Norges miljø- og biovitenskapelige universitet
Fakultet for miljøvitenskap og teknologi
Institutt for matematiske realfag og teknologi

Masteroppgave 2016
60 stp

Anvendelser av Tikhonov-regularisering på regresjon og klassifikasjon med rask «leave one out» kryssvalidering

Applications of Tikhonov Regularization on Regression and Classification with Fast «Leave One Out» Cross-Validation

Martin Seland Ansnes

Forord

Å jobbe med denne oppgaven har vært langvarig og krevende, men samtidig veldig lærerikt. Ikke bare er dette slutten på et års masterarbeid, men også slutten på et fem-årig studium der jeg har fått lov til å fordype meg i matematikk. Det er jeg veldig takknemlig for!

Jeg vil begynne med å takke Enrico for tålmodighet og støtte gjennom hele studiet og spesielt gjennom det siste året. En stor takk går også til Tordbjørn og Julie som har introdusert meg for LaTeX. Uten dette verktøyet ville det vært veldig vanskelig å skrive denne oppgaven. Samtlige tre har også inspirert meg til å ta høyere utdanning, og det er jeg veldig glad for. En stor takk går også til medstudenter jeg har jobbet sammen med gjennom studiet, og spesielt til Herman.

Jeg vil også takke seksjoner for realfag, anvendte matematiske fag og læring og lærerutdanning ved IMT for å ha latt meg jobbe med undervisning av realfag de siste fire årene. Det har vært veldig positivt for min faglige utvikling, og det har vært et stort privilegium å få lov å ha et så relevant lønnet arbeid.

Jeg vil gjerne takke min familie, som har har lært meg verdien av å jobbe for å nå sine mål. Uten dette i bagasjen ville jeg aldri klart å gjennomføre studiet.

Den største takken går til min utmerkede veileder Ulf Indahl. Takk for konstruktive tilbakemeldinger, gode råd, og et svært spennende tema for oppgaven min!

NMBU Ås, 10.05.2016

Martin Seland Ansnes

Sammendrag

I denne oppgaven har jeg sett på Tikhonov-regulasisering av regresjons- og klassifikasjonsproblemer og sammeliknet egne og veileders Matlab-script med andre regresjons- og klassifikasjonsmetoder med hensyn på tidsbruk og antatt prediksjonsevne.

Tikhonov-regularisert regresjon kan betraktes som vanlige minste kvadraters problemer, og teori herfra fungerer også for Tikhonov-regresjon. Særlig gjelder dette muligheten for å gjøre rask «leave one out» kryssvalidering: kryssvalidering uten å bygge modellen på nytt for hver utelatte måling. Sammenliknet med PLSR og PCR går funksjonene for Tikonov-regresjon svært mye raskere når man gjør «leave one out» kryssvalidering for alle datasettene jeg har analysert. Den antatte prediskjonsevnen ser også lovende ut, både PRESS-verdier og evne til å predikere test-data ser ut til å kunne konkurrere med PLSR og PCR for data med flere variable enn målinger.

Tikhonov-regularisert klassifikasjon ser også ut til å kunne konkurrere med konvensjonelle klassifikasjonsmetoder. Det viser seg at man kan gjøre rask «leave one out» kryssvalidering også for Tikhonov-regularisert lineær diskriminantanalyse, i tillegg til der man behandler klassifiseringen som et regresjonsproblem. Når man gjør «leave one out» kryssvalidering går både regularisert LDA og klassifisering ved hjelp av Tikhonov-regresjon raskere enn PLSR. Uten kryssvalidering går PLSR raskere enn enn regularisert LDA. Dette har jeg kun forsøkt for store datamengder i forbindelse med bildeanalyse, men her ser det ut til at en modifisert utgave av Tikhonov-regresjon predikerer best. Denne går også vesentlig raskere enn PLS-DA. Generelt ser det ut til at Tikhonov-regularisering kan konkurrere med mer konvensjonelle metoder for klassifisering både på tid og prediksjonsevne.

Abstract

In this thesis I have investigated Tikhonov regularization on regression and classification problems and compared my own and my supervisor's Matlab scripts with other methods with regards to time spent and prediction ability.

Tikhonov regularized regression can be treated as ordinary least squares problems, the theory from which can also be applied to Tikhonov regression, and in particular the method for doing fast «leave one out» cross-validation: cross-vaildation where without constructing the model anew for each omitted data point. Compared with PLSR and PCR the functions for Tikhonov regression run much faster when doing «leave one out» cross-validation for all the data I have analyzed. The apparent prediction ability also look promising, both PRESS values and the ability to predict test data seem to compete well with PLSR and PCR for data with more variables than data points.

Tikhonov regularized classification also seems to be able to compete with more conventional methods of classification. It turns out that it is possible to do fast «leave one out» cross-validation also for Tikhonov regularized linear discriminant analysis, in addition to the method where the classification is treated as a regression problem. When doing «leave one out» cross-validation, both regularized LDA end classification done by Tikhonov regression go faster than PLSR. When not doing cross-validation, PLSR is faster than regularized LDA. I have done this type of analysis for large datasets when doing image classification, but here it semms that a modified version of Tikhonov regression predicts better. This method is also significantly faster than PLS-DA. Overall it seems that Tikhonov regularization can compete with more conventional method of classification both on time efficiency and prediction ability.

Innhold

Kapittel 1

Innledning

1.1 Motivasjon

I romanen *The Hitch Hiker's Guide to the Galaxy* (Adams , 1980 [1]) fortelles det om datamaskinen *Deep Thought*, som skal finne svaret på det ultimate spørsmålet om livet, universet og alt mulig. Denne utregningen bruker *Deep Thought* 7,5 millioner år på, og det viser seg at svaret den kommer fram til - 42 - nok ikke ga den samme innsikten man hadde sett for seg på forhånd.

To viktige metoder innen statistikk og forskning er og regresjon og klassifisering. Det er imidlertid mange måter å bygge modeller for regresjon og klassifisering på, og det er derfor hensiktsmessig å kunne sammenlikne hvor «gyldige» de forskjellige modellene er, altså hvor godt modellene beskriver virkeligheten. Man utelater en del av dataene fra datasettet, bygger modellene og bruker disse for å predikere responsen for de utelatte dataene. Man får et avvik mellom de predikerte og de målte responsverdiene, og kan bruke dette avviket for å avgjøre hvilken modell som er best. Dette kalles *modellvalidering*.

Avhengig av datamengden kan slik analyse av gyldighet kreve stor regnekraft. I og for seg kan man hevde at hvor lang tid det tar å bygge modellen ofte er av underordnet betydning siden man kun trenger å bygge modellen én gang. Det er bruk av modellen som er interessant, og er modellen først laget - og viser seg å være en god beskrivelse av virkeligheten - kan man bruke den, noe som ofte ikke krever så mye regnekraft. *Deep Thought* bruker

svært lang tid på å bygge en modell, og denne er ikke tilfredsstillende.

En god metode for å bygge statistiske modeller, vil være en som bruker kort tid, men samtidig predikerer godt. Det er frustrerende å vente på at datamaskinen bruker lang tid på å bygge en modell som viser seg å beskrive virkeligheten dårlig, samtidig vil det gå mye mer effektivt å sammenlikne modeller dersom tidsbruken er effektiv.

Tikhonov-regularisering er en metode for å bygge regresjonsmodeller med presumptivt høy prediksjonsevne. Mens man i minste kvadraters metode skal minimere

$$\|X\hat{\boldsymbol{\beta}} - \boldsymbol{y}\|^2$$

får man i Tikhonov-regularisering en tilleggsbetingelse og skal minimere

$$\|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda\|T\hat{\boldsymbol{\beta}}\|^2. \tag{1.1}$$

Her er X datamatrisa, $\hat{\boldsymbol{\beta}} = [\hat{\beta}_1 \quad \hat{\beta}_2 \quad \cdots \quad \hat{\beta}_p]^T$ er regresjonskoeffisientene man skal estimere, $\boldsymbol{y}$ er responsvektoren, λ regulariseringsparameteren og T regulariseringsmatrisa. I spesialtilfellet Ridge-regresjon er $T = I_p$ identitetsmatrisa. Da reduseres (1.1) til

$$\|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda\|\hat{\boldsymbol{\beta}}\|^2.$$

I tillegg til å minimere normen til $X\hat{\boldsymbol{\beta}} - \boldsymbol{y}$, skal man også «straffe» normen til $\hat{\boldsymbol{\beta}}$. Den ferdige modellen vil være

$$y = \boldsymbol{x}^T\hat{\boldsymbol{\beta}} + \epsilon = \sum_{j=1}^{p} x_j\hat{\beta}_j + \epsilon,$$

der y er en responsverdi, $\boldsymbol{x}^T = [x_1 \quad x_2 \quad \cdots \quad x_p]$ en vektor av målte verdier for forklaringsvariablene og ϵ er feilen, den delen av y som ikke forklares av $\boldsymbol{x}^T\boldsymbol{\beta}$. Dersom normen til $\boldsymbol{\beta}$ er stor, vil (enkelte av) β_j-elementene også være store, og en liten forandring i en x_j-verdi vil kunne ha stor effekt på prediksjonen av y. Man har en ustabil modell. Motsatt vil y forventes å forandres lite ved en liten endring i en x_i når normen til $\boldsymbol{\beta}$ er liten. Ved å velge riktig λ er teorien den at forandringen i y vil bli akkurat passe ved en forandring i $\boldsymbol{x}^T$, og modellen forventes å ha god prediksjonsevne (Boyd og Vandenberghe, 2015 [4], s 201 - 203)

For å finne riktig λ-verdi kan man bygge modellen mange ganger for forskjellige λ-verider, for så å numerisk lete opp den verdien av regulariseringsparameteren som gir lavest prediksjonsfeil. I tillegg kan modellvalideringen gjøres ved å bygge modellen igjen og igjen med utelatte datapunkter. Dette kan bli svært mange modeller som bygges, og det tar tid. Jeg vil imidlertid se på matematiske snarveier som kan redusere antall regneoperasjoner, men gi matematisk ekvivalente resultater. Disse metodene vil jeg så sammenlikne med metoder som er mye brukt for regresjon og klassifikasjon.

1.2 Notasjon og oppbygging

Alle størrelsene i denne oppgaven er reelle tall eller vektorer og matriser med reelle elementer, og alle vektorer og matriser har endelig dimensjon. Skalare variable skrives som små bokstaver i kursiv: $x \in \mathbb{R}$. Vektorer i $\mathbb{R}^n$ skrives som små bokstaver i fet kursiv: $\boldsymbol{x} \in \mathbb{R}^n$. Matriser angis som store bokstaver i kursiv: $X \in \mathbb{R}^{n \times p}$. Transponering angis med en T som hevet skrift. X^T vil derfor aldri bety matrisa X opphøyd i T, men alltid X transponert. Alle estimater angis med cirkumfleks: $\hat{\boldsymbol{\beta}}$ er estimatet av $\boldsymbol{\beta}$. Alle normer av vektorer vil være euklidske eller 2-normer. Jeg utelater derfor å spesifisere dette videre, og $\|\cdot\| \equiv \|\cdot\|_2$ i denne oppgaven.

En datamatrise X med måleverdier av foklaringsvariable vil i denne oppgaven ha dimensjon $n \times p$. Datamatrisene er ordnet slik at hver kolonne representerer en variabel, mens hver rad representerer et målepunkt. En del litteratur gjør dette motsatt, men jeg har valgt å ordne datamatrisene slik for å slippe å transponere ved utregninger. I de tilfeller der jeg har mer enn én responsvariabel, har responsmatrisa Y dimensjon $n \times m$ med målinger og variable ordnet på tilsvarende måte. I koden som presenteres i tillegget bakerst opptrer også tredimensjonale datatyper (tensorer). Disse betraktes som ordnede mengder av matriser der den tredje indeksen angir matrisenummeret.

Kapittel 2 inneholder en del sentral teori om matrisealgebra, samt forklaring av vanlige regresjonsmetoder: Minste kvadraters metode, delvis minste kvadraters metode og prinsipalkomponentanalyse. Jeg vil også forklare detaljene i Tikhonov-regularisering her.

I kapittel 3 og 4 forklarer jeg og viser anvendelser av metoder for henholdsvis regresjon og klassifisering med modellvalidering. Det er her hoveddelen av arbeidet mitt presenteres, og jeg sammenliknger Tikhonov-regularisering med andre vanlige metoder for regresjon og klassifisering, både med hensyn på prediksjonsevne og på tidsbruk.

Kapittel 5 inneholder oppsummering av de viktigste funnene fra kapittel 3 og 4, samt noen tanker om videre stuider av Tikhonov-regularisering.

Alle referanser til likninger, figurer, kode og litteratur i teksten er klikkbare hyperlenker i .pdf-versjonen av denne oppgaven.

Kapittel 2

Teori

I dette kapittelet vil jeg ta for meg to viktige metoder for faktorisering av matriser, samt noen vanlige regresjonsmetoder.

2.1 QR-faktorisering og Gram-Schmidt

Enhver matrise $X \in \mathbb{R}^{n \times p}$ kan faktoriseres til

$$X = QR,$$

der $Q = [\boldsymbol{q}_1 \quad \boldsymbol{q}_2 \quad \cdots \quad \boldsymbol{q}_p] \in \mathbb{R}^{n \times p}$ og $R \in \mathbb{R}^{p \times p}$ er øvre triangulær. Dersom X har full rang, vil $\{\boldsymbol{q}_j\}_{j=1}^{p}$ utgjøre en ortonormal basis for søylerommet til X.

Gram-Schmidt-prosessen er en algoritme for å finne en ortogonal basis for søylerommet til en matrise. Dersom $X = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p]$, vil man kunne lage en ortogonal basis $\{\boldsymbol{v}_j\}_{j=1}^{p}$ for søylerommet ved å la den første basisvektoren være lik $\boldsymbol{x}_1$. Deretter lar man den andre basisvektoren være lik $\boldsymbol{x}_2$, men trekker fra den ortogonale projeksjonen av $\boldsymbol{x}_2$ ned på $\boldsymbol{v}_1$. $\boldsymbol{v}_1$ og $\boldsymbol{v}_2$ er nå ortogonale. Den r-te ($r \leq p$) basisvektoren $\boldsymbol{v}_r$ lar man være lik $\boldsymbol{x}_r$ minus de ortogonale projeksjonene av $\boldsymbol{x}_r$ ned på hver $\{\boldsymbol{v}_i\}_{i=1}^{r-1}$. Dette gir følgende algoritme (Lay,

2014 [17]):

$$\begin{aligned}
\boldsymbol{v}_1 &= \boldsymbol{x}_1, \\
\boldsymbol{v}_2 &= \boldsymbol{x}_2 - \frac{\boldsymbol{x}_2^T \boldsymbol{v}_1}{\|\boldsymbol{v}_1\|^2}\boldsymbol{v}_1, \\
\boldsymbol{v}_3 &= \boldsymbol{x}_3 - \frac{\boldsymbol{x}_3^T \boldsymbol{v}_1}{\|\boldsymbol{v}_1\|^2}\boldsymbol{v}_1 - \frac{\boldsymbol{x}_3^T \boldsymbol{v}_2}{\|\boldsymbol{v}_2\|^2}\boldsymbol{v}_2, \\
&\vdots \\
\boldsymbol{v}_p &= \boldsymbol{x}_p - \sum_{j=1}^{p-1} \frac{\boldsymbol{x}_p^T \boldsymbol{v}_j}{\|\boldsymbol{v}_j\|^2}\boldsymbol{v}_j.
\end{aligned}$$

Hver $\boldsymbol{v}_r$ vil være en lineærkombinasjon av $\{\boldsymbol{x}_j\}_{j=1}^r$, og dermed er $\{\boldsymbol{v}_j\}_{j=1}^p$ en ortogonal basis for søylerommet til X.

Likningene over kan alternativt uttrykkes ved

$$\boldsymbol{x}_1 = \boldsymbol{v}_1,$$

og

$$\boldsymbol{x}_r = \boldsymbol{v_r} + \sum_{i=1}^{r-1} \frac{\boldsymbol{x}_r^T \boldsymbol{v}_i}{\|\boldsymbol{v}_i\|^2}\boldsymbol{v}_i \tag{2.1}$$

der $2 \leq r \leq p$.

Ved QR-faktorisering bruker man basisvektorer $\boldsymbol{q}_r = \frac{1}{\|\boldsymbol{v}_r\|}\boldsymbol{v}_r$ slik at disse er enhetsvektorer. Dette gjør at man kan skrive søylevektorene i X som

$$\boldsymbol{x}_1 = k_{11}\boldsymbol{q}_1,$$

og

$$\boldsymbol{x}_r = k_{rr}\boldsymbol{q}_r + \sum_{j=1}^{r-1} k_{jr}\boldsymbol{q}_j.$$

Her vil hver $k_{rr} = 1/\|\boldsymbol{v}_r\| \geq 0$, mens de øvrige k-ene vil være koeffisienter fra 2.1 multiplisert med $\|\boldsymbol{v}_i\|$. Hver av søylevektorene $\boldsymbol{x}_r$ kan dermed uttrykkes som $Q\boldsymbol{k}_r$, der $\boldsymbol{k}_r = [k_{1r} \quad k_{2r} \quad \cdots \quad k_{rr} \quad 0 \quad \cdots \quad 0]^T$. Dette gjør at man kan uttrykke X som

$$X = [Q\boldsymbol{k}_1 \quad Q\boldsymbol{k}_2 \quad \cdots \quad Q\boldsymbol{k}_p] = Q[\boldsymbol{k}_1 \quad \boldsymbol{k}_2 \quad \cdots \quad \boldsymbol{k}_p] = QR,$$

der R består av søylevektorene $\{\boldsymbol{k}_j\}_{j=1}^p$ (Lay, 2014 [17]).

2.2 Singulærverdidekomposisjon (SVD)

En annen viktig metode for faktorisering av matriser, og som har mange viktige anvendelser, er *singulærverdidekomposisjon* (SVD). Enhver matrise $X \in \mathbb{R}^{n\times p}$ som har full rang, kan dekomponeres slik at

$$X = USV^T,$$

der $U \in \mathbb{R}^{n\times n}$, $S \in \mathbb{R}^{n\times p}$ og $V \in \mathbb{R}^{p\times p}$. Matrisene $U = [\boldsymbol{u}_1 \quad \boldsymbol{u}_2 \quad \cdots \quad \boldsymbol{u}_n]$ og $V = [\boldsymbol{v}_1 \quad \boldsymbol{v}_2 \quad \cdots \quad \boldsymbol{v}_p]$ har søyler med følgende egenskaper

$$\boldsymbol{u}_j^T \boldsymbol{u}_k = 0 \ \forall \ \{j,k\} \subset \{1,2,\cdots,n\}, \tag{2.2}$$

$$\|\boldsymbol{u}_j\| = 1 \ \forall \ j \in \{1,2,\cdots,n\}; \tag{2.3}$$

$$\boldsymbol{v}_j^T \boldsymbol{v}_k = 0 \ \forall \ \{j,k\} \subset \{1,2,\cdots,p\}, \tag{2.4}$$

$$\|\boldsymbol{v}_j\| = 1 \ \forall \ j \in \{1,2,\cdots,p\}. \tag{2.5}$$

$X^TX \in \mathbb{R}^{p\times p}$ vil være en symmetrisk matrise. Ved å la $\{\boldsymbol{v}_j\}_{j=1}^p$ være en ortonormal basis for $\mathbb{R}^p$ bestående av egenvektorer til X^TX og med $\{\lambda_j\}_{j=1}^p$ som tilhørende egenverdier, vil

$$\begin{aligned} 0 \leq \|X\boldsymbol{v}_j\|^2 = (X\boldsymbol{v}_j)^T X\boldsymbol{v}_j = \boldsymbol{v}_j^T X^T X \boldsymbol{v}_j \\ = \boldsymbol{v}_j^T \lambda_j \boldsymbol{v}_j = \lambda_j \end{aligned}$$

Man ordner egenverdiene $\{\lambda_j\}_{j=1}^p$ slik at

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0.$$

Singulærverdiene $\{\sigma_j\}_{j=1}^p$ til matrisa X er kvadratrota av egenverdiene til X^TX:

$$\{\sigma_j\}_{j=1}^p = \left\{\sqrt{\lambda_j}\right\}_{j=1}^p = \{\|X\boldsymbol{v}_j\|\}_{j=1}^p.$$

Man konstruerer så den ortonormale mengden

$$\{\boldsymbol{u}_i\}_{i=1}^p = \left\{\frac{1}{\|X\boldsymbol{v}_j\|} X\boldsymbol{v}_j\right\}_{j=1}^p = \left\{\frac{1}{\sigma_j} X\boldsymbol{v}_j\right\}_{j=1}^p \Rightarrow \{\sigma_j \boldsymbol{u}_j\}_{j=1}^p = \{X\boldsymbol{v}_j\}_{j=1}^p$$

Denne mengden kan utvides med $n-p$ ortonormale vektorer til $\{\boldsymbol{u}\}_{i=1}^n$, som blir en ortonormal basis for $\mathbb{R}^n$. Ved så å innføre den diagonale matrisa $\Sigma = \text{diag}([\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_p])$ og

utvidelsen $S = [\Sigma \quad 0^T]^T$, der 0 er en nullmatrise i $\mathbb{R}^{(n-p)\times p}$, samt $U = [\boldsymbol{u}_1 \quad \boldsymbol{u}_2 \quad \cdots \quad \boldsymbol{u}_n]$ og $V = [\boldsymbol{v}_1 \quad \boldsymbol{v}_2 \quad \cdots \quad \boldsymbol{v}_p]$, får man

$$US = [\sigma_1 \boldsymbol{u}_1 \quad \sigma_2 \boldsymbol{u}_2 \quad \cdots \quad \sigma_p \boldsymbol{u}_p] = XV.$$

Høyremultiplikasjon med V^T vil nå gi

$$USV^T = XVV^T = X$$

siden V er en ortonormal kvadratisk matrise (Lay, 2014 [17]). Dersom $n < p$ og X^T har full rang, gjør man analysen tilsvarende på X^T.

2.3 Minste kvadraters metode (OLS)

Regresjon brukes for å finne sammenhenger mellom to eller flere numeriske størrelser. Det enkleste eksempelet har én variabel, og målet med regresjonen er å finne en énvariabel sammenheng mellom to størrelser

$$y = \beta x \tag{2.6}$$

for proporsjonale størrelser, eventuelt

$$y = \beta_1 x + \beta_0 \tag{2.7}$$

for generelle lineære sammenhenger. Her er y responsvariabelen, x er forklaringsvariabelen og $\beta_{(j)}$ er parametrene som skal bestemmes. I anvendelser møter man også ofte på problemer av typen

$$y = \beta f(x), \tag{2.8}$$

der $f : X \mapsto Y$ er en arbitrær funksjon med $X \subseteq \mathbb{R}$ og $Y \subseteq \mathbb{R}$.

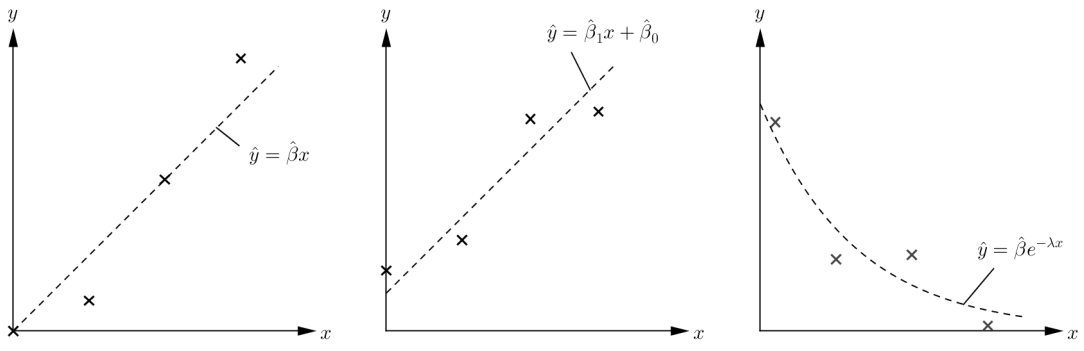
Det vil alltid være en viss uoverenstemmelse mellom modell og data, og det vil alltid være en viss usikkerhet knyttet til β. Det er derfor vanlig å innføre hattenotasjon, slik at modellene for (2.6), (2.7) og (2.8) blir

$$\hat{y} = \hat{\beta} x, \tag{2.9}$$

$$\hat{y} = \hat{\beta}_1 x + \beta_0, \tag{2.10}$$

$$\hat{y} = \hat{\beta} f(x), \tag{2.11}$$

der hattene angir at størrelsene er estimater. Man estimerer først $\beta_{(k)}$ og regner deretter ut $\hat{y}(x_0)$ for den x_0 man ønsker å predikere y for.



(a) Proporsjonal sammenheng.

(b) Lineær sammenheng.

(c) Ikke-lineær sammenheng med kjent λ.

Figur 2.1: Eksempler på grafisk fremstilling av énvariable modeller med målepunkter og regresjonskurve (striplet).

For det proporsjonale tilfellet med n datapunkter kalles differansen

$$\epsilon_i = y_i - \hat{y}_i = y_i - \hat{\beta}x_i, \quad i \in \{1, 2, \cdots, n\}$$

residual i, og man har analoge sammenhenger for de andre tilfellene. Dette gjør at måleverdiene kan uttrykkes som henholdsvis

$$\begin{aligned} y_i &= \hat{\beta}x_i + \epsilon_i, \\ y_i &= \hat{\beta}_1 x_i + \hat{\beta}_0 + \epsilon_i, \\ y_i &= \hat{\beta}f(x_i) + \epsilon_i. \end{aligned}$$

Det ikke-lineære tilfellet i figur 2.1c kan gjøres lineært ved å se på sammenhengen mellom y og f heller enn mellom y og x. Dersom vi setter $\beta_0 = 0$, er modellene i figur 2.1a og 2.1c analoge med den i figur 2.1b.

Vanlig minste kvadraters (eng: *ordinary least squares* (OLS)) metode brukes for å finne regresjonskurven ved å finne verdiene til $\beta_{(j)}$ i likning (2.9), (2.10) og (2.11). I det generelle lineære tilfellet

$$y_i = \hat{\beta}_1 x_i + \hat{\beta}_0 + \epsilon_i$$

vil minste kvadraters metode gå ut på å finne de $(\hat{\beta}_0,\hat{\beta}_1)$ som løser optimeringsproblemet

$$\min_{(\hat{\beta}_0,\hat{\beta}_1)} \sum_{i=1}^{n} \epsilon_i^2 = \min_{(\hat{\beta}_0,\hat{\beta}_1)} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \min_{(\hat{\beta}_0,\hat{\beta}_1)} \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2. \tag{2.12}$$

når man har n målinger av x og y. Ved å la $w = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$ vil løsningen av (2.12) være de $(\hat{\beta}_0, \hat{\beta}_1)$ som gir $\nabla w = 0$:

$$\begin{aligned}
\frac{\partial w}{\partial \hat{\beta}_0} &= \sum_{i=1}^{n} 2(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0 \\
\sum_{i=1}^{n} y_i - n\hat{\beta}_0 - \hat{\beta}_1 \sum_{i=1}^{n} x_i &= 0 \\
\hat{\beta}_0 &= \frac{1}{n}\left(\sum_{i=1}^{n} y_i - \hat{\beta}_1 \sum_{i=1}^{n} x_i \right). \qquad (2.13)
\end{aligned}$$

Videre blir

$$\begin{aligned}
\frac{\partial w}{\partial \hat{\beta}_1} &= \sum_{i=1}^{n} 2(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0 \\
\sum_{i=1}^{n} x_i y_i - \hat{\beta}_0 \sum_{i=1}^{n} x_i - \hat{\beta}_1 \sum_{i=1}^{n} x_i^2 &= 0 \\
&\Downarrow (2.13) \\
n \sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} y_i - \hat{\beta}_1 \sum_{i=1}^{n} x_i \right) \sum_{i=1}^{n} x_i - n\hat{\beta}_1 \sum_{i=1}^{n} x_i^2 &= 0 \\
\hat{\beta}_1 &= \frac{n \sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}
\end{aligned}$$

Man vil alltid ha like mange β-parametere som variable for proporsjonale tilfeller. For mer enn én variabel blir det mange likninger å løse, og man går over til å bruke lineær algebra.

2.3.1 Flervariabel minste kvadraters metode

Ved å ordne datapunktene i to vektorer $\boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]^T$ og $\boldsymbol{y} = [y_1 \quad y_2 \quad \cdots \quad y_n]^T$, og der (x_1, y_1) er et datapunkt, vil modellen kunne uttrykkes som

$$\hat{\boldsymbol{y}} = \boldsymbol{x}\hat{\beta},$$

for det proporsjonale tilfellet. Dersom man definerer

$$X = [\mathbf{1} \quad \boldsymbol{x}],$$

der $X \in \mathbb{R}^{n\times 2}$ og $\mathbf{1} = [1 \quad 1 \quad \cdots \quad 1]^T \in \mathbb{R}^n$, kan den lineære modellen skrives som

$$\boldsymbol{y} = \begin{bmatrix}\mathbf{1} & \boldsymbol{x}\end{bmatrix} \begin{bmatrix}\hat{\beta}_0 \\ \hat{\beta}_1\end{bmatrix} = X\hat{\boldsymbol{\beta}}.$$

Dette kan utvides til flere variable ved å la

$$X = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p]$$

og

$$\hat{\boldsymbol{\beta}} = [\hat{\beta}_1 \quad \hat{\beta}_2 \quad \cdots \quad \hat{\beta}_p]^T$$

uten konstantleddet, eller

$$X = [\mathbf{1} \quad \boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p]$$

og

$$\hat{\boldsymbol{\beta}} = [\hat{\beta}_0 \quad \hat{\beta}_1 \quad \hat{\beta}_2 \quad \cdots \quad \hat{\beta}_p]^T.$$

Man ønsker å finne den modellen som gjør differansen mellom målte y_j-verdier og modell-predikerte $\hat{y}$-verdier så liten som mulig, og den $\hat{\boldsymbol{\beta}}$ som løser dette vil være den ortogonale projeksjonen av $\boldsymbol{y}$ på søylerommet til X. Likningen $\boldsymbol{y} = X\boldsymbol{\beta}$ er generelt ikke løsbar, så man ønsker å finne den $\hat{\boldsymbol{\beta}}$ som gjør at avstanden mellom $\boldsymbol{y}$ og $\hat{\boldsymbol{y}} = X\hat{\boldsymbol{\beta}}$ blir minst mulig (figur 2.2). Dette finner man ved å minimere $\|\boldsymbol{y} - X\boldsymbol{\beta}\|$, som vil være det samme som å minimere $f(\boldsymbol{\beta}; \boldsymbol{y}, X) = \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 = (\boldsymbol{y} - X\boldsymbol{\beta})^T(\boldsymbol{y} - X\boldsymbol{\beta})$. Det er alltid mulig å gjøre et matrise-vektor-produkt om til et rent vektoruttrykk

$$\boldsymbol{y} - X\boldsymbol{\beta} = \boldsymbol{y} - [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p] \begin{bmatrix}\beta_1 \\ \beta_2 \\ \vdots \\ \beta_p\end{bmatrix} = \boldsymbol{y} - \sum_{j=1}^{p} \beta_j \boldsymbol{x}_j,$$

og minimering av et slikt flervariabelt uttrykk er gitt ved $\nabla f = 0$ der ∇-operatoren er de $\hat{\beta}$-deriverte

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial \hat{\beta}_1} & \frac{\partial}{\partial \hat{\beta}_2} & \cdots & \frac{\partial}{\partial \hat{\beta}_p} \end{bmatrix}^T.$$

Den β_r-deriverte av f vil være

$$\begin{aligned}
\frac{\partial f}{\partial \hat{\beta}_r} &= \frac{\partial}{\partial \hat{\beta}_r} \left[(\boldsymbol{y} - X\hat{\boldsymbol{\beta}})^T (\boldsymbol{y} - X\hat{\boldsymbol{\beta}}) \right] \\
&= \frac{\partial}{\partial \hat{\beta}_r} \left(\boldsymbol{y}^T \boldsymbol{y} - \boldsymbol{y}^T X\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^T X^T \boldsymbol{y} + \hat{\boldsymbol{\beta}}^T X^T X \hat{\boldsymbol{\beta}} \right) \\
&= \frac{\partial}{\partial \hat{\beta}_r} \left(\boldsymbol{y}^T \boldsymbol{y} \right) - \frac{\partial}{\partial \hat{\beta}_r} \left(\boldsymbol{y}^T X\hat{\boldsymbol{\beta}} \right) - \frac{\partial}{\partial \hat{\beta}_r} \left(\hat{\boldsymbol{\beta}}^T X^T \boldsymbol{y} \right) + \frac{\partial}{\partial \hat{\beta}_r} \left(\hat{\boldsymbol{\beta}}^T X^T X \hat{\boldsymbol{\beta}} \right) \\
&= 0 - \frac{\partial}{\partial \hat{\beta}_r} \left(\boldsymbol{y}^T \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j \right) - \frac{\partial}{\partial \hat{\beta}_r} \left(\sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j^T \boldsymbol{y} \right) + \frac{\partial}{\partial \hat{\beta}_r} \left[\left(\sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j^T \right) \left(\sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x} \right) \right] \\
&= -\boldsymbol{y}^T \boldsymbol{x}_r - \boldsymbol{x}_r^T \boldsymbol{y} + \boldsymbol{x}_r^T \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j + \sum_{j=1}^{p} \boldsymbol{x}_j^T \hat{\beta}_j \boldsymbol{x}_r \\
&= 2\boldsymbol{x}_r^T \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j - 2\boldsymbol{x}_r^T \boldsymbol{y},
\end{aligned}$$

der $r \in \{1, 2, \cdots, p\}$. Dette gir

$$\nabla f = \begin{bmatrix} 2\boldsymbol{x}_1^T \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j - 2\boldsymbol{x}_1^T \boldsymbol{y} \\ 2\boldsymbol{x}_2^T \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j - 2\boldsymbol{x}_2^T \boldsymbol{y} \\ \vdots \\ 2\boldsymbol{x}_p^T \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j - 2\boldsymbol{x}_p^T \boldsymbol{y} \end{bmatrix} = 2X^T X\hat{\boldsymbol{\beta}} - 2X^T \boldsymbol{y} = 0 \tag{2.14}$$

som igjen fører til at

$$X^T X\hat{\boldsymbol{\beta}} = X^T \boldsymbol{y}.$$

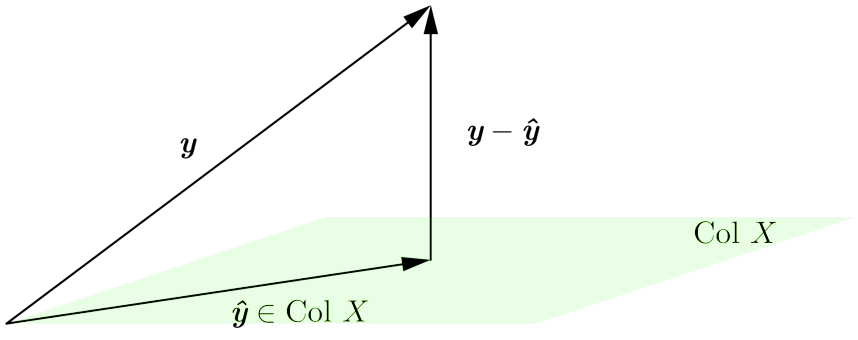
Dersom X har full rang slik at $X^T X$ er invertibel, kan man finne $\hat{\boldsymbol{\beta}}$ ved

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}, \tag{2.15}$$

og de modellpredikerte y-verdiene $\hat{\boldsymbol{y}}$ ved

$$X\hat{\boldsymbol{\beta}} = X(X^T X)^{-1} X^T \boldsymbol{y}$$

Utledningen over er delvis hentet fra Hansen et al. (2013) [11].



Figur 2.2: Geometrisk fremstilling av et minste kvadraters optimeringsproblem.

Geometrisk vil $\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|$ være den euklidske avstanden fra punktet $\boldsymbol{y}$ til $\hat{\boldsymbol{y}}$. Av figur 2.2 kan man se at $\hat{\boldsymbol{y}}$ vil være den ortogonale projeksjonen av $\boldsymbol{y}$ ned på søylerommet til X.

For at en matrise A skal være en projeksjon ned på et spesifikt vektorrom W, må den være slik at når transformerer enhver vektor $\boldsymbol{u} \in W$, vil dette gi $\boldsymbol{u}$ selv:

$$A\boldsymbol{u} = \boldsymbol{u}.$$

Videre må det være slik at ved å transformere hvilken som helst vektor $\boldsymbol{v}$ i eller utenfor W to ganger, vil være det samme som å transformere $\boldsymbol{v}$ én gang:

$$AA\boldsymbol{v} = A(A\boldsymbol{v}) = A\boldsymbol{v},$$

siden $A\boldsymbol{v} \in W$. Dette betyr at projeksjonsmatrisa A er *idempotent*, altså at $A \equiv A^2$.

Det viser seg at $X(X^TX)^{-1}X^T$ er idempotent:

$$\begin{aligned}\left[X(X^TX)^{-1}X^T\right]^2 &= X(X^TX)^{-1}X^TX(X^TX)^{-1}X \\ &= X(X^TX)^{-1}(X^TX)(X^TX)^{-1}X \\ &= X(X^TX)^{-1}X.\end{aligned}$$

Hvis $A\boldsymbol{v}$ er en ortogonal transformasjon av vektoren $\boldsymbol{v}$ må det være slik at differansen $A\boldsymbol{v} - \boldsymbol{v}$ er ortogonal med $A\boldsymbol{v}$. I figur 2.2 tilsvarer dette henholdsvis vektorene $\hat{\boldsymbol{y}} - \boldsymbol{y}$ og

$\hat{\boldsymbol{y}}$. Dette viser seg også å gjelde for $X(X^TX)^{-1}X^T$:

$$\begin{aligned}
&(\boldsymbol{v} - X(X^TX)^{-1}X^T\boldsymbol{v})^TX(X^TX)^{-1}X^T\boldsymbol{v} \\
&= \boldsymbol{v}^TX(X^TX)^{-1}X^T\boldsymbol{v} - \left[X(X^TX)^{-1}X^T\ \boldsymbol{v}\right]^TX(X^TX)^{-1}X^T\boldsymbol{v} \\
&= \boldsymbol{v}^TX(X^TX)^{-1}X^T\boldsymbol{v} - \boldsymbol{v}^TX(X^TX)^{-1}X^TX(X^TX)^{-1}X^T\boldsymbol{v} \\
&= \boldsymbol{v}^TX(X^TX)^{-1}X^T\boldsymbol{v} - \boldsymbol{v}^TX(X^TX)^{-1}X^T\boldsymbol{v} = 0,
\end{aligned}$$

og $(X^TX)^{-1}X^T$ er dermed en ortogonal projeksjonsmatrise.

Ved å QR-faktorisere X, vil man kunne uttrykke $\hat{\boldsymbol{\beta}}$ noe enklere:

$$\begin{aligned}
\hat{\boldsymbol{\beta}} = (X^TX)^{-1}X^T\boldsymbol{y} &= [(QR)^TQR]^{-1}(QR)^T\boldsymbol{y} \\
&= (R^TQ^TQR)^{-1}R^TQ^T\boldsymbol{y} \\
&= (R^TR)^{-1}R^TQ^T\boldsymbol{y} \\
&= R^{-1}(R^T)^{-1}R^TQ^T\boldsymbol{y} \\
&= R^{-1}Q^T\boldsymbol{y}, \qquad (2.16)
\end{aligned}$$

og $\hat{\boldsymbol{y}}$ kan dermet uttrykkes

$$\hat{\boldsymbol{y}} = QR\hat{\boldsymbol{\beta}}\boldsymbol{y} = QRR^{-1}Q^T\boldsymbol{y} = QQ^T\boldsymbol{y}. \qquad (2.17)$$

Tilsvarende kan man ved SVD av X uttrykke $\hat{\boldsymbol{\beta}}$ som

$$\begin{aligned}
\hat{\boldsymbol{\beta}} = (X^TX)^{-1}X^T\boldsymbol{y} &= [(USV^T)^TUSV^T]^{-1}(USV^T)^T\boldsymbol{y} \\
&= (VS^TU^TUSV^T)^{-1}VS^TU^T\boldsymbol{y} \\
&= (VS^TSV^T)^{-1}VS^TU^T\boldsymbol{y} \\
&= (V\Sigma^2V^T)^{-1}VS^TU^T\boldsymbol{y} \\
&= V\Sigma^{-2}V^{-1}VS^TU^T\boldsymbol{y} \\
&= V[\Sigma^{-1} \quad 0^T]U^T\boldsymbol{y} \\
&= V\Sigma^{-1}U_{(p)}^T\boldsymbol{y}, \qquad (2.18)
\end{aligned}$$

der $U_{(p)}$ inneholder de p første kolonnene av U. Tilvarende vil $\hat{\boldsymbol{y}}$ kunne uttrykkes som

$$\begin{aligned}
\hat{\boldsymbol{y}} &= USV^TV[\Sigma^{-1} \quad 0^T]U^T\boldsymbol{y} \\
&= U\begin{bmatrix}\Sigma \\ 0\end{bmatrix}[\Sigma^{-1} \quad 0]U^T\boldsymbol{y} = U_{(p)}U_{(p)}^T\boldsymbol{y}. \qquad (2.19)
\end{aligned}$$

Det er derfor nok med de p første kolonnene av U-matrisa for å lage regresjonsmodeller med SVD. MATLAB har en såkalt økonomisk SVD-variant, som kun regner ut $U_{(p)}$. I resten av denne oppgaven vil jeg skrive U når jeg mener $U_{(p)}$.

2.4 Prinsipalkomponentanalyse (PCA)

Et krav for å kunne regne ut minste kvadraters løsning av $\hat{\boldsymbol{\beta}}$, er at X har full rang slik at X^TX er en invertibel matrise, jf (2.15).

Kovariansen til to variable x_1 og x_2 er definert som

$$\text{Cov}(x_1, x_2) = \frac{1}{n-1}\sum_{i=1}^{n}(x_{1i} - \mu_1)(x_{2i} - \mu_2),$$

der x_{1i} og x_{2i} er målinger av hver variabel, n er antall målinger og μ_1 og μ_2 er midlene til henholdsvis x_1 og x_2. Når man har flere variable, er det vanlig å samle den totale variabiliteten til datasettet i en kovariansmatrise

$$\text{Cov}(X) = \frac{1}{n}X_s^TX_s,$$

der X_s er datamatrisa med middelet hver variablene er trukket fra korresponderende søyle i X. Dette tilsvarer kovarianser i alle elementer, og diagonalelementene blir kovariansen mellom to like variabler, det vil si variansen. Når man ikke har tilgang til alle data for en hel populasjon, og estimerer kovariansen utfra et utvalg, vil dette estimatet bli

$$\widehat{\text{Cov}(X)} = \frac{1}{n-1}X_s^TX_s. \tag{2.20}$$

I resten av dette delkapittelet antar jeg at X er en slik sentrert matrise.

Prisipalkomponentanalyse (eng: *principal component analysis*, PCA) går ut på å utelate de delene av datamengden som bidrar til lite variabilitet. Man konstruerer nye ortogonale retninger i datamengden som følger de retningene med størst varians.

