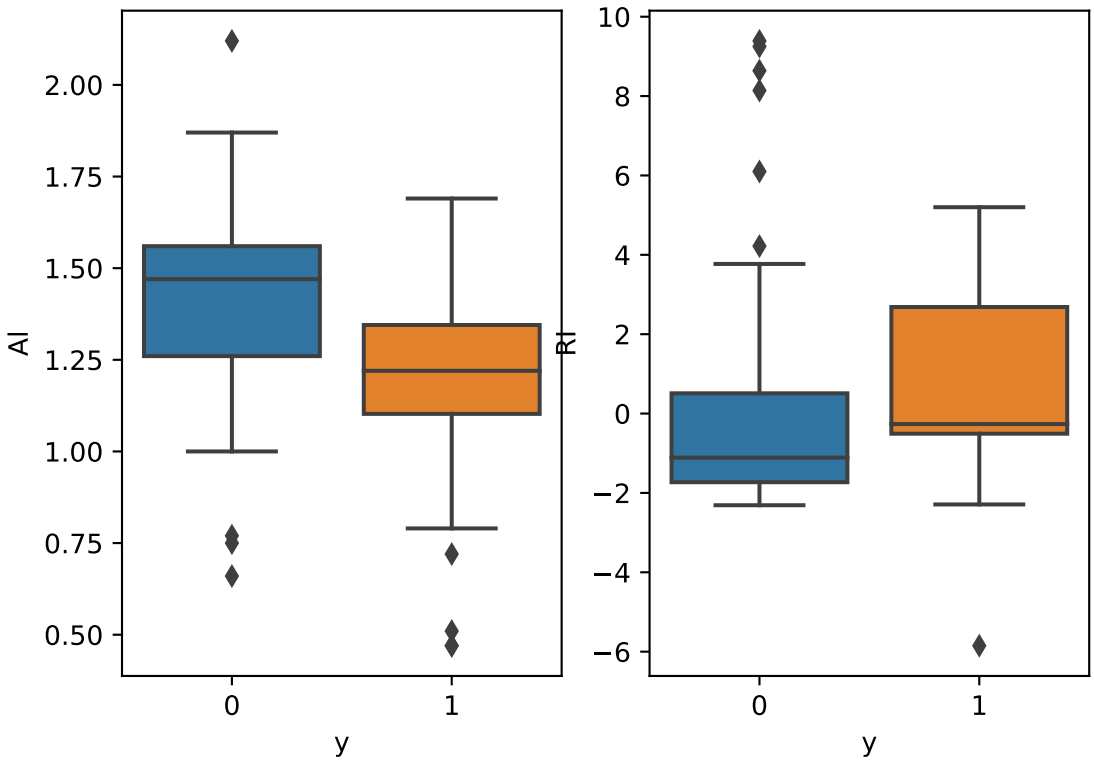
```
print("test: ",df_test.shape)
```

```
## test:  (30, 4)
```

Slik ser dataene ut for treningssettet.

```
## <seaborn.axisgrid.PairGrid object at 0x12a28e208>
```


AI
RI
y
0
1
2.0
1.5
1.0
0.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
1
2
−10
0
10



Fra boksplottene ser vi ser på at medianen for flere av forklaringsvariablene er ulike for float (1) og ikke-float (0) vindusglass. Det betyr at det er håp om å kunne å lage en god klassifikasjonsregel basert på forklaringsvariablene. I tillegg ser vi fra kryssplotten at kanskje det kan være lurt å bruke Al og RI sammen som forklaringsvariabler, fordi da ser det ut som vår kan skille de to klassene ganske bra (blå og orange punkter). Dette må vi finne ut mer om!

k-nærmeste-nabo-klassifikasjon (kNN)

Hvis vi har to klasser, koder vi gjerne responsen Y som 0 eller 1, mens er det flere enn to klasser - for eksempel M klasser - bruker vi 1, 2, ... M. I algoritmen under skriver vi $Y = j$ for en generell klasse j.

Algoritmen

Anta at vi nå har delt dataene inn trenings-, valideringss- og testsett, og nå skal vi bruke treningsdataene. kNN klassifikasjon er en svært intuitiv metode, som algoritmisk har følgende steg:

1) Se på en ny observasjon med forklaringsvariabler $\mathbf{x}_0$. Hvilken klasse bør den klassifiseres til?
2) Finn de k nærmeste naboene til $\mathbf{x}_0$ i treningssettet. Indeks til disse putter vi inn i settet $\mathcal{N}_0$.
3) Sannsynligheten for at den nye observasjonen er av klasse $Y = j$ sier vi er andelen av de k nærmeste naboene som har klasse j. Matematisk

$$P(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j).$$

der I er indikatorfunksjonen som er 1 hvis klassen til naboen er j og 0 ellers.

4) Klassen til den nye observasjonen er den som har størst sannsynlighet. Det blir det samme som å klassifisere til klassen som de fleste av de k naboene tilhører (dette kaller vi flertallsavstemming, majority vote).

Dette er en ikke-parameterisk metode, og vi skal se under at k bestemmer hvor fleksibel metoden er.

Hva betyr nærmest?

Nærmest er definert utifra at vi bruker euklidsk avstand. Den euklidske avstanden mellom en observasjon $\mathbf{x}_i$ og den nye observasjonen $\mathbf{x}_0$ er

$$\sqrt{\sum_{j=1}^{p}(x_{ij} - x_{0j})^2}$$

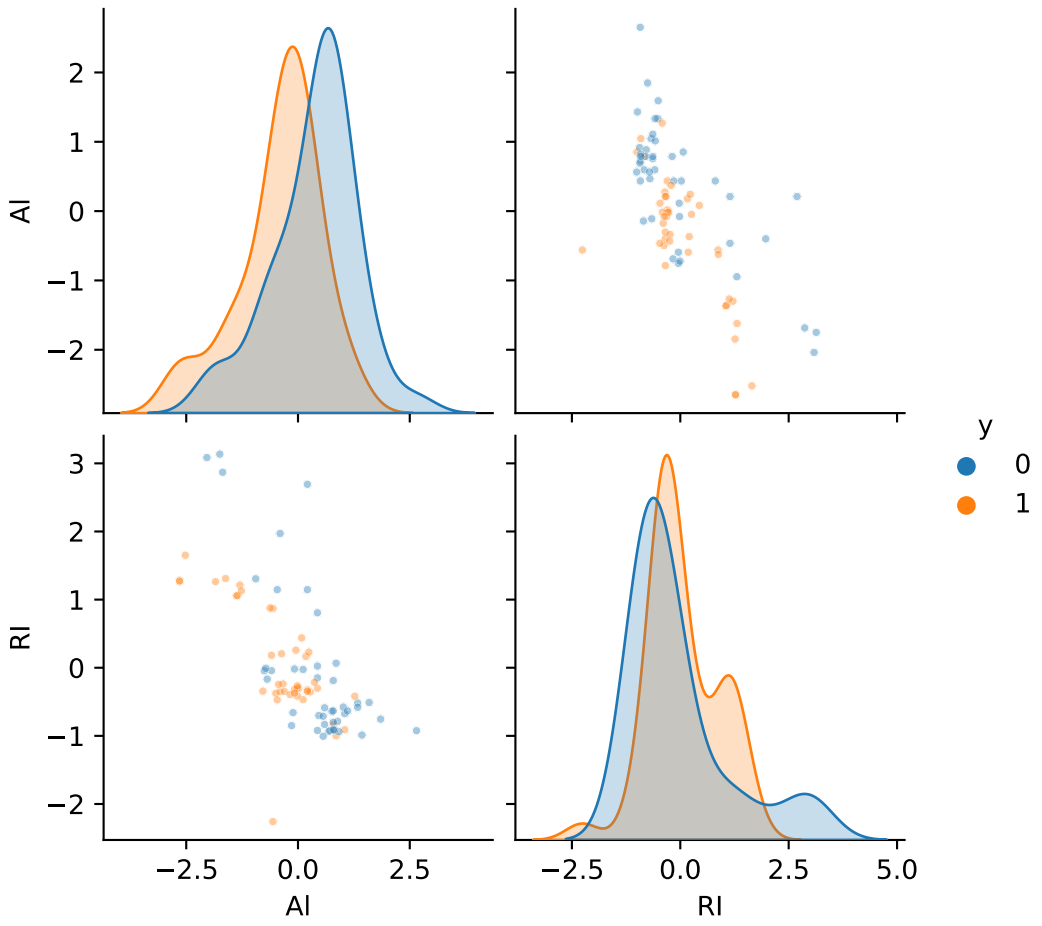
Her er x_{ij} målingen for observasjon i av variabel j, og x_{0j} målingen for den nye observasjonen $\mathbf{x}_0$ for variabel j. Her har vi totalt p forklaringsvariabler.

Hvis vi har to forklaringsvaribler kan vi tenke oss at vi slår en sirkel rundt $\mathbf{x}_0$ - med så stor radius at akkurat k naboer er inne i sirkelen. Radiusen vil være forskjellig for ulike observasjoner $\mathbf{x}_0$. For tre forklaringsvariabler blir det en kule og i høyere dimensjoner en hypersfære.

Dette betyr at det er viktig at forklaringsvariblene er på en slik skala at eukidsk avstand gir en fornuftig avstand. For eksemplet med glasskår har brytningsindeks RI et større verdiområde enn vektandel aluminiumsoksid Al, og da vil observasjoner med veldig ulike RI ligge langt fra hverandre. Det vil ikke være en god egenskap når vi skal bruke kNN og vi velger å standardisere forklaringsvariablene før vi bruker kNN. Standardisere vil for hver forklaringsvariable si å trekke fra gjennomsnittsverdi for treningsdataene og dele på det emipiriske standardavviket for treningsdataene. Alternativt kunne vi gjort dette ved å bruke en avstandsfunksjon der vi vektet de ulike forklaringsvariablene med sitt empiriske standardavvik i treningssettet.

Eksempel: standardiserte forklaringsvariabler for glass til bruk med kNN

```
## <seaborn.axisgrid.PairGrid object at 0x12de80dd8>
```



Hvilke verdier kan vi velge for k?