Man utfører SVD på $X = USV^T$, der søylene i V, $\{\boldsymbol{v}_i\}_{i=1}^p$, er en ortonormal basis for $\mathbb{R}^n$. For alle $\boldsymbol{v}_r$ og $\boldsymbol{v}_k$ med $r \neq k$ gjelder at

$$(X\boldsymbol{v}_r)^TX\boldsymbol{v}_k = \boldsymbol{v}_r^TX^TX\boldsymbol{v}_k = \boldsymbol{v}_r^T\lambda_k\boldsymbol{v}_k = 0,$$

og dermed er også $\{X\boldsymbol{v}_j\}_{j=1}^p$ en ortogonal mengde. Hvis X har r singulærverdier forskjellig fra 0, vil $\{X\boldsymbol{v}_j\}_{j=1}^r$ være lineært uavhengige, da alle vektorene i denne mengden er forskjellig fra 0. Enhver vektor $\boldsymbol{w} \in \mathbb{R}^p$ vil nå kunne skrives som en lineærkombinasjon $\sum_{j=1}^p c_j\boldsymbol{v}_j$. $X\boldsymbol{w} \in \text{Col } X$ kan skrives som

$$X\left(\sum_{j=1}^{p} c_j\boldsymbol{v}_j\right) = X\left(\sum_{j=1}^{r} c_j\boldsymbol{v}_j\right) + X\left(\sum_{j=r+1}^{p} c_j\boldsymbol{v}_j\right) = X\left(\sum_{j=1}^{r} c_j\boldsymbol{v}_j\right),$$

fordi $X\boldsymbol{v}_j = \sigma_j\boldsymbol{u}_j = 0 \;\forall\; j > r$. $\{X\boldsymbol{v}_j\}_{j=1}^r$ er dermed en ortogonal basis for søylerommet til X. Ved å bruke matrisa

$$Z = [X\boldsymbol{v}_1 \quad X\boldsymbol{v}_2 \quad \cdots \quad X\boldsymbol{v}_r] = XV = U\Sigma V^TV = U\Sigma$$

som utgangspunkt for analysen, vil man beholde all variabilitet blant søylene i X, og den tilhørende kovariansmatrisa vil være

$$\Sigma_Z = \frac{1}{n-1}Z^TZ = \frac{1}{n-1}(XV)^TXV = \frac{1}{n-1}(U\Sigma)^TU\Sigma = \frac{1}{n-1}\Sigma U^TU\Sigma = \frac{1}{n-1}\Sigma^2.$$

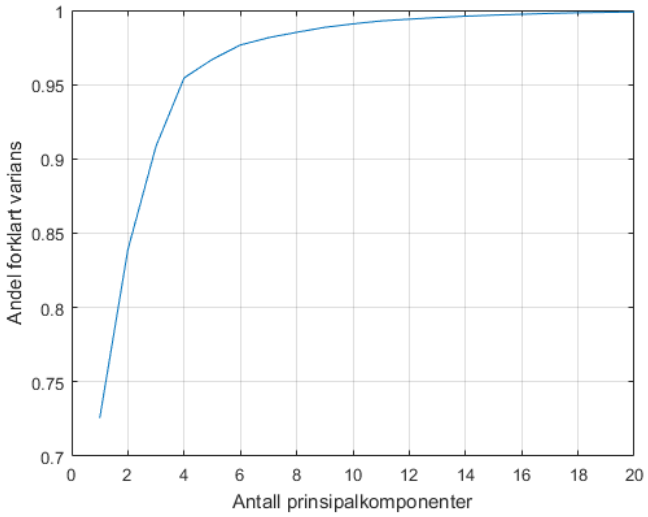
Dette er en diagonal matrise med kun varianser mellom de ortogonale $X\boldsymbol{v}_i$-retningene på diagonalen, og veridene er en skalering av de kvaderte singulærverdiene til X_s-matrisa. Denne kovariansmatrisa inneholder all variabilitet blant søylene til X. Fordi singulærverdiene er ordnet i synkende rekkefølge, vil de første diagonalelementene beskrive store deler av variansen. Ved kun å bruke et lite antall $k < \min(n, p)$ av singulærverdiene, kan man konstruere $U_{(k)} = [\boldsymbol{u}_1 \quad \boldsymbol{u}_1 \quad \cdots \quad \boldsymbol{u}_k]$, $S_{(k)} = \text{diag}([\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_k])$ og $V_{(k)} = [\boldsymbol{v}_1 \quad \boldsymbol{v}_2 \quad \cdots \quad \boldsymbol{v}_k]$. Ved så å bruke produktet $U_{(k)}S_{(k)}V_{(k)}^T$ som datamatrise, og ved samme utledning som for (2.18) og (2.19) vil man kunne modellere $\hat{\boldsymbol{\beta}}$ til å bli

$$\hat{\boldsymbol{\beta}} = V_{(k)}S_{(k)}^{-1}U_{(k)}^T$$

og $\hat{\boldsymbol{y}}$ som

$$\hat{\boldsymbol{y}} = U_{(k)}U_{(k)}^T\boldsymbol{y}$$

Dette kalles *prinsipalkomponentregresjon* (eng. principal component regression, PCR). Her vil man utelate en del av datamengden, men det man utelater er data som bidrar lite til den totale variasjonen i datamengden. Figur 2.3 viser antall prinsipalkomponenter mot andel forklart varians for et NIR-datasett 701 variable og 60 målinger (Kalivas, 1997 [16]). Datasettet er forklart nærmere i delkapittel 3.3.1.



Figur 2.3: Andel forklart varians plottet mot antall prinsipalkomponenter for de 20 første prinsipalkomponentene til Spectra-datasettet (delkapittel 3.3.1 og Kalivas (1997 [16])).

2.5 Delvis minste kvadraters metode (PLS)

Et mulig problem med PCA er komponentene konstrueres utelukkende utfra X-matrisa, mens når man skal gjøre regresjon ønsker man å se på sammenhengen mellom X-dataene og responsdataene i en $\boldsymbol{y}$-vektor. Kurven i figur 2.3 har et knekkpunkt på omlag 5 komponenter, der over 95 % av variansen i dataene er forklart. Det er imidlertid ingen garanti for at noe av den variasjonen som korrelerer med responsen $\boldsymbol{y}$ ligger i variansen forklart av disse 5 dominerende komponentene.

Delvis minste kvadraters metode (eng: partial least squares, PLS) er en annen metode for å finne såkalt latente variable i datamengden. I motsetning til PCA, som kun tar for seg X-matrisa, bruker man også $\boldsymbol{y}$ for å finne PLS-komponentene. Mens PCA produserer de ortogonale retningene i datamengden som forklarer mest variabilitet blant forklaringsvariablene, er PLS-komponentene de ortogonale retningene i datamengden som er mest korrelert med $\boldsymbol{y}$. Når korrelasjonen med $\boldsymbol{y}$ ikke finnes langs de samme retningene som det meste av den innbyrdes kovariansen i X, vil man kunne bygge modeller med høy prediksjonsevne av relativt få PLS-komponenter der man trenger mange PCA-komponenter.

Det er flere metoder for å finne PLS-komponentene, der den mest brukte er NIPALS (Wold et al., 2001 [25]): Den første PLS-komponentvektoren er den første egenvektoren

$\boldsymbol{w}_1$ til $(\boldsymbol{y}^T X)^T(\boldsymbol{y}^T X)$, det vil si den egenvektoren som tilsvarer den største egenverdien. Dette tilsvarer den første $\boldsymbol{v}$-vektoren av SVDen man gjennomfører for å finne prinsipalkomponentene i PCA. Denne kan man finne ved

$$\boldsymbol{w}_1 = \frac{1}{\|X^T\boldsymbol{y}\|} X^T\boldsymbol{y}.$$

Man finner så elementene $\boldsymbol{t}_1$ til X-punktene i $\boldsymbol{w}$-retningen ved

$$\boldsymbol{t}_1 = \frac{1}{\|X\boldsymbol{w}\|} X\boldsymbol{w}_1$$

og projeksjonen $\boldsymbol{p}_1$ av X til $\boldsymbol{w}_1$

$$\boldsymbol{p}_1 = X^T\boldsymbol{t}$$

Ved å omforme X ved

$$X \leftarrow X - \boldsymbol{t}_1\boldsymbol{p}_1^T$$

sikrer man at den nye X-matrisa er ortogonal til $\boldsymbol{t}_1$. Man kan så lage regresjonskoeffisienten for $\boldsymbol{t}_1$

$$q_1 = \boldsymbol{t}_1^T\boldsymbol{y}$$

og omforme $\boldsymbol{y}$ slik at denne er ortogonal til $\boldsymbol{t}_1$:

$$\boldsymbol{y} \leftarrow \boldsymbol{y} - q_1\boldsymbol{t}_1.$$

Man gjentar deretter prosessen til man har tilstrekkelig antall komponenter, og ordner vektorene og verdiene man har funnet slik at matrisa T inneholder alle $\boldsymbol{t}$-vektorene som søyler, W alle $\boldsymbol{w}$-vektorene, P alle $\boldsymbol{p}$-vektorene og vektoren $\boldsymbol{q}$ inneholder regresjonskoeffisientene q_i som elementer. De predikerte $\boldsymbol{y}$-verdiene blir nå

$$\hat{\boldsymbol{y}} = TT^T\boldsymbol{y} = T\boldsymbol{q}. \tag{2.21}$$

XW og T spenner ut det samme søylerommet, slik at

$$\begin{aligned} TT^TXW &= XW \\ TP^TW &= XW \\ T &= XW(P^TW)^{-1} \end{aligned}$$

Fra (2.21) kan man finne regresjonskoeffisientene $\hat{\boldsymbol{\beta}}$

$$\begin{aligned} \hat{\boldsymbol{y}} = X\hat{\boldsymbol{\beta}} &= T\boldsymbol{q} \\ X\hat{\boldsymbol{\beta}} &= XW(P^TW)^{-1}\boldsymbol{q} \\ \hat{\boldsymbol{\beta}} &= W(P^TW)^{-1}\boldsymbol{q} \end{aligned}$$

(Indahl, 2015 [14]).

2.6 Tikhonov-regularisering

Som nevnt i innledningen, gjør man Tikhonov-regularisering for å gjøre modellen mer stabil. Optimeringsproblemet går fra $\min \|X\hat{\boldsymbol{\beta}} - \boldsymbol{y}\|^2$ til $\min(\|X\hat{\boldsymbol{\beta}} - \boldsymbol{y}\|^2 + \lambda\|T\hat{\boldsymbol{\beta}}\|^2)$, der T er en regulariseringsmatrise. I innledningen tok jeg for meg Ridge-regularisering, der $T = I_p$. Det er også interessant å ta for seg andre Tikhonov-regulariseringer. Ved å la

$$T_1 = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \in \mathbb{R}^{m\times m}, \tag{2.22}$$

vil $\lambda\|T\hat{\boldsymbol{\beta}}\|$ bli stor dersom $\hat{\beta}_i$-koeffisientene viser en liten grad av kontinuitet, og man kan se på dette som en form for derivasjon. Videre vil man ved å la regulariseringsmatrisa være

$$T_2 = \begin{bmatrix} -1 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 2 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{m\times m}, \tag{2.23}$$

«straffe» på den andrederiverte av den tenkte $\hat{\beta}(\lambda)$-funksjonen, og minimeringen her vil favorisere $\hat{\boldsymbol{\beta}}$-kandidater som gir en kontinuerlig $\hat{\beta}'(\lambda)$. Jeg har også sett på tilfeller der

regulariseringsmatrisa er

$$T_3 = \begin{bmatrix} -1 & 3 & -3 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -3 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & -1 & 3 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 3 & -3 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{m \times m} \tag{2.24}$$

som vil være en slags tredjederivert-operator. Denne vil straffe på liten grad av kontinuitet i $\hat{\beta}''(\lambda)$. Figur 2.4 på neste side viser plott av $\hat{\boldsymbol{\beta}}$ produsert ved hver av de fire regulariseringsmetodene for et NIR-datasett (Spectra, se delkapittel 3.3.1) (Kalivas, 1997 [16]). Her ser man tydelig at T_1-, T_2- og T_3-regulariserte modeller gir mye glattere regresjonskoeffisientvektorer enn den Ridge-regulariserte.

Det er typen data man behandler som bestemmer om man kan forvente at $\boldsymbol{\beta}$ er kontinuerlig, eventuelt i kontinuerlig i de deriverte. I denne oppgaven vil jeg se på en mengde nær-infrarød spektroskopi-data, der hver variabel er et intervall av bølgelengder på i det elektromagnetiske spekteret. Strålingsspektra er noe man forventer at er kontinuerlig, og det gir derfor mening å regularisere med de deriverte for slike datasett.

Det er mulig å uttrykke optimeringsproblemet fra (1.1) på en enklere måte:

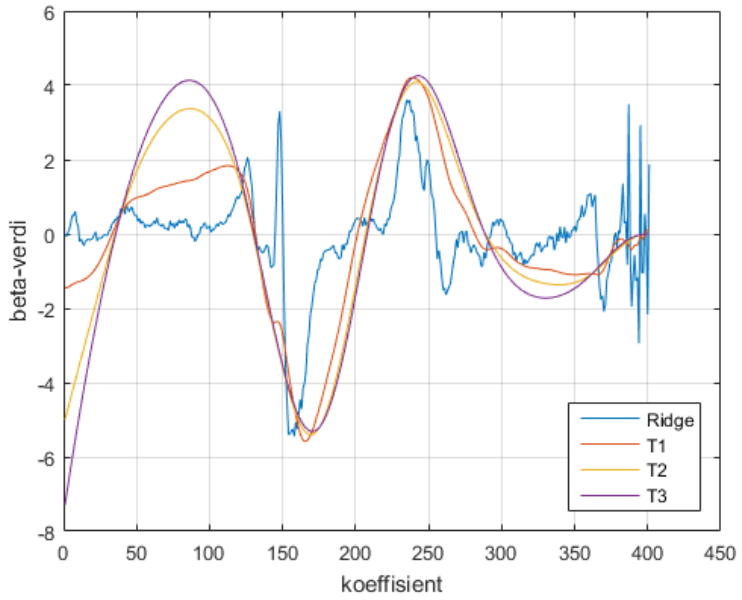
$$\|\boldsymbol{y} - X\hat{\boldsymbol{\beta}}\|^2 + \lambda\|T\hat{\boldsymbol{\beta}}\|^2 = \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T X\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^T X^T\boldsymbol{y} + \hat{\boldsymbol{\beta}}^T X^T X\hat{\boldsymbol{\beta}} + \lambda\hat{\boldsymbol{\beta}}^T T^T T\hat{\boldsymbol{\beta}}.$$

Ved å innføre

$$Z = \begin{bmatrix} X \\ \sqrt{\lambda}T \end{bmatrix} \quad \text{og} \quad \tilde{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix}, \tag{2.25}$$

blir optimeringsproblemet å minimere

$$\begin{aligned} \|\boldsymbol{y} - X\hat{\boldsymbol{\beta}}\|^2 + \lambda\|T\hat{\boldsymbol{\beta}}\|^2 &= \tilde{\boldsymbol{y}}^T\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{y}}^T Z\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^T Z^T\tilde{\boldsymbol{y}} + \hat{\boldsymbol{\beta}} Z^T Z \\ &= \left(\tilde{\boldsymbol{y}}^T - \hat{\boldsymbol{\beta}}^T Z^T\right)\left(\tilde{\boldsymbol{y}} - Z\hat{\boldsymbol{\beta}}\right) \\ &= \left(\tilde{\boldsymbol{y}} - \hat{\boldsymbol{\beta}} Z\right)^T\left(\tilde{\boldsymbol{y}} - Z\hat{\boldsymbol{\beta}}\right) \\ &= \|\tilde{\boldsymbol{y}} - Z\hat{\boldsymbol{\beta}}\|^2. \end{aligned} \tag{2.26}$$



Figur 2.4: $\hat{\boldsymbol{\beta}}$ plottet for fire forskjellige regulariserte modeller.

Altså vil Tikhonov-regulariserte minste kvadraters problemer reduseres til vanlige minste kvadraters problemer ved å la

$$X \leftarrow \begin{bmatrix} X \\ \sqrt{\lambda} T \end{bmatrix} \quad \text{og} \quad \boldsymbol{y} \leftarrow \begin{bmatrix} \boldsymbol{y} \\ \mathbf{0} \end{bmatrix}.$$

Her er $\mathbf{0}$ nullvektoren i $\mathbb{R}^p$.

2.6.1 Regularisering på flere kriterier

Det er også fullt mulig å regularisere på flere kriterier, for eksempel ved å både tvinge normen til $\hat{\boldsymbol{\beta}}$ til å være liten (Ridge-regularisering), og å tvinge $\hat{\boldsymbol{\beta}}$ til å ha en høy grad av kontinuitet der regulariseringsmatrisa T er en derivasjonsoperator beskrevet i likning (2.22) - (2.24). Optimeringsproblemet blir i så fall

$$\min \left(\|\boldsymbol{y} - X\hat{\boldsymbol{\beta}}\|^2 + \lambda \|\hat{\boldsymbol{\beta}}\|^2 + \mu \|T\hat{\boldsymbol{\beta}}\|^2 \right), \tag{2.27}$$

og man kan i prinsippet utvide til så mange regulariseringer man måtte ønske. Her vil jeg imidlertid konsentrere meg om (2.27), som allerede er en utvidelse av (1.1). Ved å

innføre

$$\Xi = \begin{bmatrix} X \\ \sqrt{\mu}T \\ \sqrt{\lambda}I \end{bmatrix} = \begin{bmatrix} Z(\mu) \\ \sqrt{\lambda}T \end{bmatrix} \in \mathbb{R}^{(n+2p)\times p} \quad \text{og} \quad \tilde{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{n+2p},$$

der $\mathbf{0}$ er den $2p$-dimensjonale nullvektoren, vil (2.27) kunne skrives som

$$\|\tilde{\boldsymbol{y}} - \Xi\hat{\boldsymbol{\beta}}\|^2.$$

Utledningen av dette er en utvidelse av (2.26). Med andre ord vil problemet igjen reduseres til et vanlig minste kvadraters problem.

2.7 Standardisering av data

Jeg har brukt forskjellige typer data til å teste ut de algoritmene for regresjon og klassifisering som jeg har brukt i arbeidet med denne oppgaven. En god del av disse dataene er fra NIR-spektroskopi, der hver forklaringsvariabel er et bølgelengdeintervall på det elektromagnetiske spekteret. Siden hver variabel måler det samme, er det ikke nødvendig å gjøre noen skalering eller annen preprosessering av data. Det samme gjelder for bildeanalysedata, der hver forklaringsvariabel er én enkelt piksel.

I tilfeller der forklaringsvariablene måler kvalitativt forskjellige parametre som må antas å ha forskjellig spredning og middel, er det viktig å standardisere dataene for at ikke noen variable skal dominere på grunn av sin distribusjon. Dette gjøres ved å trekke fra det estimerte middelet for hver forklaringsvariabel, for så å dele på det estimerte standardavviket. Resultatet blir at alle variable får middel lik 0 og standardavvik lik 1. Alle variablene får altså lik fordeling.

Kapittel 3

Regresjon

I dette kapittelet vil jeg ta for meg løsning av regresjonsproblemer og hvordan løse disse i Matlab. Jeg vil også gå inn på forskjellige metoder for modellvalidering, og hvordan kode disse effektivt. Til slutt vil jeg implementere multivariate løsninger av Tikhonov-regulariserte minste kvadraters regresjonsproblemer.

3.1 Multivariat OLS

Man refererer til et problem som *multivariat* hvis det er mer enn én responsvariabel. Hvis så er tilfellet, ordner man målingene av hver responsvariabel i en matrise

$$Y = [\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad \cdots \quad \boldsymbol{y}_m],$$

der hver $\boldsymbol{y}$ er målinger av én enkelt responsvariabel. I kapittel 1 viste jeg at løsningen på et minste kvadraters problem

$$\min \|\boldsymbol{y} - X\hat{\boldsymbol{\beta}}\|^2$$

er gitt ved $\hat{\boldsymbol{\beta}} = (X^TX)^{-1}X^T\boldsymbol{y}$ (2.15). Derom man har flere responsvariable, har man også like mange $\hat{\boldsymbol{\beta}}$-vektorer, én for hver responsvariabel. Hver av dem finnes etter metoden fra kapittel 1, men $\boldsymbol{y}$-vektoren man bruker, vil være forskjellig alt etter hvilken respons man ser på. Jeg introduserer derfor

$$\hat{B} = [\hat{\boldsymbol{\beta}}_1 \quad \hat{\boldsymbol{\beta}}_2 \quad \cdots \quad \hat{\boldsymbol{\beta}}_p]^T = (X^TX)^{-1}X^TY, \tag{3.1}$$

og denne vil være en løsning på det minste kvadraters problemet

$$\min \|Y - X\hat{B}\|^2.$$

De modellpredikerte responsverdiene samles i matrisa

$$\hat{Y} = X\hat{B} = X(X^TX)^{-1}X^TY.$$

Ved QR-faktorisering (delkapittel 2.1) vil regresjonskoeffisientene uttrykkes ved

$$\hat{B} = R^{-1}Q^TY,$$

og de predikerte responsverdiene ved

$$\hat{Y} = QQ^TY$$

fra likning (2.16) og (2.17), og ved SVD (delkappitel 2.2) av X vil regresjonskoeffisientene være på formen

$$\hat{B} = V\Sigma^{-1}U^TY$$

mens $\hat{Y}$ blir

$$\hat{Y} = UU^TY$$

fra likning (2.18) og (2.19). Selv om man har flere responsvariable, er det altså nok å gjøre én QR-faktorisering eller én SVD av X-matrisa.

3.2 Modellvalidering

Målet med å lage statistiske modeller, er at man utfra en begrenset datamengde skal kunne si noe generelt om verden utenfor datamengden man bruker til å bygge modellen. Hvordan kan man vite at modellen er god, at den inneholder sann informasjon om faktiske sammenhenger, og at ikke sammenhengene modellen viser er tilfeldige for akkurat det utvalget man har studert? Det er flere metoder for å få en pekepinn på *prediksjonsevnen*

til en modell, og felles for dem alle er at man lager modellen ved bruk av en del av datapunktene, for så å bruke modellen til å predikere responsen for resten av datapunktene. Differansene mellom prediksjon og faktisk målt verdi,

$$r_i^* = y_i - \hat{y}_i^*$$

der $\hat{y}_i$ er den modellpredikerte responsverdien for et utelatt punkt, vil være et relativt mål på hvor *god* en modell er. Disse differansene kalles kryssvaliderte residualer.

Dersom residualene er små, forventer man at modellen predikerer godt også for andre datapunkter som ikke har blitt brukt til å bygge modellen. Er residualene store, har man grunn til å betvile modellens prediksjonsevne.

3.2.1 «Leave one out»-kryssvalidering (LOOCV)

Denne metoden for modellvalidering går ut på at man utelater kun ett punkt, lager en modell for de resterende dataene, og forsøker å predikere det utelatte punktet. Dette gjør man for alle punktene i datamengden, og man får en vektor av kryssvaldierte residualer

$$\boldsymbol{r}^* = \boldsymbol{y} - \boldsymbol{y}^*$$

for hver responsvariabel. Hver respons vil få sin predikerte residualkvadratsum (eng: *Predicted REsidual Sum of Squares*, PRESS), som er definert som

$$\text{PRESS} = \sum_{i=1}^{n} (r_i^*)^2 = (\boldsymbol{r}^*)^T \boldsymbol{r}^*.$$

Dette er et hendig relativt mål på forventet prediksjonsevne for forskjellige modeller.

Når man gjør Tikhonov-regularisering, får man mange forskjellige modeller, én for hver verdi av λ. Man vil da velge den λ-verdien som gir lavest PRESS-verdi.

«Leave p out»-kryssvalidering (LpO CV)

Et problem, og dette vil jeg komme tilbake til, er at LOOCV er en kostbar prosedyre med hensyn på antall utregninger. Det å regne ut $\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}$ én gang, vil kreve

ganske mange utregninger. Hvis X har mange målinger og mange variable, vil det å regne ut regresjonskoeffisienter én gang for hvert punkt kunne ta lang tid.

Når man gjør faktiske målinger, er det vanlig å gjøre flere replikater av samme måling. I tillegg til at man får et en større datamengde, vil man også kunne si noe om usikkerheten i måleutstyret. Under en antakelse om at målefeilen er tilfeldig, vil man ved å gjøre flere replikerte målinger være sikrere på at gjennomsnittet av replikaene er et godt estimat for den faktiske verdien. Når man gjør kryssvalidering for slike datasett, kan det være hensiktsmessig å utelate alle replikaene av hver måling når man gjør kryssvalidering. Valideringen vil dermed kreve mindre regnekraft og dermed ta kortere tid.

Det er selvfølgelig også mulig å gjøre en tilsvarende «Leave p out»-kryssvalidering også for data som ikke inneholder replikaer. Da gjøres dette kun fordi LOOCV tar for mye regnekraft og for lang tid.

3.2.2 Test- og treningsdata

Dersom man har nok målinger, er det vanlig å dele inn datasettet i to deler, ett treningsdatasett og et test-datasett. Man bruker trenings-datene til å bygge modellen for så å forsøke å predikere responsene til test-dataene, og sammenlikner flere modeller utfra summen av de kvadrerte prediksjonsresidualene. Denne typen modellvalidering er nærmere det som vil være faktisk bruk av modellen: å bruke en liten datamengde til å bygge en modell som skal predikere utenfor denne lille datamengden. Denne typen modellvalidering simulerer i stor grad hvordan modellen vil brukes i virkeligheten.

3.2.3 Rask LOOCV

Det er mulig å gjøre LOOCV på en alternativ måte for minste kvadrater-problemer som drastisk reduserer antallet regneoperasjoner. For å vise dette, trenger jeg først en viktig sammenheng for invertering av modifiserte matriser:

Sherman-Morrison-Woodbury-formelen

Sherman-Morrison-Woodbury-formelen sier at

$$(P+UCV)^{-1} = P^{-1} - P^{-1}U(C^{-1}+VP^{-1}U)^{-1}VP^{-1}, \tag{3.2}$$

der $P \in \mathbb{R}^{n\times n}, U \in \mathbb{R}^{n\times q}, C \in \mathbb{R}^{q\times q}$, $V \in \mathbb{R}^{q\times n}$ og P og C antas invertible.

Bevis. Direkte utrgegning gir

$$\begin{aligned}
&(P+UCV)[P^{-1} - P^{-1}U(C^{-1}+VP^{-1}U)^{-1}VP^{-1}] \\
&= I_n - U(C^{-1}+VP^{-1}U)^{-1}VP^{-1} + UCVP^{-1} - UCVP^{-1}U(C^{-1}+VP^{-1}U)^{-1}VP^{-1} \\
&= I_n + UCVP^{-1} - (U+UCVP^{-1}U)(C^{-1}+VP^{-1}U)^{-1}VP^{-1} \\
&= I_n + UCVP^{-1} - UC(C^{-1}+VP^{-1}U)(C^{-1}+VP^{-1}U)^{-1}VP^{-1} \\
&= I_n + UCVP^{-1} - UCVP^{-1} \\
&= I_n,
\end{aligned}$$

og (3.2) følger direkte. □

På samme måte kan det vises at

$$(P-UCV)^{-1} = P^{-1} + P^{-1}U(C^{-1}-VP^{-1}U)^{-1}VP^{-1}: \tag{3.3}$$

Bevis.

$$\begin{aligned}
&(P-UCV)[P^{-1} + P^{-1}U(C^{-1}-VP^{-1}U)^{-1}VP^{-1}] \\
&= I_n + U(C^{-1}-VP^{-1}U)^{-1}VP^{-1} - UCVP^{-1} - UCVP^{-1}U(C^{-1}-VP^{-1}U)^{-1}VP^{-1} \\
&= I_n - UCVP^{-1} + (U-UCVP^{-1}U)(C^{-1}-VP^{-1}U)^{-1}VP^{-1} \\
&= I_n - UCVP^{-1} + UC(C^{-1}-VP^{-1}U)(C^{-1}+VP^{-1}U)^{-1}VP^{-1} \\
&= I_n - UCVP^{-1} + UCVP^{-1} \\
&= I_n,
\end{aligned}$$

og (3.3) følger direkte (Woodbury, 1950 [26]). □

PRESS for rask LOOCV

Man kan regne ut de kryssvaliderte residualene ved kun å lage modellen én gang. Jeg innfører notasjonen $\hat{\boldsymbol{\beta}}_{(i)}$ som løsningen på det minste kvadraters problemet

$$\hat{\boldsymbol{\beta}}_{(i)} = \left(X_{(i)}^T X_{(i)}\right)^{-1} X_{(i)}^T \boldsymbol{y}_{(i)}, \tag{3.4}$$

der datapunkt nummer j er utelatt, og $X_{(i)}^T X_{(i)}$ er invertibel. Jeg lar $\{\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \cdots, \boldsymbol{x}_n^T\}$ være radene i datamatrisa X. Dette gir

$$X^T X = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix} = \sum_{k=1}^{n} \boldsymbol{x}_k \boldsymbol{x}_k^T.$$

Vet å utelate datapunkt nummer i får jeg

$$X_{(i)}^T X_{(i)} = \sum_{k=1}^{n} \boldsymbol{x}_k \boldsymbol{x}_k^T - \boldsymbol{x}_i j \boldsymbol{x}_i^T = X^T X - \boldsymbol{x}_i \boldsymbol{x}_i^T \tag{3.5}$$

og ved å anvende (3.3) på dette uttrykket med $P = X^T X, U = \boldsymbol{x}_i, C = 1$ og $V = \boldsymbol{x}_i^T$, har jeg at

$$\begin{aligned} \left(X_{(i)}^T X_{(i)}\right)^{-1} &= (X^T X)^{-1} + (X^T X)^{-1} \boldsymbol{x}_i (1 - \boldsymbol{x}_i^T (X^T X)^{-1} \boldsymbol{x}_i)^{-1} \boldsymbol{x}_i^T (X^T X)^{-1} \\ &= (X^T X)^{-1} + \frac{(X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T (X^T X)^{-1}}{1 - \boldsymbol{x}_i^T (X^T X)^{-1} \boldsymbol{x}_i}. \end{aligned} \tag{3.6}$$

Jeg innfører matrisa $H = X(X^T X)^{-1} X^T$ med diagonalelement nummer i lik

$$h_i = \boldsymbol{x}_i^T (X^T X)^{-1} \boldsymbol{x}_i. \tag{3.7}$$

På engelsk er disse diagonalelementene kjent som *leverage values*. Det har ikke lyktes meg å finne et godt norsk uttrykk for dette, så jeg vil bruke leverage-verdier for å omtale disse. Jeg setter inn h_i i (3.6) og får

$$\left(X_{(i)}^T X_{(i)}\right)^{-1} = (X^T X)^{-1} + \frac{(X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T (X^T X)^{-1}}{1 - h_i}.$$

Jeg kan nå uttrykke regresjonskoeffientene for hver kryssvalideringsmodell som

$$\hat{\boldsymbol{\beta}}_{(i)} = \left[(X^T X)^{-1} + \frac{(X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T (X^T X)^{-1}}{1 - h_i} \right] X_{(i)}^T \boldsymbol{y}_{(i)},$$

der

$$X_{(i)}^T \boldsymbol{y}_{(i)} = X^T \boldsymbol{y} - y_i \boldsymbol{x}_i$$

etter samme argumentasjon som i (3.5), slik at de kryssvaliderte regresjonskoeffisientene kan skrives som

$$\hat{\boldsymbol{\beta}}_{(i)} = \left[(X^T X)^{-1} + \frac{(X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T (X^T X)^{-1}}{1 - h_i} \right] (X^T \boldsymbol{y} - y_i \boldsymbol{x}_i).$$

Jeg gjennomfører multiplikasjonen og bruker at $\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}$:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_{(i)} &= \hat{\boldsymbol{\beta}} - (X^T X)^{-1} y_i \boldsymbol{x}_i + \frac{(X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}} - y_i (X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T (X^T X)^{-1} \boldsymbol{x}_i}{1 - h_i} \\
&= \hat{\boldsymbol{\beta}} + \frac{(X^T X)^{-1} \boldsymbol{x}_i \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}} - y_i (X^T X)^{-1} \boldsymbol{x}_i h_i - y_i (X^T X)^{-1} \boldsymbol{x}_j (1 - h_i)}{1 - h_i} \\
&= \hat{\boldsymbol{\beta}} + \frac{(X^T X)^{-1} \boldsymbol{x}_j}{1 - h_i} \left[\boldsymbol{x}_i^T \hat{\boldsymbol{\beta}} - y_i h_i - y_i (1 - h_i) \right] \\
&= \hat{\boldsymbol{\beta}} + \frac{(X^T X)^{-1} \boldsymbol{x}_i}{1 - h_i} (\hat{y}_i - y_i) \\
&\Downarrow \\
\boldsymbol{x}_i^T \hat{\boldsymbol{\beta}}_{(i)} &= \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}} + \frac{\boldsymbol{x}_i^T (X^T X)^{-1} \boldsymbol{x}_i}{1 - h_i} (\hat{y}_i - y_i) \\
&= \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}} + \frac{h_i}{1 - h_i} (\hat{y}_i - y_i)
\end{aligned}$$

Jeg skriver den kryssvaliderte responsen for punkt i som $\hat{y}_i^* = \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}}_{(i)}$, og den tilsvarende prediksjonen for den fulle modellen som $\hat{y} = \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}}$, slik at uttrykket over blir

$$\begin{aligned}
\hat{y}_i^* &= \hat{y}_i + \frac{h_i}{1 - h_i} (\hat{y}_i - y_i) \\
&= \frac{\hat{y}_i (1 - h_i) + h_i (\hat{y}_i - y_i)}{1 - h_i} \\
&= \frac{\hat{y}_i - h_i y_i}{1 - h_i}.
\end{aligned}$$

Den kryssvaliderte residualen r_i^* er gitt som differansen mellom den faktiske måleverdien av responsen og prediksjonen av den reduserte modellen for punkt i:

$$\begin{aligned}
r_i^* &= y_i - \hat{y}_i^* \\
&= y_i - \frac{\hat{y}_i - h_i y_i}{1 - h_i} \\
&= \frac{y_i (1 - h_i) - \hat{y}_i - h_i y_i}{1 - h_i} \\
&= \frac{y_i - \hat{y}_i}{1 - h_i}
\end{aligned}$$

Dette gjør at PRESS-verdien blir

$$\text{PRESS} = \|\boldsymbol{r}^*\|^2 = \sum_{i=1}^{n}(r_i^*)^2 = \sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2.$$

Denne metoden for å finne PRESS ble først foreslått av Allen (1974) [2]. For å gjøre kryssvalidering og regne ut PRESS-verdien er det altså nok å bygge den fulle modellen og predikere samtlige responsverdier, samt å regne ut diagonalelementene i matrisa $H = X(X^TX)^{-1}X$, hvilket kan gjøres enklere ved faktorisering. Ved en QR-faktorisering (delkapittel 2.1) vil H ifølge (2.17) bli

$$H = X(X^TX)^{-1}X^T = QQ^T \tag{3.8}$$

med diagonalelementer

$$h_i = \sum_{j=1}^{p} q_{ij}^2.$$

der q_{ij} er elementet i rad i og kolonne j i Q. Det er også mulig å bruke SVD (delkapittel 2.2) for å finne H:

$$H = X(X^TX)^{-1}X^T = UU^T. \tag{3.9}$$

ifølge (2.19), med diagonalelementer

$$h_i = \sum_{j=1}^{p} u_{ij}^2,$$

der u_{ij} er elementet i rad i og kolonne j i U.

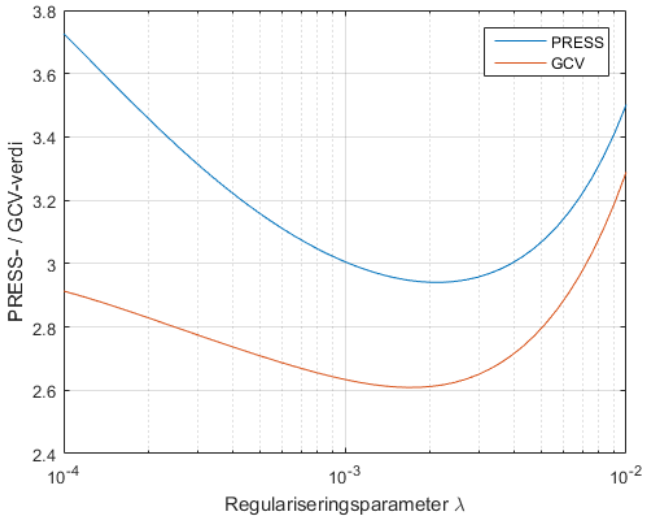
3.2.4 Generalisert kryssvalidering (GCV)

En alternativ metode for kryssvalidering beskrevet i litteraturen (Golub et al., 1979 [10]), kalt *generalisert kryssvalidering* (eng: *generalized cross-validation*, GCV), hvor man regner ut

$$\text{GCV} = \sum_{j=1}^{n}\left(\frac{y_j - \hat{y}_j}{1 - \text{tr}(H)/n}\right)^2$$

i stedet for PRESS, og velger modell GCV-verdien. Figur 3.1 viser et eksempel for Ridge-regresjon av Spectra (Kalivas, 1997 [16]). Dette er et typisk eksempel på forholdet mellom PRESS og GCV når det gjelder hvilken λ-verdi som ser ut til å predikere best.

Generalisert kryssvalidering kan være numerisk lønnsomt i enkelte sammenhenger der det å regne ut $\text{tr}(H)$ er lettere enn å regne ut hvert diagonalelement separat, ifølge Hastie et. al. (2013 [12], s 244-245). Mine implementeringer av LOOCV går så raskt at det ikke virker å være noen forskjell i tidsbruk for GCV og LOOCV.



Figur 3.1: PRESS- og GCV-verdier med hensyn på regulariseringsparameteren λ for Spectra (Kalivas, 1997 [16]). De to kurvene følger hverandre i det at de har bunnpunkt omtrent på samme sted.

3.3 MATLAB-implementeringer

I denne delen vil jeg gi en detaljert beskrivelse av veien fram mot en svært effektiv MATLAB-implementering av multivariat Tikhonov-regresjon. Jeg begynner med en presentasjon av datasettene jeg har brukt for regresjonsdelen av denne oppgaven.

3.3.1 Data for regresjon

Jeg har fortrinnsvis sett på NIR-spektroskopi-data i arbeidet med denne oppgaven. Jeg gir her en kort presentasjon av datasettene.

DiabetesForLS (Efron et al., 2004 [7])

Datasett bestående av én responsvariabel, 10 forklaringsvariable og 442 målinger, inndelt i et treningssett på 300 og et testsett på de resterende 142 målingene. Forklaringsvariablene er alder, kjønn, BMI, blodtrykk og seks forskjellige blodserum-verdier, mens responsvariabelen er et kvantitativt mål på sykdomsprogresjonen etter ett år.

Spectra (Kalivas, 1997 [16])

Dette er et innebygd datasett i MATLAB og består av NIR-spektra målt i 2 nm intervaller fra 900 til 1 700 nm av 60 bensinprøver, totalt 401 forklaringsvariable. Responsvariabel er oktantall.

Biscuit Doughs (Osborne et. al., 1984 [23])

40 målinger av NIR-data av kakedeig, målt i 2 nm intervaller fra 1100 til 2498 nm, totalt 700 forklaringsvariable. De fire responsvariablene er prosentinnhold av henholdsvis fett, sakkarose, mel og vann. Datasettet inneholder i tillegg et test-sett på 32 målinger.

Sugar (Brown, 1992 [5])

125 målinger av NIR-data av sukkerløsninger, målt i 2 nm intervaller fra 1100 til 2498 nm, totalt 700 forklaringsvariable. De tre responsvariablene er konsentrasjoner av henholdsvis sakkarose, glukose og fruktose. Datasettet inneholder i tillegg et test-sett på 21 målinger.

Fett (Næs et al., 2013 [22])

69 målinger av emulsjoner med forskjellig fettsammensetning. Det er gjort to forskjellige typer spektroskopi, NIR og Raman, for å forklare de to responsvariablene. Den første responsen er prosentandel flerumettede fettsyrer av total masse av emulsjonen, mens den andre er prosentandel av denne typen fettsyrer av det totale fettinnholdet.

3.3.2 Multivariat OLS med LOOCV

Den kanskje mest intuitive metoden å gjøre LOOCV på, er å bygge modellen på nytt for hver utelatte måling. Jeg begynner med denne implementeringen av LOOCV. Fordi OLS krever at datamatrisa X har full rang, vil jeg bruke DiabetesForLS (Efron et al., 2004 [7]) som eksempeldata.

I funksjonen `OLS_LOOCV` (vedlegg A.1.1) har jeg implementert denne intuitive tilnærmingen uten å faktorisere datamatrisa og den er å finne i vedlegg A.1.1. Her har jeg brukt at konstantleddene kan regnes ut ved å bruke

$$X_{\text{utv}} = [\mathbf{1} \quad X]$$

der $\mathbf{1} = [1 \quad 1 \quad \cdots 1]^T \in \mathbb{R}^n$. Jeg har brukt

```
1 beta = (X'*X)\(X'*Y);
```

da dette ser ut til å gå raskere enn

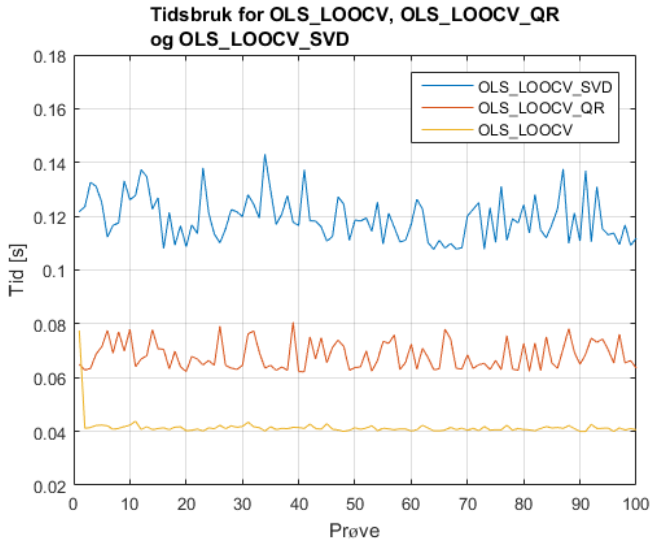
```
1 beta = X\Y;
```

Jeg vil nå forsøke å gjøre denne metoden mer tidseffektiv ved å bruke noen av grepene fra kapittel 1. I `OLS_LOOCV_QR` (vedlegg A.1.2) bruker jeg en QR-faktorisering av X-matrisa samt sammenhengene fra (2.16) og (2.17) for å bygge modellen. Jeg gjør det samme med en SVD av X-matrisa og bruker sammenhengene fra (2.18) og (2.19) som resulterer i `OLS_LOOCV_SVD` i vedlegg A.1.3. Siden de tre funksjonene er matematisk ekvivalente, gir de svært like resultater, og forskjellene skyldes kun avrundinger i utregningene:

```
1 [Beta1, Yhat1, Yhatcv1, res1, rescv1, PRESS1] = OLS_LOOCV(X, Y);
2 [Beta2, Yhat2, Yhatcv2, res2, rescv2, PRESS2] = OLS_LOOCV_QR(X, Y);
3 [Beta3, Yhat3, Yhatcv3, res3, rescv3, PRESS3] = OLS_LOOCV_SVD(X, Y);
```

```
4
5  [rank(Beta1 - Beta2, 1e-6), rank(Yhatcv1 - Yhatcv2, 1e-6), PRESS1 - PRESS2]
6  [rank(Beta1 - Beta3, 1e-6), rank(Yhatcv1 - Yhatcv3, 1e-6), PRESS1 - PRESS3]
7
8  ans =
9
10     1.0e-04 *
11
12          0         0    0.1238
13
14 ans =
15
16     1.0e-04 *
17
18          0         0    0.1231
```

Det vil så være interessant å se hvilken funksjon som bruker kortest tid. Resultatet av 100 prøver for hver av de tre funksjonene er gitt i figur 3.2



Figur 3.2: Tidsbruk for `OLS_LOOCV`, `OLS_LOOCV_QR` og `OLS_LOOCV_SVD` kjørt 100 ganger.

Selv om det går raskere å regne ut $\hat{\boldsymbol{\beta}} = R^{-1}Q^t\boldsymbol{y}$ fra (2.16), må vi gjøre en ny QR-faktorisering for hver gang vi plukker ut en måling i kryssvalideringen. Dette stjeler regnekraft, og den fulle versjonen uten QR-faktorisering går raskere. Det samme fenomenet oppstår i SVD-versjonen.