I kNN kalles k for en *hyperparameter* - det er en parameter vi må sette, og som ikke vi kan estimeres fra treningsdataene.

For k kan vi velge alle verdier fra 1 til n (antall observasjoner i treningssettet).

For et problem med to klasser er det vanlig å velge k til å være oddetall. Hvorfor det? Vi ønsker å unngå at halvparten av naboene er fra den ene klassen og halvparten fra den andre klassen - for da er det vanskelig med flertallsavstemming.

Forvirringsmatrise og feilrate

Med kNN har vi laget en klassifikasjonsregel, der vi lar de k naboene ta en flertallsavstemming for å bestemme klassen til en ny observasjon.

Hvor mye feil gjør vi ved å bruke klassifikasjonsregelen?

Vi skal her ha fokus på spesialtilfellet at vi har to klasser, som er kodet 0 og 1 (som i glasskåreksemplet vårt).

For hver observasjon vil vi klassifisere til enten klasse 0 eller 1, og vi kan gjøre riktig og galt på fire måter.

Vi kan si at en observasjon tilhører:

- klasse 1 når den i sannhet tilhører klasse 1: det er en sann positiv (true positive TP) beslutning
- klasse 1 når den i sannhet tilhører klasse 0: det kaller vi en falsk positiv (FP) beslutning

- klasse 0 når den i sannhet tilhører klasse 0: det er en sann negativ (true negative TN) beslutning
- klasse 0 når den i sannhet tilhører klasse 1: det kaller vi en falsk negativ (FN) beslutning

Hvis vi gjør dette for hver observasjon i et datasett (vi skal bruke valideringssettet) og putter tallene på følgende måte inn i en matrise, har vi laget en *forvirringsmatrise*.

	Sann klasse 0	Sann klasse 1	Totalt
Predikert klasse 0	Sann negativ (TN)	Falsk negativ (FN)	totalt predikert som klasse 0 (N)
Predikert klasse 1	Falsk positiv (FP)	Sann positiv (TP)	totalt predikert som klasse 1 (P)
Totalt	totalt antall sanne klasse 0	totalt antall sanne klasse 1	totalt antall

Ut fra forvirringsmatrisen kan vi definere veldig mange godhetsmål, statistikerne har sine favoritter og maskinlærerne sine (se under), men vi skal ha fokus på *feilrater*.

Feilraten for klasse 1 er hvor mange observasjoner der sannheten er at de tilhører klasse 1 men de er feilklassifisert til klasse 0: FN/(totalt sanne klasse 1)

Feilraten for klasse 0 er hvor mange observasjoner der sannheten er at de tilhører klasse 0 men de er feilklassifisert til klasse 1: FP/ (totalt sanne klasse 0)

Den *totale feilraten er da (FN+FP)/(totalt antall).*

Videre - til informasjon (men vi skal ikke bruke dette):

- Andelen riktige klassifikasjoner heter nøyaktighet (accuracy): (TN+TP)/(totalt antall).
- Presisjon (precision): TP/(P: totalt predikert som klasse 1). Statistikerne kaller dette positiv prediktiv verdi (PPV).
- Sensitivitet: andel observasjoner fra klasse 1 som er korrekt klassifisert: TP/(totalt antall sanne klasse 1), kalles "recall" innen maskinlæring.
- Spesifisitet: andel observasjoner fra klasse 0 som er korrekt klassifisert: TN/(totalt antall sanne klasse 0)
- F1-score: 2· (presisjon·recall)/(presisjon+recall).
- Reciever-operator kurve (ROC): sett ulike cut-off p, og regn ut sensitivitet og spesifisitet for hvert cut-off. Plott 1−spesifisitet mot sensitivitet.
- Areal under ROC (AUC): Arealet under ROC-kurven
- Presisjon-recall curve (PR): sett ulike cut-off p, og regn ut presisjon (PPV) og recall (sensitivitet) for hvert cut-off. Plott sensitivitet mot presisjon.

Vi kan definere forvirringsmatrisen på tilsvarende måte for mer enn to klasser, da blir forvirringsmatrisen $M \times M$ for M klasser. Vi kan også definere feilraten som antall gale klassifikasjoner delt på antall observasjoner som skal klassifiseres, men begrepene sanne og falske positive og negative er det litt mer arbeid å finne generalisering for.

Eksempel: Forvirringsmatrise og feilrate for glassskår kNN

La de nye dataene vi skal klassifisere være observasjonene i *valideringssettet*, og fyller inn i forvirringsmatrisen for $k = 1$.

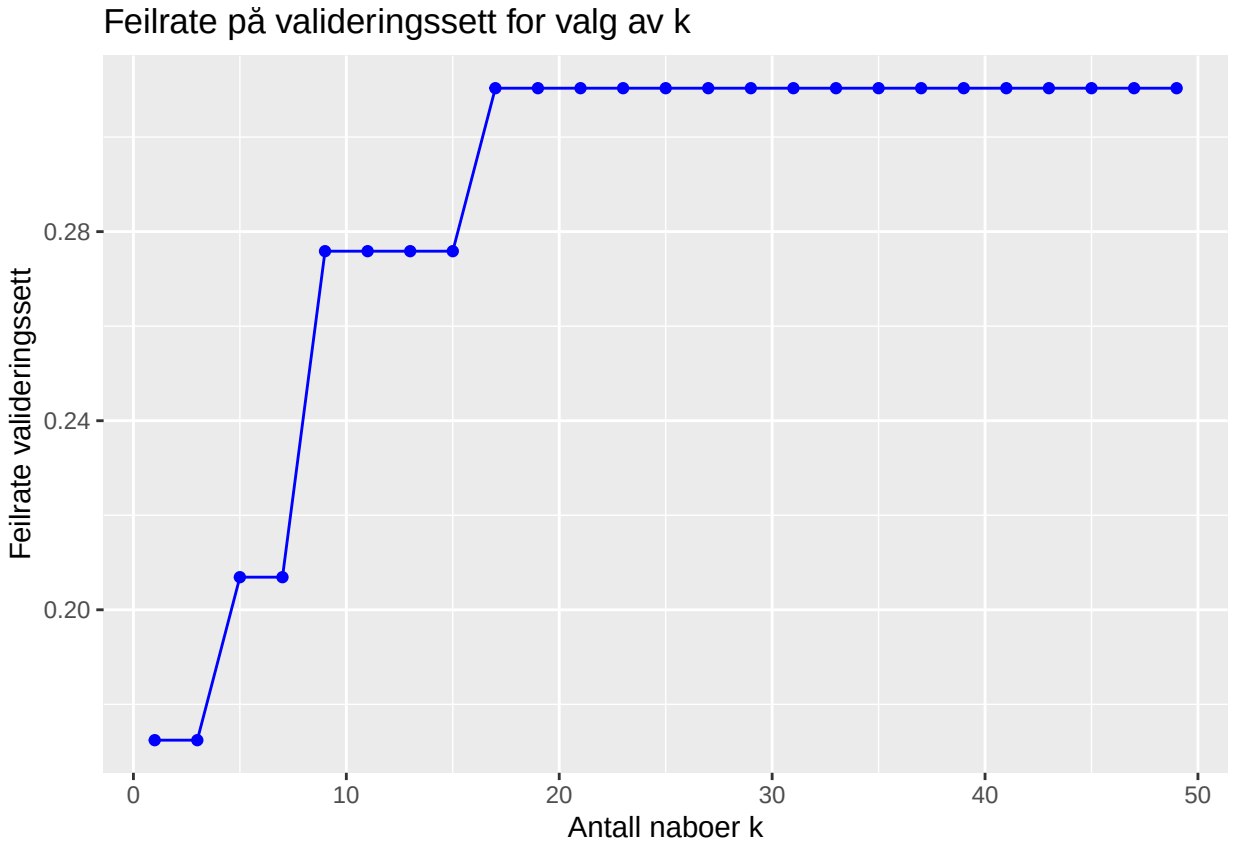
	Sann 0 (ikke-float)	Sann 1 (float)	Totalt
Predikert 0 (ikke-float)	11	4	15
Predikert 1 (float)	1	13	14
Totalt	12	17	29

Andelen korrekte klassifikasjoner (nøyaktighet=accuracy) og blir (11 + 13)/29=0.83. Det motsatte er andelen feil: det er da først teller: 4 + 1 feile klassifikasjoner, slik at *feilraten* er (4 + 1)/29=0.17. som også er 1 minus nøyaktigheten.

Hvordan skal vi velge k i kNN?

Vi velger den k som gir lavest feilrate på *valideringssettet*.

Vi lar k være alle oddetall fra 1 til 49. I Python kan vi regne ut feilraten på valideringssettet og plotter som en funksjon av k.