3.3.3 Multivariat OLS med rask LOOCV

Jeg prøver videre en rask LOOCV-implementering med de samme tre tilnærmingene: den fulle $\hat{\boldsymbol{y}} = X(X^TX)^{-1}X^t\boldsymbol{y}$, QR-faktoriserte $\hat{\boldsymbol{y}} = QQ^T\boldsymbol{y}$ og SVD-versjonen $\hat{\boldsymbol{y}} = VS^{-1}U^T\boldsymbol{y}$.

I alle funksjonene regner jeg ut leverage-verdiene ved å sentrere dataene, for så å bruke sammenhengen at

$$\begin{aligned}
[1 \quad \boldsymbol{x}^T](X_{\text{utv}}^TX_{\text{utv}})^{-1}\begin{bmatrix}1\\ \boldsymbol{x}\end{bmatrix} &= [1 \quad \boldsymbol{x}^T]\begin{bmatrix}n & \mathbf{0}^T\\ \mathbf{0} & X^TX\end{bmatrix}^{-1}\begin{bmatrix}1\\ \boldsymbol{x}\end{bmatrix}\\
&= [1 \quad \boldsymbol{x}^T]\begin{bmatrix}\frac{1}{n} & \mathbf{0}^T\\ \mathbf{0} & (X^TX)^{-1}\end{bmatrix}\begin{bmatrix}1\\ \boldsymbol{x}\end{bmatrix}\\
&= \frac{1}{n} + \boldsymbol{x}^T(X^TX)^{-1}\boldsymbol{x}
\end{aligned}$$

Dette vil si at det ikke er nødvendig å legge til en kolonne med enere for å få med konstantleddet, men at man bare kan legge $1/n$ til hver leverage-verdi man finner for den sentrerte datamatrisa. Konstantleddet finner man så ved å ta middelverdien for responsen og trekke fra middelverdiene til den ikke-sentrerte X-matrisa mutiplisert med regresjonskoeffisientene. Matlab-koden for versjonen uten QR eller SVD ligger i vedlegg A.1.4.

I `OLS_fastLOOCV_QR` (vedlegg A.1.5) bruker jeg at $Q = [\boldsymbol{q}_1 \quad \boldsymbol{q}_2 \quad \cdots \quad \boldsymbol{q}_p]$ og sammenhengen at $H = QQ^T = \sum_{i=1}^{p}\boldsymbol{q}\boldsymbol{q}^T$ fra (3.8). Leverage-verdiene blir diagonalelementene i denne matrisa, som blir summer av kvadrerte elementer fra Q-matrisa. Første leverage-verdi er summen de kvadrerte første elementene fra hver $\boldsymbol{q}$-søylevektor, andre leverage-verdi er summen av de kvadrerte andreelementene og så videre. I Matlab kan man regne ut dette enkelt ved å kvadrere hvert element av Q-matrisa for så å summere langs radene og legge til $1/n$:

```
1  h = sum(Q.^2, 2) + 1/n;
```

Her blir `h` en vektor bestående av alle leverage-verdiene.

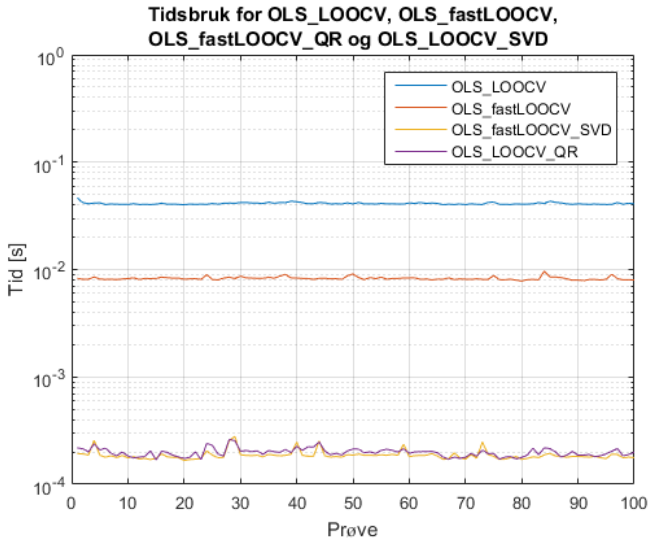
I `OLS_fastLOOCV_SVD` (vedlegg A.1.6) bruker jeg at $H = UU^T$ fra (3.9), og finner deretter leverage-verdiene ved kommandoen

```
1  h = sum(U.^2, 2) + 1/n;
```

Som tidligere vist er disse fremgangsmåtene matematisk ekvivalente, og produserer noenlunde samme resultater:

```
[rank(Beta1 - Beta2, 1e-6), rank(Yhatcv1 - (Yhatcv2 + mean(Y)), 1e-6), PRESS1 - PRESS2]
[rank(Beta1 - Beta3, 1e-6), rank(Yhatcv1 - (Yhatcv3 + mean(Y)), 1e-6), PRESS1 - PRESS3]
[rank(Beta1 - Beta4, 1e-6), rank(Yhatcv1 - (Yhatcv4 + mean(Y)), 1e-6), PRESS1 - PRESS4]

ans =

   1.0e-04 *

         0         0    0.1197

ans =

   1.0e-04 *

         0         0    0.1238

ans =

   1.0e-04 *

         0         0    0.1238
```

Jeg har igjen brukt DiabetesForLS-datasettet (Efron et al., 2004 [7]) for å teste disse tre metodene mot hverandre på tidsbruk og sammenlikne med den raskeste implementeringen uten rask LOOCV. Resultatene er gitt i figur 3.3.



Figur 3.3: Sammenlikning av de tre raske LOOCV-metodene med den raskeste metoden uten å bruke Allens (1974 [2]) PRESS.

Det er tydelig at QR- og SVD-implementeringene går svært mye raskere, og det er to grunner til dette. For det første trenger man kun å gjøre én faktorisering i hvert av tilfellene. For andre det krever utregning av leverage-verdiene betydelig færre regneoperasjoner i QR- og SVD-tilfellet.

3.3.4 Multivariat Ridge-regresjon med rask LOOCV

Jeg ser så på implementeringer av Tikhonov-regularisering. Jeg begynner med den enkleste formen, Ridge-regularisering, der regulariseringsmatrisa $T = I_p$. Jeg skal løse problemet $\|Y - X\hat{\boldsymbol{\beta}}\|^2 + \lambda\|\hat{\boldsymbol{\beta}}\|^2$, og vil løse numerisk ved å prøve mange forskjellige verdier av regulariseringsparameteren λ og finne den som gir lavest PRESS-verdi. Som foreslått av Boyd og Vandenberghe (2015 [4]) lar jeg λ-verdiene være $\log_{10}$-distribuert og bruker 50 λ-verdier mellom 10^{-4} og 10^4. En implementering av Ridge-regresjon uten faktorisering av X, `Ridge_fastLOOCV`, ligger i vedlegg A.1.7.

Jeg har vist at full LOOCV der man bygger modellen på nytt for hver utelatte verdi i kryssvalideringen går raskest når man lar være å faktorisere datamatrisa i analysen, mens når man gjør rask LOOCV går det raskest med QR-faktorisering. Både med og uten QR-faktorisering må man regne ut nye leverage-verdier for hver verdi av λ, men ved QR-faktorisering kan man gjøre en ytterligere snarvei ved at man bare har behov for de n første diagonalelementene i QQ^T. Dette fordi man kun trenger n leverage-verdier til å predikere de n y-verdiene. Videre regner man ut $\hat{\boldsymbol{\beta}}$ ved

$$\hat{B} = R^{-1}Q\tilde{Y},$$

der elementene på de siste p radene i $\tilde{Y}$ har verdien 0. Det vil dermed være nok å regne med de første n radene av Q og den originale Y-matrisa. Det samme vil gjelde for utregning av av de modellpredikerte y-verdiene $\hat{Y} = QQ^T\tilde{Y}$. Siden de siste p radene i $\tilde{Y}$ bare inneholder nuller, vil det være nok å bruke de n første radene i Q-matrisa og den originale Y-matrisa. Dette er implementert i funksjonen `Ridge_fastLOOCV_QR` (vedlegg A.1.8).

Ved å sette $X = U\Sigma V^T$ der U og V henholdsvis består av søylevektorne $\{\boldsymbol{u}_i\}_{i=1}^n$ og $\{\boldsymbol{v}_j\}_{j=1}^p$, vet jeg at $\{\boldsymbol{u}_i\}_{i=1}^n$ er egenvektorer til XX^T og $\{\boldsymbol{v}_j\}_{j=1}^p$ er egenvektorer til X^TX fra utledningen av SVD (delkapittel 2.2).

Definisjonen av den utvidede datamatrisa Z for Ridge-regresjon,

$$Z = \begin{bmatrix} X \\ \sqrt{\lambda} I_p \end{bmatrix},$$

gjør dermed at

$$\begin{aligned} ZZ^T U &= (XX^T + \lambda I)U \\ &= XX^T U + \lambda U. \end{aligned}$$

Dette betyr at for hver søylevektor i $\boldsymbol{u}_r$ i U vil

$$\begin{aligned} ZZ^T \boldsymbol{u}_r &= XX^T \boldsymbol{u}_r + \lambda \boldsymbol{u}_r \\ &= \sigma_r^2 \boldsymbol{u}_r + \lambda \boldsymbol{u}_r \\ &= (\sigma_r^2 + \lambda)\boldsymbol{u}_r \end{aligned} \tag{3.10}$$

når σ_r er diagonalelement r i Σ. Altså er $\{\boldsymbol{u}_j\}_{j=1}^n$ også egenvektorer for ZZ^T, med egenverdier like dem for XX^T, bortsett fra et et λ-tillegg. Tilsvarende argument kan gjøres for V og $Z^T Z$, slik at singulærverdidekomposisjonen av Z er identisk med den av X bortsett fra for egenverdiene i Σ-matrisa. Dette betyr at man bare trenger å gjøre dekomposisjonen én gang, noe som kan gjøre kryssvalideringen svært regneeffektiv.

I SVD-tilfellet bytter jeg derfor ut Σ med

$$\tilde{\Sigma} = \operatorname{diag}\left(\begin{bmatrix} \sqrt{\sigma_1^2 + \lambda} & \sqrt{\sigma_2^2 + \lambda} & \cdots & \sqrt{\sigma_n^2 + \lambda} \end{bmatrix}\right) \tag{3.11}$$

som beskrevet i (3.10) og bruker videre den λ-oppdaterte $\tilde{X} = U\tilde{\Sigma}V^T$. For X-matrisa regnes leverage-verdi k ut ved $\boldsymbol{x}_k^T (X^T X)^{-1} \boldsymbol{x}_k$ der $\boldsymbol{x}_k^T$ er radvektoren for måling k. I det λ-oppdaterte tilfellet, vil man finne leverage-verdiene som diagonalelementer i $X(\tilde{X}^T \tilde{X})^{-1} X^T$. Matrisene lengst til høyre og lengst til venstre trenger man ikke å oppdatere med λ fordi det ikke er interessant å prøve å predikere 0-elementene i den utvidede respons-matrisa. Dette kan dermed skrives som

$$\begin{aligned} H &= X(\tilde{X}^T \tilde{X})^{-1} X^T \\ &= U\Sigma V^T [(U\tilde{\Sigma}V^T)^T U\tilde{\Sigma}V^T]^{-1} (U\Sigma V^T)^T \\ &= U\Sigma V^T (V\tilde{\Sigma}U^T U\tilde{\Sigma}V^T)^{-1} V\Sigma U^T \\ &= U\Sigma\tilde{\Sigma}^{-2}\Sigma U^T \end{aligned}$$

og fordi Σ og $\tilde{\Sigma}$ er diagonale matriser

$$\begin{aligned} &= U\Sigma^2\tilde{\Sigma}^{-2}U^T \\ &= U\check{S}(U\check{S})^T, \end{aligned} \tag{3.12}$$

der $\check{S} = \Sigma\tilde{\Sigma}^{-1}$. Diagonalelementene finner man ved å summere de kvadrerte elementene i $U\check{S}$-matrisa rekke for rekke. På samme måte blir $\hat{B} = V\tilde{\Sigma}^{-2}\Sigma U^T Y$ og $\hat{Y} = U\Sigma^2\tilde{\Sigma}^{-2}U^T Y$. Dette er implementert i funksjonen `Ridge_fastLOOCV_SVD` (Vedlegg A.1.9).

Ved QR-faktorisering må man gjøre en ny faktorisering for hver λ-verdi, mens ved å bruke SVD vil man altså klare seg med én enkelt dekomposisjon.

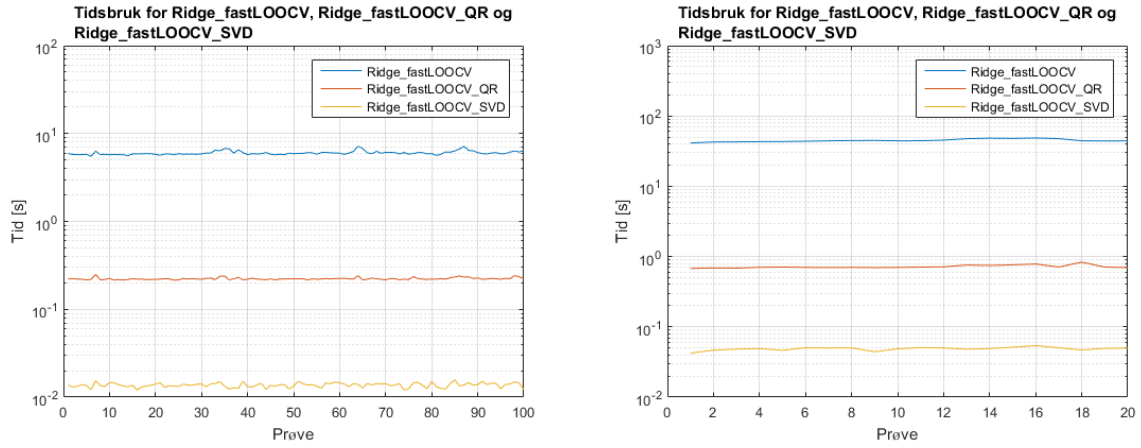
Nok en gang har jeg tre matematisk ekvivalente funksjoner, og jeg prøver dem ut på Spectra-datasettet (Kalivas, 1997 [16]):

```
[Beta1, Betas1, H1, PRESS1, lambda1] = Ridge_fastLOOCV(X, Y, lambdas);
[Beta2, Betas2, H2, PRESS2, lambda2] = Ridge_fastLOOCV_QR(X, Y, lambdas);
[Beta3, Betas3, H3, PRESS3, lambda3] = Ridge_fastLOOCV_SVD(X, Y, lambdas);
[rank(Betas1 - Betas2, 1e-10), rank(H1 - H2, 1e-10), rank(PRESS1 - PRESS2, 1e-10)]
[rank(Betas1 - Betas3, 1e-10), rank(H1 - H3, 1e-10), rank(PRESS1 - PRESS3, 1e-10)]

ans =

     0     0     0

ans =

     0     0     0
```

Figur 3.4 på neste side viser tidsbruken til de tre funksjonene for dette datasettet. Som ventet går SVD-versjonen svært mye raskere enn de andre to.

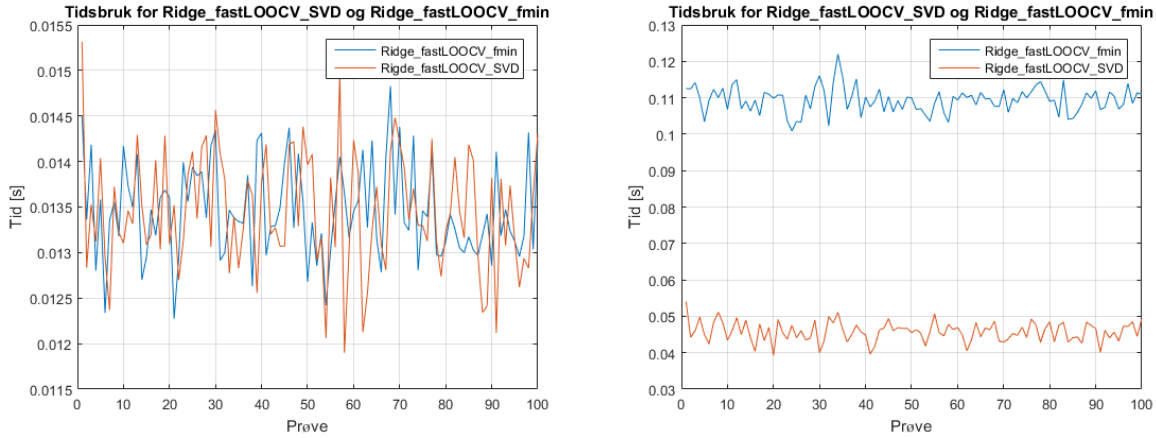
fminbnd

Matlab har en innebygd funksjon som minimerer funksjoner av én variabel kalt `fminbnd`. Jeg har laget en funksjon som bruker denne `fminbnd` for å finne lambda med lavest PRESS-verdi og bruke denne til å bygge modeller.`Ridge_fastLOOCV_fmin` (vedlegg A.1.10) er en omforming av `Ridge_fastLOOCV_SVD` som jobber med funksjonen PRESS(λ). Figur 3.5 viser at det er liten forskjell mellom de to framgangsmåtene med én respons, men at `fminbnd`-versjonen bruker tydelig lenger tid i det multivariate tilfellet.



(a) Spectra (b) Sugar

Figur 3.4: Tidsbruk for Ridge-regularisert regresjon av Spectra-datasettet (Kalivas, 1997 [16]) og Sugar-datasettet (Brown, 1992 [5]) med 50 verdier av λ $\log_{10}$-distribuert mellom 10^{-4} og 10^{4} for `Ridge_fastLOOCV`, `Ridge_fastLOOCV_QR` og `Ridge_fastLOOCV_SVD`.

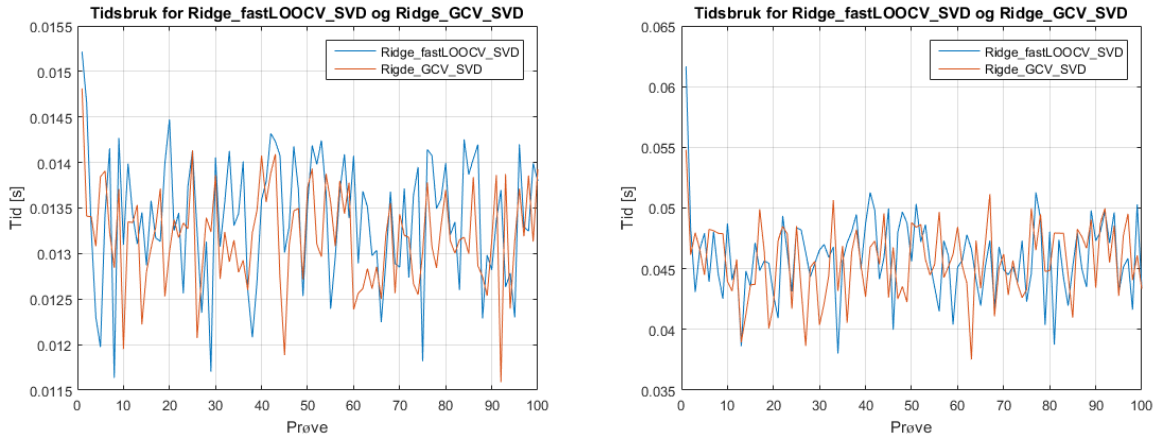


(a) Spectra (b) Sugar

Figur 3.5: Tidsbruk for Ridge-regularisert regresjon av Spectra-datasettet (Kalivas, 1997 [16]) og Sugar-datasettet (Brown, 1992 [5]) med 50 verdier av λ $\log_{10}$-distribuert mellom 10^{-4} og 10^{4} for `Ridge_fastLOOCV_SVD` og `Ridge_fastLOOCV_fmin`.

GCV

Jeg har også laget en funksjon `Ridge_GCV_SVD`, som er en implementering av generalisert kryssvalidering (vedlegg A.1.11). Figur 3.6 viser at det ikke er særlig forskjell i tidsbruk for de to implementeringene.



(a) Spectra

(b) Sugar

Figur 3.6: Tidsbruk for Ridge-regularisert regresjon av Spectra-datasettet (Kalivas, 1997 [16]) og Sugar-datasettet (Brown, 1992 [5]) med 50 verdier av λ $\log_{10}$-distribuert mellom 10^{-4} og 10^{4} for `Ridge_fastLOOCV_SVD` og `Ridge_GCV_SVD`.

3.3.5 Multivariat Tikhonov-regresjon med rask LOOCV

Når regulariseringsmatrisa $T \neq I_p$, er det ikke mulig å bruke sammenhengen fra 3.12 direkte fordi man ikke har en like klar sammenheng mellom singulærveridene til Z og X som det er i Ridge-tilfellet. Det er imildertid mulig å gå fram på en liknende måte ved å høyremultiplisere Z med T^{-1}:

$$ZT^{-1} = \begin{bmatrix} X \\ \sqrt{\lambda}T \end{bmatrix} T^{-1} = \begin{bmatrix} XT^{-1} \\ \sqrt{\lambda}I_p \end{bmatrix}.$$

Analyse av denne matrisa er vanlig Ridge-regresjon, og vil produsere en matrise av regresjonskoeffisienter, $\hat{B}_{\mathrm{T}}$, som er slik at

$$XT^{-1}\hat{B}_{\mathrm{T}} = \hat{Y}.$$

Ved så å bruke innføre $\hat{B}_{\mathrm{T}} = T\hat{B}$, vil

$$XT^{-1}T\hat{B} = X\hat{B} = \hat{Y}.$$

For å få regresjonskoeffisientene for den originale X-matrisa, er det med andre ord nok å transformere $\hat{B}_{\mathrm{T}}$ tilbake igjen ved å venstremultiplisere med T^{-1}.

Med dette i bakhodet er det to ting man kan merke seg. Det ene er at derivajonsregulariseringer og Ridge-regularisering egentlig koker ned til det samme problemet, men at man må transformere med T^{-1}.

Det andre er at det gir mening å bruke derivasjonsregulariseringer der man antar B består av noe som nærmer seg en kontinuerlig distribusjon av regresjonskoeffisienter i hver søyle. Når dette viser seg å være en transformert utgave av Ridge-regresjon, vil det også gi mening å utføre den samme transformasjonen også for andre regresjonsmetoder, som PCR og PLS.

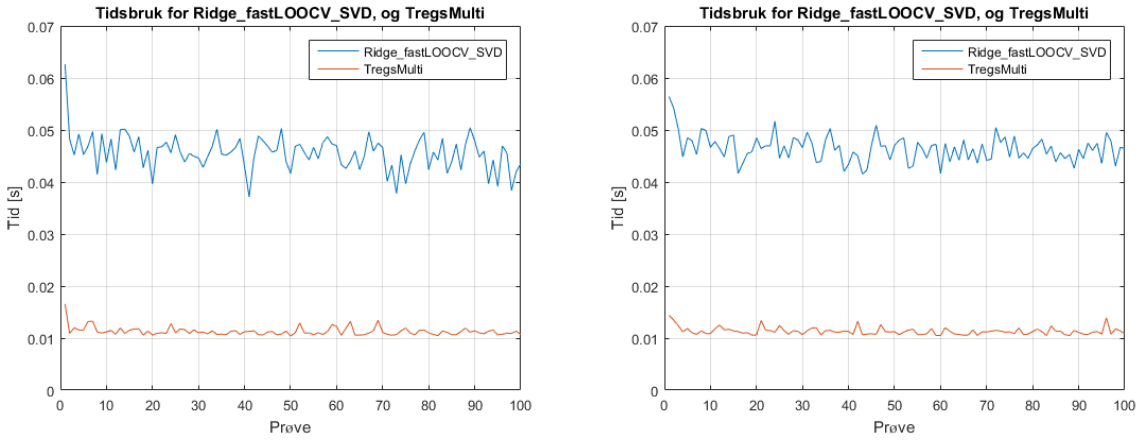
Derivasjonsregulariseringer

Regularisering med derivasjonsmatriser er altså i bunn og grunn det samme som Ridge-regresjon av XT^{-1}, der T er en invertibel regulariseringsmatrise. Ved å la derivasjonsordenen være en input-verdi, er det mulig å lage en funksjon som bygger Tikhonov-regulariserte regresjonsmodeller for angitt derivasjonsorden. `TregsMulti`-funksjonen gjør dette ved å ta input-variabelen `d` som derivasjonsorden. Ved `d` > 0 genereres regulariseringsmatrisa `T` ved

```
1 T = diff([speye(p);sparse(d, p)], d);
```

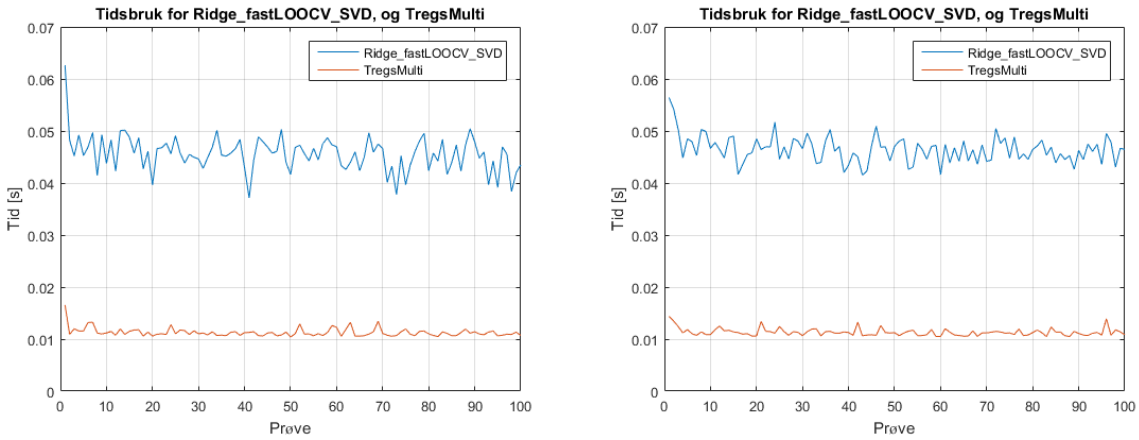
`diff`-operatoren i Matlab regner ut forskjellen mellom to etterfølgende elementer i en matrise, mens `diff(X, n)` anvender `diff`-operatoren `n` ganger på matrisa n. Dette tilsvarer den n-te deriverte. Fordi utgangsmatrisa er $p \times p$-diagonal, vil også `T` være $p \times p$. Det brukes `sparse` og `speye` for å gjøre utregningene raskere da man ikke trenger å ta med unødvendige nullelementer. `TregsMulti`-funksjonen (vedlegg A.1.12) er en utvidelse av `Ridge_fastLOOCV_SVD` som er laget for å gå så raskt som mulig. Den bygger på en funksjon jeg har fått av min veileder modifisert til å kunne behandle mer enn én variabel. Tidsbruk for `TregsMulti` og `Ridge_fastLOOCV_SVD` er illustrert i figur 3.7.

I figur 3.8 er det vist en sammenlikning av `TregsMulti` og `Ridge_fastLOOCV_SVD` for uni- og multivariat tilfelle der jeg har brukt regularisering med T_2. Her har jeg brukt sistnevnte funksjon på den transformerte datamatrisa XT_2^{-1}. Man kan også se på standardisering av data som en form for regularisering, der man analyserer X transformert med den inverse



(a) Spectra (b) Sugar

Figur 3.7: Tidsbruk for `TregsMulti` og `ls_full_LOOCVfast_svd_Ridge`. I begge tilfeller er det brukt 50 λ-verdier $\log_{10}$-distribuert mellom 10^{-4} og 10^4.



(a) Spectra (b) Sugar

Figur 3.8: Tidsbruk for `TregsMulti` og `ls_full_LOOCVfast_svd_Ridge`. I begge tilfeller er det brukt 50 λ-verdier $\log_{10}$-distribuert mellom 10^{-4} og 10^4.

av den diagonale matrisa S bestående av standardavvikene til hver forklaringsvariabel. I tilfeller der det er naturlig å standardisere data vil det være lite trolig at man kan forvente kontinuitet, så standardisering vil først og fremst bli brukt sammen med Ridge-regularisering.

Regularisering på flere kriterier

Som foreslått i del 2.6.1 er det mulig å regularisere på flere kriterier. Jeg har sett på to måter å gå fram på for å få til dette.

Den første metoden går ut på å anta at μ og λ er uavhengige i

$$\Xi = \begin{bmatrix} X \\ \sqrt{\mu} I_p \\ \sqrt{\lambda} T \end{bmatrix} = \begin{bmatrix} Z(\mu) \\ \sqrt{\lambda} T \end{bmatrix},$$

slik at den verdien av μ som gir lavest PRESS ved Ridge-regresjon av X, vil være den samme som gir lavest press for analyse av Ξ definert over. Det skal sies at dette er en noe ullen antakelse.

For å bygge modellen, gjør man derfor først en Ridge-regresjon med LOOCV av Z for å finne ut hvilken μ som minimerer PRESS. Deretter gjør man en ny Tikhonov-regresjonsanalyse av problemet

$$ZB = \tilde{Y},$$

hvor de p siste radene av $\tilde{Y}$ kun består av nullelementer. For univariate data må man gjøre to SVDer, mens for multivariate med m responser, må man gjøre $m + 1$ SVDer. Dette er likevel ganske billig med tanke på antall regneoperasjoner. Jeg har laget en implementering av dette i funksjonen `TregsMulti2C` i vedlegg A.1.13.

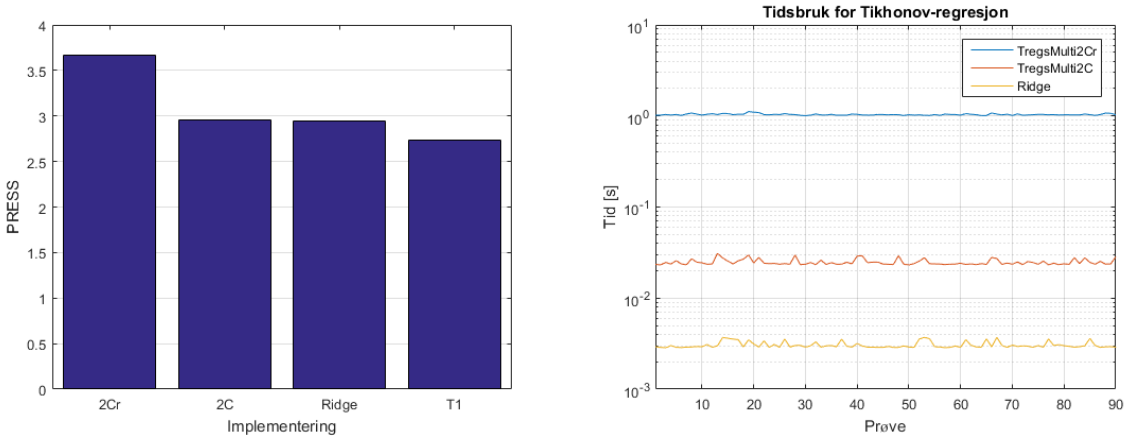
En annen metode å gjøre dette på er å gjøre regulariseringen for mange forskjellige μ- og λ-verdier, slik at antallet modeller man til slutt sammenlikner er antallet μ-verdier multiplisert med antall λ-verdier. Jeg går man frem på samme måte som over, men for hver μ-verdi gjør man en egen λ-analyse. Hvis r er antall λ-verdier, blir antall SDVer man må gjøre $r + 1$. Selv om man her ikke straffes for å ha flere responsvariable, vil dette som regel kreve flere regneoperasjoner og ta lenger tid enn metoden beskrevet der man bare tester λ-verdier for én μ-verdi. Til gjengjeld har man ingen antakelse her om at μ og λ er uavhengige. Det viktigste argumentet for denne antakelsen var imidlertid at den gir billig utregning. I funksjonen `TregsMulti2Cr` (vedlegg A.1.14) har jeg implementert denne tilnærmingen.

Jeg forsøker så å teste disse to metodene mot `TregsMulti`. Det jeg er interessert i, er tidsbruk og PRESS. Jeg vil også sjekke hvor gode de forskjellige modellene er til å predikere test-data for de datasettene som har en test-/trenings-inndeling. Dette gjør jeg ved å sammenlikne summen av de kvadrerte testresidualene

$$\sum_{i=1}^{n}(y_i - y_{\text{pred},i})^2$$

der y_i er en faktisk målt responsverdi og $y_{\text{pred},i}$ er den modellpredikerte verdien.

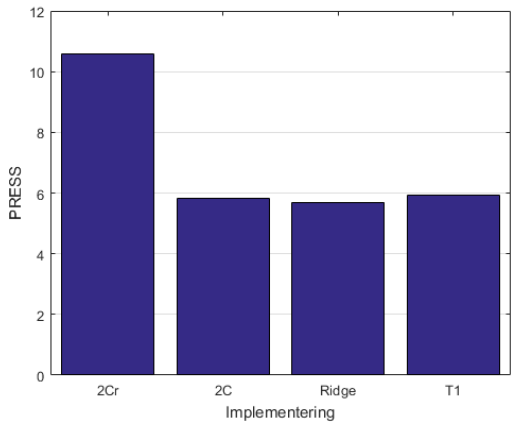
Ved sammenlikning av tidsbruk ser jeg kun på Ridge-regularisering når jeg bruker `TregsMulti`. Dette fordi denne går om lag like raskt uavhengig av type regularisering. Jeg har gjort analyser av Spectra (Kalivas, 1997 [16]), Biscuit Doughs (Osborne et al., 1984 [23]), Sugar (Brown, 1992 [5]) og Fett (Næs et al., 2013 [22]) i figur 3.9 - 3.14. I alle tilfeller er det brukt 30 λ- og μ-verdier $\log_{10}$-fordelt mellom 10^{-4} og 10^4. Ved sammenlikning av PRESS og eventuell test-prediksjon, har jeg sett på de to implementeringene med to kriterier regularisert på norm og førstederiverte, samt ren Ridge-regresjon og ren T1-regresjon.



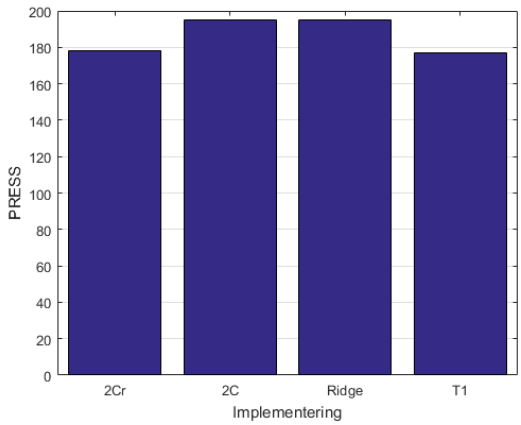
(a) PRESS

(b) Tidsbruk

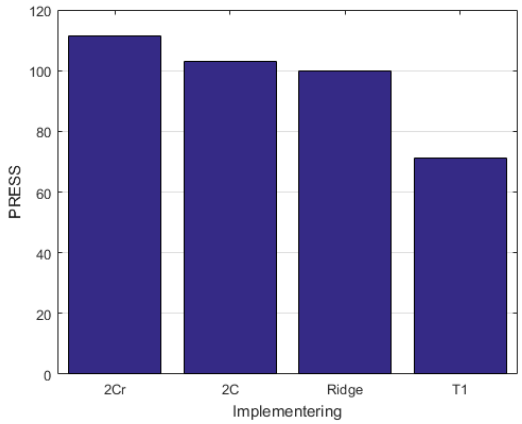
Figur 3.9: PRESS og tidsbruk for Spectra-datasettet.



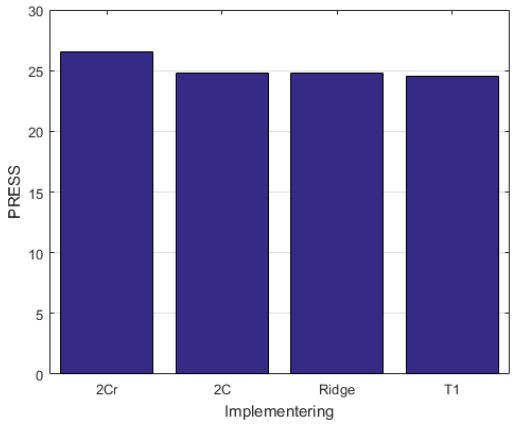
(a) PRESS, første respons



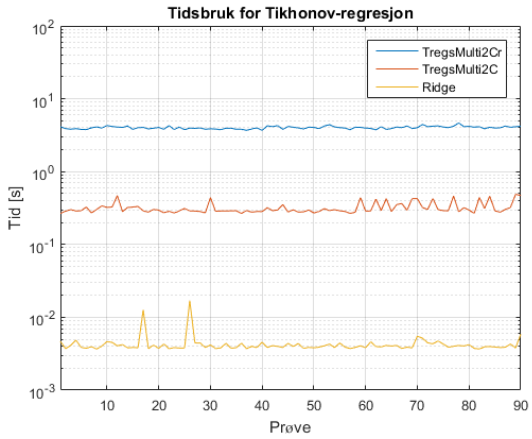
(b) PRESS, andre respons



(c) PRESS, tredje respons

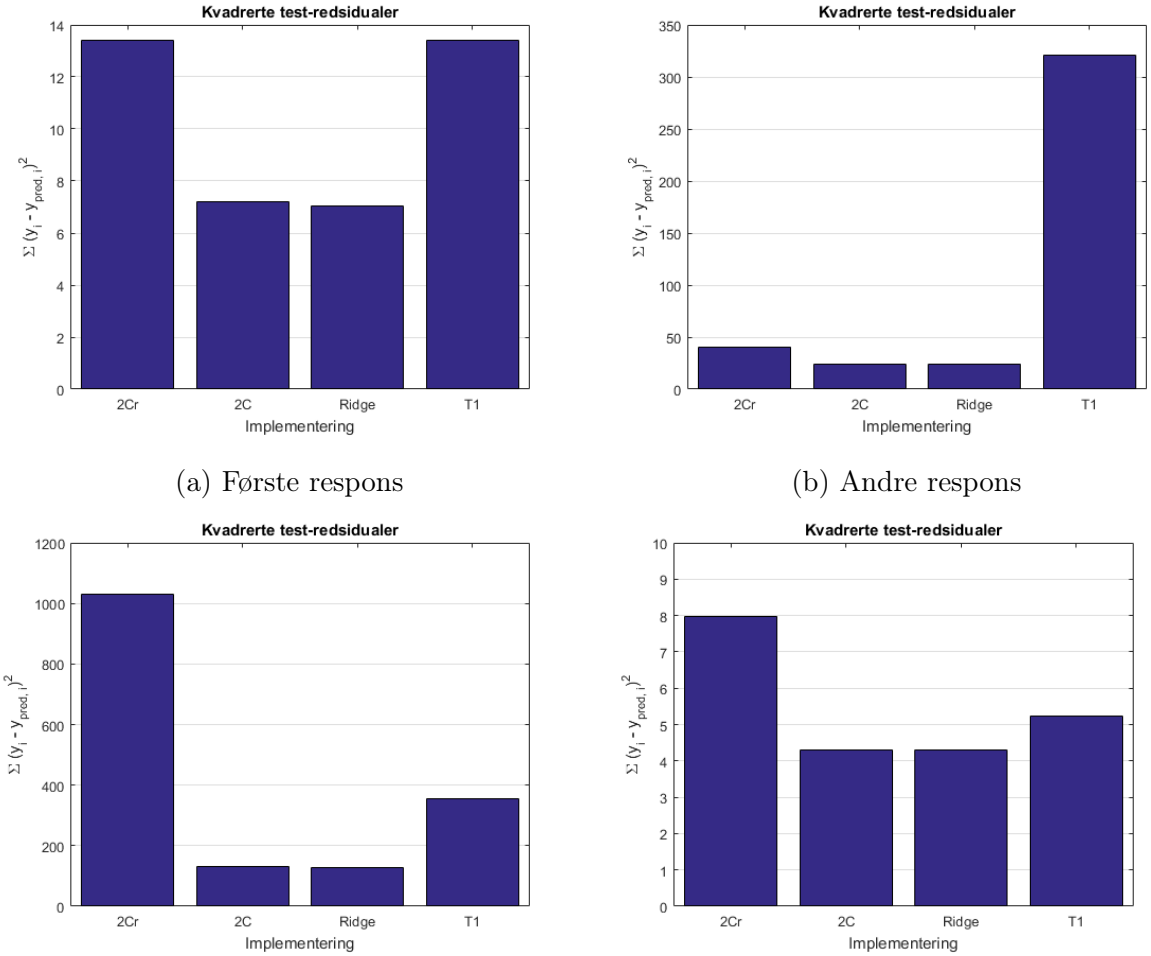


(d) PRESS, fjerde respons



(e) Tidsbruk

Figur 3.10: PRESS og tidsbruk for Biscuit Doughs



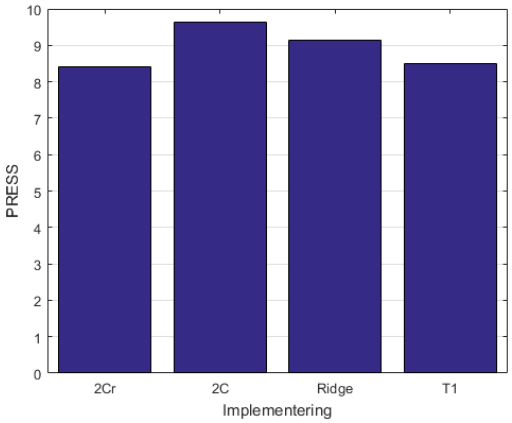
(a) Første respons

(b) Andre respons

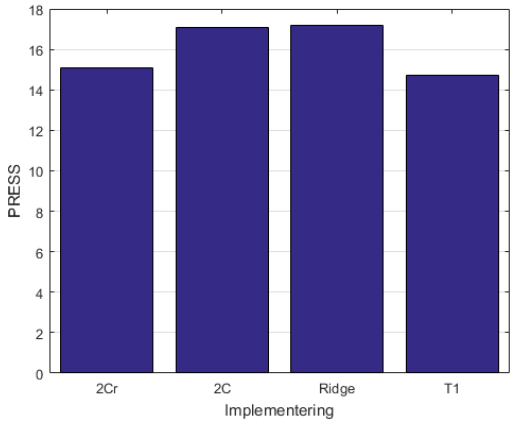
(c) Tredje respons

(d) Fjerde respons

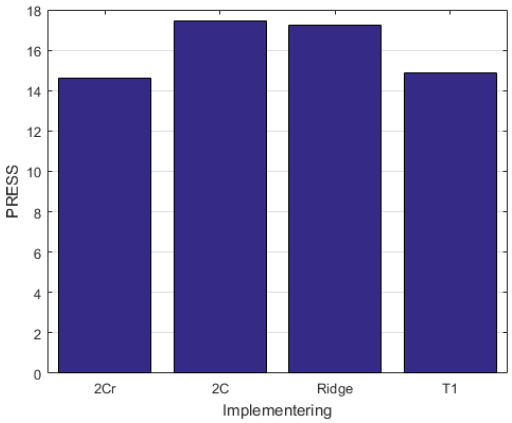
Figur 3.11: Kvadrert testprediksjonsfeil for Biscuit Doughs.