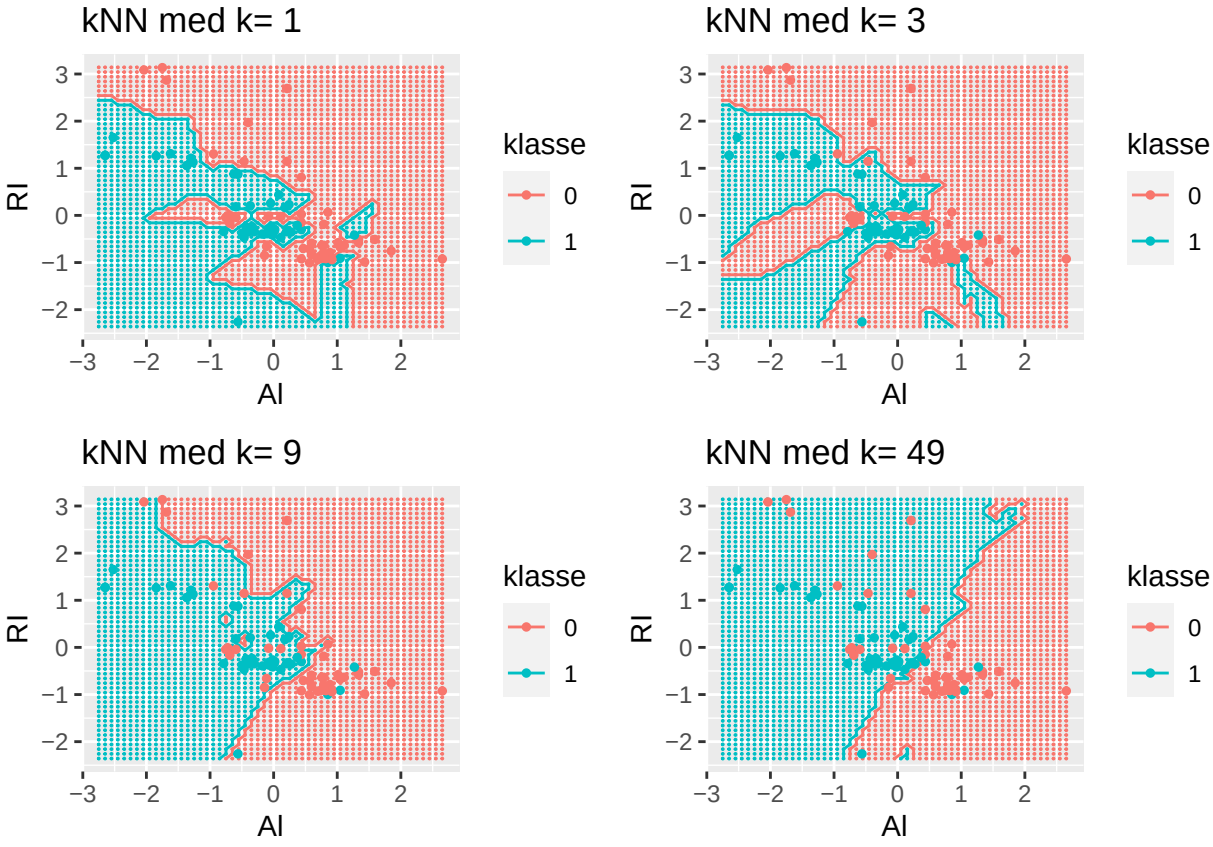



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np

df_trenS=pd.read_csv("glassdftrenS.csv")
df_valS=pd.read_csv("glassdfvalS.csv")

knaboer = np.arange(1,50,step=2)
val_feilrate = np.empty(len(knaboer))

X_tren=df_trenS[['Al','RI']] #bruker bare disse to forklaringsvariablene
X_val=df_valS[['Al','RI']]

for i,k in enumerate(knaboer):

#Initialiser kNN med  k neighbors
    knn = KNeighborsClassifier(n_neighbors=k,p=2) # p=2 gir euklidsk avstand
```

```
# Tilpass modellen med treningssettet
    knn.fit(X_tren, df_trenS['y'])

# Beregn feilrate på valideringssett
# score er accuracy= "andel korrekt"
    val_feilrate[i] = 1-knn.score(X_val, df_valS['y'])

plt.title('k-NN for ulike verdier av antall naboer k')
plt.plot(knaboer, val_feilrate, label='Feilrate på valideringssettet')
plt.legend()
plt.xlabel('Antall naboer k')
plt.ylabel('Feilrate')
plt.show()

mink_valfeilrate = knaboer[np.where(val_feilrate == val_feilrate.min())]
print(mink_valfeilrate[0])
```

Her ser vi at det er lik feilrate for $k = (1, 3)$, så bare ved bruk av valideringssettet kan vi ikke skille mellom disse valgene av k. Noe av grunnen til dette er nok at vi har et *veldig lite valideringssett*. (Det ville i vårt tilfelle vært gunstigere å bruke det som heter kryss-validering, der vi deler opp i treningssett og valideringssett mange ganger og baserer avgjørelsen vår på en sum over alle oppdelingen.) Men, vi må velge enten $k = 1$ eller $k = 3$.

Vi kan lage oss et grid av verdier for Al og RI og predikere klassen til hvert punkt. Deretter kan vi plotte hva klassifikasjonen ville være i hvert gridpunkt. I tillegg plotter vi dataene i treningssettet. Under finner vi plott for k: 1 og 3, og så har vi tatt med $k = 9$ og $k = 49$ for vise at grensene blir mer glatte når flere naboer er involvere.



Et generelt prinsipp innen statistikk er parsimoni - hvis to modeller passer like godt som løsning av problemet skal man velge den enkleste modellen. Men, hva er den enkleste modellen av $k = 1$ og $k = 3$, det er den som er minst fleksibel. Generelt vil vi tro at en løsning basert på den nærmeste naboen er mer fleksibel enn en basert på de tre nærmeste, men akkurat i vårt tilfelle ser løsningen med $k = 3$ ut til å ha to litt rare øyer - og klassifikasjonsgrensene ser ikke ut som de vil generalisere godt til nye data (i testsettet og andre fremtidige data). Vi velger tross dette å bruke $k = 3$ som optimal k.

I maskinlæring i dag er det veldig ofte komplekse modeller som er brukt, og det er viktig at vi blir kjent med bruk av valideringssett. Men, hvis vi har et veldig lite valideringssett må vi forvente at feilandelen på valideringssettet ikke er et godt estimat for feilandelen på fremtidige observasjoner. Vi har også testsettet, som vi tar frem når vi har bestemt oss for endelig modell.

Egenskaper til kNN

En styrke ved kNN er at metoden kan brukes for mange klassser, ikke bare for to klasser (metoden vi skal se på etterpå, logistisk regresjon, virker bare for to klasser).

En annen styrke er at metoden er grei å forstå, det inngår ikke vanskelige matematiske formler, men vi må velge hvilken verdi av k (hyperparameteren) vi skal bruke.

Hvis vi ikke har så mange forklaringsvariabler, p er liten, og antallet observasjoner i treningssettet n er stort, så kan kNN være en veldig god metode, men vi trenger mange nok nære naboer for å lage en regel som kan klassifisere godt.

Hvis antall forklaringsvariabler blir stort, kan den nærmeste naboen til et punkt være langt fra punktet, og da virker ikke kNN gå godt. Dette er det såkalte *curse of dimensionality*.

Eksempel: feilrate for kNN for testsettet for glasskår-klassifikasjon

```
from sklearn.neighbors import KNeighborsClassifier

df_testS=pd.read_csv("glassdftestS.csv")

X_tren=df_trenS[['Al','RI']]
X_test=df_testS[['Al','RI']]

knn = KNeighborsClassifier(n_neighbors=3,p=2)
knn.fit(X_tren, df_trenS['y'])
```

```
## KNeighborsClassifier(n_neighbors=3)
```

```
print(1-knn.score(X_test, df_testS['y']))
```

```
## 0.33333333333333337
```

Regresjon for klassifikasjon med to klasser

Eksempel: oppsett for glasskår-dataene

Vi skal se på situasjonen med to klasser, det er vår response. For observasjon i er $Y_i = 0$ den ene klassen og $Y_i = 1$ den andre klassen.

Dermed er Y_i en diskret variabel, som kan ta to verdier 0 og 1. Hvilken fordeling kan Y_i ha? (Svar: binomisk!)

I glassdataene så er $Y_i = 1$ vindusglass av typen float og $Y_i = 0$ vindusglass av typen ikke-float.

Vi har også data for brytningsindeks og vektprosent aluminumoksid, og det er våre forklaringsvariabler.

Det blir for mye å starte med alle variablene, så vi ser først på om vi kan bruke en av disse, $x = \text{Al}$ (vektprosent av aluminiumoksid), til å lage en regel for klassifisere en glassprøve fra et åsted.

For hele datasettet hadde vi 70 float-glassprøver ($y = 1$) og 76 ikke-float-glassprøver ($y = 0$). Vi jobber nå med treningssettet av glassprøver, og der har vi 42 float ($y = 1$) og 45 ikke-float-glassprøver ($y = 0$). I valideringssettet er det 15 float ($y = 1$) og 14 ikke-float-glassprøver ($y = 0$).

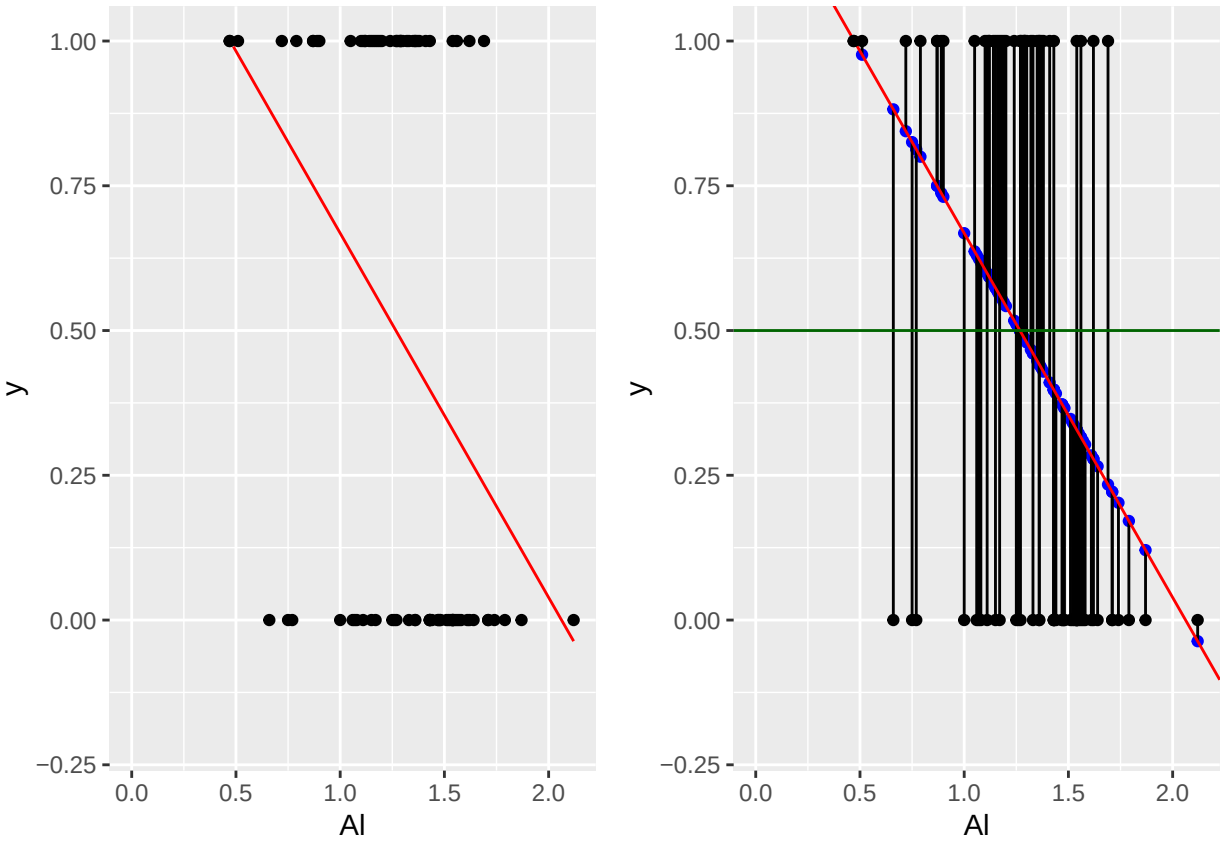
Eksempel: Enkel lineær regresjon for klassifikasjon for glasskår-dataene

I enkel lineær regresjon antar vi at responsen vår er kontinuerlig og normalfordelt. Kan vi bruke det til å klassifisere glasskåret - selv om responsen vår bare kan ta to verdier? Vi prøver!

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                      y   R-squared:                       0.152
## Model:                            OLS   Adj. R-squared:                  0.142
## Method:                 Least Squares   F-statistic:                     15.22
## Date:                Fri, 20 Nov 2020   Prob (F-statistic):           0.000191
## Time:                        17:03:19   Log-Likelihood:                -55.928
## No. Observations:                  87   AIC:                             115.9
## Df Residuals:                      85   BIC:                             120.8
## Df Model:                           1
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept      1.2972      0.215      6.043      0.000       0.870       1.724
## Al            -0.6291      0.161     -3.901      0.000      -0.950      -0.308
## ==============================================================================
## Omnibus:                      102.262   Durbin-Watson:                   2.572
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):                7.626
## Skew:                           0.006   Prob(JB):                       0.0221
## Kurtosis:                       1.550   Cond. No.                         8.84
## ==============================================================================
##
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Vi ser at Al er en signifikant kovariat (p-verdi mindre enn 0.05) med estimert koeffisient -0.63, og det er fullt mulig å bruke den enkle lineære regresjonen til å lage en klassifikasjonsregel.

I treningsettet bruker vi regresjonslinjen til å predikere verdi for responsen. Det blir et reelt tall, ofte ikke mellom 0 og 1. Se plott av regresjonslinja sammen med observasjonene for treningssettet. I plottet til høyre har vi også tegnet inn residualene som sorte streker mellom de sorte observasjonen og de predikerte observasjonene (blå) på regresjonslinjen. Det er klart at modellen ikke passer veldig godt til observasjonene.



Vi har lært at hvis vi ser på den estimerte koeffisienten for Al så er den -0.63, som betyr at når Al øker med 1 så minker prediksjonen vår med 0.63. Men hva er det egentlig vi har predikert? Hva betyr vår $\hat{y}$? Vår $\hat{y}$ kan ta verdier fra minus uendelig til uendelig, og det kan jo ikke være en sannsynlighet - den skal jo ta verdier fra 0 til 1. Det betyr at det er vanskelig å bruke lineær regresjon på en binær respons hvis vi vil tolke prediksjonen som en sannsynlighet.

Hva hvis vi hadde mer enn to klasser? Da ville det være enda vanskeligere å bruke enkel lineær regresjon fordi analysene ville avhenge av hvordan vi valgte å sette numeriske verdier for responsen.

Løsningen vi har valgt i statistikk er for to klasser å tilpasse en såkalt *logitisk* regresjon, der utgangspunktet er at responsen vår nå er binomisk fordelt - og ikke normalfordelt slik vi antar i lineær regresjon.

Logistisk regresjon

Modell for enkel logistisk regresjon

I logistisk regresjon ser vi på data der responsen er to mulige klasser, 0 eller 1. Ofte kaller vi utfallet til klassen 1 som en suksess og til klassen 0 som fiasko.

Vi antar at responsen fra observasjon i, Y_i, er binomisk fordelt med 1 forsøk og en sannsynlighet for suksess p_i, som kan være forskjellig for hver observasjon.

Vi skriver det slik:

$$Y_i = \begin{cases} 1 \text{ med sannsynlighet } p_i, \\ 0 \text{ med sannsynlighet } 1 - p_i. \end{cases}$$

Husk at i en binomisk fordeling er forventningsverdien antall forsøk ganger suksessannsynligheten, og her er det 1 forsøk og suksessannsynlighet p_i, så da er p_i *også lik forventningsverdien til* Y_i. Vi har at $\mathrm{E}(Y_i \mid \mathbf{x}_i) = \mu_i = p_i$.

Denne sannsynligheten p_i er det vi vil koble til forklarigsvariablene til observasjon i, og det gjør vi for logistisk regresjon med en såkalt *logistisk funksjon*.

I en enkel lineær regresjon har vi en forklaringsvariabel. Da snakker vi om at vårt beste gjett på den betingede forventningsverdien til Y_i er $\beta_0 + \beta_1 x_{1i}$.

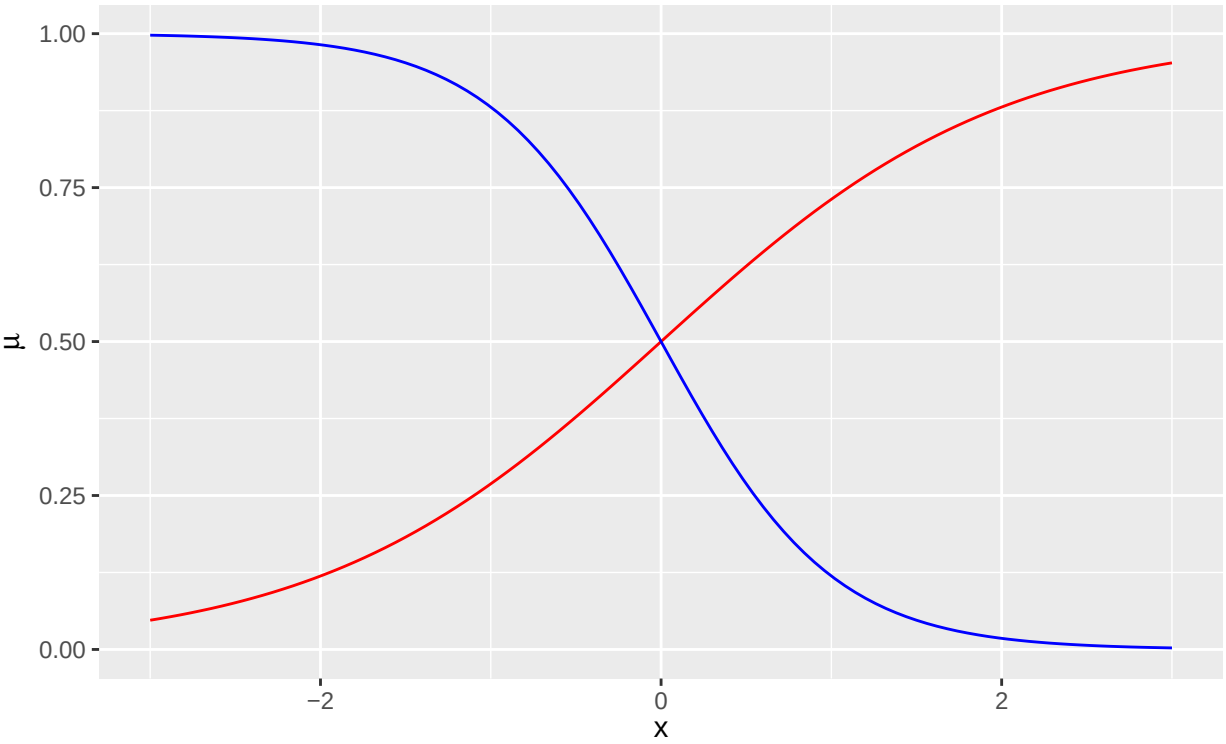
Hvis vi har en forklaringsvariabel x_{1i}, som Al i eksemplet med float eller ikke-float vindusglass, så er formelen for den logistiske funksjonen slik:

$$p_i = \frac{e^{\beta_0+\beta_1 x_{1i}}}{1 + e^{\beta_0+\beta_1 x_{1i}}}.$$

Vi har byttet ut en lineær funksjon (i linær regresjon) med en logistisk funksjon!

Den logistiske funksjonen har en S-form, og går fra 0 til 1 eller fra 1 til 0 - slik at nå er suksessannsynligheten $p_i \in [0, 1]$. Absoluttverdien til parameteren β_1 bestemmer helningsgraden til denne S-formede kurven, og fortegnet viser om kurven stiger eller synker.

Under ser vi den logistiske funksjonen for $\beta_0 = 1$, og $\beta_1 = 1$ (rød linje) og $\beta_1 = -2$ (blå linje) (og her er $\beta_0 = 1$ for begge kurver).

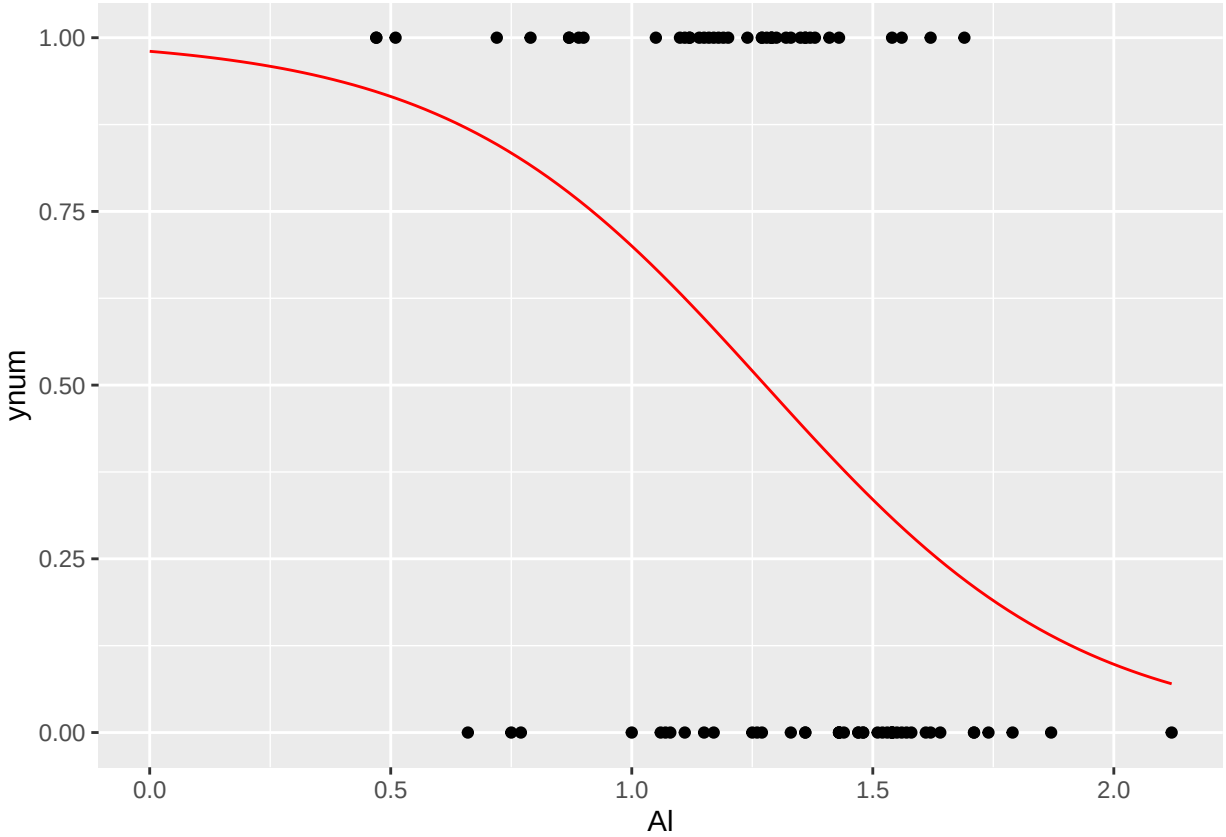


Det er andre kurveformer som også har egenskapen at den transformerer fra $[-\infty, \infty]$ til $[0, 1]$ - men den logistiske funksjonen er klart den mest brukte.

Eksempel: Enkel logistisk regresjon for glass med Al som forklaringsvariabel

Hvis vi tilpasser en logistisk regresjon til glasstype som respons og vektandel av aluminiumsoksid som forklaringsvaribel finner vi at $\hat{\beta}_0 = 3.91$ og $\hat{\beta}_1$ =-3.07 . Vi skal komme tilbake til hvordan vi estimerer de to parameterne β_0 og β_1, nå skal vi ha fokus på hvordan vi kan plotte suksessannsynlighet som en funksjon av vektandel av aluminiumsoksid - og det gjør vi ved å sette inn de to $\hat{\beta}_0$ og $\hat{\beta}_1$ inn i formelen for den logistiske funksjonen.

Under er den estimerte logistiske kurven plottet sammen med observasjonene. Ser den logistiske kurven ut som den passer litt bedre til data enn den lineære gjorde? (Svar: ja, noe bedre.)



Hvordan skal vi tolke β_1 i en enkel logistisk regresjon?

I en enkel lineær regresjon så har β_1 følgende tolkning: når x_i øker med 1 så vil y_i øke med β_1.

For logistisk regresjon er det litt vanskeligere. Hvis x_i øker med 1 så vil ikke endringen i suksesssannsynligheten p_i være den samme for alle verdier av x_i - endringen er avhengig av hva x_i er. Kort fortalt,
når x øker med 1 så vil *oddsen* multipliseres med e^{β_1}.

Odds

Odds er definert som ratioen mellom sannsynlighet for suksess og fiasko: $\frac{p_i}{1-p_i}$.

Hvis suksessannsynligheten er $p_i = \frac{1}{2}$ blir oddsen 1, og hvis $p_i = \frac{1}{4}$ da er oddsen $\frac{1}{3}$. Her er en liten tabell