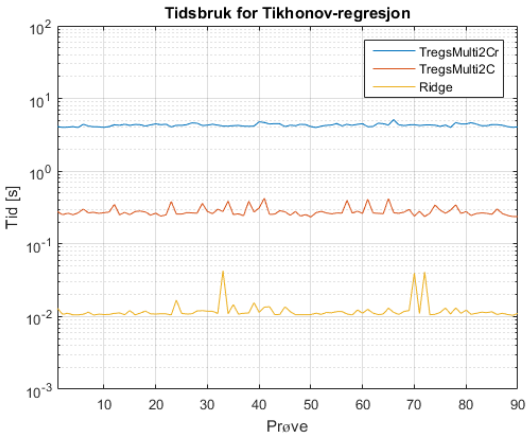
(a) PRESS, første respons



(b) PRESS, andre respons

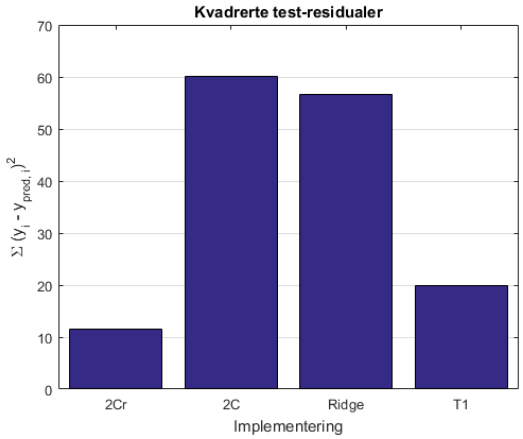


(c) PRESS, tredje respons

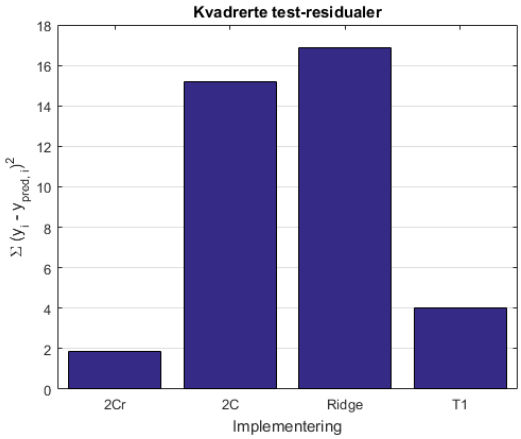


(d) Tidsbruk

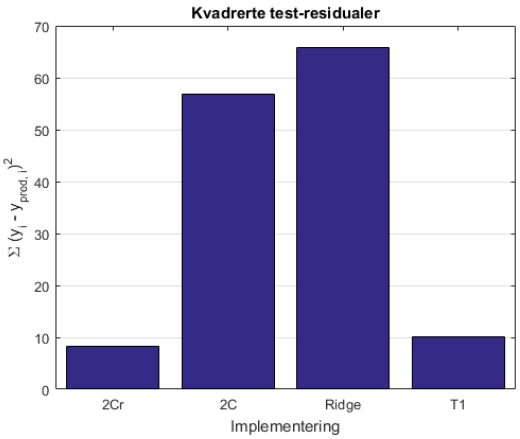
Figur 3.12: PRESS og tidsbruk for Sugar.



(a) Første respons

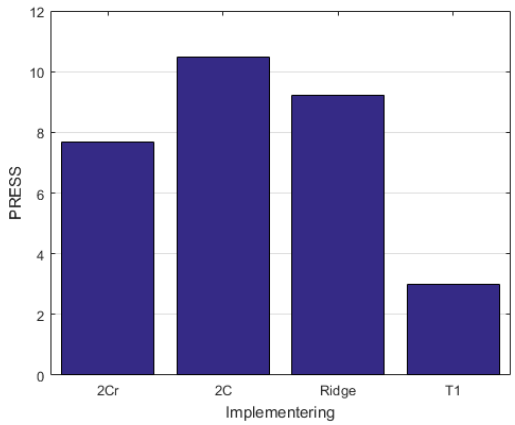


(b) Første respons

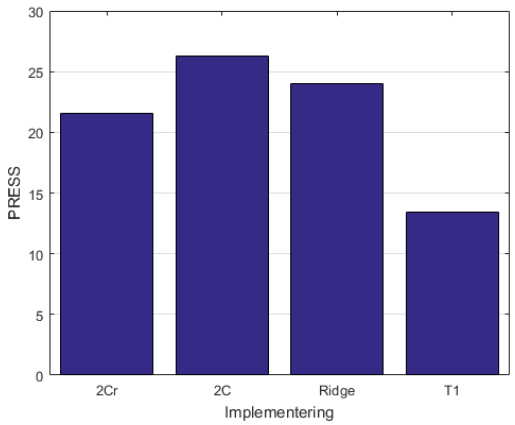


(c) Første respons

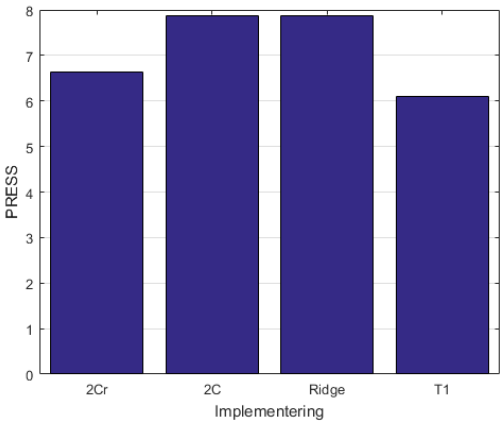
Figur 3.13: Kvadrert testprediksjonsfeil for Sugar.



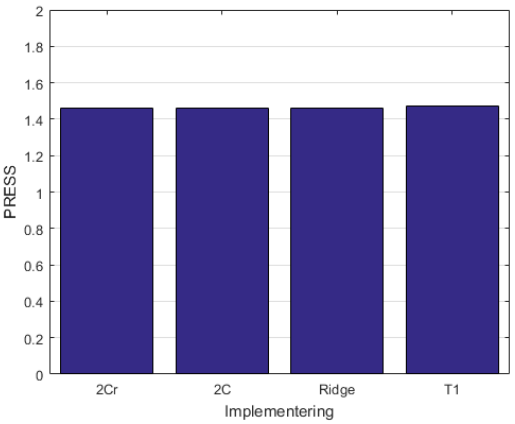
(a) PRESS, første respons, NIR



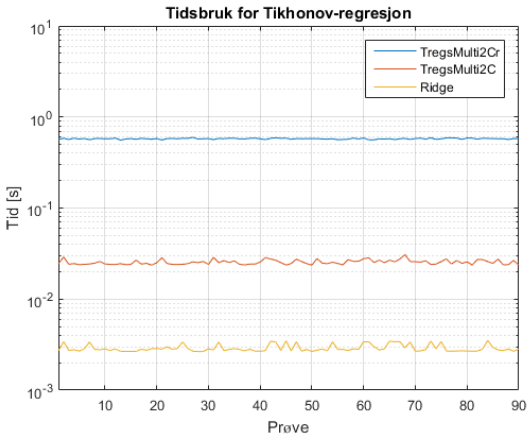
(b) PRESS, andre respons, NIR



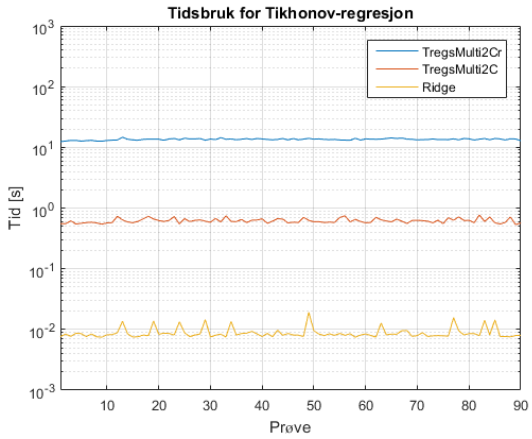
(c) PRESS, første respons, Raman



(d) PRESS, andre respons, Raman



(e) Tidsbruk NIR



(f) Tidsbruk Raman

Figur 3.14: PRESS og tidsbruk for Fett.

Det er kun testprediksjon av Sugar at noen av modellene med regularisering på to kriterier ser ut til å prestere bedre, og ikke uventet er dette den tyngste modellen å bygge fra `TregsMulti2Cr`. Selv om `TregsMulti2Cr` har en sum av kvadrerte testresidualer som er omlag halvparten av den for T_1-modellen, presterer begge disse modellene svært mye bedre enn `TregsMulti2C` og Ridge. Det samme gjelder for den tredje responsen, men her er forskjellen i sum av kvadrerte testresidualer relativt mindre.

Det er helt tydelig at Ridge-regresjon går svært mye raskere enn regresjon med regularisering på to kriterier for alle datasettene.

Det ser ut som om regularisering på flere kriterier kan bidra til økt prediksjonsevne, og dette gjelder særlig implementeringen der man ikke gjør antakelsen om at de to kriteriene er uavhengige. Dette er imidlertid den tyngste modellen å bygge fordi den krever ganske mange SVDer.

3.4 Sammenlikning med andre metoder

For Tikhonov-regularisering har jeg vist at derivasjonsregulariseringer kan tolkes som Ridge-regularisering av transformerte datamatriser (del 3.3.5), og jeg vil derfor gjøre sammenlikninger av derivasjonsregulariseringer med tilsvarende transformerte data jeg har anvendt PCR og PLSR på. Dette gjør at det ikke blir like naturlig å bruke regulariseringer på flere kriterier her.

3.4.1 PLSR og PCR

Når man gjør Tikhonov-regularisert regresjon, lager man modeller for mange forskjellige λ-verdier, og velger modellen med lavest PRESS-verdi. I PCR og PLSR har man ingen λ å bygge modeller utfra, men her kan man optimere PRESS-verdier utfra antall komponenter man tar med i modellen. For datasett med mer enn én respons, er det ikke sikkert det er det samme antallet komponenter som gir den beste modellen for alle responsene. Derfor er dette tatt høyde for i funksjonene `PCR2LOOCV` (vedlegg A.1.15) og `PLS2fastLOOCV` (vedlegg

A.1.16), der begge plukker ut det antallet komponenter som gir lavest PRESS for hver responsvariabel.

`PCR2LOOCV` og `PLS2fastLOOCV` er utvidelser av funksjoner for PCR og PLSR skrevet av min veileder Ulf Indahl. Jeg har modifisert disse til å kunne behandle multivariat respons samt å gjøre LOOCV. I `PLS2fastLOOCV` refererer «fast» ikke til kryssvalideringen, men til PLSR-implementeringen. Denne implementeringen går raskere enn den vanlige NIPALS-implementeringen blant annet ved at man gjør en SVD av X^TY og finner $\boldsymbol{w}$-vektoren utfra dette.

Jeg har forsøkt å gjøre utregning av PRESS så rask som mulig for PLSR- og PCR-funksjonene. Jeg skal blant annet teste om `TregsMulti` går raskere enn PCR og PLSR, og da er det et poeng å gjøre de sistnevnte så tidseffektive som mulig. Det finnes ingen snarvei for rask kryssvalidering for PCR og PLSR slik det gjør for OLS og Tikhonov-regularisert regresjon, så her må modellen bygges på nytt for hver utelatte måling. PRESS-verdiene kan man imidlertid regne ut ganske effektivt. I stedet for å ta vare på hver residual:

```
res = zeros(n, m, n -1);
for i = 1:n
    inds = setdiff(1:n, i);
    betas = PLS2fast(X(inds, :), Y(inds, :), n-1);
    for j = 1:n - 1
        res(i, :, j) = bsxfun(@minus, Y(i, :), X(i, :)*betas);
    end
end
```

for så å kvadrere og summere disse, kan man summere opp PRESS direkte inni løkka. Dette viser seg å gå ganske mye raskere:

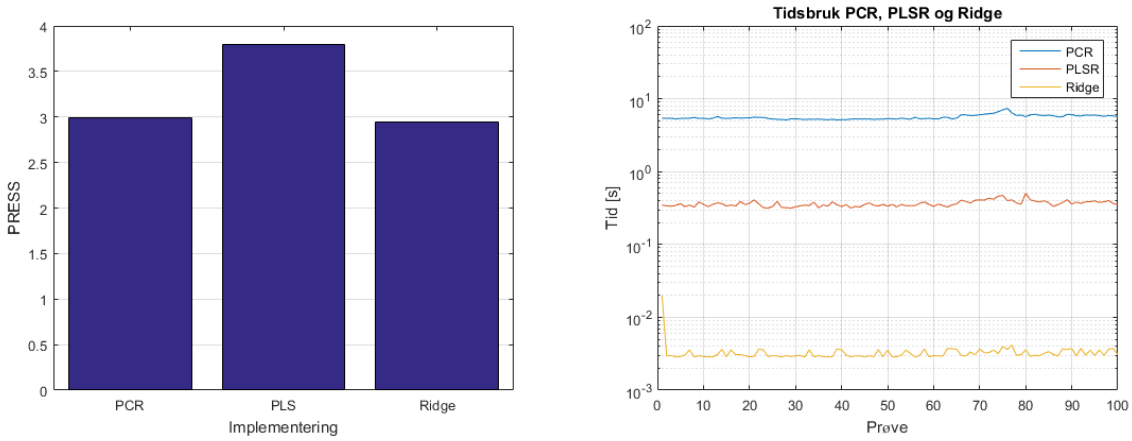
```
PRESS = zeros(n - 1, m);
for i = 1:n
    inds = setdiff(1:n, i);
    betas = PLS2fast(X(inds, :), Y(inds, :), n-1);
    for j = 1:n - 1
        res2 = bsxfun(@minus, Y(i, :), X(i, :)*betas);
        PRESS(j, :) = PRESS(j, :) + res2
    end
end
```

Jeg vil gjøre sammenlikning av PRESS og tidsbruk for Ridge, PCR og PLS, og for Sugar og Biscuit Doughs vil jeg også se på hvor godt de forskjellige metodenes modeller predikerer

test-data. Jeg vil også sammenlikne Tikhonov-regresjon med T_1-, T_2- og T_3-regularisering med tilsvarende transformerte data for PCR og PLSR. I alle tilfellene bruker jeg 30 λ-verdier $\log_{10}$-fordelt mellom 10^{-4} og 10^4.

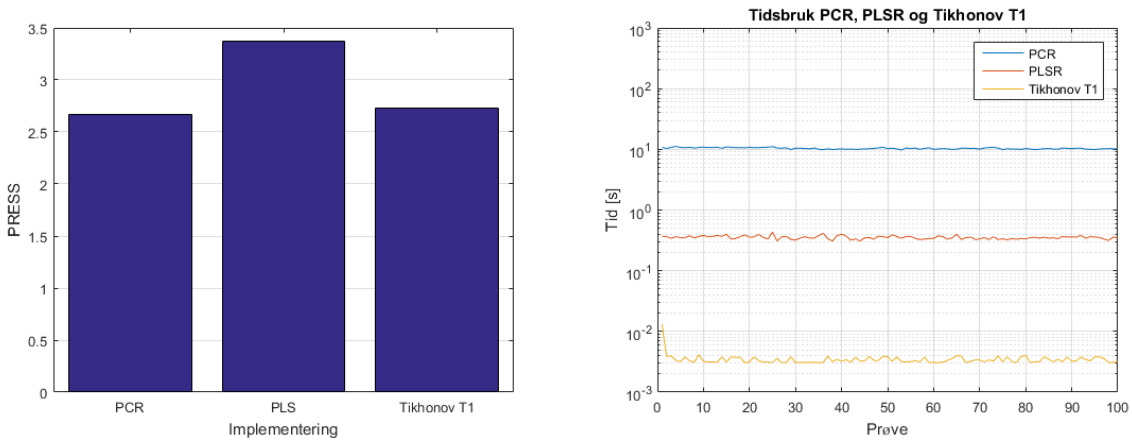
3.4.2 Spectra

Jeg tar først for meg det univariate Spectra-datasettet (Kalivas, 1997 [16]).



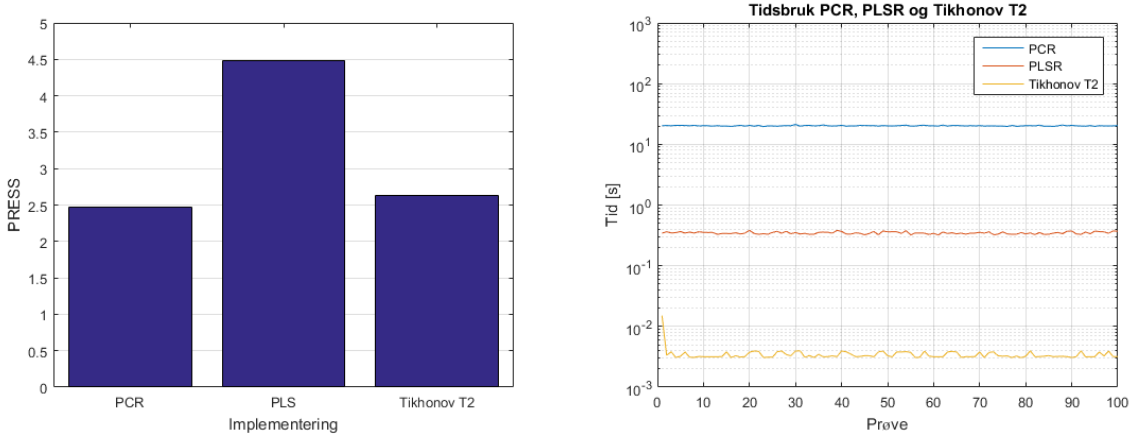
(a) PRESS (b) Tidsbruk

Figur 3.15: PRESS og tidsbruk for PCR, PLSR og Ridge-regresjon anvendt på Spectra-datasettet.



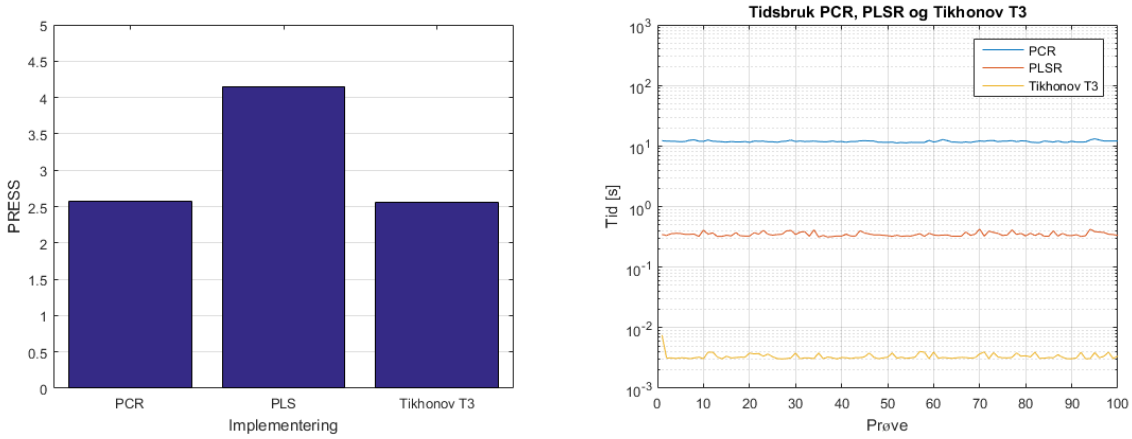
(a) PRESS (b) Tidsbruk

Figur 3.16: PRESS og tidsbruk for PCR, PLSR og T1-regresjon anvendt på Spectra-datasettet.



(a) PRESS (b) Tidsbruk

Figur 3.17: PRESS og tidsbruk for PCR, PLSR og T2-regresjon anvendt på Spectra-datasettet.

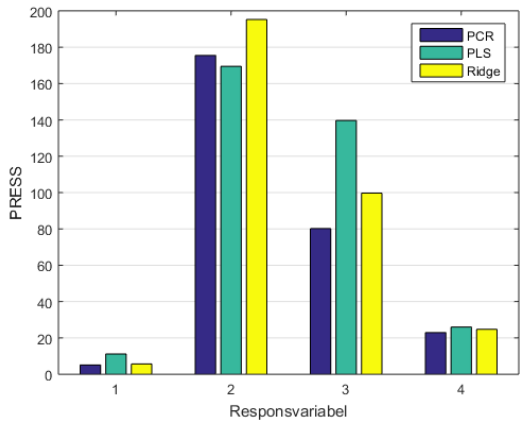


(a) PRESS (b) Tidsbruk

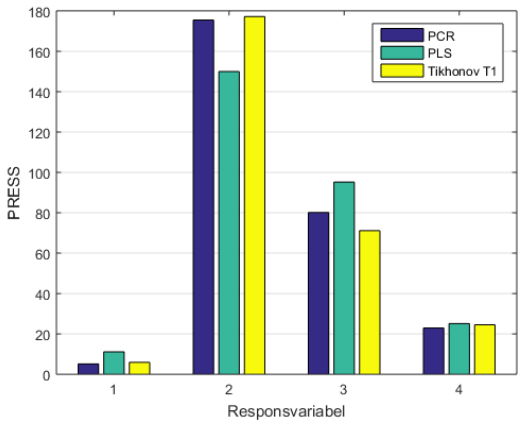
Figur 3.18: PRESS og tidsbruk for PCR, PLSR og T3-regresjon anvendt på Spectra-datasettet.

3.4.3 Biscuit Doughs

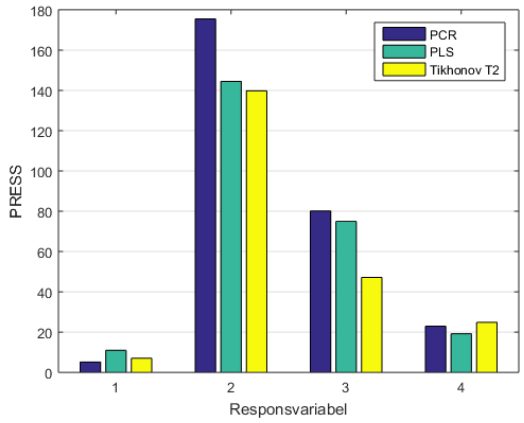
Jeg ser så på Biscuit Doughs-datasettet (Osborne et al., 1984 [23]). Jeg vil kun ta med ett plott av tidsbruk (Ridge) da antall utregninger blir de samme i alle tilfeller



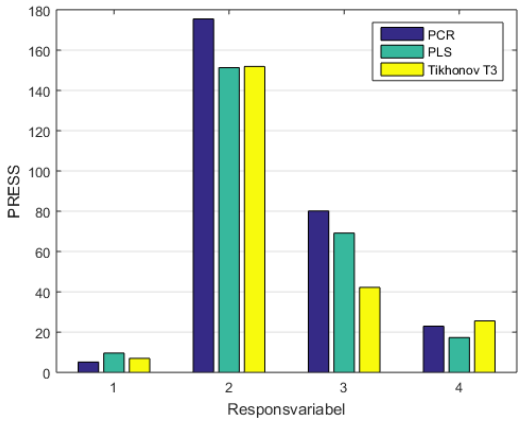
(a) PRESS for PCR, PLS og Ridge



(b) PRESS for T_1^{-1}-transformert PCR, PLS og Tikhonov

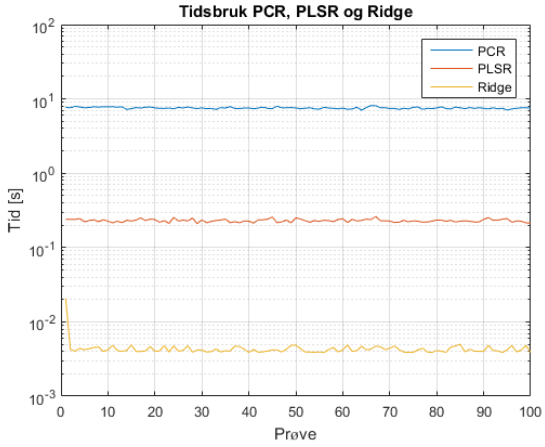


(c) PRESS for T_2^{-1}-transformert PCR, PLS og Tikhonov

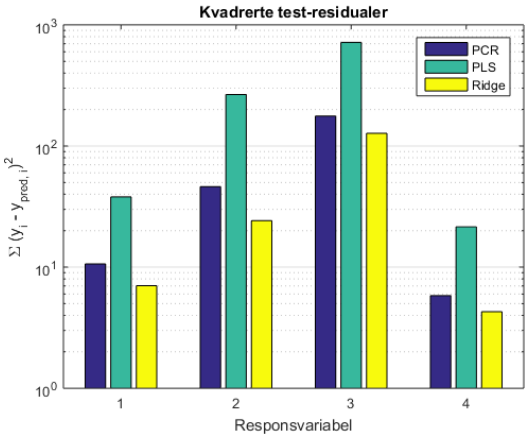


(d) PRESS for T_3^{-1}-transformert PCR, PLS og Tikhonov

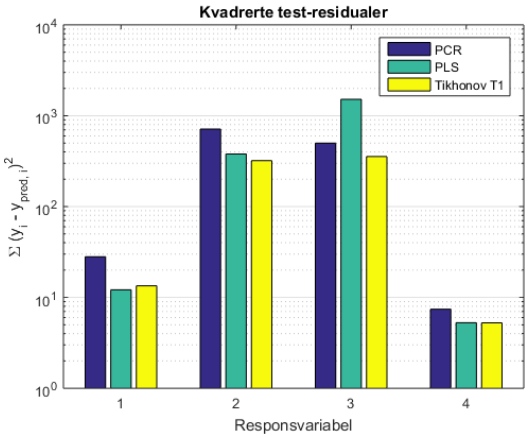
Figur 3.19: PRESS for PCR, PLSR og Tikhonov anvendt på Biscuit Doughs-datasettet.



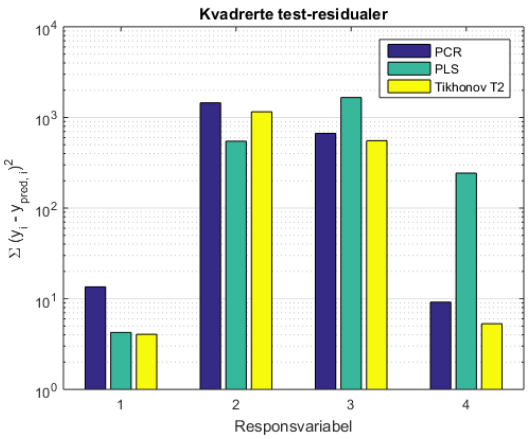
Figur 3.20: Tidsbruk for PCR, PLSR og Ridge-regresjon av Biscuit Doughs



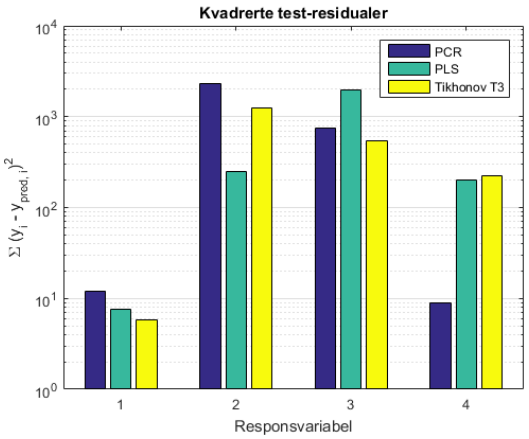
(a) Prediksjon av testsett for PCR, PLS og Ridge



(b) Prediksjon av testsett for T_1^{-1}-transformert PCR, PLS og Tikhonov



(c) Prediksjon av testsett for T_2^{-1}-transformert PCR, PLS og Tikhonov

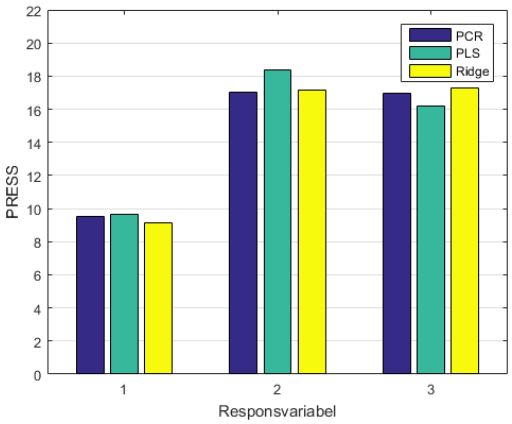


(d) Prediksjon av testsett for T_3^{-1}-transformert PCR, PLS og Tikhonov

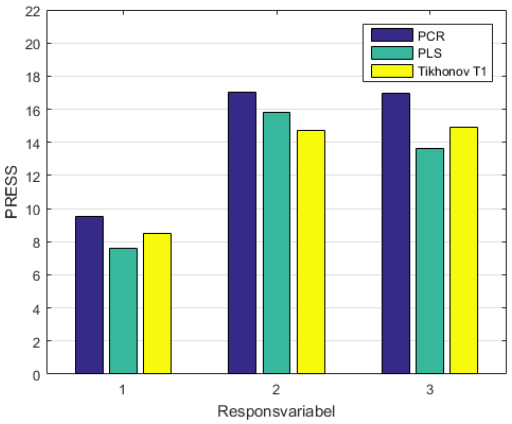
Figur 3.21: Prediksjon av testsett for PCR, PLSR og Tikhonov anvendt på Biscuit Doughs-datasettet.

3.4.4 Sugar

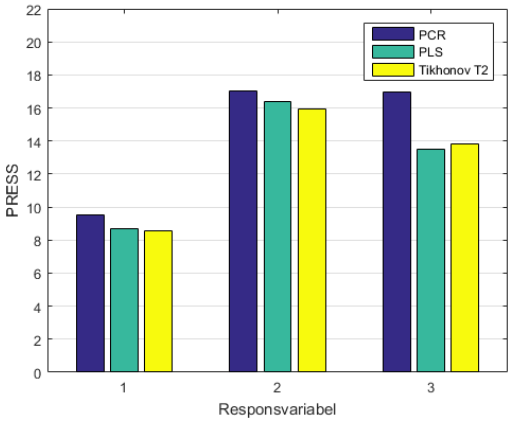
Under er det gitt plot av PRESS, tidsbruk og prediksjonsfeil for testdatasettet ved analyse av Sugar-datasettet (Brown, 1992 [5]).



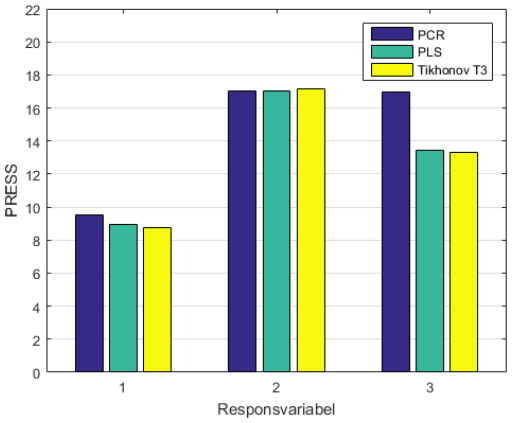
(a) PRESS for PCR, PLS og Ridge



(b) PRESS for T_1^{-1}-transformert PCR, PLS og Tikhonov

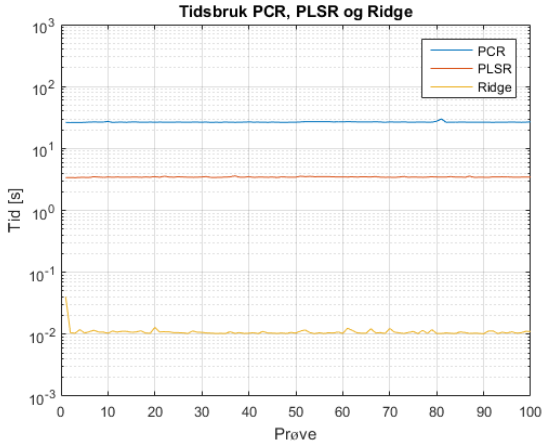


(c) PRESS for T_2^{-1}-transformert PCR, PLS og Tikhonov

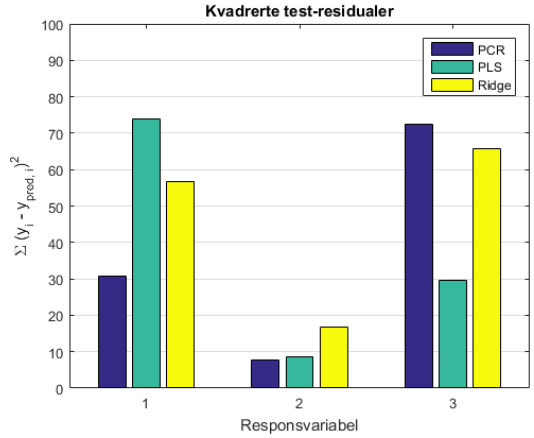


(d) PRESS for T_3^{-1}-transformert PCR, PLS og Tikhonov

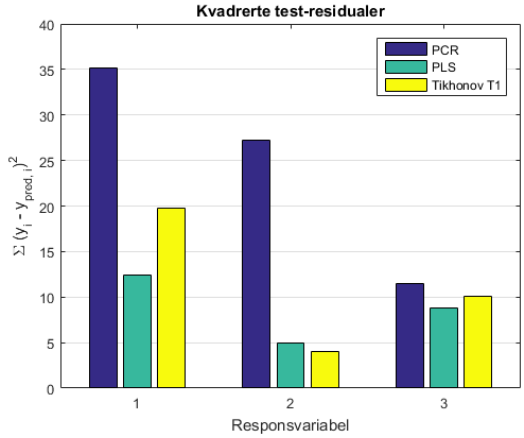
Figur 3.22: PRESS for PCR, PLSR og Tikhonov anvendt på Sugar-datasettet.



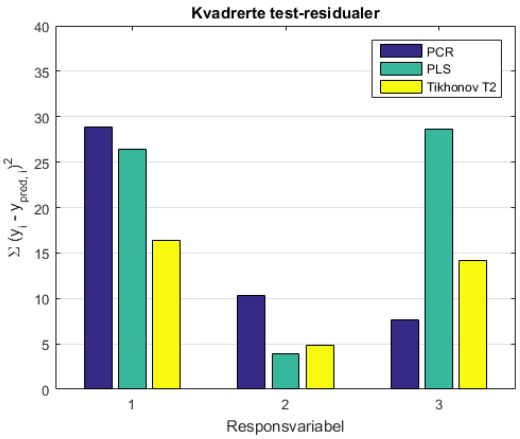
Figur 3.23: Tidsbruk for PCR, PLSR og Ridge-regresjon av Sugar



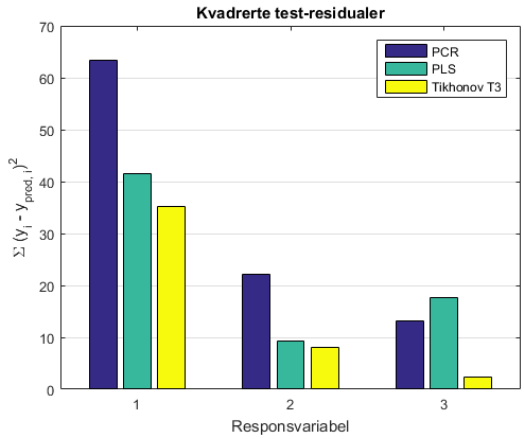
(a) Prediksjon av testsett for PCR, PLS og Ridge



(b) Prediksjon av testsett for T_1^{-1}-transformert PCR, PLS og Tikhonov



(c) Prediksjon av testsett for T_2^{-1}-transformert PCR, PLS og Tikhonov

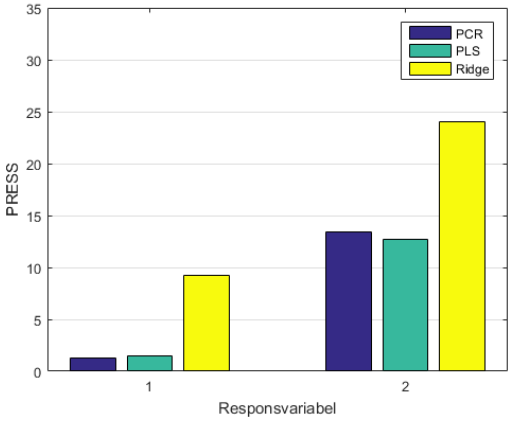


(d) Prediksjon av testsett for T_3^{-1}-transformert PCR, PLS og Tikhonov

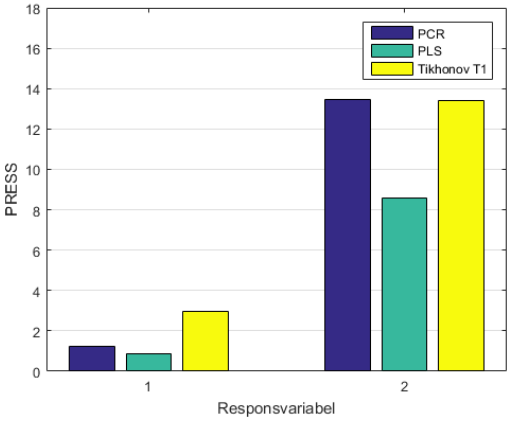
Figur 3.24: Prediksjon av testsett for PCR, PLSR og Tikhonov anvendt på Sugar-datasettet.

3.4.5 Fett - NIR

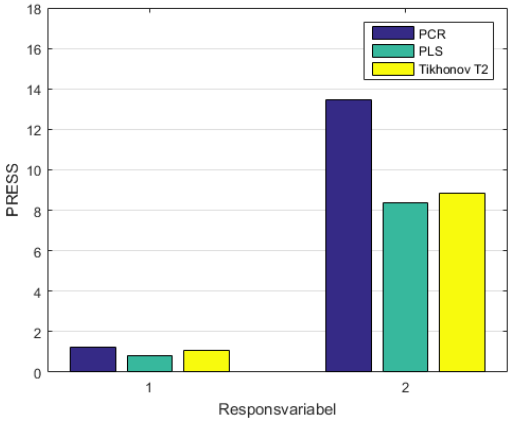
Videre ser jeg på NIR-datasettet av Fett (Næs et al, 2013 [22]).



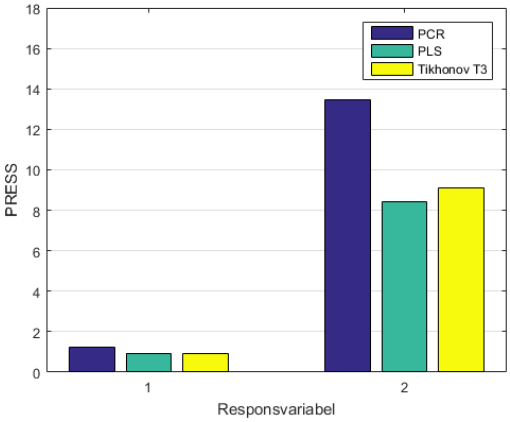
(a) PRESS ikke-transformert



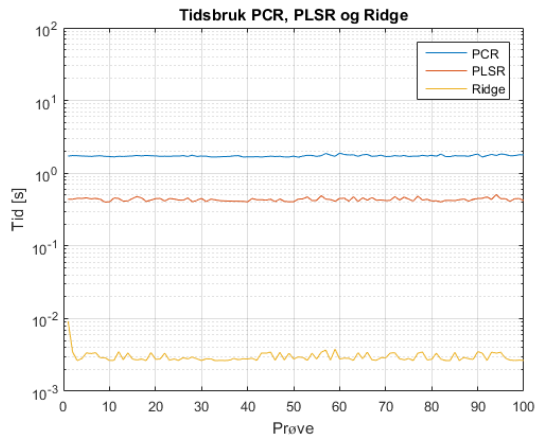
(b) PRESS T_1^{-1}-transformert



(c) PRESS T_2^{-1}



(d) PRESS T_3^{-1}-transformert

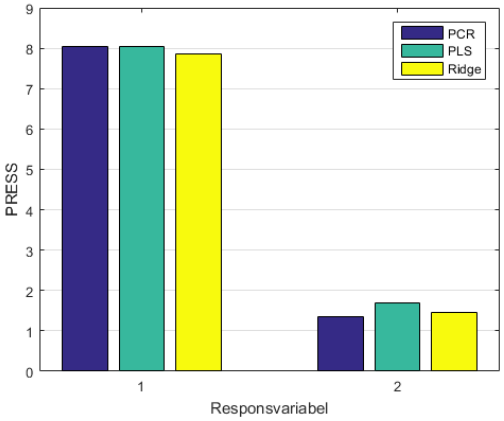


(e) Tidsbruk Ridge

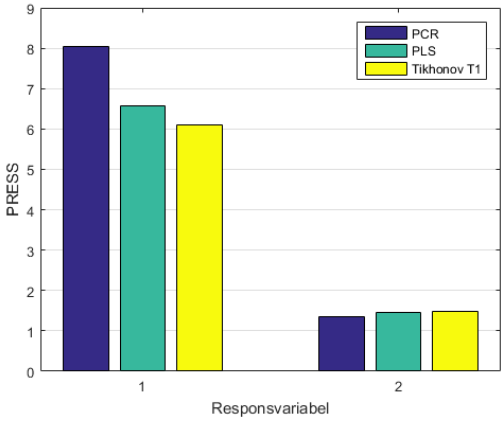
Figur 3.25: PRESS og tidsbruk for PCR, PLSR og Tikhonov anvendt på NIR-delen av Fett-datasettet.

3.4.6 Fett - Raman

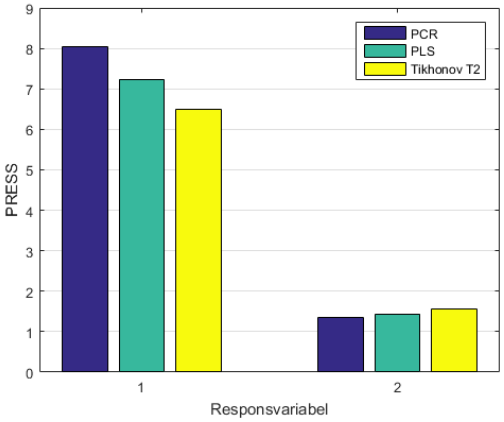
Til sist ser jeg på Raman-datasettet av Fett (Næs et at., 2013 [22]).



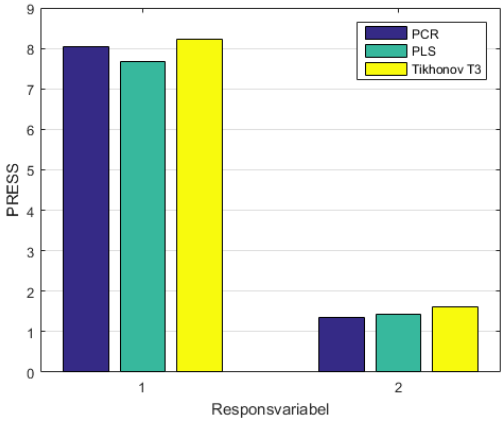
(a) PRESS ikke-transformert



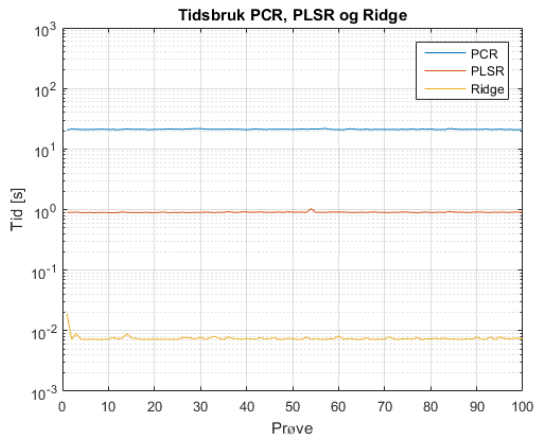
(b) PRESS T_1^{-1}-transformert



(c) PRESS T_2^{-1}



(d) PRESS T_3^{-1}-transformert



(e) Tidsbruk Ridge

Figur 3.26: PRESS og tidsbruk for PCR, PLSR og Tikhonov anvendt på NIR-delen av Fett-datasettet.

3.4.7 Test- og training-data

Som nevnt i del 3.2.2 er en av hovedgrunnene til at man bygger statistiske modeller at man ønsker å si noe om virkeligheten, man ønsker å bruke målinger man har gjort til å predikere en respons. Biscuit Doughs (Osborne et al., 1984 [23]) og Sugar (Brown, 1992 [5]) inneholder egne test- og training-datasett, så jeg har sett på hvilken av metodene som best predikerer datasettet ved å sammenlikne

$$\|\boldsymbol{r}_{\text{test}}\| = \|\boldsymbol{y}_{\text{test}} - \boldsymbol{y}_{\text{pred}}\|,$$

der $\boldsymbol{y}_{\text{test}}$ er de målte responsverdiene for test-datasettet og $\boldsymbol{y}_{\text{pred}}$ er de LOOCV-predikerte verdiene $X_{\text{text}}\hat{\boldsymbol{\beta}}$. Den modellen som gir lavest verdi for denne residualen, predikerer test-settet best. Resultater er gitt i tabell 3.1. Resten av regresjonsdatasettene har ingen egne

Tabell 3.1: Prediksjonsfeil for Tikhonov-regularisert minste kvadraters regresjon og PLSR. Den andre sølyen angir transformasjon, der data er blitt tranformert med T^{-1}. I de to første tilfellene er det ikke skjedd noen tranformasjon, mens i de neste er transformasjon gjort med T-matrisene definert i (2.22) - (2.24).

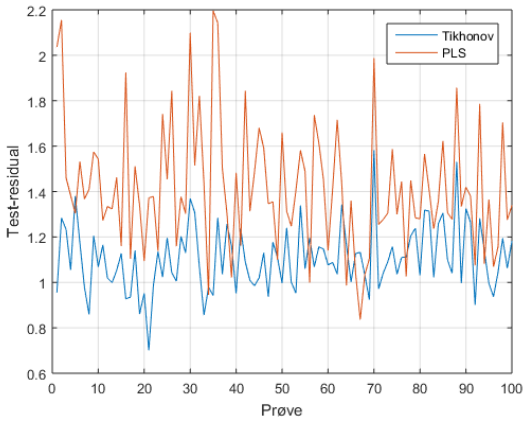
	T	Dough				Sugar		
		1	2	3	4	1	2	3
$\|\boldsymbol{r}_{\text{test}}\|_{\text{Ridge}}$	I	3,1980	4,7414	12,5049	2,0156	7,7462	4,1006	8,1129
$\|\boldsymbol{r}_{\text{test}}\|_{\text{PLS}}$		6,1675	16,3137	26,7907	4,6374	8,5910	2,2953	5,4451
$\|\boldsymbol{r}_{\text{test}}\|_{\text{Ridge}}$	T_1	3,7410	19,8873	18,6286	2,1647	4,2670	2,0151	2,2360
$\|\boldsymbol{r}_{\text{test}}\|_{\text{PLS}}$		3,4920	19,4728	38,9422	2,2919	3,5170	2,2366	2,9685
$\|\boldsymbol{r}_{\text{test}}\|_{\text{Ridge}}$	T_2	2,0320	35,7736	22,7390	1,9208	4,1547	2,2338	3,7715
$\|\boldsymbol{r}_{\text{test}}\|_{\text{PLS}}$		2,0616	23,3907	40,7801	15,5949	5,1354	1,9774	5,3500
$\|\boldsymbol{r}_{\text{test}}\|_{\text{Ridge}}$	T_3	2,7460	37,9830	23,6442	13,9255	5,8775	2,4990	2,4020
$\|\boldsymbol{r}_{\text{test}}\|_{\text{PLS}}$		2,7432	15,7325	44,1641	14,0960	6,4455	3,0576	4,1987

test- og training-datasett, så jeg har gjennomført 100 analyser av hver variabel der jeg har plukket ut tilfeldige del-datasett. Dersom det er et mønster i forskjellen på testresidualnormen for Tikhonov-regresjon og PLSR, gir dette noen indikasjoner om prediksjonsevnen til

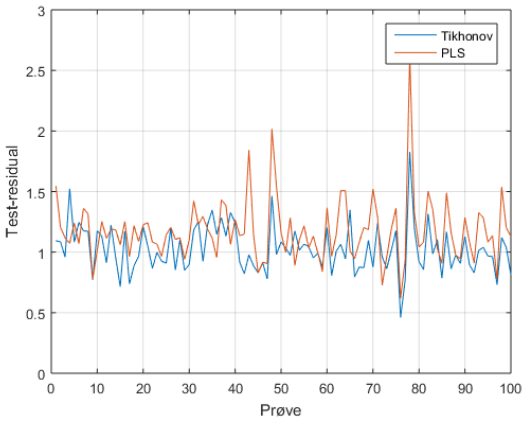
de to modellene. Som for Dough- og Sugar-datasettene har jeg gjennomført sammenlikning mellom Ridge-regresjon og PLSR, både uten transformasjoner og etter å ha transformert datamatrisene med henholdsvis T_1^{-1}, T_2^{-1} og T_3^{-1} definert i (2.22) - (2.24).

Spectra (Kalivas, 1997 [16])

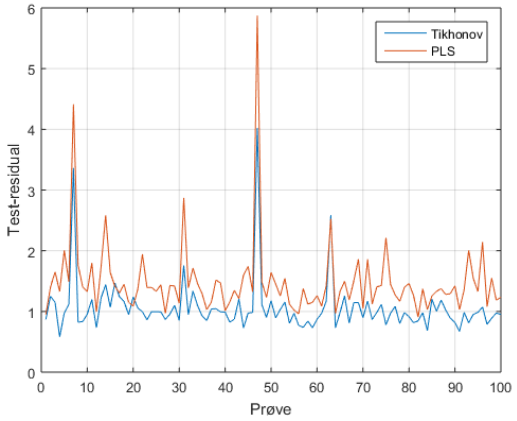
Jeg ser først på Spectra-datasettet, som kun har én responsvariabel. Datasettet består av totalt 60 målepunkter, så jeg har tilfeldig fordelt 40 til training- og 20 til test-datasettene. Resultatene er gitt i figur 3.27.



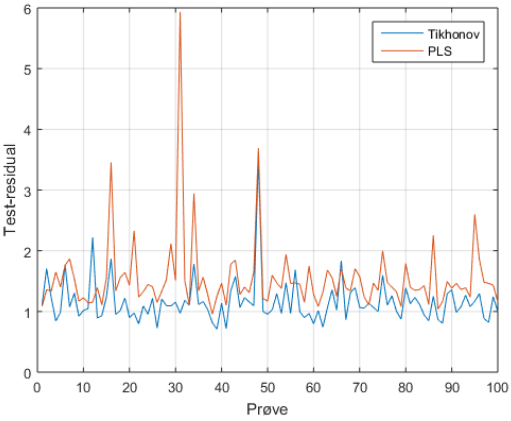
(a) Ikke-transformert data



(b) T_1^{-1}-transformert data



(c) T_2^{-1}-transformtert data

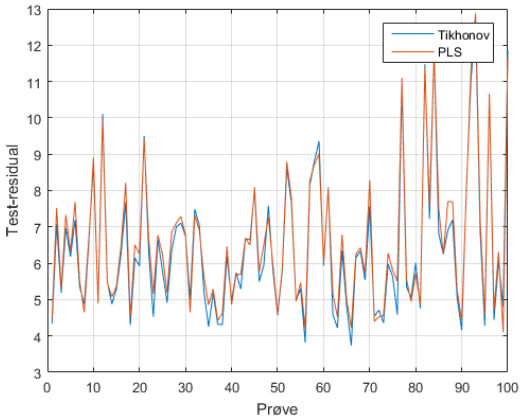


(d) T_3^{-1}-transformert data

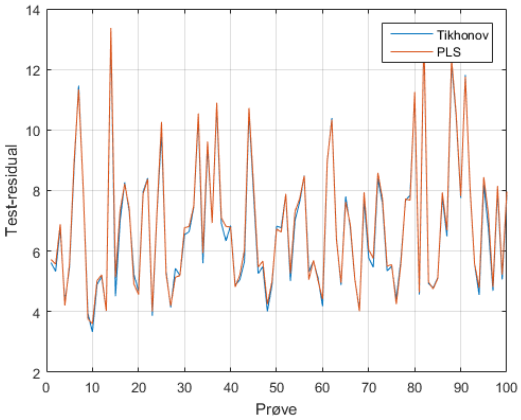
Figur 3.27: Sammenlikning av PLSR og Tikhonov-regresjon for 100 tilfeldige uttak av test- og training-datasett av Spectra (Kalivas, 1997 [16]). Det kan se ut til at Tikhonov-regresjon gir modeller med bedre prediksjonsevne for dette datasettet.

Fett - NIR

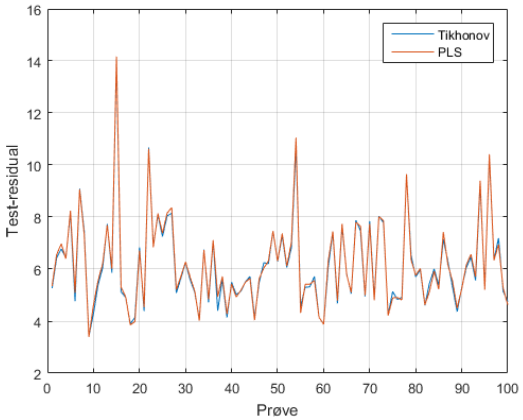
NIR-L-datasettet har to responsvariable. Disse variablene har tallverdier med flere størrelsesordner forskjell, og som i tidligere analyse velger jeg derfor å standardisere disse. Datasettet har 69 målinger, og jeg velger å bruke 45 av disse til training-data og de resterende 24 til test-data. Resultater er gitt i figur 3.28



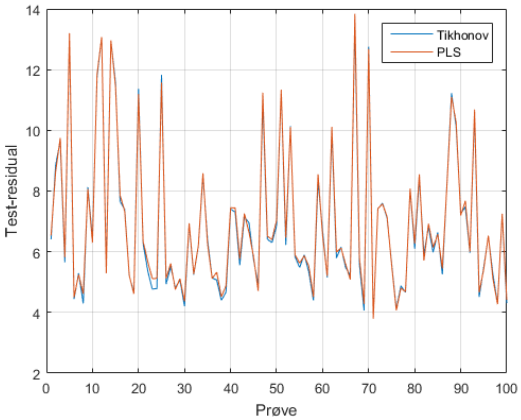
(a) PUFA1, ikke-transformert.



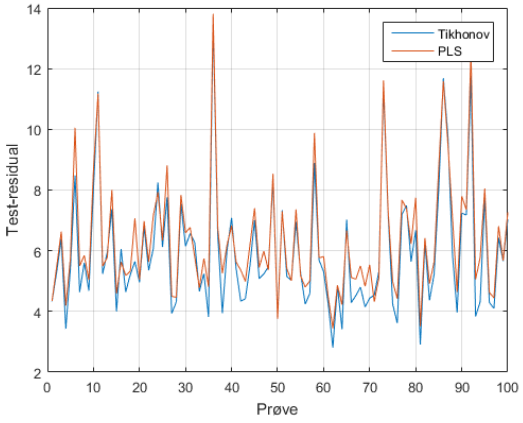
(b) PUFA1, T_1^{-1}-transformert.



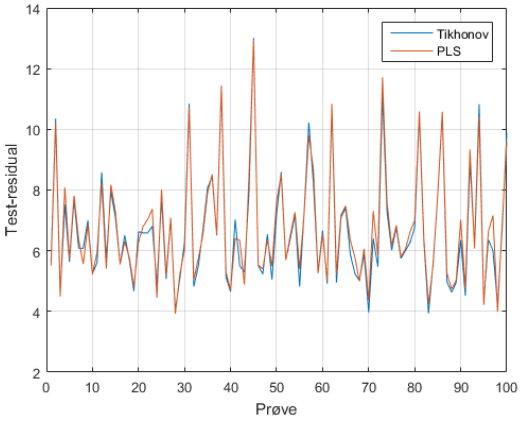
(c) PUFA1, T_2^{-1}-transformert.



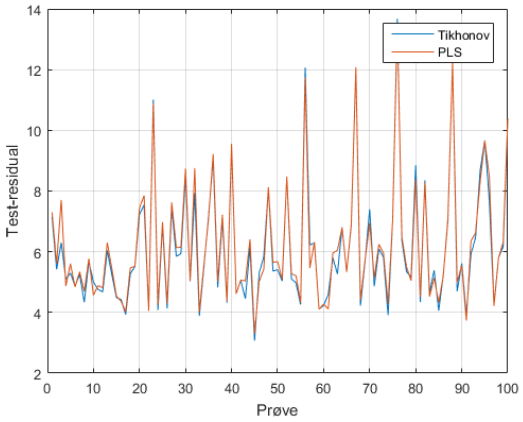
(d) PUFA1, T_3^{-1}-transformert.



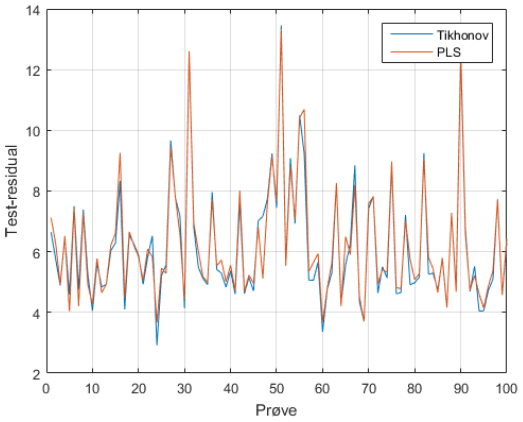
(a) PUFA2, ikke-transformert.



(b) PUFA2, T_1^{-1}-transformert.



(g) PUFA2, T_2^{-1}-transformert.

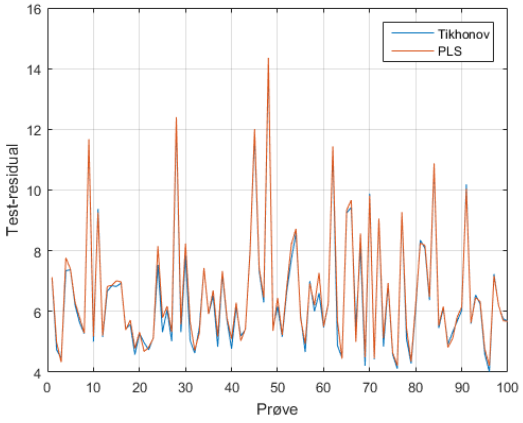


(h) PUFA2, T_3^{-1}-transformert.

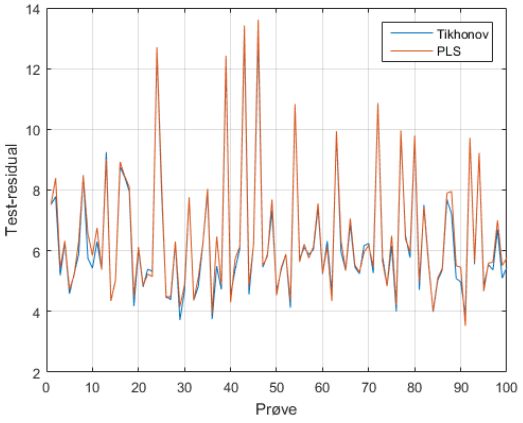
Figur 3.28: Plott av test-residualer for 100 tilfeldige fordelinger i test- og training-data for NIR-L-datasettet. Figur (a) - (d) viser resultater for den første PUFA-responsen, mens figur (e) - (h) viser resultater for den andre. Det ser ikke ut som det er noe åpenbart mønster i at den ene metoden predikerer bedre enn den andre.

Raman-L

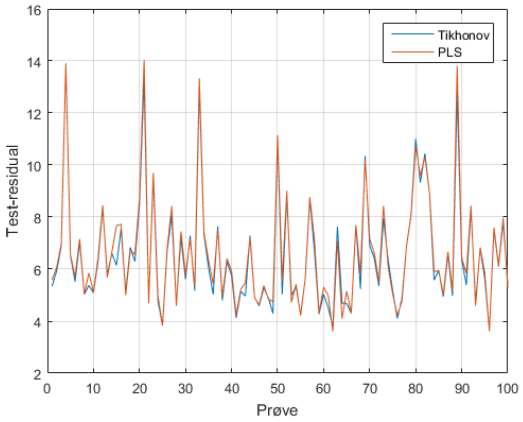
Raman-L har de samme to responsvariable som NIR-L, og disse er standardisert i analysen. Datasettet har 69 målinger, og jeg velger å bruke 45 av disse til training-data, mens de resterende 24 målingene utgjør test-datasettet. Resultater er gitt i figur 3.29.



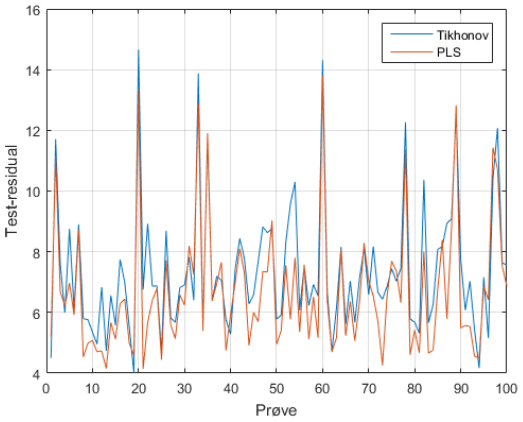
(a) PUFA1, ikke-transformert



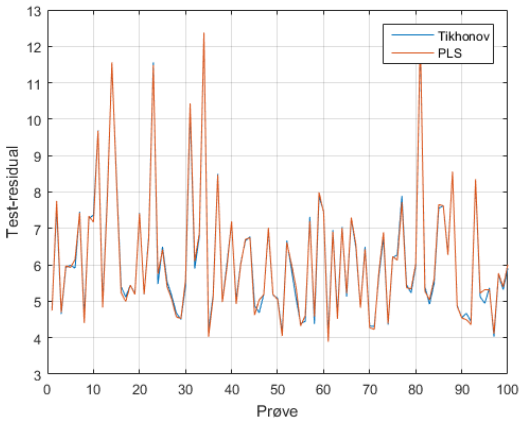
(b) PUFA1, T_1^{-1}-transformert



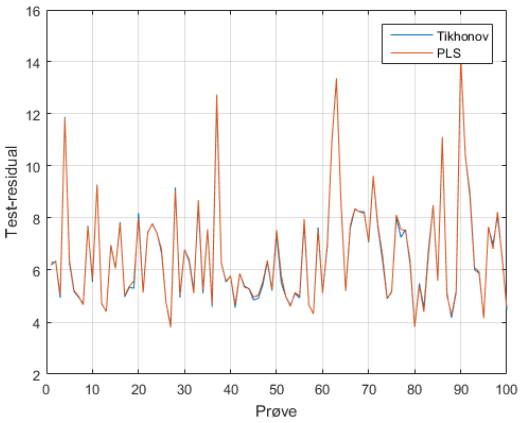
(c) PUFA1, T_2^{-1}-transformert



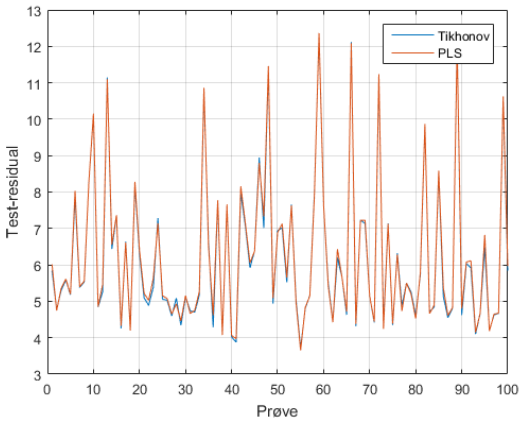
(d) PUFA1, T_3^{-1}-transformert



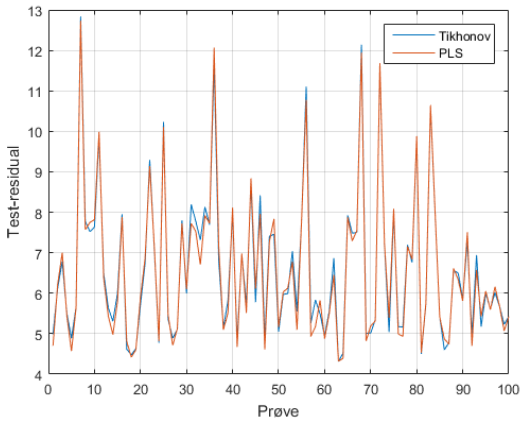
(e) PUFA2 ikke-transformert



(f) PUFA2, T_1^{-1}-transformert



(g) PUFA2, T_2^{-1}-transformert

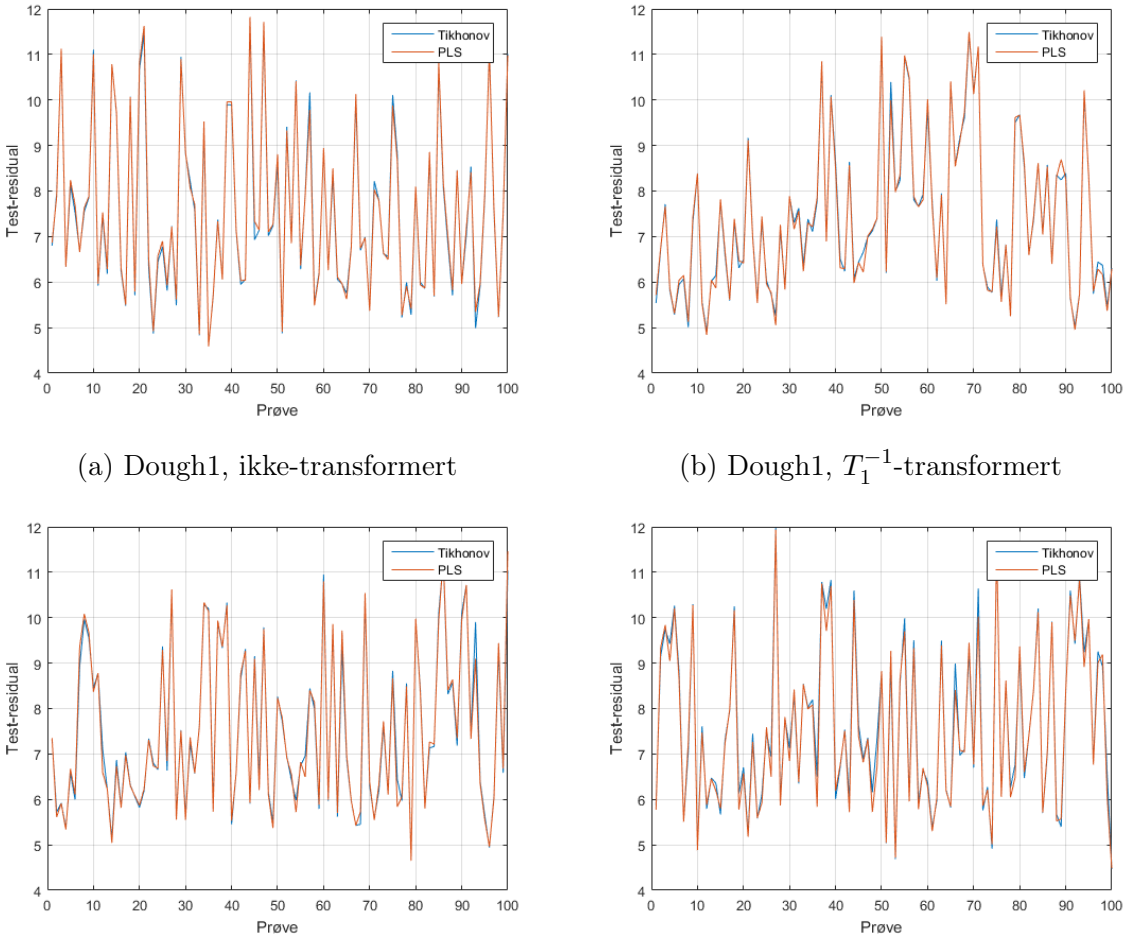


(h) PUFA2, T_3^{-1}-transformert

Figur 3.29: Plott av test-residualer for 100 tilfeldige fordelinger i test- og training-data for Ramnan-L-datasettet. Figur (a) - (d) viser resultater for den første PUFA-variabelen, mens figur (e) - (h) viser resultater for den andre.

Dough og sugar - tilfeldig valgte datainndelinger.

Den samme typen analyse kan selvfølgelig gjøres for de datasettene som allerede er inndelt i test- og training-data, ved å gjøre tilfeldige utplukk av training-data fra hele datamengden. Dette er gjort i figur 3.30 og 3.31 for Dough-datasettet (Osborne et al., 1984 [23]) og figur 3.32 for Sugar-datasettet (Brown, 1992 [5]). Det er ikke mulig å se noen store forskjeller i prediksjonsevne for de to metodene her heller.



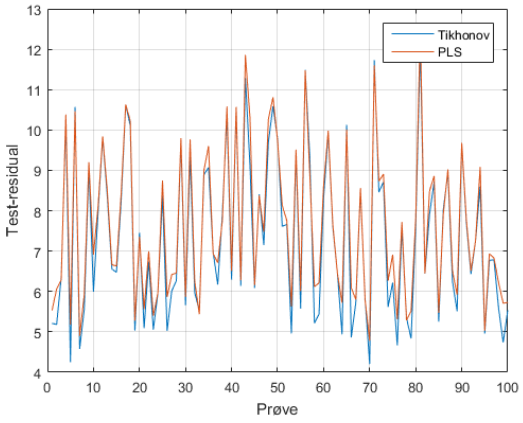
(a) Dough1, ikke-transformert

(b) Dough1, T_1^{-1}-transformert

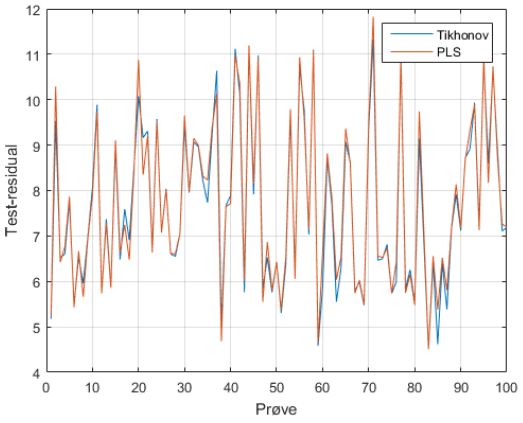
(c) Dough1, T_2^{-1}-transformert

(d) Dough1, T_3^{-1}-transformert

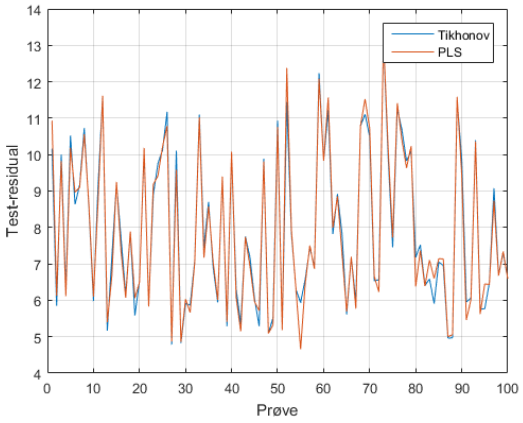
Figur 3.30: Plott av test-residualer for 100 tilfeldige fordelinger i test- og training-data for første responsvariabel av Dough-datasettet (Osborne et al., 1984 [23]).



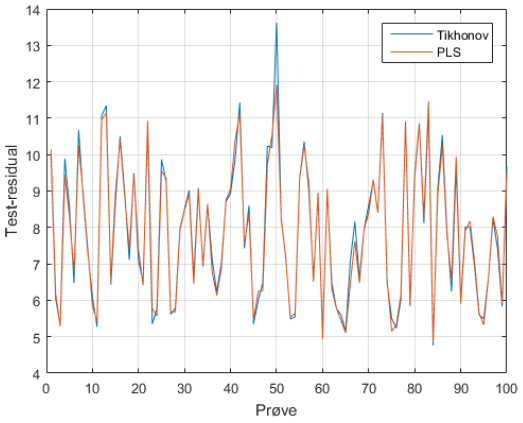
(a) Dough2, ikke-transformert



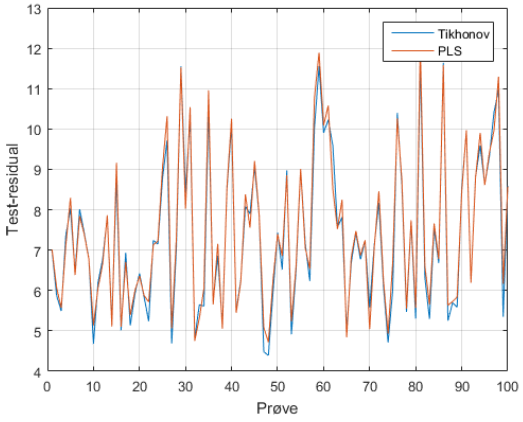
(b) Dough2, T_1^{-1}-transformert



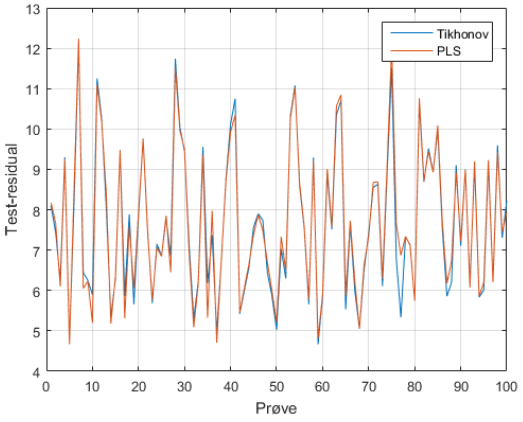
(c) Dough2, T_2^{-1}-transformert



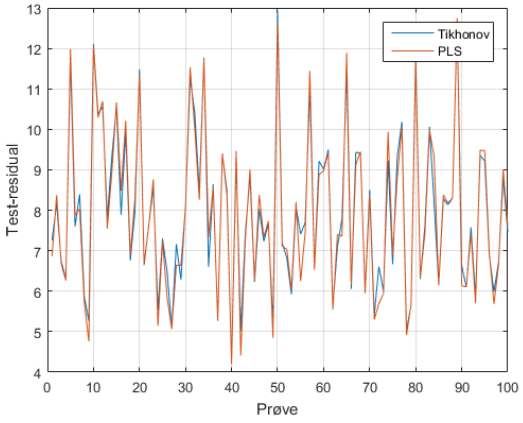
(d) Dough2, T_3^{-1}-transformert



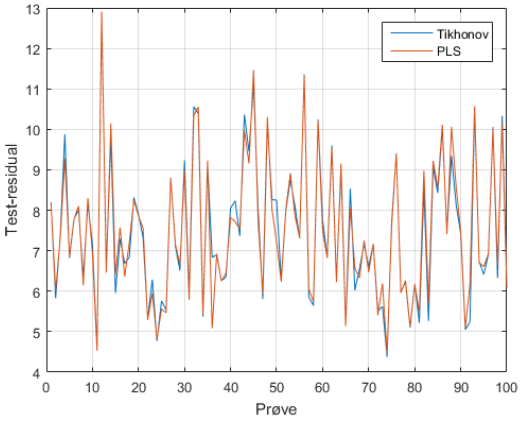
(e) Dough3, ikke-transformert



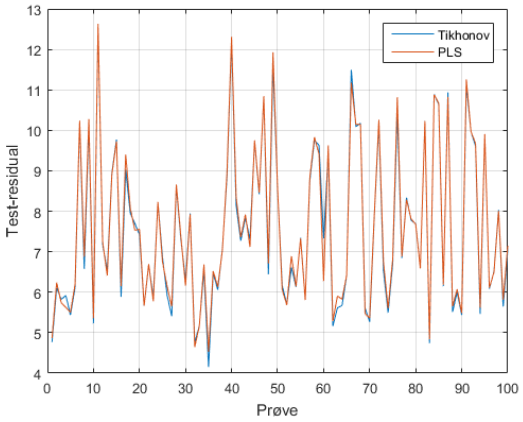
(f) Dough3, T_1^{-1}-transformert



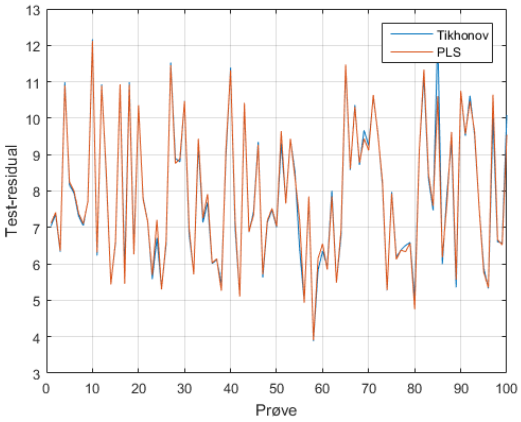
(g) Dough3, T_2^{-1}-transformert



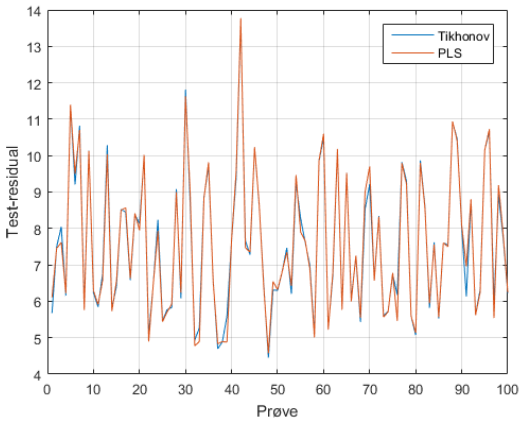
(h) Dough3, T_3^{-1}-transformert



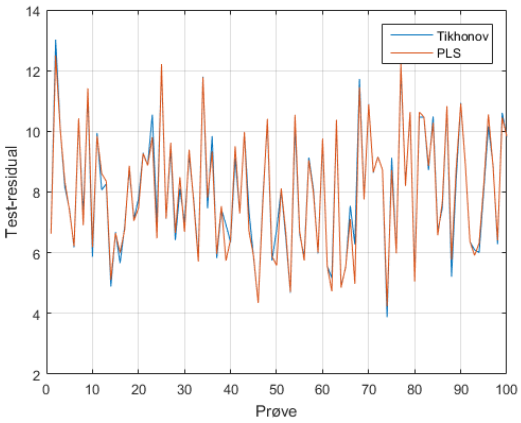
(i) Dough4, ikke-transformert



(j) Dough4, T_1^{-1}-transformert

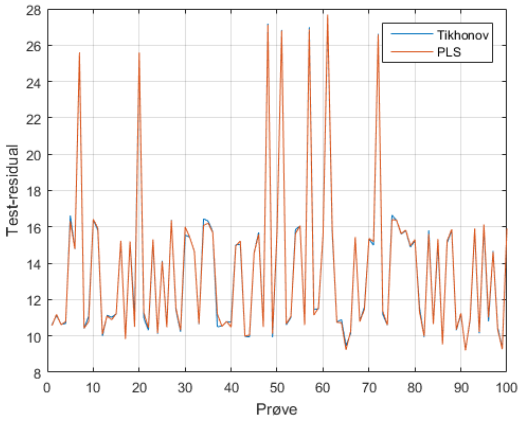


(k) Dough4, T_2^{-1}-transformert

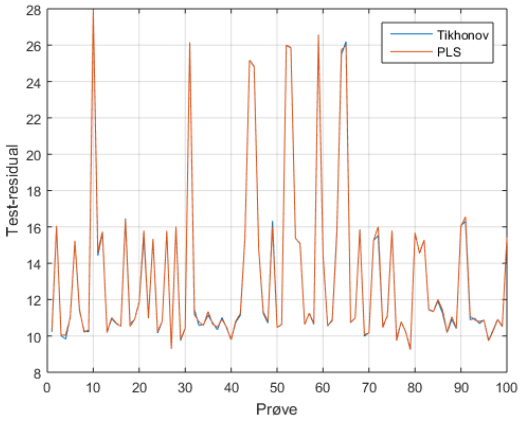


(l) Dough4, T_3^{-1}-transformert

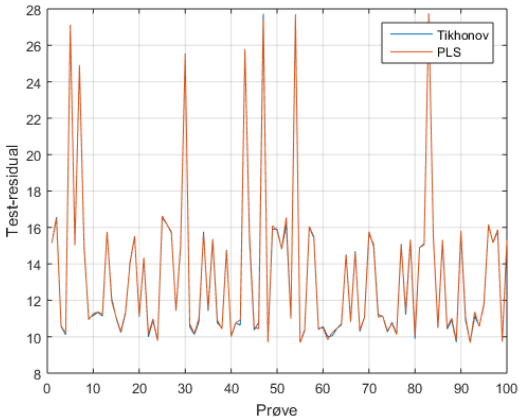
Figur 3.31: Plott av test-residualer for 100 tilfeldige fordelinger i test- og training-data for Dough-datasettet (Osborne et al., 1984 [23]). Figur (a) - (d) viser resultater for den andre responsvariabelen,figur (e) - (h) for den tredje, mens figur (i) - (l) viser resultater for den fjerde responsvariabelen.



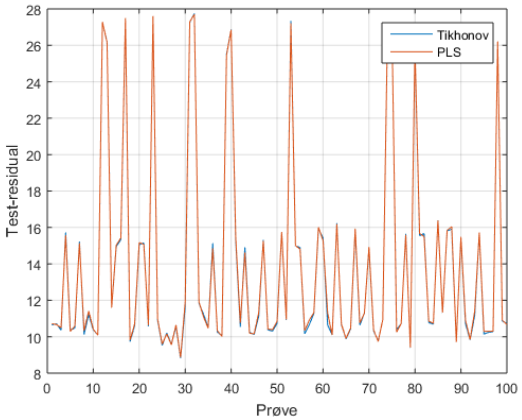
(a) Sugar1, ikke-transformert



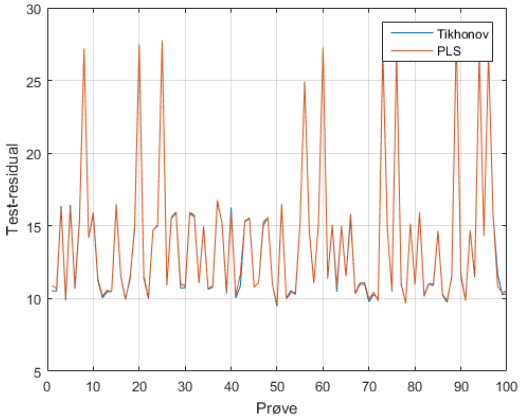
(b) Sugar1, T_1^{-1}-transformert



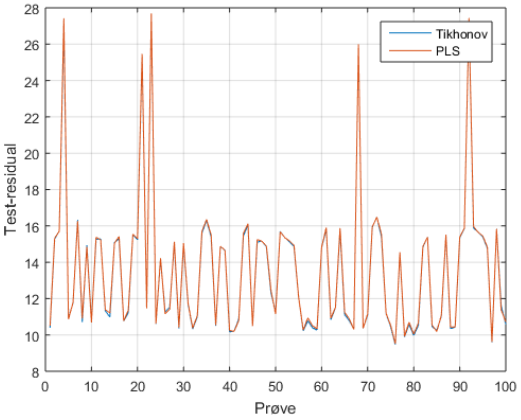
(c) Sugar1, T_2^{-1}-transformert



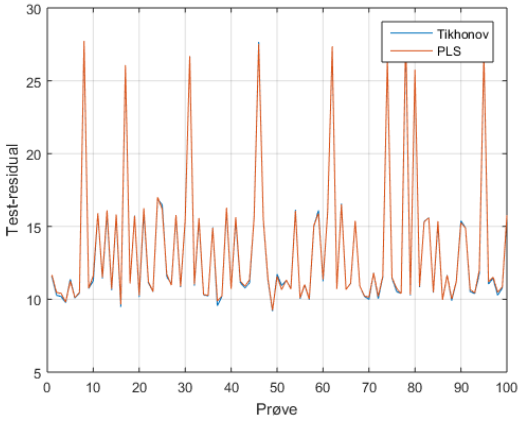
(d) Sugar1, T_3^{-1}-transformert



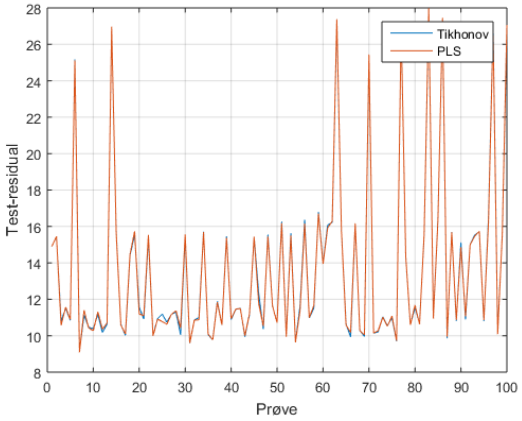
(e) Sugar2, ikke-transformert



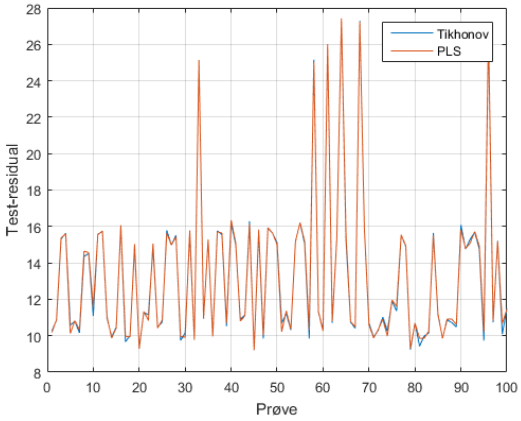
(f) Sugar2, T_1^{-1}-transformert



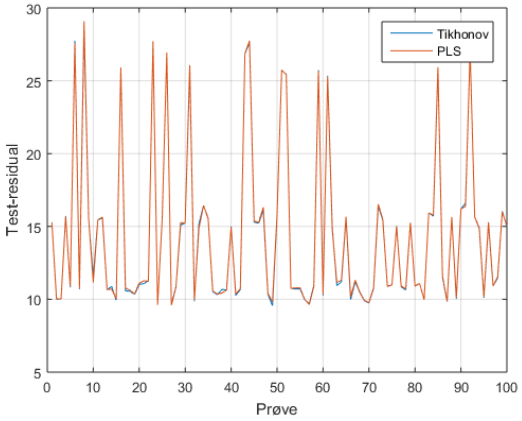
(g) Sugar1, T_2^{-1}-transformert



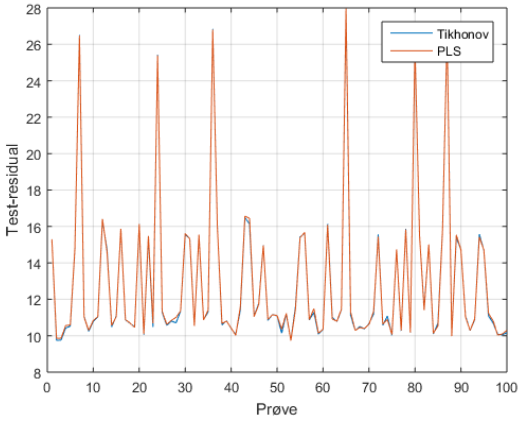
(h) Sugar2, T_3^{-1}-transformert



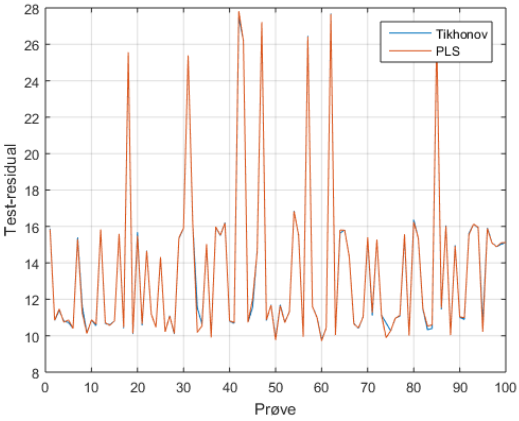
(i) Sugar3, ikke-transformert



(j) Sugar3, T_1^{-1}-transformert



(k) Sugar3, T_2^{-1}-transformert



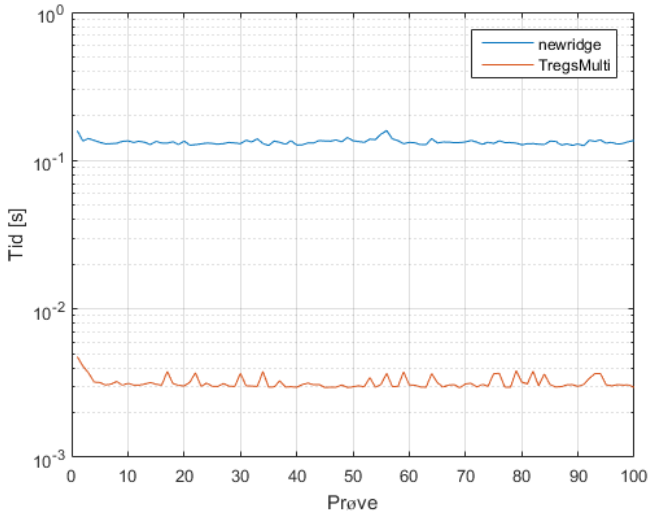
(l) Sugar3, T_3^{-1}-transformert

Figur 3.32: Plott av test-residualer for 100 tilfeldige fordelinger i test- og training-data for første responsvariabel av Sugar-datasettet (Brown, 1992 [5]). Figur (a) - (d) har resultater for den første responsvariabelen, (e) - (h) for den andre, og figur (i) - (l) har resultater for den tredje responsen.