```
##
## p    0.10 0.20 0.30 0.40 0.5 0.6 0.7 0.8 0.9
## odds 0.11 0.25 0.43 0.67 1.0 1.5 2.3 4.0 9.0
```

Observasjon i har forklaringsvariabelverdi x_i, og suksessannsynlighet p_i og odds $\frac{p_i}{1-p_1}$. Hvis vi øker forklaringsvariabelens verdi til $x_i + 1$ så vil den nye oddsen bli $\frac{p_i}{1-p_i} \cdot \exp(\beta_1)$, dvs. den gamle multiplisert med e^{β_1}.

Dette betyr at hvis x_{i1} øker med 1:

- hvis $\beta_1 < 0$ minker oddsen (ganges med et tall mindre enn 1),
- hvis $\beta_1 = 0$ endres oddsen ikke ($e^0 = 1$)
- hvis $\beta_1 > 0$ øker oddsen (ganges med et tall større enn 1)

Dermed er e^{β_1} "lettere" å tolke enn β_1.

Forklaring på resultatet om odds (for deg som lurer)

Følgende er en kort forklaring (men den forventes ikke at du kan).

Vi starter med uttrykket for suksessannsynligheten i en enkel logistisk regresjon.

$$p_i = \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}$$

Dette er suksessannsynligheten, og fiaskosannsynligheten blir da

$$1-p_i = \frac{1+e^{\beta_0+\beta_1 x_i}-e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}} = \frac{1}{1+e^{\beta_0+\beta_1 x_i}}$$

Da kan vi lage odds ved å dele disse to uttrykkene på hverandre

$$\frac{p_i}{1-p_i} = \frac{\frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}}{\frac{1}{1+e^{\beta_0+\beta_1 x_i}}} = e^{\beta_0+\beta_1 x_i}$$

Hvis $x_i = 0$ så blir oddsen e^{β_0}, mens hvis $x_i = 1$ så blir oddsen $e^{\beta_0+\beta_1} = e^{\beta_0} \cdot e^{\beta_1}$, det vil si at forholdet mellom disse to oddsene som

$$\frac{e^{\beta_0+\beta_1}}{e^{\beta_0}} = e^{\beta_1}$$

Dette oddsforholdet får vi også hvis vi sammenligner en hvilken som helst x_i med $x_i + 1$, fordi

$$\frac{e^{\beta_0+\beta_1(x_i+1)}}{e^{\beta_0+\beta_1 x_i}} = \frac{e^{\beta_0+\beta_1 x_i} \cdot e^{\beta_1}}{e^{\beta_0+\beta_1 x_i}} = e^{\beta_1}$$

I logistisk regresjon er vårt naturlige effektmål ikke regresjonskoeffisienten β_1, men e^{β_1} siden dette gir oss en tolkning via oddsforholdet.

Estimere regresjonsparametere for logistisk regresjon

I lineær regresjon fant vi estimater av regresjonsparameterne ved å minimere det kvadratiske avviket mellom observerte verdier og predikerte verdier, det vil si, summen av kvadratene av residualene. Vi sier at vi bruker en *kvadratisk tapsfunksjon* i lineær regresjon.

For logistisk regresjon minimerer man heller noe som heter *binært kryssentropi-tap*:

$$-\frac{1}{n}\sum_{i=1}^{n}[y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i)]$$

der n er antall observasjoner i treningssettet og $\hat{y}_i = \hat{p}_i$ er den predikerte verdien til observasjon x_i (et tall i intervallet fra 0 til 1) og y_i er den observerte klassen (0 eller 1).

Når vi minimere binært kryssentropi-tap med hensyn på regresjonskoeffisientene våre må vi bruke numeriske metoder, og vi kan ikke skrive ut en matematisk formel for de estimerte regresjonskoeffisientene.

Her er det ikke så naturlig å definere residualer - slik vi definerte dem for lineær regresjon, siden det ikke er en funksjon av residualene vi minimerer. Det finnes for logistisk regresjon andre varianter av residualer for å bruke til å evaluere modellen, men vi skal ikke se på det her.

Det finnes mye interessant statistisk teori for parameterestimering, og begrepet sannsynlighetsmaksimering (maximum likelihood) er et kjernebegrep. Både valget av kvadratisk tap for normalfordelte reponser og binært kryssentropitap for binomisk respons kan forklares fra sannsynlighetsmaksimeringsprinsippet.

Inferens for β_1

Når vi har minimert det binære kryssentropi-tapet finner vi estimert verdi for de to regresjonsparameterne β_0 og β_1.

Hva vil vi gjøre etter vi har estimert regresjonsparameterne? Vi vil gjøre inferens - det betyr her at vi vil ha konfidensintervall for β_1 og teste hypotesen om $\beta_1 = 0$. Til dette brukes en variant av sentralgrenseteoremet, slik at det er en normalfordeling vi bruker som fordeling til $\hat{\beta}_1$. (Vi bryr oss lite om β_0 nå.)

I utskriften fra en logistisk regresjon er det følgende vi skal bruke:

- "coef": $\hat{\beta}_1$
- "std err": estimert standardavvik til $\hat{\beta}_1$
- "P>|z|": p-verdi for å teste hypotesen $H_0 : \beta_1 = 0$ mot $H_1 : \beta_1 \neq 0$.
- "[0.025 0.975]": 95% konfidensintervall for β_1.

Eksempel: Logistisk regresjon for glass og Al (forts.)

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

#pydf=r.df
pydf=pd.read_csv("glassdftren.csv")
# Steg 2: spesifiser matematisk modell
formel='y~Al'

# Steg 3: Initaliser og tilpass en lineær regresjonsmodell
# først initialisere
modell = smf.logit(formel,data=pydf)
# deretter tilpasse
resultat = modell.fit()

# Steg 4: Presenter resultater fra den tilpassede regresjonsmodellen
```

```
## Optimization terminated successfully.
##          Current function value: 0.609160
##          Iterations 6
```

```
print(resultat.summary())
```

```
##                            Logit Regression Results
## ==============================================================================
## Dep. Variable:                      y   No. Observations:                   87
## Model:                          Logit   Df Residuals:                       85
## Method:                           MLE   Df Model:                            1
## Date:                Fri, 20 Nov 2020   Pseudo R-squ.:                  0.1204
## Time:                        17:03:21   Log-Likelihood:                -52.997
## converged:                       True   LL-Null:                       -60.252
## Covariance Type:            nonrobust   LLR p-value:                 0.0001394
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept      3.9141      1.232      3.176      0.001       1.499       6.330
## Al            -3.0656      0.924     -3.317      0.001      -4.877      -1.254
## ==============================================================================
```

```
print("FLERE utregninger:")
```

```
## FLERE utregninger:
```

```
print("exp(beta): ",np.exp(resultat.params))
```

```
## exp(beta):  Intercept    50.103371
## Al            0.046625
## dtype: float64
```

Hva kan vi lære av denne utskriften? Vi er (først) bare interessert i effekten Al har på å klassifisere glasskåret, linjen som begynner med Al i det nedre panelet.