3.4.8 MATLABs innebygde Ridge-funksjon.

Det fins en innebygd Ridge-funksjon i MATLAB (MathWorks [21]). Denne gjør Ridge-regresjon, men uten LOOCV. Jeg har derfor laget funksjonen `newridge` (ikke vedlagt på grunn av kopibeskyttelse) som er en utvidelse av MATLABs Ridge-funksjon. Denne gjør rask LOOCV og plukker ut modellen med lavest PRESS. Jeg har også endret på utregningen av $\hat{\boldsymbol{\beta}}$ slik at denne går mer effektivt.

Et problem med den innebygde Ridge-funksjonen er at den ikke støtter multivariate responser, til tross for at en slik utvidelse er triviell. Jeg har imidlertid latt være å gjøre en slik utvidelse, og vil derfor teste kun på Spectra-datasettet. Jeg antar at det jeg eventuelt finner i forskjell i tidsbruk for dette datasettet også vil gjelde de andre. Figur 3.33 viser tidsbruk for `newridge` og `TregsMulti` kjørt 100 ganger.



Figur 3.33: Sammenlikning av tidsbruk for `TregsMulti` og `newridge`. Begge er brukt for å analysere Spectra (Kalivas, 1997 [16]) med 30 λ-verdier mellom 10^{-4} og 10^{4}.

Kapittel 4

Klassifisering

I forarbeidet til denne oppgaven har jeg jobbet med Majones-datasettet (Indahl et al., 1999 [15], også forklart i del 4.1), som består av NIR-data fra 162 majonesprøver. Majonesen i prøvene var laget med 6 forskjellige typer oljer, og analysen har i stor grad gått ut på å plassere hvert datapunkt i riktig oljegruppe. Klassifiseringsproblemer er optimaliseringsproblemer der responsvariabelen er *kategorisk*, og ikke numerisk. Responsvariabelen er i tillegg definert kun for et endelig antall verdier, ofte kalt *nivåer* eller *klasser*. Sammenklynging, eller clustering, er problemer der vi ikke vet hvor mange klasser eller nivåer dataene egentlig har. Clustering er derfor et eksempel på *unsupervised learning*, mens klassifisering, som omhandler et kjent antall nivåer eller klasser for responsvariabelen er et eksempel på *supervised learning*. I dette kapittelet vil jeg ta for meg klassifiseringsproblemer. Analyse av klassifiseringsproblemer kalles *diskriminantanalyse*.

4.1 Data for klassifisering

Fishers iris (Fisher [9])

Klassisk datasett av metriske mål av kron- og blomsterblader på forskjellige iris-arter. Datasettet består av 150 målepunkter og fire variable, der responsen er artsnavn.

Majones (Indahl et al., 1999 [15])

162 målinger NIR-data av majones, målt i 2 nm intervaller fra 1100 til 2500 nm, totalt 351 forklaringsvariable. De seks forskjellige klassene representerer en oljetype brukt i majonesprøvene (soya, solsikke, raps, oliven, mais og druekjerne).

Ovarian Cancer (The FDA-NCI Clinical Proteomics Program Databank [6])

216 målinger med 4000 forklaringsvariable av proteomisk fingerprinting. Hver måling er gruppert som "cancer" eller "normal".

Zip (Hull, 1994 [13])

USPS håndskrevne postnumre, bilder med 256 piksler (16 ganger 16). Hvert bilde er lagret som en 256-dimensjonal radvektor, og klassen er sifferet bildet representerer.

MNIST (LeCun et al. [18])

Håndskrevne tall, bilder med 784 piksler (28 ganger 28). Hvert bilde er lagret som en 784-dimensjonal radvektor, og klassen er sifferet bildet representerer.

4.2 Klassifisering som multivariat OLS

Når man har analyserer datamengder som inneholder kategoriske data, er det ofte man koder disse numerisk. I Diabetes-datasettet (del 3.3.1) er en av forklaringsvariablene kjønn, men her er mann kodet som 0 og kvinne som 1.

For å gjøre diskriminantanalyse kan man gå frem på tilsvarende måte. Man representerer hver klasse med en responsvariabel, og koder denne binært: 1 hvis målingen tilhører den aktuelle klassen, -1 ellers. Disse verdiene samler man i responsmatrisa G, som tilsvarer Y i regresjonskapittelet. Ved å utføre en multivariat regresjonsanalyse får man de predikerte

verdiene $\hat{G}$, og man plukker ut det elementet i hver rad som har den høyeste verdien. Kolonnen til hver av disse maksimalverdiene vil være klasseprediksjonene.

Dersom antallet målinger er færre enn antall variable, slik at X ikke har full rang og $X^T X$ ikke er invertibel, kan man gjøre tilsvarende analyse med PCR eller PLS. Funksjonen `RDA` (vedlegg A.2.1) gjør dette, og man kan spesifisere metoden. Den gir $\hat{G}$, forvirringsmatrise og andel korrekte klassifiseringer som output-verdier. I forvirringsmatrisa (`confmat`) viser hver rad de faktiske klassifiseringene, mens hver kolonne viser de predikerte klassifiseringene. `pcc` er andelen korrekte klassifiseringer.

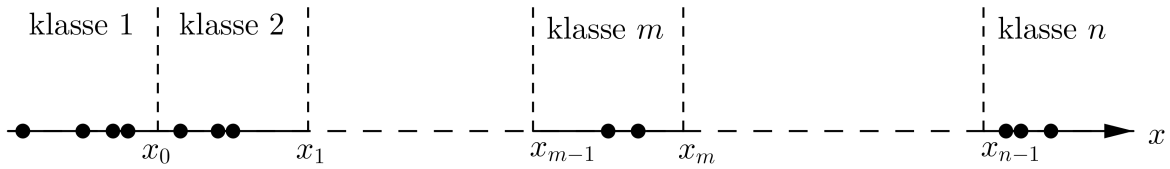
Ved å anvende `RDA` med OLS på Fishers (1936 [9]) iris-data, gir dette følgende resultat:

```
>> [~, confmat, pcc] = RDA(X, G)

confmat =

    50     0     0
     1    34    15
     0    11    39


pcc =

    0.8200
```

Det er altså 82 % korrekte klassifiseringer, og det er særlig klasse 2 og 3 som resiprokt feilklassifiseres.

4.3 Lineær diskriminantanalyse (LDA)

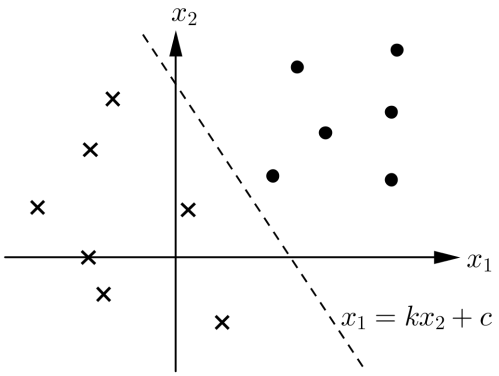
Det enkleste tilfellet av klassifisering har én variabel. Her avhenger klassifiseringen kun av størrelsen på denne ene variabelen. For et tre-klassesystem der $x \in \mathbb{R}$ er forklaringsvariabelen og $-\infty < a < b < \infty$, vil $x < a$ predikere klasse 1, $x \in (a, b)$ klasse 2 og $x > b$ klasse 3. Videre vil et n-klassesystem der $-\infty < x_0 < x_1 < \cdots < x_{n-1} < \infty$ gjøre at $x < x_0$ predikerer klasse 1, $x \in (x_{m-1}, x_m)$ predikerer klasse m og $x > x_{n-1}$ predikerer klasse n for $m < n$ (figur 4.1)



Figur 4.1: Klassifisiering av n klasser i én dimensjon. Forlaringsvariabelens verdi bestemmer klasseprediksjonen.

I det énvariable tilfellet er grensene mellom de forskjellige prediksjonsområdene for hver klasse punkter langs en akse. Som regel har man imidlertid mer enn én variabel når man gjør diskriminantanalyse. Med n variable befinner datapunktene seg i et n-dimensjonalt rom, og grensene mellom prediksjonsområdene vil være $n-1$-dimensjonale geometrier. Mens det kun fins én måte å representere noe 0-dimensjonalt (og ikke-tomt) på, nemlig punktet, vil det i høyere dimensjoner være mange forskjellige representasjoner. Både kuleflata og planet er for eksempel 2-dimensjonale geometrier.

I lineær diskriminantanalyse (LDA) antar man at grensene mellom prediksjonsområdene generelt er hyperplan i et hyperrom (Hastie et al., 2009, s 101 [12]). Er dataene 2-dimensjonale, vil dermed grensene være være linjer (figur 4.2).



Figur 4.2: Et toklassesystem i to dimensjoner. Her er grensen mellom de to klassene en linje.

For å predikere klassetilhørighet trenger man i utgangspunktet bare klassesenterne man får ved vektorsummen av hvert punkt dividert med antall punkter, for hver klasse. Man sjekker så avstanden en måling har til de forskjellige klassesenterne, og predikerer at den

tilhører klassen den ligger nærmest senteret til.

Den euklidske avstanden mellom et punkt $\boldsymbol{x}_i = [x_{i1} \quad x_{i2} \quad \cdots \quad x_{ip}]^T$ og senter av en klasse $\boldsymbol{\mu}_c = [\mu_{c1} \quad \mu_{c2} \quad \cdots \quad \mu_{cp}]$ er definert som

$$d_{\text{Eu}} = \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\| = \left[\sum_{j=1}^{p}(x_{ij} - \mu_{cj})^2\right]^{1/2}, \tag{4.1}$$

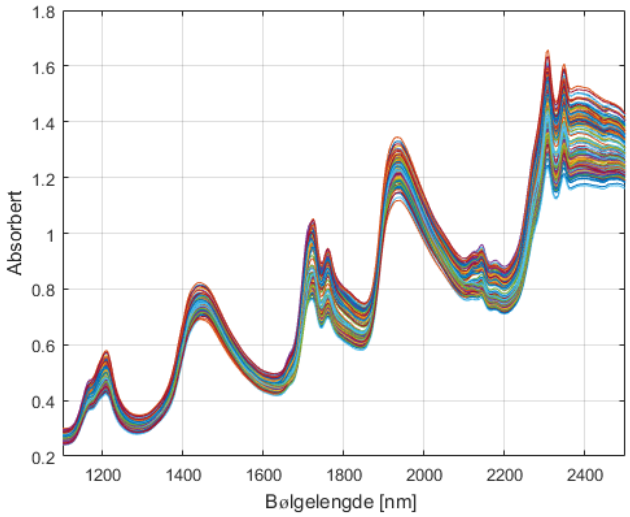
og funksjonen `LDA_Eu` (vedlegg A.2.2) klassifiserer utfra dette avstansmålet. Ved å kjøre denne funksjonen på Fishers iris-data, får jeg

```
[~, confmat, pcc] = LDA_Eu(X, G)

confmat =

    50     0     0
     0    46     4
     0     7    43


pcc =

    0.9267
```

Igjen er det klasse 2 og 3 som blir predikert feil, men her er det vesentlig færre enn i regresjonstilfellet, og jeg får over 92 % korrekte klassifiseringer.

Et problem med å bruke euklidsk avstandsmål er at man lar alle variablene telle like mye og man antar at de er uavhengige. Den første delen av dette problemet kan man unngå ved å standardisere dataene, slik at alle variablene får lik spredning. Det andre problemet er litt verre. Det er ikke ofte dataene er uavhengige. Fishers datasett består av målinger av lengde og bredde på kron- og blomsterblad for tre forskjellige iris-arter. Det synes rimelig å anta at dersom blomsterbladet er langt, vil også kronbladet være langt. Har man en unormalt stor blomst, er gjerne alle målene store. Disse variablene er altså overhodet ikke uavhengige. I majonesdataene er variablene intervaller av bølgelengder og verdien er mengde absorbert stråling. Slike data er så godt som alltid kontinuerlige, og det er derfor rimelig å anta at en høy verdi for én variabel vil gi høye verdier også for nabovariablene. Figur 4.3 viser et plott av majonesdataene for å illustrere denne avhengigheten.

Lineær diskriminantanalyse med euklidsk avstandsmål krever ikke at datamatrisa har full



Figur 4.3: Plott av hver måling i Majones-datasettet. Den tilsynelatende kontinuiteten og glattheten i kurvene støtter opp under antakelsen om at dataene ikke er uavhengige.

rang, noe majonesdataene ikke har. Jeg forsøker derfor å kjøre `LDA_Eu` på disse dataene:

```
>> [~, confmat, pcc] = LDA_Eu(X, G)

confmat =

    22     2     3     9     2     4
    12     0     0    12     0     0
    12     1     0    10     1     0
    11     0     1    12     0     0
    12     0     0    12     0     0
    12     0     0    12     0     0


pcc =

    0.2099
```

Datasettet har seks klasser, slik at ren gjetting ville gitt en forventet andel på $1/6 \approx 0{,}17$ riktige klassifiseringer. Dette er ikke langt unna, og `LDA_Eu` fungerer svært dårlig som klassifikator for dette datasettet.

4.3.1 Mahalanobis-avstand

I sin artikkel *The Use of Multiple Measurements in Taxonomic Problems* (1936 [9]) viste R. A. Fisher at det var mulig å artsbestemme iris-blomstene temmelig nøyaktig utfra lengde og bredde på kron- og blomsterblader. I sin analyse brukte ikke Fisher den euklidske avstanden, men en *standardisert* avstand. I motsetning til vanlig standardisering av variablene, tar dette standardiserte avstandsmålet hensyn til kovariansen mellom variablene. Avstanden defineres som

$$d_{\mathrm{M}} = \sqrt{(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T S^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)}, \tag{4.2}$$

der S er (en skalering av) kovariansmatrisa til datasettet. Dette generaliserte avstandsmålet ble utviklet av P. C. Mahalanobis tidlig på 1900-tallet, og ble publisert i 1936 [19]. Vi kjenner dette avstandsmålet i dat som *Mahalanobis-avstand*. Mens indreproduktet i euklidske vektorrom er

$$\langle \boldsymbol{u} \mid \boldsymbol{v} \rangle = \boldsymbol{u}^T \boldsymbol{v},$$

vil indreproduktet i et Mahalanobis-rom være

$$\langle \boldsymbol{u} \mid \boldsymbol{v} \rangle = \boldsymbol{u}^T S^{-1} \boldsymbol{v},$$

med Mahalanobis-normen

$$\|\boldsymbol{u}\|_{\mathrm{M}} = \sqrt{\langle \boldsymbol{u} \mid \boldsymbol{u} \rangle} = \sqrt{\boldsymbol{u}^T S^{-1} \boldsymbol{u}}.$$

Mahalanobis-avstanden mellom $\boldsymbol{x}_i$ og $\boldsymbol{\mu}_c$ er altså Mahalanobis-normen til differansen mellom disse vektorene.

Kovariansmatrisa inneholder alle kovariansene og variansene til variablene i datasettet, og er definert i (2.20) som $\frac{1}{n-1} X_s^T X_s$ der X_s er den sentrerte datamatrisa. For å finne den *interpolerte* datamatrisa som antas felles for alle klassene, må man ta hensyn til at klassene har forskjellig middel og det muligens ikke er like mange målinger i hver klasse. Man møter dette problemet ved å sentrere datamatrisa utfra klassetilhørighet, der man trekker fra middelet til hver variabel innenfor den klassen hver måling tilhører. Denne klassesentrerte matrisa angir jeg som X_c og den interpolerte kovariansmatrisa blir

$$\hat{\Sigma}_{\mathrm{p}} = \frac{1}{n-1} X_c^T X_c.$$

Den kvadrerte Mahalanobis-avstanden blir her

$$d_{\mathrm{M}}^2 = \frac{1}{n-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T (X_c^T X_c)^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c).$$

Siden alle avstandene vil være skalert med $\frac{1}{n-1}$, kan man utelate denne i diskriminantanalysen og få identiske resultater. Man kan også bruke de kvadrerte avstandene direkte i analysen, siden kvadratrotfunksjonen er en monoton funksjon og derved bevarer ulikheter. Prediksjon av klasse på bakgrunn av Mahalanobis-avstand vil derfor skje ved å sammenlikne den kvadrerte Mahalanobis-avstanden

$$d_{\mathrm{M}}^2 = (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T (X_c^T X_c)^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c).$$

Probabilistisk LDA forutsetter at dataene er normalfordelte og tar utgangspunkt i den multivariate normalfordelingsfunksjonen (Bickel & Doksum, 1977 [3]):

$$f(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^p |\Sigma|}} e^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}{2}},$$

der sannsynligheten for at en måling i tilhører klasse c er blir

$$P(\boldsymbol{x} \mid \boldsymbol{\mu}_c, \Sigma) = \frac{1}{\sqrt{(2\pi)^p |\Sigma|}} e^{-\frac{(\boldsymbol{x}_i-\boldsymbol{\mu}_c)^T \Sigma^{-1} (\boldsymbol{x}_i-\boldsymbol{\mu}_c)}{2}}. \tag{4.3}$$

Siden $|\Sigma|$ er en konstant, vil $[(2\pi)^p |\Sigma|]^{-1/2}$ også være en konstant, så denne vil ikke bidra noe i diskriminantanalysen. Eksponenten er halvparten av den negative kvadrerte Mahalanobis-avstanden, og stor avstand gjør at verdien for (4.3) blir liten, mens liten avstand gir stor sannsynlighet. Det er derfor nok å regne ut Mahalanobis-avstanden for å gjøre diskriminantanalysen.

Det er verdt å legge merke til at kovariansmatrisa skal inverteres, noe som ikke er mulig dersom X ikke har full rang. Jeg har implementert diskriminantanalyse med Mahalanobis-avstand i funksjonen `LDA_M` (vedlegg A.2.3). Anvendelse på iris-dataene gir

```
>> [~, confmat, pcc] = LDA_M(X, G)

confmat =

    50     0     0
     0    48     2
     0     1    49

```

```
 9
10  pcc =
11
12      0.9800
```

Dette er den beste klassifiseringen av dette datasettet så langt.

Det er mulig å regne ut Mahalanobis-avstanden ved å gå veien om SVD. Ved å la $X_c = U\Sigma V^T$, vil man kunne regne ut den kvadrerte avstanden mellom måling i og sentrum av klasse c ved

$$\begin{aligned} d_{\mathrm{M}}^2 &= (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T (X_c^T X_c)^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T [(U\Sigma V^T)^T U\Sigma V]^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T (V\Sigma^2 V^T)^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T V\Sigma^{-2} V^T (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T V\Sigma^{-1} (V\Sigma^{-1})^T (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T WW^T (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= \|W^T (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T\|^2, \end{aligned} \tag{4.4}$$

der $W = V\Sigma^{-1}$.

4.3.2 Regularisert LDA

Selv om det er mulig å bruke euklidsk avstandsmål for datasett uten full rang, har jeg vist at dette ikke nødvendigvis gir gode resultater, i alle fall ikke for majones-datasettet. Det skyldes i all hovedsak at variablene ikke er uavhengige.

For Tikhonov-regularisering av regresjonsproblemer ble det foretatt en utvidelse av data:

$$X\hat{B} = Y \leftarrow \begin{bmatrix} X \\ \sqrt{\lambda}T \end{bmatrix} \hat{B} = \begin{bmatrix} Y \\ 0 \end{bmatrix},$$

der 0 er nullmatrisa i $\mathbb{R}^{p\times m}$. I klassifikasjon kan man gjøre noe tilsvarende:

$$X\hat{B} = G \leftarrow \begin{bmatrix} X \\ \sqrt{\lambda}T \end{bmatrix} \hat{B} = \begin{bmatrix} G \\ 0 \end{bmatrix}.$$

I lineær diskriminantanalyse skal man ikke gjøre regresjon, og man er ikke interessert i å finne noen $\hat{B}$-matrise. Man kan imidlertid utvide X-matrisa med flere målinger, men det gir ikke mening å utvide G-matrisa med nullverdier da det ikke fins noen klasse 0. Ved SVD av den utvidede Z-matrisa har jeg vist at man vil få de samme U og V-matrisene, men at Σ-matrisa blir annerledes (likning (3.11)). Siden Z har full rang, vil alle diagonalelementene i Σ_Z ha full rang.

Uttrykket for Mahalanobis-avstanden i (4.4) er ikke intuitivt når X ikke har full rang, fordi her er $V^{-1} \neq V^T$ ettersom V ikke er kvadratisk når man bruker den økonomiske SVD-varianten. Når man gjør SVD av Z-matrisa får man flere søyler i V-matrisa, som hver svarer til egenverdien $\sqrt{\lambda}$. Her er V en kvadratisk matrise, og (4.4) holder. Hastie et al. (2009, s 113) foreslår at man kan gjøre tilsvarende for V og Σ man får fra SVD av X_c, så lenge man oppdaterer Σ med λ som i (3.11). En kjapp sjekk i MATLAB viser at dette stemmer:

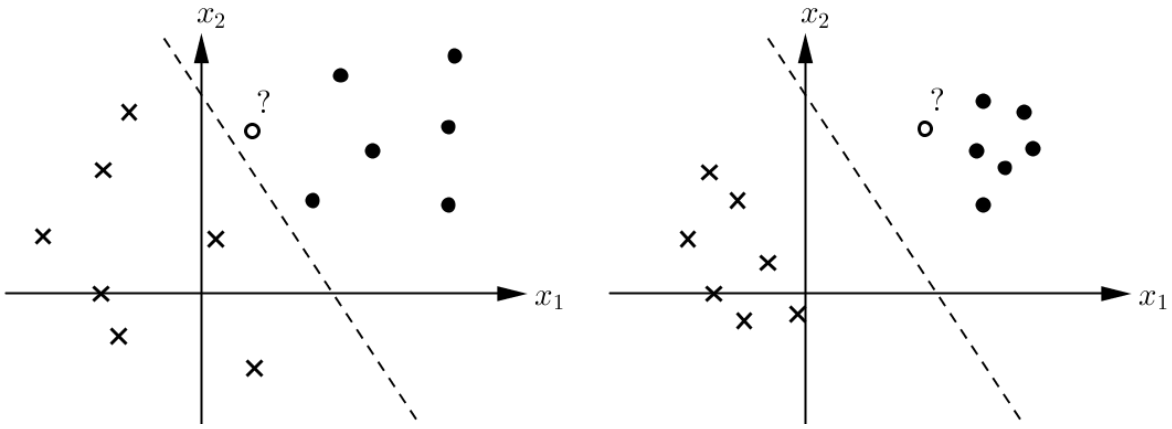
```
>> clc, clear, close all
load Mayo
[n, p] = size(X);                                % antall rader og kolonner
mX = mean(X);                                    % Middelet av X
Xc = X - ones(n, 1)*mX;                          % Xc (globalt sentrert
Z = [Xc; sqrt(0.0022)*eye(p)];                   % Z
[U, S, V] = svd(Xc, 'econ');                     % SVD av Xc
[Uz, Sz, Vz] = svd(Z, 'econ');                   % SVD av Z
S = sqrt(S.^2 + 0.0022*speye(n));                % lambda-oppdatert Sigma fra X
d = (X(1, :) - mX)*V*S^-2*V'*(X(1, :) - mX)';    % kvadrert Mahalanobis-avst fra X
dz = (X(1, :) - mX)*(Z'*Z)^(-1)*(X(1, :) - mX)'; % kvadrert Mahalanobis-avst fra Z
rank(d - dz, 1e-10)

ans =

     0
```

Man får altså den samme avstanden ved å bruke SVD av X med oppdatert Σ.

4.3.3 Modellvalidering

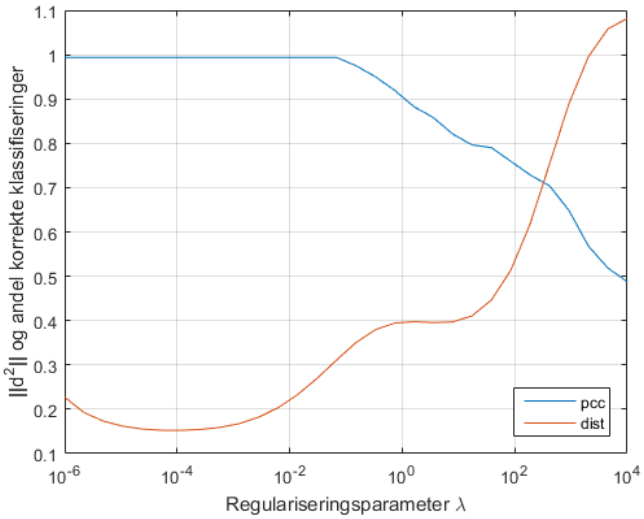
LOOCV i diskriminantanalyse foregår på samme måte som i regresjon. Siden man ikke har noen PRESS-verdi i diskriminantanalyse, velger man den modellen som har flest kor-

rekte klassifiseringer. Det er imidlertid ofte flere λ-verdier som gir det maksimale antallet korrekte kryssvaliderte klassifiseringer, så hvordan velge bare én? En løsning på dette er å bruke et sekundært kriterium der Mahalanobis-avstandene fra klassesenterne også skal minimeres. Filosofien her er at en modell der punktene klynges tettest mulig sammen rundt et klassesenter (som forøvrig er uavhengig av λ) vil være en modell der det er lettere å predikere riktig klasse (figur 4.4). Noen klasser kan helt naturlig være spredt mer utover i rommet enn andre, og det er derfor lurt å normere avstandene til hvert av klassesenterne for hvert målepunkt. Man vil så velge den modellen som har den laveste sum av slike normerte avstander. Figur 4.5 et plot av disse normerte avstandene og andel korrekte klassifiseringer.



(a) Store avstander innen klassen (b) Små avstander innen klassen

Figur 4.4: Fordi man jobber med estimater, og grenselinja har en viss usikkerhet, er man på mye tryggere grunn når man skal klassifisere det ukjente punktet når avstandene innen gruppene er små.



Figur 4.5: Plott av andel korrekte klassifiseringer og normerte avstander til CV-predikert klassesentrum mot regulariseringsparameteren λ for majones-datasettet (Indahl et al., 1999 [15])

Ved å velge den modellen er differansen mellom `pcc` og `dist` er størst, vil man altså få den modellen man venter vil predikere best. Funksjonen `TLDCA_LOOCV` (vedlegg A.2.4) har dette implementert, og gir følgende resultater for majones-dataene:

```
>> [~, ~, ~, ~, ~, ~, pcc, confmat] = TLDA_LOOCV(X, G, lambdas, d)

pcc =

    0.9938


confmat =

    42     0     0     0     0     0
     1    23     0     0     0     0
     0     0    24     0     0     0
     0     0     0    24     0     0
     0     0     0     0    24     0
     0     0     0     0     0    24
```

Her er derivasjonsordenen `d` satt lik 0, og `lambdas` består av 30 $\log_{10}$-distribuerte verdier mellom 10^{-4} og 10^{4}.

Rask LOOCV for LDA

Et problem med `TLDA_LOOCV` er at den tar ganske så lang tid:

```
1 >> tic, TLDA_LOOCV(X, G, lambdas, d); toc
2 Elapsed time is 69.111469 seconds.
```

Det er heldigvis også mulig å gjøre en rask LOOCV for klassifikasjonsproblemer, der man unngår å bygge modellen på nytt for hver utelatte måling. I tilfellet over ble det gjort en ny SVD for hver λ-verdi, noe som betyr svært mange regneoperasjoner.

Fordi man utelukkende bruker avstander til gruppesentra til å predikere klassetilhørighet, vil det å finne disse avstandene være nok. LOOCV Mahalanobis-avstand fra målepunkt i i gruppe c til sentrum av gruppe c vil være

$$d^2_{\mathrm{M},ic} \leftarrow d^2_{\mathrm{M},ic} \left(\frac{n_c}{n_c - 1}\right)^2 \Bigg/ \left(1 - \frac{n_c}{(n_c-1)} d^2_{\mathrm{M},ic}\right), \tag{4.5}$$

mens tilsvarende avstand til sentrum av gruppe $k \neq c$ vil være

$$d^2_{\mathrm{M},ik} \leftarrow d^2_{\mathrm{M},ik} \left(1 + \frac{\left[(\boldsymbol{x}_i - \boldsymbol{\mu}_i)(X^T X)^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^T\right]^2}{\left[(n_c-1)/n_c - d^2_{\mathrm{M},ic}\right] d^2_{\mathrm{M},ik}}\right) \text{ (Ripley, 1996, s. 100 [24]).} \tag{4.6}$$

Her er n_c antall målinger i gruppe c og $d^2_{\mathrm{M},ik}$ er Mahalanobis-avstand fra måling i til klassesenter k. Som vist i (4.4) kan Mahalanobis-avstanden også uttrykkes som en tranformasjon av den euklidske avstanden med $W = V\Sigma^{-1}$, der V og Σ er SVD-matriser til X_s, og dette gjør at (4.6) kan skirves som

$$d^2_{\mathrm{M},ik} \leftarrow d^2_{\mathrm{M},ik} \left(1 + \frac{\left[(\boldsymbol{x}_i - \boldsymbol{\mu}_i)WW^T(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^T\right]^2}{\left[(n_c-1)/n_c - d^2_{\mathrm{M},ic}\right] d^2_{\mathrm{M},ik}}\right).$$

Dersom man bruker euklidsk avstandsmål, settes $W = I_p$.

Denne metoden er implementert i `TLDAfastLOOCV` (vedlegg A.2.5).

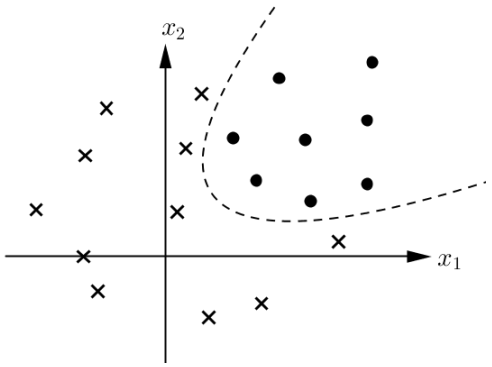
4.4 Kvadratisk diskriminantanalyse (QDA)

Mens man i lineær diskriminantanalyse antok at hele datasettet har en felles kovariansmatrise, vil man i kvadratisk diskriminantanalyse (eng: *Quadratic Discriminant Analysis*,

QDA) anta at hver klasse har sin egen korvariansmatrise. Dette gjør at man ikke lenger kan gå snarveien om kun å diskriminere med hensyn må Mahalanobis-avstander, men man er nødt til å regne probabilistisk. Dersom kovariansmatrisa til klasse c er angitt som Σ_c, blir sannsynligheten for at en måling $\boldsymbol{x}_i$ tilhører gruppe c med middel $\boldsymbol{\mu}_c$ en modifikasjon av (4.3):

$$P(c \mid \boldsymbol{x}_i) = \frac{1}{\sqrt{(2\pi)^p |\Sigma_c|}} e^{-\frac{(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T \Sigma_c^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)}{2}}.$$

Her er ikke koeffisienten foran eksponentialuttrykket en konstant, men forandrer seg for hver klasse. Det er derfor nødvendig å bruke hele denne utregningene i klassifiseringen, og ikke bare Mahalanobis-avstanden, slik vi kunne i LDA.



Figur 4.6: Et toklassesystem i to dimensjoner. Her er grensen mellom de to klassene en parabel.

En konsekvens av at man lar hver klasse ha sin egen kovariansmatrise, og dermed egen fordeling, er at grensene mellom prediksjonsområdene er kvadratiske og ikke lineære, og det er derfor denne typen diskriminantanalyse omtales som kvadratisk. Et eksempel på avgrensing av av prediksjonsområdene for QDA er gitt i figur 4.6.

Jeg har implementert denne fremgangsmåten i funksjonen `QDA` (vedlegg A.2.6). Den gir det samme gode resultatet som LDA for Fishers iris-data:

```
>> [~, ~, ~, ~, ~, confmat, pcc] = QDA(X, G)

confmat =

    50     0     0
     0    48     2
     0     1    49
```

```
8
9
10  pcc =
11
12      0.9800
```

Det er også mulig å gå veien om SVD i QDA. Fordi kovariansmatrisene er forskjellige for hver klasse, og ikke alle klasser har det samme antallet målinger, kan vi her ikke sløyfe konstantleddet, og får at kovariansmatrisa for klasse c blir

$$\text{Cov}(X_c) = \frac{1}{n_c - 1} V \Sigma^2 V^T,$$

der V og Σ kommer fra SVD av den delen av X-matrisa som tilhører klasse c og n_c er antallet målinger i denne klassen. Den inverse kovariansmatrisa blir dermed

$$\text{Cov}(X_c)^{-1} = (n_c - 1) V \Sigma^{-2} V^T$$

hvis X har full rang. Siden Σ er en diagonal matrise, blir dette veldig greit å regne ut.

Determinanten til kovariansmatrisa trengs også for å regne ut QDA-sannsynlighetene. Denne kan man regne ut ved

$$\begin{aligned}
|\text{Cov}(X_c)| &= |(n_c - 1)^{-1} V \Sigma^{-2} V^T| \\
&= |V (n_c - 1)^{-1} \Sigma^2 V^T| \\
&= |V||(n_c - 1)^{-1} \Sigma^2||V^T| \\
&= |(n_c - 1)^{-1} \Sigma^2||V^T V| \\
&= |(n_c - 1)^{-1} \Sigma^2|
\end{aligned}$$

fordi Σ er en diagnoal matrise og $(n_c - 1)$ er en konstant, blir determinanten derfor

$$|\text{Cov}(X_c)| = (n_c - 1)^{-p} \prod_{j=1}^{p} \sigma_j^2.$$

Sannsynlighetsfunksjonen kan nå skrives som

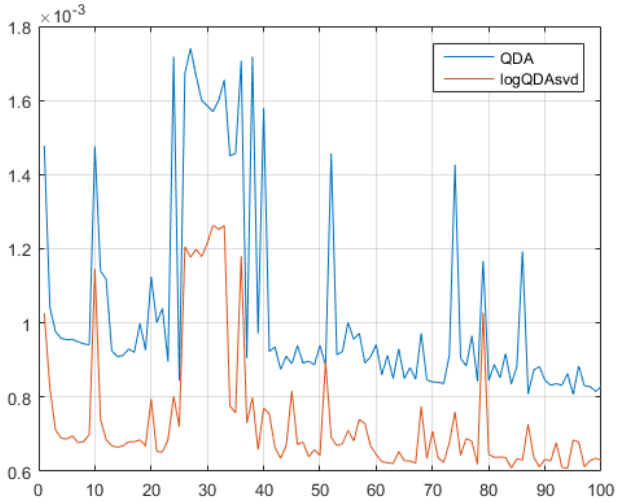
$$P(\boldsymbol{x} \mid c) = \left(\frac{n_c - 1}{2\pi} \right)^{p/2} \prod_{j=1}^{p} \sigma_j^{-1} e^{-\frac{\left\| W^T (\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T \right\|^2}{2}}$$

Denne funksjonen kan fort inneholde svært store og svært små tall dersom p blir stor nok. Koeffisienten før eksponentialuttrykket blir større og større for store n_c og store p,

mens eksponentialuttrykket kan bli forsvinnende lite for store Mahalanobis-avstander. Man trenger faktisk ikke ha så veldig store datasett før tallene i uttrykket blir for store for Matlab. Dette kan man løse ved å gå logaritmisk til verks. Den naturlige logaritmen til sannsynligehtsfunksjonen over kan uttrykkes som

$$\ln P(\boldsymbol{x} \mid c) = \frac{p}{2} \ln\left(\frac{n_c - 1}{2\pi}\right) - \sum_{j=1}^{p} \ln \sigma_j - \frac{1}{2} \left\| W^T(\boldsymbol{x}_i - \boldsymbol{\mu}_c) \right\|^2 \tag{4.7}$$

Siden logaritmefunksjoner er monotone kan verdiene fra denne funksjonen sammenliknes direkte. Ønsker man pene, pyntelige og positive sannsynlighetsverdier som summeres til 1, er det en smal sak å anvende eksponentialfunksjonen på verdiene for så å dele på summen av dem. Dette er implementert i funksjonen `logQDAsvd` (vedlegg A.2.7), som viser seg å bruke litt kortere tid på analysen enn `QDA`-funksjonen (figur 4.7).



Figur 4.7: Tidsbruk for `QDA`-funksjonen og `logQDAsvd` anvent 100 ganger på iris-datasettet (Fisher, 1936 [9]).

4.4.1 Regularisert QDA

Det er mulig å regularisere QDA på samme måte som LDA. Her blir uttrykket i (4.7) modifisert til

$$\ln P(\boldsymbol{x} \mid c) = \frac{p}{2} \ln\left(\frac{n_c - 1}{2\pi}\right) - \frac{1}{2} \sum_{j=1}^{p} \ln(\sigma_j^2 + \lambda) - \frac{1}{2} \left\| W(\lambda)^T(\boldsymbol{x}_i - \boldsymbol{\mu}_c) \right\|^2,$$

der $W(\lambda)$ er den λ-regulariserte matrisa $V\tilde{D}^{-1}(\lambda)$. $\tilde{D}$ er en regularisert utgave av singulærverdimatrisa Σ og mens man i LDA kan bruke $D^2(\lambda) = \Sigma^2 + \lambda I$, må man i QDA passe på at ikke total varians forandres i kovariansmatrisa. Dette fordi de forskjellige determinantene til kovariansmatrisene også er med på å bestemme sannsynligheten for klassifiseringen. I LDA er determinanten til kovariansmatrisa konstant, og vi kan derfor se bort fra dette.

Siden man kun forandrer på diagonalen, må man passe på at tr $\tilde{D}^2$ blir den samme som tr Σ^2. Dette oppnår man ved å multiplisere hvert av diagonalelementene i $D^2 = \Sigma^2 + \lambda I$ med konstanten

$$k = \frac{\text{tr } \Sigma^2}{\text{tr } D^2}.$$

4.4.2 Rask LOOCV for QDA

Også for QDA fins det en LOOCV-snarvei. I LDA er det to forskjellige uttrykk for den oppdaterte Mahalanobis-avstanden til for et utelatt punkt, ett for avstanden til klassesenteret der målingen hører hjemme (4.6) og et annet for de andre klassesenterne (4.5). Det er to forskjellige uttrykk fordi hele den felles kovariansmatrisa og det esitmerte klassesenteret må oppdateres i det første tilfellet, mens kun kovariansmatrisa oppdateres for avstandene i det andre.

I QDA har ikke klassene noen felles kovariansmatrise, det holder å kun oppdatere kovariansmatrisa og Mahalanobis-avstanden for den klassen den utelatte målingen tilhører. Ifølge Ripley (1996, s 100 [24]) blir determinanten til den oppdaterte kovariansmatrisa

$$|\text{Cov}(X_c)| \leftarrow |\text{Cov}(X_c)| \left(\frac{n_c - 1}{n_c - 2}\right)^p \left(1 - \frac{n_c}{(n_c - 1)^2} d^2_{\text{M},ic}\right),$$

mens den oppdaterte Mahalanobis-avstanden blir

$$d^2_{\text{M},ic} \leftarrow d^2_{\text{M},ic} \frac{n_c^2(n_c - 2)}{(n_c - 1)^3} \Bigg/ \left(1 - \frac{n_c}{(n_c - 1)^2} d_{\text{M},ic}\right)$$

Fordi man ikke kun ser på Mahalanobis-avstand når man gjør klassifiseringen, vil tilleggskriteriet for modellvalg være sannsynlighet, og man velger den modellen som har høyest norm av sannsynligheter for tildelte klassene.

De logaritmiske sannsynlighetene kan nå skrives som for de utelatte punktene blir nå

$$\ln P(\boldsymbol{x}_i \mid c) = a_c(\lambda) - \frac{p}{2}\ln\frac{n_c-1}{n_c-2} - \frac{1}{2}\ln\left(1 - \frac{n_c}{(n_c-1)^2}d^2_{\mathrm{M},ic}\right) - \frac{1}{2}d^{2*}_{\mathrm{M},ic},$$
$$a_c(\lambda) = \frac{p}{2}\ln\left(\frac{n_c-1}{2\pi}\right) - \frac{1}{2}\sum_{j=1}^{p}\ln(\sigma_j^2+\lambda),$$

der $d^{2*}_{\mathrm{M},ic}$ er den CV-oppdaterte kvadrerte Mahalanobis-avstanden. Dette er implementert i funksjonen `TQDAfastLOOCV` (vedlegg A.2.8).

Et problem jeg har støtt på her, er at faktoren

$$1 - \frac{n_c}{(n_c-1)^2}d^2_{\mathrm{M},ic} \tag{4.8}$$

vil gå som $1 - n_c^{-1}d^2_{\mathrm{M},ic}$ for store n_c, men dersom $d^2_{\mathrm{M},ic} > n_c$ kan man risikere at (4.8) blir negativ. Dette gjør at den logaritmiske sannsynlighetsverdien blir kompleks, og den ikke-logaritmiske blir negativ, noe som ikke gir mening. Den oppdaterte Mahalanobis-avstanden blir også negativ, så en stor oppdatert avstand vil kunne øke sannsynligheten i stedet for å minske den. I `TQDAfastLOOCV` har jeg derfor valgt å burke absoluttverdien av (4.8), men jeg har ikke fått denne til å gi samme resultater som den manuell LOOCV.

4.5 Bildeanalyse

Jeg skal ta for meg noe bildeanalyse i denne delen. Klassifisering av bilder er ofte noe som krever svært mange utregninger og tar lang tid. Jeg vil derfor konsentrere meg om enkle bilder av håndskrevne tall.

Bilder består av piksler, og jeg ser på hver piksel som en variabel. Hver gruppe er et siffer, og hvert bilde er en måling i datasettet. Jeg tar kun for meg bilder i sorthvitt, og verdien på en variabel vil være gråhetsgraden. Bilder er ikke én-dimensjonale, slik at hvert bilde blir en egen datamatrise. Til nå i klassifiseringskapittelet og i hele regresjonskapittelet har en måling vært en vektor, slik at denne kan representeres som et punkt i et multidimensjonalt euklidsk eller Mahalanobis-rom. I denne delen gjør jeg det samme, men for å representere bildene vektorielt, legger jeg hver pikselkolonne etter hverandre i en vektor. Et 10×10 pikslet bilde, vil dermed representeres som en vektor i $\mathbb{R}^{100}$.