- coef: $\hat{\beta}_1$ =-3.0656
- std err: estimert standardavvik til $\hat{\beta}_1 = 0.9241$ (men dette tallet skal vi ikke bruke direkte)
- P>|z|: p-verdi for å teste hypotesen $H_0 : \beta_1 = 0$ mot $H_1 : \beta_1 \neq 0$. Her er den 0.001, så den er mindre enn favorittsignifikansnivået 0.05 - og betyr at vi forkaster nullhypotesen at koeffisienten foran Al, β_1, er 0.
- [0.025 0.975]: 95% konfidensintervall for β_1: dette sier hva vi har av usikkerhet i estimatet vårt av regresjonskoeffisienten β_1. Observer at ikke 0 er i intervallet (det visste vi på forhånd fordi p-verdien var mindre enn 0.05).

Vi kan tolke verdien av $\hat{\beta}_1$ med oddsforkaring: For hver økning med "ett prosentpoeng i vektprosent av aluminiumoksid i glasskåret" (forklaringsvariablene Al øker med 1 enhet) vil oddsen multipliseres med $e^{\hat{\beta}_1} = 0.047$. Det er ikke så sannsynlig at det er et vindusglass av typen float hvis vi har en høy vektprosent av aluminimumsoksid.

Klassifikasjonsregel fra enkel logistisk regresjon

Da har vi sett at det kan ha noe for seg å bruke logistisk regresjon til å tilpasse data fra en kovariat og to klasser. Men, hvor er sannsynligheten for å tilhøre hver klasse og denne klassifikasjonsregelen?

Først: estimert sannsynlighet for å tilhøre klassen som er kodet som 1 (her er det float-glass i eksemplet) finner vi ved å sette inn estimerte regresjonskoeffisienter inn i den logistiske formelen for suksessannsynligheten (være klassen kodet som 1) i en enkel logistisk regresjon:.

$$\hat{p}_i = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 x_i}}{1 + e^{\hat{\beta}_0+\hat{\beta}_1 x_i}}$$

og det er den kurven vi allerede har tegnet sammen med observasjonene våre - og den tegner jeg igjen under.

Og hva med klassifikasjonsregelen? Hvor skal vi sette cut-off for sannsynligheten for at en observasjon er fra klasse 1?

Her er det mange muligheter, men i utgangspunktet er det vanlig å sette cut-off på sannsynlighet 0.5. Da trenger vi minst 50% sjanse for at observasjonen vi skal klassifisere er fra klasse 1.

Hvis vi velger cut-off 0.5 blir regelen vår: Klassifiser til klasse 1 hvis

$$\hat{p}_i = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 x_i}}{1 + e^{\hat{\beta}_0+\hat{\beta}_1 x_i}} \geq 0.5$$

Hvis $\hat{p}_i = 0.5$ vil oddsen $\hat{p}_i/(1 - \hat{p}_i) = 1$, og deler på $1 - \hat{p}_i$ (som da blir mindre eller lik 0.5) kan vi skrive

$$\frac{\hat{p}_i}{1 - \hat{p}_i} = e^{\hat{\beta}_0+\hat{\beta}_1 x_i} \geq 1$$

og tar vi den naturlige logaritmen på hver side, og så er $\ln(1) = 0$ og vi får klassifikasjonsregelen

$$\hat{\beta}_0 + \hat{\beta}_1 x_i \geq 0$$

Det vil si at

$$\hat{\beta}_1 x_i \geq -\hat{\beta}_0$$

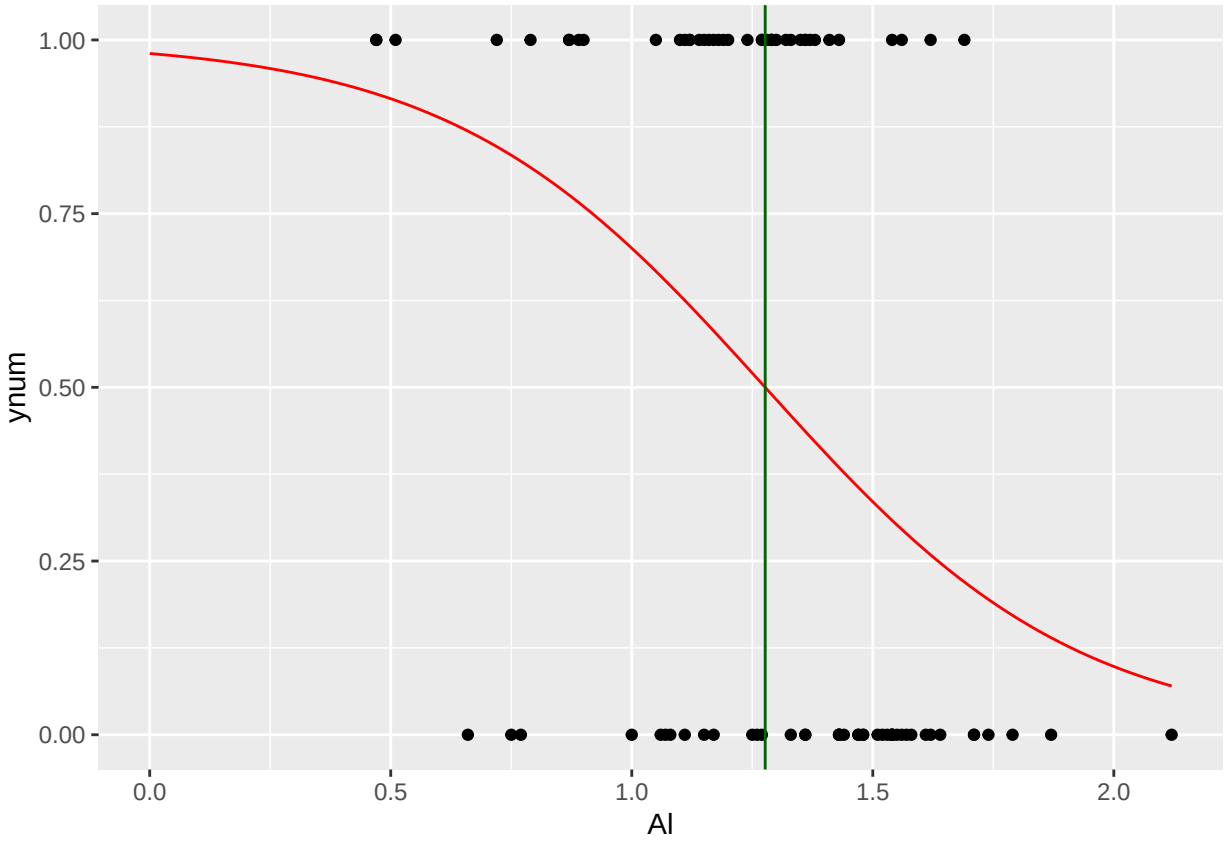
Hvis $\hat{\beta}_1$ er positiv så blir regelen å klassifisere til klasse 1 hvis

$$x_i \geq -\frac{\hat{\beta}_0}{\hat{\beta}_1}$$

mens hvis $\hat{\beta}_1$ er negativ blir regelen å klassifisere til klasse 1 hvis

$$x_i \leq -\frac{\hat{\beta}_0}{\hat{\beta}_1}$$

For eksemplet vårt er med glasskår $\hat{\beta}_1$ negativ og vi klassifiserer til float-glass hvis vektprosenten av aluminimumsoksid blir mindre enn 1.28. Dette ser vi også lett på figuren under, det er jo der p_i er lik 0.5.



Forvirringsmatrise og feilrate

Vi har tidligere definert forvirringsmatrise og feilrate for k-nærmeste-nabo-klassifikasjon, og definisjonene er akkurat den samme for logistisk regresjon.

Eksempel: Forvirringsmatrise og feilrate for glassskår logistisk regresjon

Når vi jobber med å finne frem til den beste modellen for klassifikasjon kan vi gjøre noe a la det vi gjorde i multippel lineær regresjon da vi brukte justert R^2. For logistisk regresjon kunne vi valgt den modellen som har den beste verdien (på treningssettet) av noe som heter AIC. Men, for andre klassifikasjonsmetoder finnes ikke tilsvarende godhetsmål, og derfor vil vi her ha fokus på å velge en modell som har lav feilrate på *valideringssettet*.

Vi har jo allerede laget klassifiksjonsregelen, og da er det bare for hver observasjon i valideringssettet å sjekke om Al er mindre eller større enn 1.28, og så teller vi opp og fyller ut forvirringsmatrisen.