Når man gjør en slik vektorisering, er det ingenting i veien for å benytte de tidligere presenterte metodene for klassifisering. Både multivariat regresjon, LDA og QDA vil fungere, men det er også metoder tilgjengelige som tar opp i seg bilders egenart. Et bilde er har ofte stor grad av kontinuitet, og når jeg tar for meg håndskrevne tall gir det også mening å anta en viss kontinuitet mellom den nederste delen av en pikselsøyle og den øverste delen av neste pikselsøyle. Dette fordi tallet antas å befinne seg midt i bildet, og randen som regel vil være ensfarget. Ved derivasjons-regularisering vil man se på forskjell mellom to etterfølgende punkter, eller en serie av etterfølgende punkter for høyere ordens deriverte. Man tar altså ikke hensyn til kontinuitet mellom to nabopunkter på samme rad, ei heller nabopunkter på diagonalen.

4.5.1 SVD-basis-metoden

Eldén (2007, kapittel 10 [8]) presenterer en metode der man lager basisvektorer som representerer hvert siffer. I den enkleste utgaven vil hvert siffer ha én basisvektor, og denne vil være gjennomsittet av alle målingene for hver gruppe. Man klassifiserer så nye data ved til den klassen punktet ligger nærmest middelet til. Dette er det samme som lineær diskriminantanalyse.

Det er imidlertid mulig å utvide denne tankegangen, slik at man har mer enn én basis for hvert siffer. Man ordner bildevektorene i matriser, slik at matrise A_1 kun inneholder bildene av 1-ere og så videre. Ved å gjøre SVD av hver av disse matrisene, vil man få ortogonale basiser for hvert tall i V-matrisa, siden denne representerer radrommet til matrisa. Eldén refererer til disse singulærvektorene som *singulærbilder*. Man bruker et lite antall k av singulærbildene og ordner disse i matrisa $V_{(k)}$. Når man skal klassifisere en bildevektor $\boldsymbol{x}_i^T$, projiserer man denne ned i singulærbilderommet til hvert av sifrene

$$\boldsymbol{x}_{\text{proj}}^T = V_{(k)} V_{(k)}^T \boldsymbol{x}_i^T.$$

Deretter sjekker man hvilken av disse projeksjonene selve bildevektoren ligger lengst unna, og man har i prinsippet et minimeringsproblem

$$\min \|\boldsymbol{x} - \boldsymbol{x}_{\text{proj}}\| = \|\boldsymbol{x}_i^T - V_{(k)} V_{(k)}^T \boldsymbol{x}_i^T\| = \|(I - V_{(k)} V_{(k)}^T)\boldsymbol{x}_i^T\|$$

for hvert siffer. Dette er i prinsippet et minste kvadraters problem, hvor man skal gjøre projeksjonen av $\boldsymbol{x}_i^T$ ned på det ortogonale komplementet av basisbilderommet så liten som mulig. Siden antall klasser er et endelig og lite tall, finner man den beste klassifiseringen ved å lete etter den klassen der denne verdien er minst direkte.

I denne analysen bruker man ikke noen form for regularisering, men ved å variere antall basisvektorer i analysen, vil man kunne velge det antallet som gir høyest antall korrekte klassifiseringer, for eksempel ved å gjøre en LOOCV. En del av datamengdene er så store at det imidlertid kan være krevende å gjøre en full LOOCV. MNIST-datasettet (LeCun et al. [18]) har for eksempel 60 000 bilder i treningsdatasettet, og klassifisering med 30 basisvektorer tar 10,5 sekunder. Å gjøre full LOOCV med dette settet vill dermed tatt omlag 3,6 millioner sekunder eller nesten 41 dager. Jeg har derfor ikke gjort noen implementering av LOOCV for denne metoden, men velger antall basisbilder utfra hvor mange som ser ut til å predikere datene best. Jeg har implementert dette i funksjonen `class_numbers_svd` (vedlegg A.2.9).

Selv om det ikke er sikkert at det er det samme antallet basisvektorer som predikerer hvert siffer best, vil et stort sprik i antall basisvektorer for de forskjellige tallene gjøre at man får en svært ustabil modell. Tar man med svært mange basisvektorer for et siffer, vil man ta med svært mye støy, og det ser ut til at jo flere vektorer man tar med, dess større sjanse er det at et tall klassifiseres som dette sifferet.

Et annet problem er at treningsdataene ikke alltid er gode representasjoner for klassen. I konstruksjon av basisvektorene er gjerne også utydelige bilder med, bilder man ville hatt problemer med å tyde også ved visuell inspeksjon direkte. Disse kan bidra til at mer støy enn nødvendig blir med i modellen. En metode for å unngå dette, er å gjøre modellbyggingen i to omganger: først på vanlig vis, for så å manuelt gjennomgå de feilklassifiserte bildene og fjerne dem som ikke likner tallet de skulle representere. Deretter bygger man modellen igjen uten disse bildene, og bruker denne nye modellen til å predikere testdataene.

4.5.2 Tikhonov-regularisert regresjon med randomisert kontinuitetskriterium som klassifikator

Boyd & Vandenberghe (2015, delkapittel 13.4 [4]) beskriver en metode for bildeklassifisering, der man preprosesserer datamatrisa for å ta med kontinuiteten i bildet i analysen. Fordi en piksel ofte likner på sine nabopiksler, er det vanlig å ta dette med i analysen, men det som faktisk gir informasjon i et bilde, er diskontinuitetene. I tillegg vil et bilde av et håndskrevet tall for det meste bestå av bakgrunn, mens selve tallet er en tydelig kontrast til denne bakgrunnen. Boyd & Vandenberghe foreslår derfor å skjøte på randomiserte differanser mellom variablene for hvert datapunkt.

Man definerer matrisa $R \in \mathbb{R}^{p\times 1000}$ der p er antall piksler i bildet. Hvert element i R skal være enten -1 eller 1, tilfeldig valgt. For hver rad i datamatrisa X, skjøter man på $(R\boldsymbol{x}_i)^T$, men med alle negative elementer satt til 0. Man antar at dataene er kodet slik at hver piksel har en positiv verdi, og det vil derfor være unaturlig å legge til negative elementer. Man gjør så multivariat regresjon med den utvidede datamatrisa

$$X_{\text{utv}} = [X \quad XR^T].$$

Man kan se på Tikhonov-regresjon som vanlig regresjon der man skjøtet på ekstra målinger på datamatrisa. Her har man skjøtet på ekstra variable, og disse utgjør til sammen en slags randomisert derivasjonsoperator.

Man kan gjøre multivariat regresjon på dette utvidede datasettet direkte, og eventuelt kan man gjøre en Ridge-regularisering. Selv om man nå har 1000 ekstra variable å behandle, har både Zip-datasettet (Hull, 1994 [13]) og MNIST nok målinger til at man ikke behøver å regularisere. Det vil imidlertid kunne oppstå problemer med kollinearitet fordi egenverdiene i Σ-matrisa blir veldig små. Å gjøre en ytterligere derivasjonsregularisering vil imidlertid være unaturlig både fordi det nok er for mye forlangt at radene i X er kontinuerlige, men særlig fordi det vil være svært spesielt å kreve kontinuitet i XR^T.

Funksjonen `class_rand_cont_Ridge` (vedlegg A.2.10) har implementert denne metoden.

4.6 Sammenlikning med andre klassifikasjonsmetoder

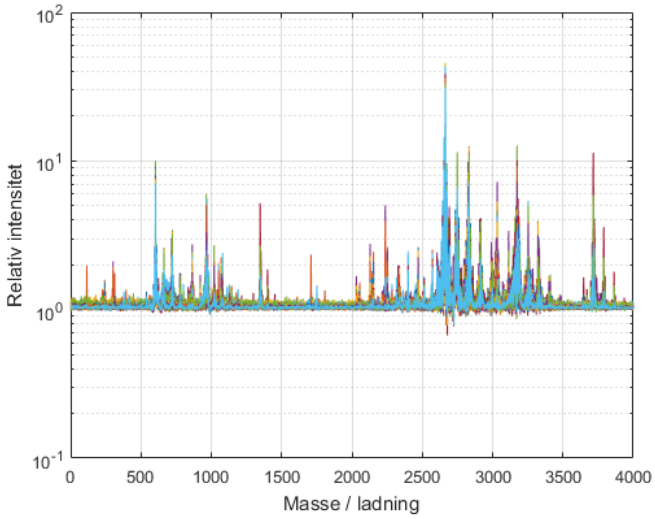
Jeg har vist i regresjonskapittelet at PCR er en metode som tar lang tid og ikke ser ut til å predikere noe særlig bedre enn PLS eller Tikhonov-regluarisert OLS. Jeg vil derfor avstå fra å bruke PCR i denne delen, da jeg ikke har noen indikasjoner på at PCR skulle oppføre seg annerledes her.

Jeg vil gjøre to forskjellige analyser, der den første består i sammenlikning av antall korrekte klassifiseringer og tid for majones- og Ovarian Cancer-datasettene, mens den andre tar for seg tilsvarende for bildeanalysendataene.

`RDA`-funksjonen har implementeringer for å gjøre diskriminantanalyse med PLS og PCR, og i disse tilfellene velges antall komponenter ut ved å gjøre full LOOCV. Fordi datasettene med håndskrevne tall er såpass store, vil jeg la være å gjøre LOOCV på disse. Dette fordi det å fjerne én måling blant flere tusen vil ha liten innvirkning på modellen, i tillegg til at LOOCV vil ta svært lang tid når man må bygge modellen på nytt for hver utelatte måling.

4.6.1 Tikhonov-regularisert LDA og RDA versus PLS-DA med LOOCV

I denne delen bruker jeg PLS, regularisert LDA og regularisert regresjon for å gjøre prediksjon av Majones- og Ovarian Cancer-datasettene. For alle tre funksjonene sammenlikner jeg LOOCV-prediksjoner. Jeg vil først se på de fulle datasettene og hvor godt disse ser ut til å predikeres, for deretter å se på tidsbruk og så til sist oppdeling i test- og treningssett for å se på eventuelle forskjeller i prediksjon her. Majones-dataene er kontinuerlige (figur 4.3), og det gir derfor mening å bruke derivasjonsregulariseringer. Figur 4.8 viser et plot av Ovarian Cancer-datasettet, og dette ser ikke spesielt kontinuerlig ut. Jeg vil derfor ha Ridge-regularisering som enenste regularisering ved analyse av dette datasettet.



Figur 4.8: Plot av de individuelle målingene i Ovarian Cancer-datasettet. Fordi noen av verdiene er negative, har jeg lagt til 1 på alle målinger for å kunne plotte logaritmisk.

Fullt datasett

Jeg analyserer her hele datasettet med det som skal være den beste modellen for hver metode. Disse er fremkommet ved å velge λ-verdi eller antall komponenter for hver gruppe ved LOOCV. Prediksjonene er ikke LOOCV-verdier, men prediksjoner der det er brukt hele datasettet. For majones-dataene gir dette følgende forvirringsmatrise:

```
>> confmat

confmat =

    42     0     0     0     0     0
     1    23     0     0     0     0
     0     0    24     0     0     0
     0     0     0    24     0     0
     0     0     0     0    24     0
     0     0     0     0     0    24
```

Denne er lik for alle metoder og regularisering med 0. - 3. deriverte, bortsett fra PLS med en andrederivertregularisering. Mens resten har andel korrekte klassifiseringer på 99,38 %, gir denne ene PLS-modellen 100 % korrekte klassifiseringer.

For Ovarian Cancer er det annerledes. Dette datasettet består av to klasser, som både PLS og TregsMulti predikerer 100 % riktig:

```
>> confmat_tregs

confmat_tregs =

   121     0
     0    95

>> confmat_pls

confmat_tregs =

   121     0
     0    95
```

I PLS-versjonen får jeg imidlertid problemer med kollinearitet.

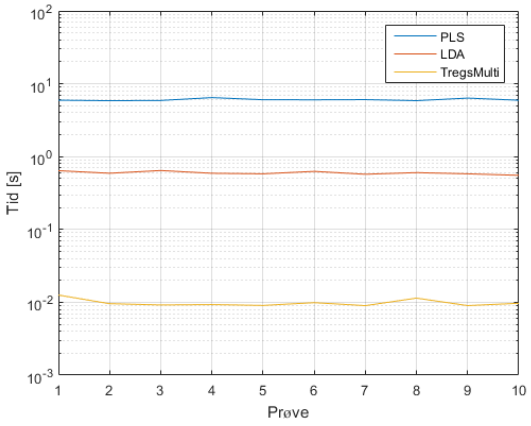
Den regulariserte LDA-metoden gir et dårligere resultat:

```
>> confmat_lda

confmat_lda =

   117     4
     2    93
```

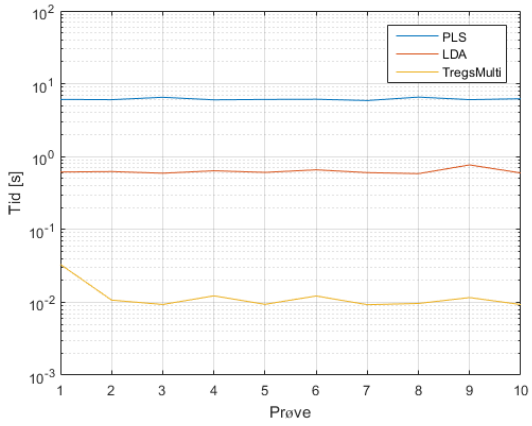
med 97,22 % korrekte klassifiseringer.

Tidsbruk

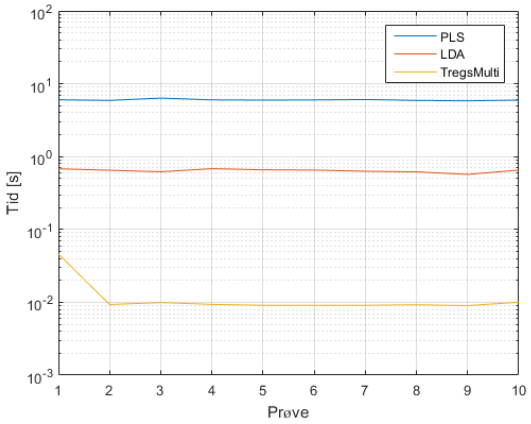
Jeg sjekker så tidsbruken. Fordi PLS-funksjonen ser ut til å ta temmelig lang tid, nøyer jeg meg med å gjøre analysen 10 ganger. Figur 4.9 viser tidsbruk for fire forskjellige regulariseringer av Majones samt Ridge-regularisering av Ovarian Cancer.



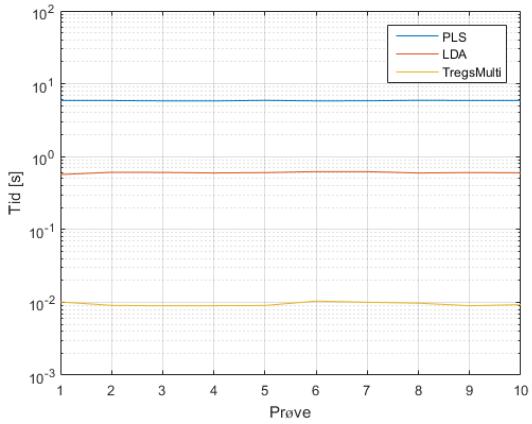
(a) Ridge-regularisering av Majones



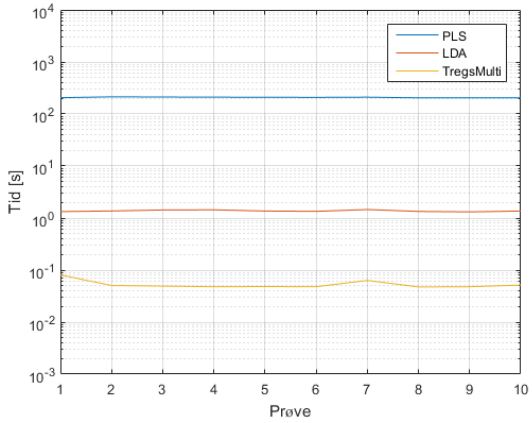
(b) T_1-regularisering av Majones



(c) T_1-regularisering av Majones



(d) T_1-regularisering av Majones

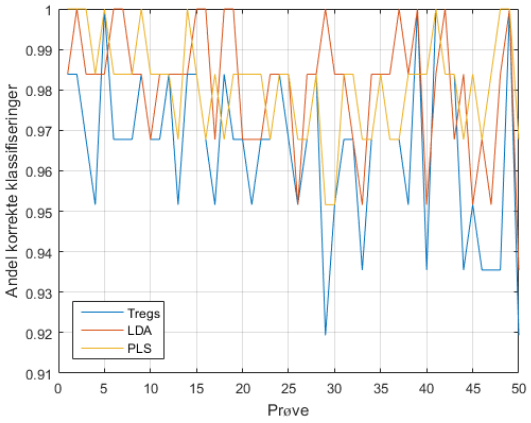


(e) Ridge-regularisering av Ovarian Cancer

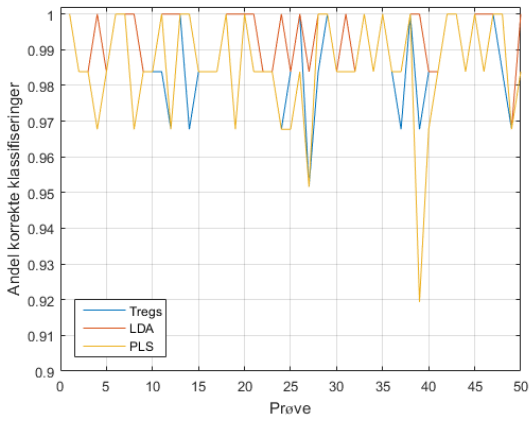
Figur 4.9: Tidsbruk av de tre klassifiseringsmetodene for Majones (Indahl et al., 1999 [15]) og Ovarian Cancer (The FDA NCI Clinical Proteomics Program [6]).

Test- og treningsdata

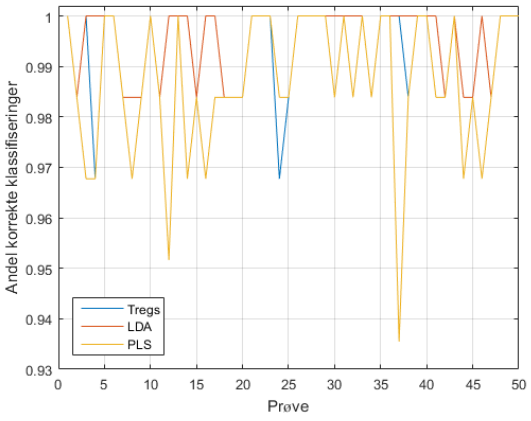
Jeg har delt inn Majones-dataene i test- og treningssett med henholdsvis 62 og 100 tilfeldige målinger i hver. Jeg har så brukt treningsdataene til å forsøke å predikere testsettet ved hjelp av PLS-regresjon, Tikhonov-regularisert LDA og Tikhonov-regresjon. Jeg har også gjort tilsvarende for Ovarian Cancer-datasettet, men her med 150 målinger i treningssettet og 66 i testsettet. Figur 4.10 viser andel korrekte klassifiseringer av test-settene for Majones, der jeg også har gjort analyser av T_1-, T_2- og T_3-transformerte data.



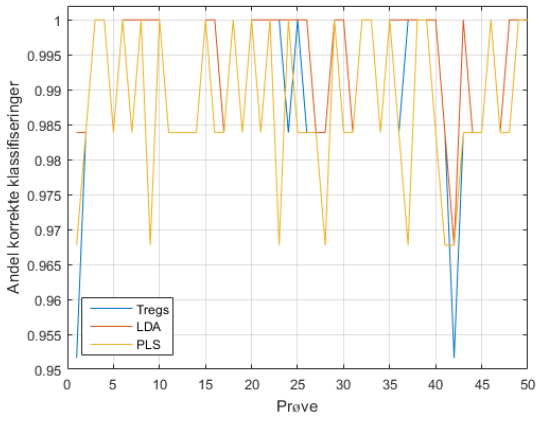
(a) Ridge-regularisering av Majones



(b) T_1-regularisering av Majones



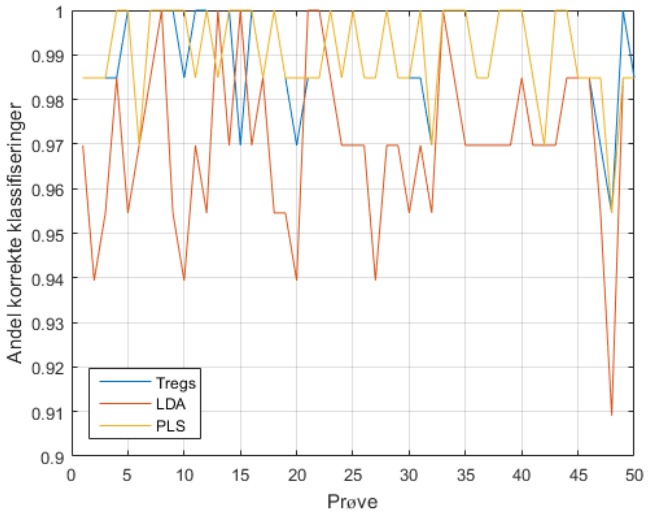
(c) T_2-regularisering av Majones



(d) T_3-regularisering av Majones

Figur 4.10: Prediksjon av tilfeldig utvalgte testsett med 62 målinger ved bruk av PLS-regresjon, Tikhonov-regresjon og Tikhonov-regularisert LDA på Majones-datasettet.

Figur 4.11 viser tilsvarende analyse av Ovarian Cancer-datasettet. Fordi dette ikke består av kontinuerlige målinger, er det ikke gjort noen analyser av transformerte data her.



Figur 4.11: Prediksjon av tilfeldig utvalgte testsett med 66 målinger ved bruk av PLS-regresjon, Ridge-regresjon og Ridge-regularisert LDA på Ovarian Cancer-datasettet

Det kan se ut som PLS er har en litt dårligere prediksjonsevne i for Majones i de transformerte tilfellene, mens det er Ridge-regresjon som predikerer dårligst i det ikke-transformerte tilfellet. For Ovarian Cancer er det derimot LDA som er minst stabil. Alle tre metodene predikerer uansett godt med over 90 % korrekte klassifiseringer.

4.6.2 Bildeanalyse

Til sist ser jeg på metoder for bildeanalyse. I tillegg til Ridge-regresjon, Ridge-regularisert LDA og PLS-regresjon vil jeg også analysere datasettene med SVD-basisvektor-metoden og Tikhonov-regularisert regresjon med randomisert kontinuitetskriterium.

Jeg vil først bruke de forskjellige metodene til å predikere treningsdatasettene. Siden datasettene er såpass store, vil jeg ikke bruke LOOCV her, men velge den λ-verdien eller det antallet PLS-komponenter som predikerer best direkte. Deretter vil jeg se på tidsbruk, og til sist hvor godt de forskjellige metodene predikerer test-datasettene.

I alle regresjonsanalysene er responsen kodet som binære matriser, med 1 som tilhørende gruppe og -1 ellers. Zip-dataene er transformert slik at verdiene er mellom 0 og 255 heller enn mellom -1 og 1 for `class_rand_cont_Ridge`.

Jeg får følgende kovariansmatriser ved direkte prediksjon av treningsdata:

Ridge-regresjon på Zip

```
>> confmat

confmat =

  1002     1     0     0     0     0     0     1     1     0
     4   649    24    13     2    12     8    13     2     4
     0     4   615     0    10     0     8    15     3     3
    21     7     0   592     3     5     2     4    17     1
     2     3    28     9   486     9     0     5     3    11
     7     8     1    14    10   613     0     7     0     4
     2     0     0     5     2     0   595     3    36     2
     5     3    18    14    12     4     4   463     7    12
     5     0     1    26     6     0    20     6   579     1
     0     1     5     3     3    10     0     3     0  1169
```

Dette 92,76 % korrekte klassifiseringer.

Ridge-regularisert LDA på Zip

```
>> confmat

confmat =

  1001     1     0     0     0     0     0     1     2     0
     1   657    21    14     3     7     4    18     3     3
     0     3   616     0    11     0     6    17     3     2
    16     9     0   591     1     4     0     7    23     1
     0     2    16     7   508     4     0     9     2     8
     6     4     0    10     6   625     0     9     0     4
     1     0     1    11     1     0   588     2    41     0
     2     2    22    14     9     4     2   481     3     3
     2     0     2    19     0     0    13     2   606     0
     0     1     3     4     2    11     0     7     0  1166
```

Dette gir 93,80 % korrekte klassifiseringer.

PLS-DA på Zip

```
>> confmat

confmat =

 1002     1     0     0     0     0     0     1     1     0
    5   650    18    15     2    12     6    12     2     9
    0     6   611     0     9     0     8    14     2     8
   25     7     0   590     3     5     1     3    16     2
    1     3    29    10   479     9     0     5     3    17
    7     9     1    12     9   610     0     7     0     9
    4     1     0     4     1     0   593     3    35     4
    6     4    16    14    12     5     3   453     6    23
    7     0     1    27     6     0    20     5   577     1
    0     1     5     3     1     7     0     1     0  1176
```

Dette gir 92,46 % korrekte klassifiseringer.

Klassifisering av Zip med SVD-basis

```
>> confmat

confmat =

 1005     0     0     0     0     0     0     0     0     0
    6   723     0     2     0     0     0     0     0     0
    4     0   649     0     2     0     0     3     0     0
    7     0     0   632     0     0     1     0    12     0
    0     0     0     1   551     1     0     2     0     1
   10     0     0     0     1   652     0     0     0     1
   11     0     0     2     0     0   623     1     8     0
   22     0     0     0     0     0     0   519     0     1
    2     0     0     0     0     0     3     0   639     0
    4     0     0     0     0     0     0     0     0  1190
```

Dette gir 98,52 % korrekte klassifiseringer

Ridge-regresjon-klassifisering av Zip med randomisert kontinuitetskriterium

```
>> confmat

confmat =

 1003     0     0     0     1     0     0     1     0     0
    1   722     1     4     0     0     2     1     0     0
    0     0   653     0     2     0     0     3     0     0
    3     0     0   640     0     3     0     0     6     0
    0     1     1     1   550     2     0     0     1     0
    1     0     0     2     1   658     0     1     0     1
    0     0     0     2     0     0   634     2     7     0
    1     0     1     3     3     1     2   531     0     0
    0     0     0     0     0     0     3     0   641     0
    0     0     1     0     0     0     0     1     0  1192
```

Dette gir 99,08 % korrekte klassifiseringer.

Ridge-regresjon på MNIST

```
>> confmat

confmat =

 6531    39    15    20    37    15    13    64     6     2
  249  4806   146   109    14   235    87   193    18   101
  154   181  5147    32   139    58   113   137   128    42
   92    40     4  5225    52    38    21    59   301    10
   87    26   396   104  4063   186    34   224   138   163
   66    62     0    72    94  5479     0    35     2   108
  179    37    50   175    10     2  5413    11   332    56
  461    64   222   106   235    55    19  4437   177    75
   57    20   114   373    15     4   476    41  4781    68
    7    18    11    24    45    64     2    61     7  5684
```

Dette gir 85,94 % korrekte klassifisieringer.

Ridge-regularisert LDA på MNIST

```
>> confmat

confmat =

  6454    38    18    11    44     6     8   152    11     0
   190  4868   180   118    25   189    42   257    31    58
    90   160  5179    25   241    21    88   174   139    14
    56    30     2  5247    45    29     2    54   372     5
    55    26   232    51  4503   112    28   225   132    57
    54    60     3    84   141  5434     0    79     4    59
   141    38    46   180    14     1  5222    28   560    35
   324    46   190    75   281    29    13  4699   162    32
    26    16    92   320    28     0   261    58  5116    32
     4    21    30    23   107    52     2    91     9  5584
```

Dette gir 87,18 % korrekte klassifiseringer.

PLS-DA på MNIST

```
>> confmat

confmat =

  6515    36    16    10    29    15    13   102     5     1
   258  4803   147   103    12   237    84   203    19    92
   141   183  5210    29    91    57   109   139   130    42
    97    55     8  5127    49    48    24    82   343     9
    70    30   521    82  3787   196    38   401   152   144
    65    64     1    60    81  5492     0    46     2   107
   186    43    56   153     9     3  5394    18   351    52
   529    58   223   123   235    51    20  4355   175    82
    60    25   113   356     8     4   489    53  4777    64
     8    20    18    26    43    72     3    62     6  5665
```

Dette gir 85,21 % korrekte klassifiseringer.

Klassifisering av MNIST med SVD-basis

```
>> confmat

confmat =

  6662    36     2    14     0     4    13    11     0     0
    57  5651    48    20     1    12    44    80    12    33
    18    76  5761     2    46     2    32   136    41    17
    28     8     0  5634     2    17    29    12   103     9
    14     7    97     1  5070    50     3   121    29    29
    19     2     2     5    84  5748     0    26     0    32
    23    31     6    23     5     2  5971    14   178    12
   113    44    73    14    46    29    12  5437    57    26
    18     6    37    54    10     2    84    54  5662    22
    16     3     0     2     8    11     1    17     1  5864
```

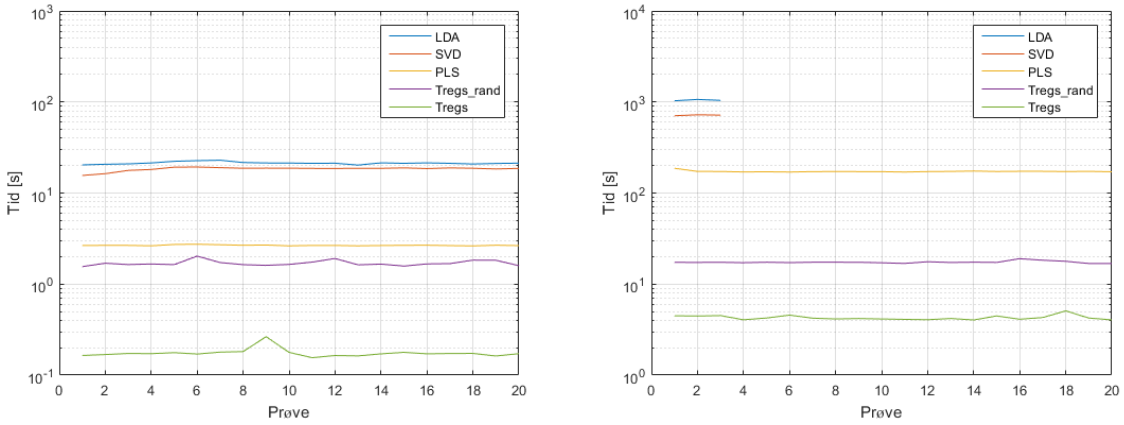
Dette gir 95,77 % korrekte klassifiseringer.

Ridge-regresjon-klassifisering av MNIST med randomisert kontinuitetskriterium

```
>> confmat

confmat =

  6637    39    10    14     8     4    10    18     2     0
    28  5644    40    35     6    21    57    81    11    35
    14    68  5750     6    96    15    49    80    45     8
    21    13     0  5598     6    36     9    22   130     7
    13    12    82    23  5139    63     7    30    22    30
    13     6     1    16    53  5786     0    17     1    25
    46    42    15    49     2     1  5972     9   115    14
    51    31    64    27    70    33    11  5480    66    18
    18    12    61   105    32     2    90    48  5562    19
     0     7     2     6    17    33     2    47     3  5806
```

Dette gir 95,62 % korrekte klassifiseringer.

Jeg ser så på tidsbruk. LDA og SVD-basis tar såpass lang tid at jeg kun har kjørt disse tre ganger for MNIST. De resterende har jeg kjørt 20 ganger. Resultatene er gitt i figur 4.12.



(a) Tidsbruk Zip.

(b) Tidsbruk MNIST.

Figur 4.12: Tidsbruk for bildeanalyse.

Jeg undersøker til slutt de forskjellige metodenes evne til å predikere testsettene.

Klassifisering av testsett til Zip med Ridge-regresjon

```
>> confmat

confmat =

   253     0     2     4     0     3     0     1     1     0
     2   157     9     9     2     1     1    11     0     6
     0     4   139     2     9     0     2     3     4     3
     4     4     0   172     1     3     2     1    11     2
     0     0    20     2   124     0     1     2     4     7
     0     4     0     6     4   153     0     1     0     2
     1     1     1     7     0     0   131     0     4     2
     1     2    13     4    10     1     1   123     4     7
     1     0     0     7     0     0     7     2   160     0
     1     0     2     3     1     6     0     1     2   343
```

Dette gir 87,44 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med Ridge-regularisert LDA

```
>> confmat

confmat =

   251     0     2     5     0     3     0     1     2     0
     2   155     4    12     2     1     1    14     0     7
     0     3   142     3     9     0     1     4     1     3
     4     6     0   174     0     2     2     1    10     1
     0     0    17     3   125     0     0     6     5     4
     0     3     0     3     3   158     0     3     0     0
     1     0     2     7     0     0   129     1     7     0
     0     2    11     7     4     0     0   135     2     5
     0     0     0     4     0     0     5     3   165     0
     0     0     4     3     1     4     0     4     1   342
```

Dette gir 88,49 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med PLS-DA

```
>> confmat

confmat =

   254     0     2     3     0     3     0     1     1     0
     3   158     7    10     1     1     1     9     0     8
     0     4   138     2     8     0     2     3     4     5
     7     4     0   169     1     3     2     1    10     3
     1     0    20     2   122     0     1     1     4     9
     2     4     0     5     4   151     0     1     0     3
     1     1     1     6     0     0   131     0     4     3
     3     3    15     5     8     1     2   117     4     8
     3     0     0     7     0     0     7     2   158     0
     2     0     1     3     1     4     0     0     1   347
```

Dette gir 86,95 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med SVD-basis

```
>> confmat

confmat =

   257     0     0     3     1     3     0     0     0     0
     0   177     2     5     1     0     1     7     0     5
     0     2   147     0    10     0     0     4     0     3
     6     4     0   179     0     0     2     0     9     0
     1     0     3     1   144     1     0     4     2     4
     1     1     0     2     5   159     0     2     0     0
     1     1     0     3     0     0   136     2     4     0
     6     4     2     0     3     2     0   143     2     4
     5     0     0     0     1     0     3     1   167     0
     3     3     0     1     0     0     0     1     0   351
```

Dette gir 92,68 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med Ridge-regresjon med randomisert kontinuitetskrieterium

```
>> confmat

confmat =

   254     1     1     4     0     3     0     0     1     0
     0   179     2     4     2     2     2     7     0     0
     0     3   145     1    10     0     1     4     1     1
     2     3     0   183     2     1     1     1     7     0
     0     0     7     1   146     1     0     2     2     1
     0     3     0     2     1   163     0     0     0     1
     0     2     0     4     1     0   136     1     3     0
     0     3     2     1     3     0     0   149     6     2
     0     1     0     4     0     0     1     1   170     0
     0     4     0     2     0     1     0     0     1   351
```

Dette gir 93,47 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med Ridge-regresjon

```
>> confmat

confmat =

  1102     2     2     3     2     5     1    18     0     0
    50   816    25    15     1    42    21    39     5    18
    16    23   880     5    17     9    21    23    12     4
    19     6     1   883     5    10     2    11    45     0
    16     3    70    24   667    23    13    38    15    23
     9     9     0    22    19   874     0     7     0    18
    36    16     6    26     1     1   885     1    51     5
    43    11    30    29    42    15    11   759    20    14
    10     2    17    82     1     1    74     4   803    15
     0     1     2     2     8    14     2     7     1   943
```

Dette gir 86,12 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med Ridge-regularisert LDA

```

>> confmat

confmat =

  1094     4     3     2     3     3     0    26     0     0
    32   816    33    21     5    37     9    58     6    15
     5    25   881     4    27     3    16    29    15     5
    12     6     0   889     4     7     1    10    53     0
     8     4    44    12   736    15    10    37    18     8
     8    11     0    25    29   857     0    16     0    12
    30    15     9    22     3     0   863     4    80     2
    27     7    25    20    54    10     5   794    25     7
     7     1    13    63     6     0    36    12   862     9
     0     0     4     2    13     9     1    10     1   940
```

Dette gir 87,32 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med PLS-DA

```
>> confmat

confmat =

  1107     2     2     1     1     5     2    15     0     0
    55   808    28    16     0    43    21    39     5    17
    15    26   888     2    14     9    21    20    11     4
    23     6     3   872     5    10     2    13    48     0
    18     2    84    19   625    22    13    67    22    20
     9    10     0    21    20   872     0     9     0    17
    37    18     8    20     0     2   877     3    58     5
    55     9    31    27    40    15    12   745    23    17
    10     2    15    69     2     1    77    13   802    18
     0     2     2     2     7    15     2     7     2   941
```

Dette gir 85,37 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med SVD-basis

```

>> confmat

confmat =

  1127     5     2     0     0     1     0     0     0     0
     8   969     7     3     0     2    14    21     1     7
     2     7   947     0    15     0     7    17     6     9
     4     3     0   955     0     5     4     2     9     0
     1     1    17     0   838     6     3    19     5     2
     4     0     0     2    16   926     1     3     0     6
     7    17     1     4     2     0   965     2    29     1
     7     5    16     4     5     1     5   918     5     8
     7     2    12    11     1     0     9    10   950     7
     0     2     0     0     1     4     1     2     0   970
```

Dette gir 95,65 % korrekte klassifiseringer.

Klassifisering av testsett til Zip med Ridge-regresjon med randomisert kontinuitetskrieterium

```
>> confmat

confmat =

  1122     3     1     1     1     3     0     4     0     0
     1   961     8    10     1     6    14    22     1     8
     0     4   957     0    21     2     9    12     5     0
     5     4     0   927     0     9     3     5    28     1
     0     1    17     3   832    13     4     9     6     7
     3     1     0     6     9   927     1     3     0     8
    11    21     2    12     2     0   949     1    29     1
     2     7     8     6    20     4     7   907     5     8
     4     1    12    26     6     2    10     8   932     8
     1     2     0     0     1     6     2     4     0   964
```

94,78 % korrekte klassifiseringer.

Metodene som er spesifikke for bildeanalyse predikerer ikke uventet best.

Kapittel 5

Resultater og konklusjoner

I dette kapittelet vil jeg oppsummere og diskutere resultatene jeg har fått fra arbeidet med denne oppgaven, samt å prøve å si noe om hva som kan og bør undersøkes videre.

5.1 Regresjon

Jeg begynner regresjonsdelen med å vise at multivariat regresjonsanalyse er en ganske triviell utvidelse av dens univariate motpart, og dette brukes videre gjennom hele resten av oppgaven. Jeg viser videre hvordan rask LOOCV kan utføres ved hjelp av singulærverdidekomposisjon. Siden leverage-verdiene regnes ut kun ved hjelp av datamatrisa X, vil også PRESS kunne regnes ut uten veldig mye ekstra regnekraft for multivariate datasett.

Mens man i vanlige minste kvadraters problemer skal minimere

$$\|Y - X\hat{B}\|^2,$$

blir det tilsvarende Tikhonov-regulariserte problemet

$$\min \|Y - X\hat{B}\|^2 + \lambda\|T\hat{B}\|^2.$$

Jeg har vist at dette kan behandles som et vanlig minste kvadraters problem, men der man har utvidet datamatrisa med p rader bestående av $\sqrt{\lambda}T$ og Y-matrisa tilsvarende med nullelementer. Det er dermed mulig å gjøre rask LOOCV også for Tikhonov-regulariserte problemer.

En interessant sammenheng er den jeg har vist i del 3.3.5. Ved å analysere XT^{-1} i stedet for X, vil man kunne gjøre Tikhonov-regularisering med enhver invertibel T-matrise $\in \mathbb{R}^{p\times p}$ ved å gjøre Ridge-regresjon på XT^{-1}. Dette lønner seg fordi kryssvalideringen for Ridge går mye lettere enn for andre regulariseringer. Der det gir mening å gjøre regularisering med derivasjonsmatriser, vil det derfor også gi mening å gjøre andre analyser av XT^{-1}, for eksempel PCR og PLS. Dette er spennende, og det virker som det ikke er publisert noe særlig om slike metoder.

Jeg bruker en del sider på å forklare `TregsMulti`-funksjonen (vedlegg A.1.12). Denne funksjonen er en utvidelse av en funksjon laget av min veileder Ulf Indahl. Funksjonen er kommentert og utvidet til å gjøre multivariat Tikhonov-regresjon. Det viser seg at `TregsMulti` går svært raskt sammenliknet med alle andre multivariate regresjonsimplementeringer jeg har forsøkt. Jeg har gjort en grundig undersøkelse av både antatt prediksjonsevne for modellene denne produserer, og hvor lang tid det tar å lage disse modellene. `TregsMulti` bruker en brøkdel av tiden på å bygge modeller sammenliknet med multivariat PCR og PLSR med LOOCV. Jeg antar at dette skyldes jeg ikke har noen snarvei til PRESS-verdiene for disse to metodene, slik jeg har for Tikhonov-regresjon.

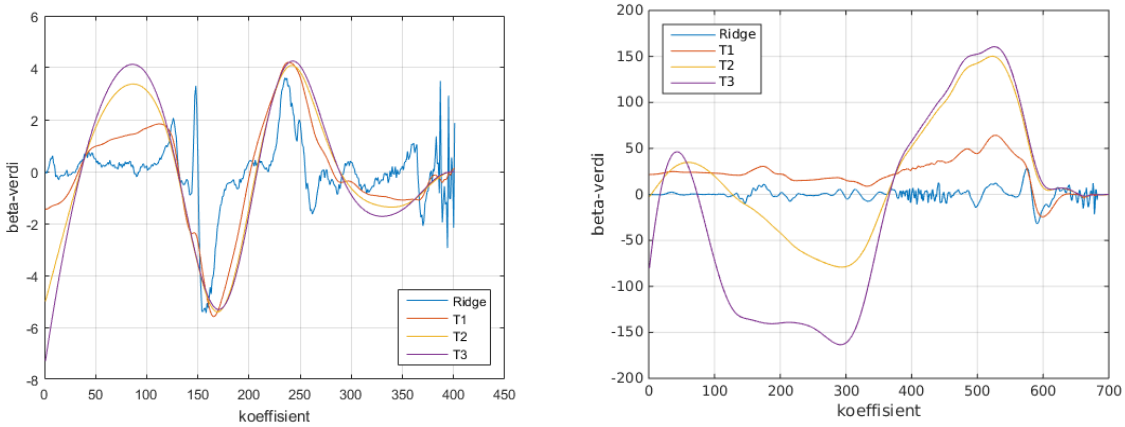
`TregsMulti` ser også ut til å lage modeller som predikerer godt. Selv om Tikhonov-modellen ikke alltid har de laveste PRESS-verdiene, er de aldri svært mye høyere enn verdiene for verken PLSR eller PCR. For prediksjon av ukjent data, ser Tikhonov-regresjon ut til å gjøre det svært godt, det er kun for Sugar-datasettet at PLSR ser ut til å prestere litt bedre. Dette til tross for at Tikhonov-analysene jevnt over ikke gir noen høyere PRESS-verdi, hvilket viser viktigheten av ikke å stole blindt på PRESS for å si noe om prediksjonsevne. Forskjellene er uansett så små at det godt kan hende at det i virkeligheten ikke er noen forskjell på prediksjonsevnen til de forskjellige modellene, men at det skyldes datautvalget.

Det skal sies at datasettene jeg har brukt for regresjon, er svært like. De er kjemometriske data, stort sett av NIR-spektroskopi. Jeg vet derfor ikke hvordan Tikhonov-regresjon vil fungere for andre typer data. Jeg har imidlertid ingen indikasjon på at det skulle forholde seg veldig annerledes, og dette vil være gjenstand for videre studier.

Jeg har forsøkt å gjøre PCR- og PLSR-funksjonene så tidseffektive som mulig. Det kan

godt tenkes at det er implementeringer av disse metodene som går raskere. Særlig LOOCV-algoritmen har jeg jobbet en del med, og den jeg har implementert går en god del raskere enn mer intuitive tilnærminger. I `TregsMulti` lar jeg hver respons få lov å velge sin egen verdi av regulariseringsparameteren λ, heller enn å ha en felles for alle responsvariablene. Jeg har gjort det på samme måte for PLSR og PCR, og her har jeg regnet ut PRESS for forskjellig antall komponenter inntil det maksimale antallet, som bør være lavere enn datamatrisas rang. Et videre arbeid her kunne vært å se på implementeringer der alle responsene hadde samme λ-verdi eller samme antall komponenter, og gjerne begrense antallet komponenter ytterligere.