	Sann 0 (ikke-float)	Sann 1 (float)	Totalt
Predikert 0 (ikke-float)	7	5	12
Predikert 1 (float)	8	9	17
Totalt	15	14	29

Andelen korrekte klassifikasjoner kalles *nøyaktighet* (accuracy) og blir (7 + 9)/29=0.55. Det motsatte er andelen feil: det er da først teller: 5 + 8 feile klassifikasjoner, slik at *feilraten* er (5 + 8)/29=0.45. som også er 1 minus nøyaktigheten.

I Python regner vi ut feilraten som følger

```
from sklearn.metrics import classification_report, accuracy_score,confusion_matrix

# Spesifiser verdi for cutoff
cutoff = 0.5

# Prediker verdi for valideringssettet
val_pred = resultat.predict(exog = df_val)

# klassifiser med cut-off 0.5
y_valpred = np.where(val_pred > cutoff, 1, 0)
y_valobs = df_val['y']

# Finn feilrate
print("Feilrate:", 1-accuracy_score(y_true=y_valobs, y_pred=y_valpred))
```

```
## Feilrate: 0.4482758620689655
```

Dette er ikke veldig bra, kanskje vi får en bedre regel hvis vi tar med flere forklaringsvariabler?

Hvorfor kan vi ikke bare heller regne ut feilraten på treningssettet? Joda, det kan vi - og vi har en veldig enkel modell som ikke overtilpasses til data. Hvis antall parametere i modellen er stort (veldig mange forklaringsvariabler eller mer fleksible kurver) så er det tryggere å regne feilrate på valideringssett.

Fra enkel til multippel logistisk regresjon

Når vi i logistisk regresjon går fra å ha med en forklaringsvariabel (enkel logistisk regresjon) til å ha med flere forklaringsvariabler (multippel logistisk regresjon) er endringen vi gjør å utvide $\beta_0 + \beta_1 x_{i1}$. Dette uttrykket kalles gjerne vår *lineærprediktor*.

Først, vi omdøper vår ene forklaringsvariabel fra x_i til x_{1i}.

Vi utvider lineærprediktoren $\beta_0 + \beta_1 x_{1i}$ til å ta med flere ledd - med k forklaringsvariabler får vi

$$\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_k x_{ki}$$

Dermed utvider vi uttrykket for suksessannsynligheten til:

$$p_i = \frac{e^{\beta_0+\beta_1 x_{1i}+\beta_2 x_{2i}+\cdots+\beta_k x_{ki}}}{1 + e^{\beta_0+\beta_1 x_{1i}+\beta_2 x_{2i}+\cdots+\beta_k x_{ki}}}.$$

Vi får da tilsvarende et utvidet uttrykk for $\hat{y}_i = \frac{e^{\beta_0+\beta_1 x_{1i}+\beta_2 x_{2i}+\cdots+\beta_k x_{ki}}}{1+e^{\beta_0+\beta_1 x_{1i}+\beta_2 x_{2i}+\cdots+\beta_k x_{ki}}}$, som inngår i minimeringskriteriet for å estimere koeffisientene i modellen.

Helt praktisk blir det liten forskjell i hvordan vi utfører analysen, men vi må tolke de estimerte koeffisienten på en litt annen måte. Nå må vi si at *gitt at de andre forklaringsvariablene i modellen holdes konstante* så vil når x_{1i} øker med 1 oddsen multipliseres med e^{β_1}.

Eksempel: Logistisk regresjon for glass med aluminiumoksid og brytningsindeks som forklaringsvariabler

Når vi gjør logistisk regresjon med pakken statsmodels i Python kan vi bruke modellformel (som vi kjenner fra multippel lineær regresjon), og vi skriver bare "y~Al+RI" for å få med to forklaringsvariabler Al og RI (refractive index=brytningsindeks).

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

#pydf=r.df
pydf=pd.read_csv("glassdftren.csv")
# Steg 2: spesifiser matematisk modell
formel='y~Al+RI'

# Steg 3: Initaliser og tilpass en lineær regresjonsmodell
# først initialisere
modell = smf.logit(formel,data=pydf)
# deretter tilpasse
resultat = modell.fit()

# Steg 4: Presenter resultater fra den tilpassede regresjonsmodellen
```

```
## Optimization terminated successfully.
##          Current function value: 0.548177
##          Iterations 6
```

```
print(resultat.summary())

# Tolkning av estimerte regresjonsparametere er på exp-skala (odds)
```

```
##                           Logit Regression Results
## ==============================================================================
## Dep. Variable:                      y   No. Observations:                   87
## Model:                          Logit   Df Residuals:                       84
## Method:                           MLE   Df Model:                            2
## Date:                Fri, 20 Nov 2020   Pseudo R-squ.:                  0.2085
## Time:                        17:03:21   Log-Likelihood:                -47.691
## converged:                       True   LL-Null:                       -60.252
## Covariance Type:            nonrobust   LLR p-value:                 3.507e-06
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept      8.4824      2.203      3.851      0.000       4.165      12.800
## Al            -6.4327      1.643     -3.915      0.000      -9.653      -3.212
## RI            -0.4473      0.156     -2.874      0.004      -0.752      -0.142
## ==============================================================================
```

```
print("FLERE utregninger:")
```

```
## FLERE utregninger:
```

```
print("exp(beta): ",np.exp(resultat.params))
```

```
## exp(beta):  Intercept    4828.993795
## Al              0.001608
## RI              0.639331
## dtype: float64
```

Vi ser at koeffisientestimatet foran Al har endret seg mye når vi får med RI. Nå er effekten av at vi øker vektandel aluminiumsoksid med ett prosentpoeng så multipliseres oddsen med 0.0016.

For brytningsindeksen er effekten at oddsen for at det er float-glass ganges med 0.64.

Vi observerer også at begge forklaringsvariablene er signifikante i analysen.

Da har vi to ting igjen: hvilken klassifikasjonsregel har vi, og hva er feilraten nå på valideringssettet.

Klassifikasjonsregel for multippel logistisk regresjon

Gitt estimater for regresjonskoeffisientene. Vi vil klassifisere observasjon $\mathbf{x}_i$ (med k forklaringsvariabler). Anta at vi vil bruke cut-off 0.5 på suksessannsynligheten (sannsynlighet for å tilhøre klasse 1).

1) regn ut estimert suksessannsynlighet
2) klassifiser til klasse 1 hivs den estimerte suksessannsynligheten er større enn valg cut-off (vanlig med 0.5).

Det kan vises at grensen mellom de to klassens med en logistisk regresjon med k forklaringsvariabler vil være et hyperplan i $k-1$ dimensjoner. For to forklaringsvariabler får vi da en rett linje, og formelen for grensen mellom de to klassene er da: $x_2 = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2}x_1$. Dette betyr at klassegrensen er en rett linje hvis vi plotter x_1 mot x_2.

Eksempel: Forvirringsmatrise og feilrate for glasskår logistisk regresjon

	Sann 0 (ikke-float)	Sann 1 (float)	Totalt
Predikert 0 (ikke-float)	10	2	12
Predikert 1 (float)	5	12	17
Totalt	15	14	29

Andelen korrekte klassifikasjoner (accuracy) og blir (10 + 12)/29=0.76. Det motsatte er andelen feil: det er da først teller: 2 + 5 feile klassifikasjoner, slik at *feilraten* er (2 + 5)/29=0.24. som også er 1 minus nøyaktigheten.

Den nye modellen med både Al og RI har lavere feilrate enn modellen med Al, så da velger vi å bruke den modellen videre og sjekke hvordan den fungere på testsettet som vi har hatt i "bankboksen vår".

Eksempel: Endelig evaluering av modellen med testsett

	Sann 0 (ikke-float)	Sann 1 (float)	Totalt
Predikert 0 (ikke-float)	10	2	12
Predikert 1 (float)	6	12	18
Totalt	16	14	30

Andelen korrekte klassifikasjoner blir (10 + 12)/30=0.73. Det motsatte er andelen feil: det er da først teller: 2 + 6 feile klassifikasjoner, slik at *feilraten* er (2 + 6)/30=0.27. som også er 1 minus nøyaktigheten.

```
# Spesifiser verdi for cutoff
cutoff = 0.5

# Prediker verdi for testsettet
test_pred = resultat.predict(exog = df_test)

# klassifiser som float hvis sannsynlighet er over 0.5
y_testpred = np.where(test_pred > cutoff, 1, 0)
y_testobs = df_test['y']
print(y_testpred)
# Finn feilrate
```

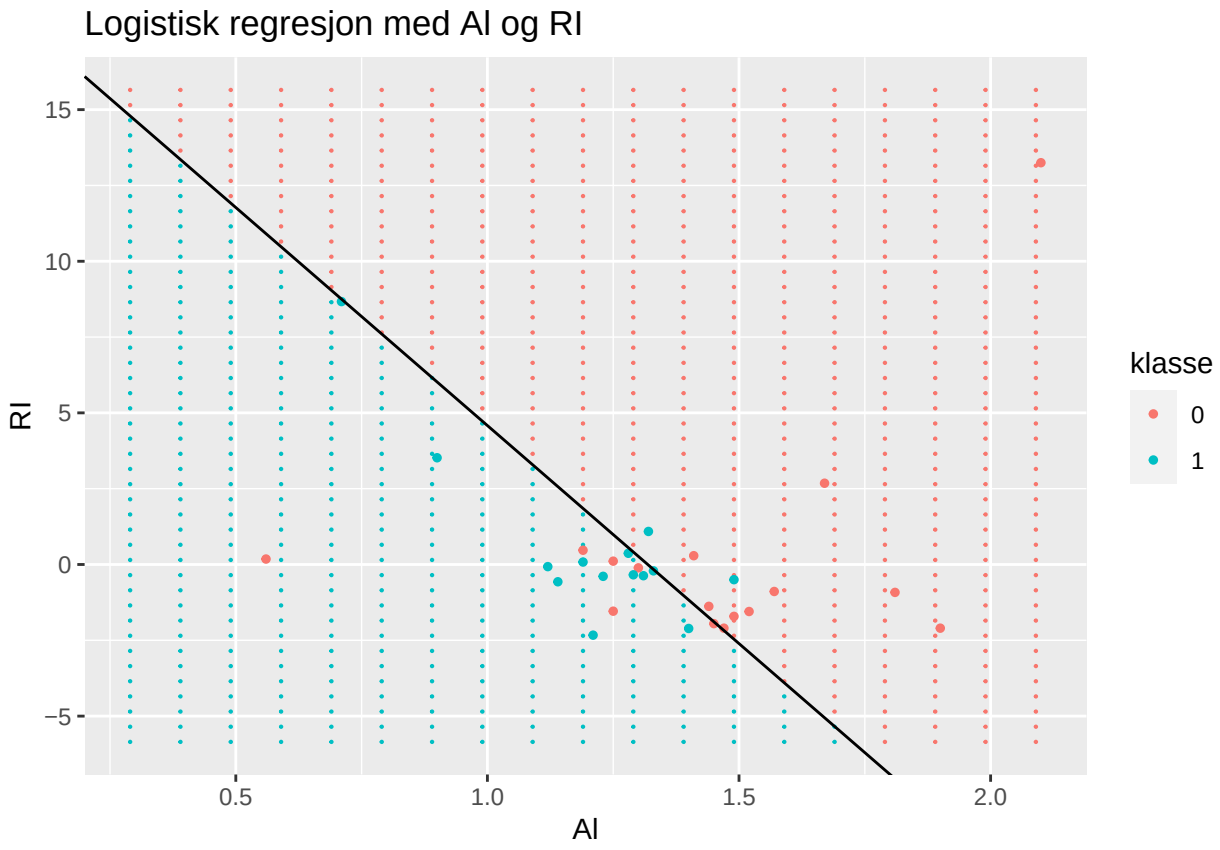
```
## [1 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1]
```

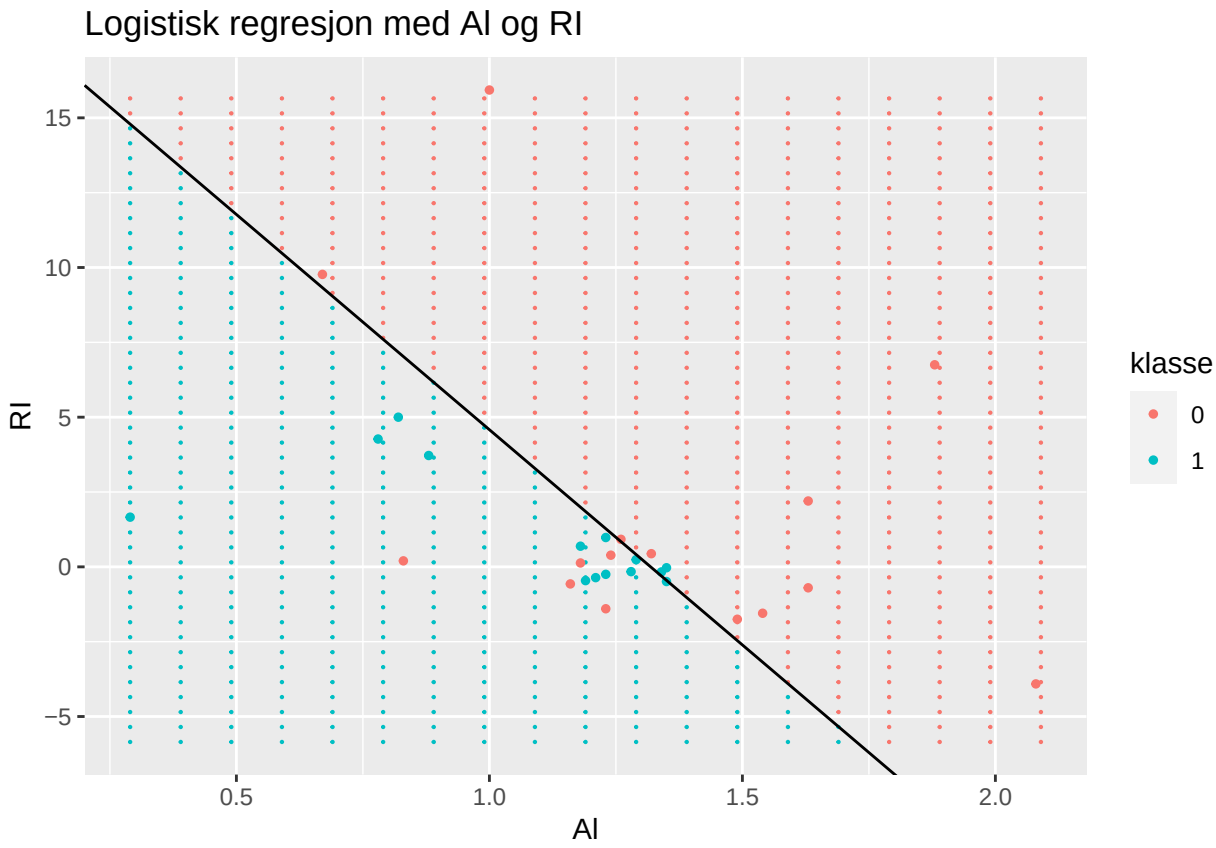
```
print("Feilrate:", 1-accuracy_score(y_true=y_testobs, y_pred=y_testpred,normalize=True))
```

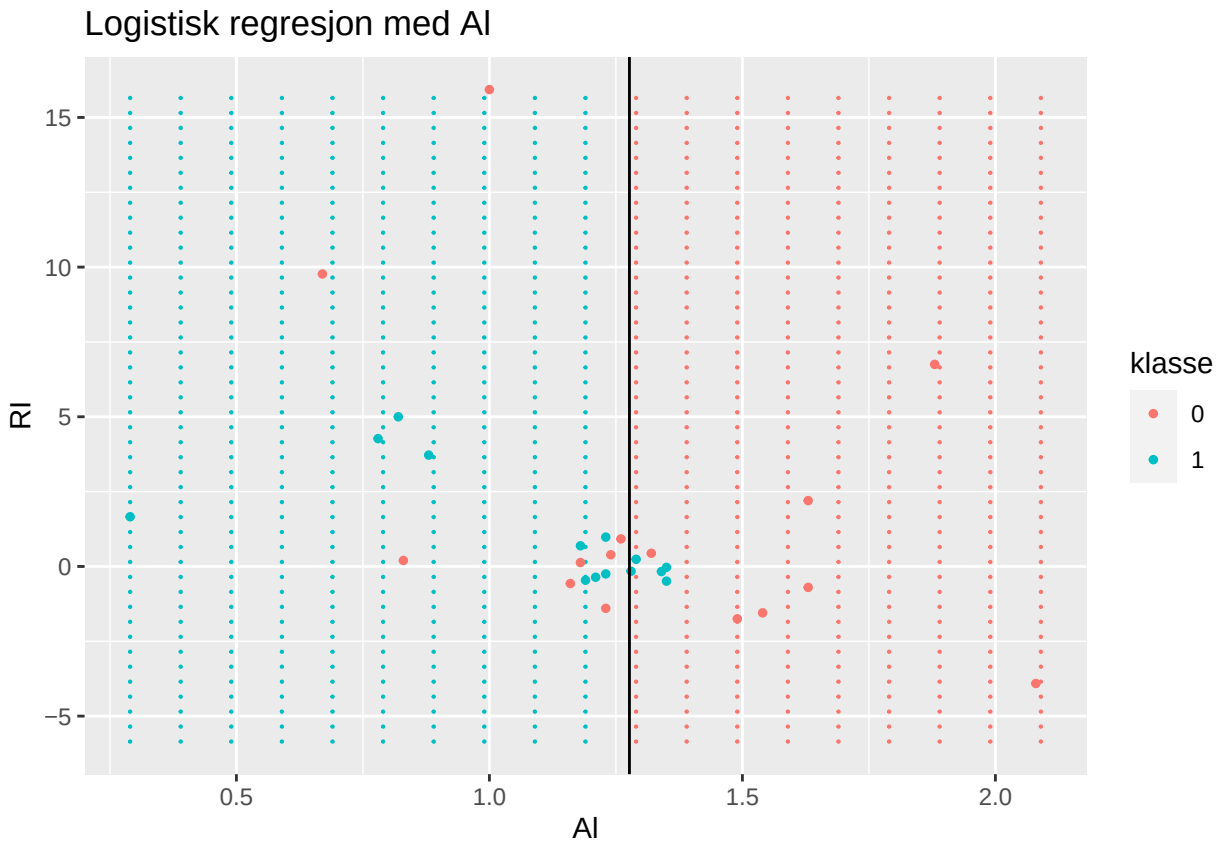
```
## Feilrate: 0.2666666666666667
```

Eksempel: Grafisk fremstilling av klassegrensen for logistisk regresjon

Klassegrensen vises sammen med dataene i testsettet, der feilraten var 0.27.




Logistisk regresjon med Al og RI
15
10
5
0
−5
RI
0.5
1.0
1.5
2.0
Al
klasse
0
1


Logistisk regresjon med Al
15
10
5
0
−5
RI
0.5
1.0
1.5
2.0
Al
klasse
0
1

Analyse av et datasett med logistisk regresjon

Alt vi har lært om enkel logistisk regresjon (en forklaringsvariabel) kan generaliseres til multippel logistisk regresjon (flere forklaringsvariabler), og tolkningene blir de samme.

Vi bruker samme fremgangsmåte for logistisk regresjon som for lineær regresjon (men vi bruker lineær regresjon ved en normalfordelt respons og så vil vi bruke logistisk regresjon for en binomisk fordelt respons).

- Steg 1: Bli kjent med dataaene ved å se på oppsummeringsmål og ulike typer plott
- Steg 2: Spesifiser en matematisk modell
- Steg 3: Initialiser og tilpass modellen
- Steg 4: Presenter resultater fra den tilpassede modellen
- Steg 5: Evaluere om modellen passer til dataene

Vi skal se på en Jupyter-notatbok for å utføre logistisk regresjon på et datasett med mange forklaringsvariabler i zoom-forelesningen.

Referanser

Glass på åstedet er beskrevet her: https://archive.ics.uci.edu/ml/datasets/glass+identification, og datasettet er en modifisert versjon fra Rs MASS pakke, datasett fgl.