For å velge modell, har jeg satt som eneste krav at PRESS-verdien skal være så liten som mulig. Det er imidlertid ikke alltid slik at PRESS(λ)-funksjonen har noe bunnpunkt, og da kan man oppleve å få veldig små eller veldig store λ-verdier. Dette vil gjenspeiles i at normen til $\hat{\boldsymbol{\beta}}$ blir henholdsvis veldig stor eller veldig liten (figur 5.1). Stor norm på $\hat{\boldsymbol{\beta}}$ betyr store verdier for regresjonskoeffisientene, noe som gjerne gir en mer ustabil modell. Det kan dermed tenkes at en liten trade-off der man godtar en litt mindre PRESS-verdi kan gi modeller med bedre prediksjonsevne. Dette vil være en naturlig utvidelse.



(a) Spectra

(b) Første respons av Sugar.

Figur 5.1: Et eksempel på forskjell i størrelse på regresjonskoeffisientene. I (a) ser ikke de forskjellige regulariseringene ut til å ha noen særlig forskjell i størresle på regresjonskoeffisientene, mens forskjellen ser ut til å være betydelig i (b). De forskjellige kurvene viser $\hat{\boldsymbol{\beta}}$-verdier for forskjellige regulariseringer for de samme dataene i hver figur.

5.2 Klassifikasjon

Klassifikasjonskapittelet begynner jeg med å vise hvordan man kan se på klassifikasjonsproblemer som multivariate regresjonsproblemer ved å kode responsen numerisk. Denne ideen er grunnlaget for at man kan bruke PCR og PLS i klassifisering.

Videre bruker jeg en del spalteplass på å vise hvordan man kan gjøre Tikhonov-regularisering av LDA med rask LOOCV. Jeg forklarer hvorfor Mahalanobis-avstand er et bedre avstandsmål enn euklidsk, og viser eksempler som støtter dette. Selv om det er mulig å gjøre probabilistisk LDA, og man da får faktiske sannsynligheter for klassetilhørighet for hver måling, er det ikke nødvendig å gjøre dette for selve klassifiseringen, da man bruker den felles kovariansmatrisa for utregning av sannsynlighet i alle grupper, og det eneste som endrer seg er den kvadrerte Mahalanobis-avstanden i eksponenten på sannsynlighetsfunksjonen i (4.3). Klassifisering med sannsynligheter fra denne funksjonen og kun med hensyn på Mahalanobis-avstand er derfor ekvivalent.

Jeg jobber videre med kvadratisk diskriminantanalyse, og viser hvordan snarveien ved å kun se på Mahanlanobis-avstand ikke fungerer når man antar at kovariansmatrisene er forskjellige for hver gruppe. Det er derfor nødvendig å jobbe med sannsynligheter, men det er mulig å gjøre dette litt raskere ved å se på de naturlige logaritmene til sannsynlighetene. Siden $\ln(x)$ er en monoton funksjon, vil den bevare ulikheter, og det å plassere hver måling i den klassen med høyest logaritmisk sannsynlighet vil være det samme som å bruke vanlig sannsynlighet.

Når jeg skal lage implementering for rask LOOCV for QDA støter jeg på problemer. I LDA-tilfellet skal Mahalanobis-avstandene i klassen for den utelatte målingen blant annet multipliseres med

$$\left(1 - \frac{n_c}{(n_c - 1)n} d^2_{\mathrm{M},ic}\right)^{-1} \tag{5.1}$$

Her kan man i ekstreme tilfeller støte på problemet at den kryssvaliderte Mahalanobisavstanden blir negativ, men det vil kun skje hvis den faktiske avstanden er svært stor og n er liten. I QDA skal man multiplisere determinanten til kovariansmatrisa og dele den

kvadrerte Mahalanobis-avstanden med

$$1 - \frac{n_c}{(n_c - 1)^2} d^2_{\mathrm{M},ic}.$$

Denne faktoren har en mye større mulighet enn (5.1) til å bli negativ, og dette skjer når jeg forsøker å bruke metoden på Majones-datasettet. På grunn av dette velger jeg å ikke bruke QDA i videre analyse.

Jeg presenterer også to metoder for bildeanalyse. SVD-basis-metoden er mye brukt, men man betrakter ikke kontinuitet i bilder med denne metoden. Denne er det heller ikke mulig å regularisere, da man kun bruker V-matrisa fra SVDen, og denne vil være lik for regulariserte og ikke-regulariserte data. Metoden til Boyd og Vandenberghe (2015, [4]) med et randomisert kontinuitetskriterium tar hensyn til at bilder ofte i stor grad er kontinuerlige, men ikke overalt. Dette er en spennende metode, og det viser seg at den produserer gode resultater.

Det viser seg at de Tikhonov-regulariserte metodene predikerer bedre enn PLSDA og metoden for bildeanalyse med SVD-basis i alle tilfeller jeg har testet. Tikhonov-metodene er også mer tidseffektive når man gjør LOOCV, og de regulariserte regresjonsmetodene er de mest tidseffektive jeg har sett på for bildeanalyse.

Også for klassifikasjon har datamengden vært begrenset i denne oppgaven, så jeg skal være forsiktig med å generalisere funnene jeg har gjort, spesielt på antatt prediksjonsevne. Figur 4.11 viser at LDA ser ut til å være mindre stabil for Ovarian Cancer-datene enn den ser ut til å være for Majones-datene. Dette kan være en indikasjon på at resultatene er data-avhengige. Selv om de to datasettene er forskjellige, ville det helt klart vært spennende å anvende disse metodene på flere og andre typer datasett, for å få et klarere bilde av deres generelle atferd.

I bildeanalyse har jeg brukt to veldig like datasett, begge er av håndskrevne sifre. Selv om de har ulik oppløsning og antall målinger, måler de i bunn og grunn det samme. En grunn til at disse dataene er brukt, er at bildeanalyse ofte er krever mye regnekraft, mens både Zip-datasettet og MNIST har ganske lav oppløsning og derfor tar relativt kort tid å analysere. En naturlig videre studie ville være å se hvordan den Ridge-regulariserte bildeanalysen med randomisert kontinuitetskriterium vil klassifisere andre typer bilder og

bildeserier. Boyd og Vandenberghe (2015 [4], s 206) beskriver en metode der man ved å legge til enda flere randomiserte størrelser kan testsettet til MNIST med mindre enn 2 % feilmargin, noe som er bedre enn manuell klassifisering. Dette hadde det vært svært interessant å undersøke dette nærmere.

Jeg har laget en funksjon som har implementert rask LOOCV for QDA som beskrevet av Ripley (1996, [24]). Denne har ikke fungert for å analysere Majones-dataene fordi den har gitt komplekse logaritmiske sannsynligheter. Ripley (s. 106) henviser til Marks & Dunn (1974 [20]) og skriver at LDA kan prestere bedre for små datasett der kovariansmatrisene for de forskjellige klassene er like. Det kan godt være at Majones-dataene er et slikt datasett, og det kunne vært interessant å undersøke dette nærmere. I bildeanalyse tar LDA svært lang tid, til tross for at det ikke gjøres LOOCV her. Da QDA forventes å kreve flere utregninger og dermed mer regnekraft, har jeg valgt å ikke bruke QDA videre.

5.3 Konklusjon

Jeg har gjennom denne oppgaven undersøkt Tikhonov-regularisering på regresjon og klassifikasjon, og kommet til at slike metoder i alle tilfeller går raskere enn alternative metoder uten regularisering, både med og uten LOOCV. Selv om mine undersøkelser har vært begrenset, gir de likevel en indikasjon på potensialet til slike metoder. Jevnt over ser Tikhonov-regularisert regresjon ut til å predikere vel så godt som PLSR og PCR for datamengder med flere variable enn antall målinger, i tillegg til at det ser ut til å gå svært mye raskere å lage modeller med Tikhonov-regresjon. Det samme er tilfellet for klassifikasjon.

Jeg har ingen mening om hvorvidt Tikhonov-regresjon vil ta over for andre metoder, til det har jeg for lite kunnskap om og erfaring med anvendt statistikk. Jeg tror imidlertid at Tikhonov-regularisert regresjon og klassifikasjon etterhvert kan bli mer brukt i forskning på den måten at slik analyse svært raskt kan gi en pekepinn på sammenhenger i store datamengder. Tikhonov-regularisering gir ikke nødvendigvis modeller med høyest prediksjonsevne av metodene jeg har undersøkt, men ser ikke ut til å være langt unna. På den måten ser jeg for meg at Tikhonov-regularisert regresjon og klassifikasjon kan bli en av

mange metoder man bruker for å få den beste modellen.

Vedlegg

Tillegg A

MATLAB-script

A.1 Regresjons-script

A.1.1 OLS_LOOCV

```
1  function [Beta, Yhat, Yhatcv, res, rescv, PRESS] = OLS_LOOCV(X, Y)
2
3  % Function finding the least squares solution with full leave one out cross
4  % validation for a data matrix X and corresponding response matrix Y.
5  %
6  % ------------------------------------------------------------------------
7  %
8  % Inputs:
9  % X: Data matrix with rows as individual measurements and colums
10 %    representing each variable. Must be "slim" => # measurements >
11 %    # variables.
12 % Y: Corresponding column vector or matrix of response measurements
13 %
14 % ------------------------------------------------------------------------
15 %
16 % Outputs:
17 % Beta:   Least squares regression coefficients, including constant terms.
18 % Yhat:   Fitted values for the full data set.
19 % Yhatcv: CV-fitted values for the full data set.
20 % res:    Residual matrix = Y - Yhat.
21 % rescv:  CV-residual matrix = Y - Yhatcv.
22 % PRESS:  Statistic used for comparing prediction ability of model =
23 %         sum(rescv.^2).
24 %
```

```
25 % ------------------------------------------------------------------------
26
27 % Initializations:
28 n       = size(X, 1);       % # measurements
29 m       = size(Y, 2);       % # response variables
30 X       = [ones(n, 1), X]; % Expanded data matrix to get the constant term.
31 Yhatcv = zeros(n, m);      % Initialization of Yhatcv
32
33 % LS model:
34 Beta = (X'*X)\(X'*Y); % Regression coefficients
35 Yhat = X*Beta;         % Fitted response matrix
36 res  = Y - Yhat;       % Residual matrix
37
38 % LOOCV:
39 for i = 1:n
40     inds          = setdiff(1:n, i);                % Indices where i is removed
41     Xred          = X(inds, :);                     % Reduced data matrix
42     Bcv           = (Xred'*Xred)\(Xred'*Y(inds, :)); % CV model regression coeff's
43     Yhatcv(i, :) = X(i, :)*Bcv;                    % CV-fitted response values
44 end
45
46 rescv = Y - Yhatcv;     % CV residual martix
47 PRESS = sum(rescv.^2); % PRESS statistic
```

A.1.2 OLS_LOOCV_QR

```
1  function [Beta, Yhat, Yhatcv, res, rescv, PRESS] = OLS_LOOCV_QR(X, Y)
2
3  % Function finding the least squares solution with full leave one out cross
4  % validation for a data matrix X and corresponding response matrix y. Uses
5  % QR-factorization of the data matrix.
6  %
7  % ------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X: Data matrix with rows as individual measurements and colums
11 %    representing each variable. Must be "slim" => # measurements >
12 %    # variables.
13 % Y: Corresponding column vector or matrix of response measurements.
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % Beta:   Least squares regression coefficients, including constant terms.
19 % Yhat:   Fitted values for the full data set.
20 % Yhatcv: CV-fitted values for the full data set.
```

```
21 % res:     Residual matrix = Y - Yhat.
22 % rescv:   CV-residual matrix = Y - Yhatcv.
23 % PRESS:   Statistic used for comparing prediction ability of model =
24 %          sum(rescv.^2).
25 %
26 %--------------------------------------------------------------------------
27
28 % Initializations:
29 n       = size(X, 1);       % # measurements
30 m       = size(Y, 2);       % # response variables
31 X       = [ones(n, 1), X]; % Expanded data matrix to get the constant term.
32 Yhatcv = zeros(n, m);      % Initialization of Yhatcv
33
34 % LS model:
35 [Q, R] = qr(X, 0); % QR factorization where R i square.
36 Beta    = R\(Q'*Y); % Regression coefficients
37 Yhat    = Q*(Q'*Y); % Fitted response matrix
38 res     = Y - Yhat; % Residual matrix
39
40 % LOOCV:
41 for i = 1:n
42     inds          = setdiff(1:n, i);          % Indices where i is removed
43     [Qred, Rred] = qr(X(inds, :), 0);        % QR-factorization of reduced data
44     Bcv           = Rred\(Qred'*Y(inds, :)); % CV model regression coefficients
45     Yhatcv(i, :) = X(i, :)*Bcv;              % CV-fitted response values
46 end
47
48 rescv = Y - Yhatcv;     % CV residual matrix
49 PRESS = sum(rescv.^2); % PRESS statistic
```

A.1.3 OLS_LOOCV_SVD

```
1  function [Beta, Yhat, Yhatcv, res, rescv, PRESS] = OLS_LOOCV_SVD(X, Y)
2
3  % Function finding the least squares solution with full leave one out cross
4  % validation for a data matrix X and corresponding response matrix Y. Uses
5  % singular value decomposition of the data matrix.
6  %
7  %--------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X: Data matrix with rows as individual measurements and colums
11 %    representing each variable. Must be "slim" => # measurements >
12 %    # variables.
13 % y: Corresponding column vector or matrix of response measurements.
14 %
```

```
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % Beta:   Least squares regression coefficients, including constant terms.
19 % Yhat:   Fitted values for the full data set.
20 % Yhatcv: CV-fitted values for the full data set.
21 % res:    Residual matrix = Y - Yhat
22 % rescv:  CV-residual matrix = Y - Yhatcv
23 % PRESS:  Statistic used for comparing prediction ability of model =
24 %         sum(rescv.^2).
25 %
26 % ------------------------------------------------------------------------
27
28 % Initializations:
29 n       = size(X, 1);       % # measurements
30 m       = size(Y, 2);       % # response variables
31 X       = [ones(n, 1), X]; % Expanded data matrix to get the constant term.
32 Yhatcv = zeros(n, m);      % Initialization of yhatcv
33
34 % LS model:
35 [U, S, V] = svd(X, 'econ'); % "Economical" SVD of data matrix
36 Beta      = V*(S\U')*Y;      % Regression coefficients
37 Yhat      = U*U'*Y;          % Fitted response matrix
38 res       = Y - Yhat;        % Residual martix
39
40 % LOOCV:
41 for i = 1:n
42     inds         = setdiff(1:n, i);          % Indices where i is removed
43     [Ur, Sr, Vr] = svd(X(inds, :), 'econ'); % SVD of reduced data
44     bcv          = Vr*(Sr\Ur')*Y(inds, :);  % CV model regression coefficients
45     Yhatcv(i, :) = X(i, :)*bcv;             % CV-fitted response value
46 end
47
48 rescv = Y - Yhatcv;     % CV residual matrix
49 PRESS = sum(rescv.^2); % PRESS statistic
```

A.1.4 OLS_fastLOOCV

```
1 function [Beta, Yhat, Yhatcv, res, h, rescv, PRESS] = OLS_fastLOOCV(X, Y)
2
3 % Function finding the least squares solution with fast leave one out cross
4 % validation for a data matrix X and corresponding response matrix y.
5 %
6 % ------------------------------------------------------------------------
7 %
8 % Inputs:
```

```
9  % X: Data matrix with rows as individual measurements and colums
10 %    representing each variable. Must be "slim" => # measurements >
11 %    # variables.
12 % y: Corresponding column vector or matrix of response measurements.
13 %
14 % ------------------------------------------------------------------------
15 %
16 % Outputs:
17 % Beta:   Least squares regression coefficients, including constant terms.
18 % Yhat:   Fitted values for the full data set.
19 % Yhatcv: CV-fitted values for the full data set.
20 % res:    Residual matrix = y - yhat
21 % h:      Vector of leverage values
22 % rescv:  CV-residual matrix = y - yhatcv
23 % PRESS:  Statistic used for comparing prediction ability of model =
24 %         sum(rescv.^2).
25 %
26 % ------------------------------------------------------------------------
27
28 % Initializations
29 n  = size(X, 1);          % # measurements
30 h  = zeros(n, 1);         % Initialization of h
31 mX = mean(X);             % Mean data values, one for each variable
32 mY = mean(Y);             % Mean response value
33 X  = X - ones(n, 1)*mX; % Centered data matrix
34 Y  = Y - ones(n, 1)*mY; % Centered response values
35
36 Beta = (X'*X)\(X'*Y); % Regression coefficients
37 Yhat = X*Beta;         % Fitted response vector
38 res  = Y - Yhat;       % Residual vector
39
40 % Leverage
41 for i = 1:n
42     x = X(i, :);                % the i-th sample
43     h(i) = x*((X'*X)\x') + 1/n; % corresponding leverage value
44 end
45
46 % LOOCV values
47 Yhatcv = bsxfun(@rdivide, Yhat - bsxfun(@times, Y, h), 1 - h); % CV fitted values
48 rescv  = Y - Yhatcv;              % CV residuals
49 PRESS = sum(rescv.^2);           % PRESS statistic
50
51 Beta = [mY - mX*Beta; Beta]; % reg coeffs with constant term
```

A.1.5 OLS_fastLOOCV_QR

```
1  function [Beta, Yhat, Yhatcv, res, h, rescv, PRESS] = OLS_fastLOOCV_QR(X, Y)
2
3  % Function finding the least squares solution with fast leave one out cross
4  % validation for a data matrix X and corresponding response matrix y. Uses
5  % QR-factorization.
6  %
7  % ------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X: Data matrix with rows as individual measurements and colums
11 %    representing each variable. Must be "slim" => # measurements >
12 %    # variables.
13 % y: Corresponding column vector or matrix of response measurements.
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % Beta:   Least squares regression coefficients, including constant term.
19 % Yhat:   Fitted values for the full data set.
20 % Yhatcv: CV-fitted values for the full data set.
21 % res:    Residual matrix = y - yhat
22 % h:      Vector of leverage values
23 % rescv:  CV-residual matrix = y - yhatcv
24 % PRESS:  Statistic used for comparing prediction ability of model =
25 %         sum(rescv.^2).
26 %
27 % ------------------------------------------------------------------------
28
29 % Initializations
30 n  = size(X, 1);          % # measurements
31 mX = mean(X);             % Mean data values, one for each variable
32 mY = mean(Y);             % Mean response value
33 X  = X - ones(n, 1)*mX; % Centered data matrix
34 Y  = Y - ones(n, 1)*mY; % Centered response values
35
36 % QR model building & leverage
37 [Q, R] = qr(X, 0);        % QR-factorization of centered data
38 Beta = R\(Q'*Y);          % Regression coefficients
39 Yhat = Q*(Q'*Y);          % Fitted response vector
40 res  = Y - Yhat;          % Residual vector
41 h = sum(Q.^2, 2) + 1/n; % Leverage
42
43 % LOOCV values
44 Yhatcv = bsxfun(@rdivide, Yhat - bsxfun(@times, Y, h), 1 - h); % CV fitted values
45 rescv  = Y - Yhatcv;              % CV residuals
```

```
46 PRESS = sum(rescv.^2);              % PRESS statistic
47
48 Beta = [mY - mX*Beta; Beta]; % reg coeffs with constant term
```

A.1.6 OLS_fastLOOCV_SVD

```
1  function [Beta, Yhat, Yhatcv, res, h, rescv, PRESS] = OLS_fastLOOCV_SVD(X, Y)
2
3  % Function finding the least squares solution with fast leave one out cross
4  % validation for a data matrix X and corresponding response matrix y. Uses
5  % singular value decomposition.
6  %
7  % ------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X: Data matrix with rows as individual measurements and colums
11 %    representing each variable. Must be "slim" => # measurements >
12 %    # variables.
13 % y: Corresponding column vector or matrix of response measurements.
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % Beta:   Least squares regression coefficients, including constant term.
19 % Yhat:   Fitted values for the full data set.
20 % Yhatcv: CV-fitted values for the full data set.
21 % res:    Residual matrix = y - yhat
22 % h:      Vector of leverage values
23 % rescv:  CV-residual matrix = y - yhatcv
24 % PRESS:  Statistic used for comparing prediction ability of model =
25 %         sum(rescv.^2).
26 %
27 % ------------------------------------------------------------------------
28
29 % Initializations
30 n  = size(X, 1);          % # measurements
31 mX = mean(X);             % Mean data values, one for each variable
32 mY = mean(Y);             % Mean response value
33 X  = X - ones(n, 1)*mX; % Centered data matrix
34 Y  = Y - ones(n, 1)*mY; % Centered response values
35
36 % SVD model building & leverage
37 [U, S, V] = svd(X, 'econ');    % "Economical" SVD of centered data
38 Beta = V*(sparse(S)\(U'*Y)); % Regression coefficients
39 Yhat = U*(U'*Y);              % Fitted response vector
40 res  = Y - Yhat;              % Residual vector
```

```
41 h = sum(U.^2, 2) + 1/n;          % Leverage
42
43 % LOOCV values
44 Yhatcv = bsxfun(@rdivide, Yhat - bsxfun(@times, Y, h), 1 - h); % CV fitted values
45 rescv  = Y - Yhatcv;               % CV residuals
46 PRESS = sum(rescv.^2);             % PRESS statistic
47
48 Beta = [mY - mX*Beta; Beta]; % reg coeffs with constant term
```

A.1.7 Ridge_fastLOOCV

```
1  function [Beta, Betas, H, PRESS, lambda] = Ridge_fastLOOCV(X, Y, lambdas)
2
3  % Function finding the least squares solution with Ridge regularization and
4  % fast leave one out cross validation for a data matrix X and corresponding
5  % response vector y over an array of lambdas.
6  %
7  % ------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X:       Data matrix with rows as individual measurements and colums
11 %           representing each variable.
12 % Y:       Corresponding column vector or matrix of response measurements.
13 % lambdas: Array of regularization parameter values
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % beta:   The "winning" regression coefficients, including constant term.
19 % betas:  Tensor containing matrices with all regression coefficient
20 %         vectors as rows, one for each lambda value
21 % H:      Matrix of leverage values, each column represents a lambda value
22 % PRESS:  Matrix of PRESS values. Each row represents a response
23 %         variable, each column a response variable
24 % lambda: The "winning" regularization parameter value for each response
25 %         variable
26 %
27 % ------------------------------------------------------------------------
28
29 % Initializations
30 [n, p]  = size(X);              % Sample size
31 m       = size(Y, 2);           % # of response variables
32 q       = length(lambdas);      % # of regularization parameter values
33 H       = zeros(n, q);          % Initialization of H
34 Betas   = zeros(p, m, q);       % Initialization of Betas
35 Beta    = zeros(p + 1, m);      % Initialization of Beta
```

```
36  PRESS    = zeros(q, m);         % Initialization of PRESS
37  mX       = mean(X);             % Mean data values, one for each variable
38  mY       = mean(Y);             % Mean response value
39  X        = X - ones(n, 1)*mX; % Centered data matrix
40  Y        = Y - ones(n, 1)*mY; % Centered response values
41  Y        = [Y; zeros(p, m)];  % Augmented response vector
42
43  for i = 1:q
44      Z = [X;sqrt(lambdas(i))*eye(p)]; % Augmented data matrix
45      Betas(:, :, i) = (Z'*Z)\(Z'*Y);  % Regression coefficients
46      Yhat = X*Betas(:, :, i);         % Model fitted response predictions
47
48      % Leverage
49      for j = 1:n
50          z = Z(j, :);                    % the i-th sample
51          H(j, i) = z*((Z'*Z)\z') + 1/n; % corresponding leverage value
52      end
53
54      % LOOCV values
55      h = H(1:n, i);
56      Yhatcv = bsxfun(@rdivide, Yhat - bsxfun(@times, Y(1:n, :), h), 1 - h); % CV fitted values
57      rescv  = Y(1:n, :) - Yhatcv;              % CV residuals
58      PRESS(i, :) = sum(rescv.^2);              % PRESS statistic
59  end
60  [~, inds] = min(PRESS);
61  lambda = lambdas(inds);
62  for i = 1:m
63      Beta(:, i) = [mY(i) - mX*Betas(:, i, inds(i)); Betas(:, i, inds(i))];
64  end
65  if m == 1
66      Betas = permute(Betas, [1, 3, 2]);
67  end
```

A.1.8 Ridge_fastLOOCV_QR

```
1  function [Beta, Betas, H, PRESS, lambda] = Ridge_fastLOOCV_QR(X, Y, lambdas)
2
3  % Function finding the least squares solution with Ridge regularization and
4  % fast leave one out cross validation for a data matrix X and corresponding
5  % response vector y over an array of lambdas. Uses QR-factorization.
6  %
7  % ------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X:        Data matrix with rows as individual measurements and colums
11 %           representing each variable.
```

```
12 % Y:        Corresponding column vector or matrix of response measurements.
13 % lambdas: Array of regularization parameter values
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % beta:   The "winning" regression coefficients, including constant term.
19 % betas:  Tensor containing matrices with all regression coefficient
20 %         vectors as rows, one for each lambda value
21 % H:      Matrix of leverage values, each column represents a lambda value
22 % PRESS:  Matrix of PRESS values. Each row represents a response
23 %         variable, each column a response variable
24 % lambda: The "winning" regularization parameter value for each response
25 %         variable
26 %
27 % ------------------------------------------------------------------------
28
29 % Initializations
30 [n, p]   = size(X);              % Sample size
31 m        = size(Y, 2);           % # of response variables
32 q        = length(lambdas);      % # of regularization parameter values
33 H        = zeros(n, q);          % Initialization of H
34 Betas    = zeros(p, m, q);       % Initialization of Betas
35 Beta     = zeros(p + 1, m);      % Initialization of Beta
36 PRESS    = zeros(q, m);          % Initialization of PRESS
37 mX       = mean(X);              % Mean data values, one for each variable
38 mY       = mean(Y);              % Mean response value
39 X        = X - ones(n, 1)*mX;    % Centered data matrix
40 Y        = Y - ones(n, 1)*mY;    % Centered response values
41
42 for i = 1:q
43     Z              = [X;sqrt(lambdas(i))*eye(p)]; % Augmented data matrix
44     [Q, R]         = qr(Z, 0);                    % QR-factorization of Z
45     Q              = Q(1:n, :);                   % First n rows of Q
46     Betas(:, :, i) = R\(Q'*Y);                    % Reg coefficients
47     Yhat           = Q*(Q'*Y);                    % Response predictions
48     H(:, i)        = sum(Q.^2, 2) + 1/n;          % Leverage
49
50     % LOOCV values
51     h = H(:, i);
52     Yhatcv = bsxfun(@rdivide, Yhat - bsxfun(@times, Y, h), 1 - h); % CV fitted values
53     rescv  = Y - Yhatcv;                          % CV residuals
54     PRESS(i, :) = sum(rescv.^2);                  % PRESS statistic
55 end
56 [~, inds] = min(PRESS);
57 lambda = lambdas(inds);
58 for i = 1:m
```

```
59      Beta(:, i) = [mY(i) - mX*Betas(:, i, inds(i)); Betas(:, i, inds(i))];
60  end
61  if m == 1
62      Betas = permute(Betas, [1, 3, 2]);
63  end
```

A.1.9 Ridge_fastLOOCV_SVD

```
1   function [Beta, Betas, H, PRESS, lambda] = Ridge_fastLOOCV_SVD(X, Y, lambdas)
2
3   % Function finding the least squares solution with Ridge regularization and
4   % fast leave one out cross validation for a data matrix X and corresponding
5   % response vector y over an array of lambdas. Uses SVD.
6   %
7   % ------------------------------------------------------------------------
8   %
9   % Inputs:
10  % X:       Data matrix with rows as individual measurements and colums
11  %          representing each variable.
12  % Y:       Corresponding column vector or matrix of response measurements.
13  % lambdas: Array of regularization parameter values
14  %
15  % ------------------------------------------------------------------------
16  %
17  % Outputs:
18  % beta:   The "winning" regression coefficients, including constant term.
19  % betas:  Tensor containing matrices with all regression coefficient
20  %         vectors as rows, one for each lambda value
21  % H:      Matrix of leverage values, each column represents a lambda value
22  % PRESS:  Matrix of PRESS values. Each row represents a response
23  %         variable, each column a response variable
24  % lambda: The "winning" regularization parameter value for each response
25  %         variable
26  %
27  % ------------------------------------------------------------------------
28
29  % Initializations
30  [n, p]  = size(X);              % Sample size
31  m       = size(Y, 2);           % # of response variables
32  q       = length(lambdas);      % # of regularization parameter values
33  H       = zeros(n, q);          % Initialization of H
34  Betas   = zeros(p, m, q);       % Initialization of Betas
35  Beta    = zeros(p + 1, m);      % Initialization of Beta
36  PRESS   = zeros(q, m);          % Initialization of PRESS
37  mX      = mean(X);              % Mean data values, one for each variable
38  mY      = mean(Y);              % Mean response value
```

```
39 X       = X - ones(n, 1)*mX; % Centered data matrix
40 Y       = Y - ones(n, 1)*mY; % Centered response values
41
42 [U, S, V] = svd(X, 'econ');
43 S2 = S.^2;
44
45 for i = 1:q
46     Betas(:, :, i) = V*(S./(S2 + lambdas(i)))*U'*Y;
47     Yhat = X*Betas(:, :, i);
48     H(:,i) = sum(bsxfun(@times,U,(diag(S./sqrt(S2+lambdas(i))))').^2,2)+1/n;     % LOOCV values
49     h = H(:, i);
50     Yhatcv = bsxfun(@rdivide, Yhat - bsxfun(@times, Y, h), 1 - h); % CV fitted values
51     rescv  = Y - Yhatcv;              % CV residuals
52     PRESS(i, :) = sum(rescv.^2);      % PRESS statistic
53 end
54 [~, inds] = min(PRESS);
55 lambda = lambdas(inds);
56 for i = 1:m
57     Beta(:, i) = [mY(i) - mX*Betas(:, i, inds(i)); Betas(:, i, inds(i))];
58 end
59 if m == 1
60     Betas = permute(Betas, [1, 3, 2]);
61 end
```

A.1.10 Ridge_fastLOOCV_fmin

```
1  function [Beta, H, PRESS, lambda] = Ridge_fastLOOCV_fmin(X, Y, lambdas)
2
3  % Function finding the least squares solution with Ridge regularization and
4  % fast leave one out cross validation for a data matrix X and corresponding
5  % response vector y over an array of lambdas. Uses SVD.
6  %
7  % ------------------------------------------------------------------------
8  %
9  % Inputs:
10 % X:       Data matrix with rows as individual measurements and colums
11 %           representing each variable.
12 % Y:       Corresponding column vector or matrix of response measurements.
13 % lambdas: Array consisting of the two boundaries for the lambda interval.
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % beta:   The "winning" regression coefficients, including constant term.
19 % betas:  Tensor containing matrices with all regression coefficient
20 %          vectors as rows, one for each lambda value
```

```
% H:       Matrix of leverage values, each column represents a lambda value
% PRESS:   Matrix of PRESS values. Each row represents a response
%          variable, each column a response variable
% lambda:  The "winning" regularization parameter value for each response
%          variable
%
% ------------------------------------------------------------------------

% Initializations
[n, p]   = size(X);             % Sample size
m        = size(Y, 2);          % # of responses
mX       = mean(X);             % Mean data values, one for each variable
mY       = mean(Y);             % Mean response value
X        = X - ones(n, 1)*mX;   % Centered data matrix
Y        = Y - ones(n, 1)*mY;   % Centered response values
PRESS    = zeros(1, m);         % init of PRESS
lambda   = zeros(1, m);         % init of lambda
Beta     = zeros(p, m);         % init of Beta

[U, S, V] = svd(X, 'econ');
S2 = S.^2;
for k = 1:m
    Betaf = @(lambda) V*(S./(S2 + lambda))*U'*Y(:, k); % reg coeffs
    H = @(lambda) sum(bsxfun(@times,U,(diag(S./sqrt(S2+lambda)))').^2,2)+1/n; % leverage
    res = X*Betaf(lambda) - bsxfun(@times, Y(:, k), H(lambda)); % residuals
    Yhatcv = @(lambda) bsxfun(@rdivide, res, 1 - H(lambda)); % CV fitted values
    PRESSs = @(lambda) sum((Y(:, k) - Yhatcv(lambda)).^2);      % PRESS statistic
    [PRESS(k), lambda(k)] = fminbnd(PRESSs, lambdas(1), lambdas(2)); % PRESS & lambda
    Beta(:, k) = V*(S./(S2 + lambda(k)))*(U'*Y(:, k)); % Winning reg coeffs
end

Beta = [mY - mX*Beta; Beta]; % reg coeffs with constant term
```

A.1.11 Ridge_GCV_SVD

```
function [Beta, Betas, H, GCV, lambda] = Ridge_GCV_SVD(X, Y, lambdas)

% Function finding the least squares solution with Ridge regularization and
% fast leave one out cross validation for a data matrix X and corresponding
% response vector y over an array of lambdas. Uses SVD.
%
% ------------------------------------------------------------------------
%
% Inputs:
% X:       Data matrix with rows as individual measurements and colums
%          representing each variable.
```

```
12 % Y:        Corresponding column vector or matrix of response measurements.
13 % lambdas: Array of regularization parameter values
14 %
15 % ------------------------------------------------------------------------
16 %
17 % Outputs:
18 % beta:   The "winning" regression coefficients, including constant term.
19 % betas:  Tensor containing matrices with all regression coefficient
20 %         vectors as rows, one for each lambda value
21 % H:      Matrix of leverage values, each column represents a lambda value
22 % PRESS:  Matrix of PRESS values. Each row represents a response
23 %         variable, each column a response variable
24 % lambda: The "winning" regularization parameter value for each response
25 %         variable
26 %
27 % ------------------------------------------------------------------------
28
29 % Initializations
30 [n, p] = size(X);              % Sample size
31 m      = size(Y, 2);           % # of response variables
32 q      = length(lambdas);      % # of regularization parameter values
33 H      = zeros(n, q);          % Initialization of H
34 Betas  = zeros(p, m, q);       % Initialization of Betas
35 Beta   = zeros(p + 1, m);      % Initialization of Beta
36 GCV    = zeros(q, m);          % Initialization of PRESS
37 mX     = mean(X);              % Mean data values, one for each variable
38 mY     = mean(Y);              % Mean response value
39 X      = X - ones(n, 1)*mX;    % Centered data matrix
40 Y      = Y - ones(n, 1)*mY;    % Centered response values
41
42 [U, S, V] = svd(X, 'econ');
43 S2 = S.^2;
44
45 for i = 1:q
46     Betas(:, :, i) = V*(S./(S2 + lambdas(i)))*U'*Y;
47     Yhat = X*Betas(:, :, i);
48     H(:,i) = sum(bsxfun(@times,U,(diag(S./sqrt(S2+lambdas(i))))').^2,2)+1/n;     % LOOCV values
49     h = sum(H(:, i));
50     resGCV = bsxfun(@rdivide, Y - Yhat, 1 - h/n); % CV fitted values
51     GCV(i, :) = sum(resGCV.^2);      % PRESS statistic
52 end
53 [~, inds] = min(GCV);
54 lambda = lambdas(inds);
55 for i = 1:m
56     Beta(:, i) = [mY(i) - mX*Betas(:, i, inds(i)); Betas(:, i, inds(i))];
57 end
58 if m == 1
```

```
59      Betas = permute(Betas, [1, 3, 2]);
60  end
```

A.1.12 TregsMulti

```
1   % Function that does Tikhonov regression on data with univariate or
2   % multivariate response. Uses the internal MATLAB function min() to
3   % optimize with regard to PRESS.
4   %
5   % Inputs:
6   % ———————
7   % X:        Matrix of data with rows as data points.
8   % Y:        Matrix or possibly vector of response data
9   % lambdas: Array of (log-spread) regularization parameter candidates
10  % type:     Type of regularization; 0 for norm, n > 0 for nth derivative.
11  %           Default value 0 (Ridge regularization)
12  % strd:     Omits standardization of data if this value is set to 0.
13  %           Optional with default set to standardization.
14  %
15  % Outputs:
16  % ————————
17  % Beta:    Matrix of regression coefficients produced by LOOCV. Each column
18  %          is a set of coefficients, one for each response variable.
19  % lambda: The best lambdas for each response produced by LOOCV.
20  % PRESS:   Matrix of PRESS values for each lambda. Each column represents a
21  %          response variable.
22  % Betas:   Collection of matrices containing all regression coefficients.
23  %          Each row is a set of regression coefficients for one
24  %          regularization parameter value, whereas each matrix respresents a
25  %          response variable.
26
27  function [Beta, lambda, PRESS, Betas] = TregsMulti(X, Y, lambdas, d, strd)
28
29  % Setting default values
30  if nargin == 3
31      d    = 0;
32      strd = 0;
33  elseif nargin == 4
34      strd = 0;
35  end
36
37  % Initializations:
38  [n, p] = size(X);          % Size of data
39  m      = size(Y, 2);       % Number of response variables
40  q      = length(lambdas); % Number of regularization parameter candidates
41  Betas  = cell(m, 1);       % Init of Betas
```

```
42 PRESS  = zeros(q, m);      % Init of PRESS
43 Beta   = zeros(p + 1, m); % Init of Beta
44 inds   = zeros(1, m);      % Init of inds
45
46 % Centering of data:
47 mX = mean(X);             % Mean value of each X column
48 mY = mean(Y);             % Mean value of each Y column
49 X  = X - ones(n, 1)*mX; % Centered explanatory data
50 Y  = Y - ones(n, 1)*mY; % Centered response data
51
52 % Regularization type:
53 if d ~= 0
54     T = diff([speye(p);sparse(d, p)], d); % Regularization matrix
55     X = X/T;                              % "Regularized" data w/o lambda
56 end
57
58 % Standardization of data:
59 if strd == 1
60     St = std(X);
61     X = bsxfun(@rdivide, X, St); % Standardized data
62 end
63
64 % SVD:
65 [U, S, V] = svd(X, 'econ'); % Economy version of singular value decomp.
66
67 % Leverage:
68 s  = diag(S);                                % "Linearization" of the singular values
69 s2 = s.^2;                                   % Squaring the singular values
70 D  = bsxfun(@plus, s2, lambdas);             % Matrix of singular values for each lambda
71 H  = (U.^2)*bsxfun(@rdivide, s2, D) + 1/n; % Leverage values for each lambda
72
73 % Regression coefficients, GCV, and PRESS:
74
75 UYS = bsxfun(@times, U'*Y, s); % Matrix product used in the loop ahead
76
77 for k = 1:m % Values for response variable i
78     Betas{k}        = V*bsxfun(@rdivide, UYS(:, k), D); % Regression coefficients
79     res             = bsxfun(@minus, Y(:, k), X*Betas{k}); % Residuals
80     PRESS(:, k)     = sum(bsxfun(@rdivide, res, (1-H)).^2)'; % PRESS
81     [~, inds(k)]    = min(PRESS(:, k)); % lambdas-indices for lowest PRESS
82     beta            = Betas{k}(:, inds(k)); % "Winning" reg coeffs
83     if d > 0
84         beta = T\beta;  % Transforms beta
85     end
86     if strd ~= 0
87         beta = St'.\beta; % "Unstandardizes" beta
88     end
```

```
89      Beta(:, k)    = [mY(k) - mX*beta; beta]; % Reg coeffs with constant term
90  end
91
92  % Regularization parameter values
93  lambda = lambdas(inds); % LOOCV regularization parameters for each response
```

A.1.13 TregsMulti2C

```
1   % Function that does Tikhonov regression on two criteria for data with
2   % univariate or multivariate response. Uses the internal MATLAB function
3   % min() to optimize with regard to PRESS. Keeps the best mu parameter to
4   % find the best lambda parameter under the assumption that the two
5   % parameters are independent.
6   %
7   % Inputs:
8   % ————————
9   % X:        Matrix of data with rows as data points.
10  % Y:        Matrix or possibly vector of response data
11  % mus:      Array of (log-spread) first regularization parameter candidates
12  % lambdas:  Array of (log-spread) second regularization parameter candidates
13  % d1:       First regularization type
14  % d2:       Second regularization type
15  %
16  % Outputs:
17  % ————————
18  % Beta:     Matrix of regression coefficients produced by LOOCV. Each column
19  %           is a set of coefficients, one for each response variable.
20  % mu:       The best mus for each resopnse produced by LOOCV.
21  % lambda:   The best lambdas for each response produced by LOOCV.
22  % PRESS:    Matrix of PRESS values for each lambda. Each column represents a
23  %           response variable.
24
25  function [Beta, mu, lambda, PRESS] = TregsMulti2C(X, Y, lambdas, mus, d1, d2)
26
27  % Initializations
28  p      = size(X, 2);       % # of predictor variables
29  m      = size(Y, 2);       % # of responses
30  q      = length(lambdas); % # of lambda values
31  mX     = mean(X);          % mean values of each X column
32  mY     = mean(Y);          % mean values of each Y column
33  Beta   = zeros(p + 1, m); % init of Beta
34  lambda = zeros(1, m);      % init of lambda
35  PRESS  = zeros(q, m);      % init of PRESS
36  Z      = cell(m, 1);       % init of Z
37
38  % First d1-regression
```

```
39 [~, mu] = TregsMulti(X, Y, mus, d1); % returns best mu regul. param for first regression
40
41 % Second d2-regression
42 T = diff([speye(p); sparse(d2, p)], d2); % regularization matrix
43 Y = [Y; zeros(p, m)];                    % augmented Y matrix
44 for k = 1:m
45     Z{k} = [X;sqrt(mu(k))*T];            % augmented data matrix
46     % Tikh. regression for kth response:
47     [Beta(1:p +1, k), lambda(k), PRESS(:, k)] = TregsMulti(Z{k}, Y(:, k), lambdas, d2); % Treg for kth
48     Beta(1, k) = mY(k) - mX*Beta(2:end, k); % reg coeffs with constant term
49 end
```

A.1.14 TregsMulti2Cr

```
1  % Function that does Tikhonov regression on two criteria for data with
2  % univariate or multivariate response. Uses the internal MATLAB function
3  % min() to optimize with regard to PRESS.
4  %
5  % Inputs:
6  % ————————
7  % X:       Matrix of data with rows as data points.
8  % Y:       Matrix or possibly vector of response data
9  % mus:     Array of (log-spread) first regularization parameter candidates
10 % lambdas: Array of (log-spread) second regularization parameter candidates
11 % d1:      First regularization type
12 % d2:      Second regularization type
13 %
14 % Outputs:
15 % —————————
16 % Beta:   Matrix of regression coefficients produced by LOOCV. Each column
17 %         is a set of coefficients, one for each response variable.
18 % mu:     The best mus for each resopnse produced by LOOCV.
19 % lambda: The best lambdas for each response produced by LOOCV.
20 % PRESS:  Matrix of PRESS values for each lambda. Each column represents a
21 %         response variable.
22
23 function [Beta, mu, lambda, PRESS] = TregsMulti2Cr(X, Y, lambdas, mus, d1, d2)
24 % Initializations
25 [n, p] = size(X);              % sample size
26 m      = size(Y, 2);           % # of response variables
27 q      = length(lambdas);      % # of lambda values
28 r      = length(mus);          % # of mu values
29 mX     = mean(X);              % means of X columns
30 mY     = mean(Y);              % means of Y columns
31 X      = X - ones(n, 1)*mX; % centers X
32 Y      = Y - ones(n, 1)*mY; % centers Y
```

```
33 mu     = zeros(m, 1);        % init of mu
34 lambda = zeros(m, 1);        % init of lambda
35 Beta   = zeros(p, m);        % init of Beta
36 PRESS  = cell(m, 1);         % init of PRESS
37 Betas  = cell(m, 1);         % init of Betas
38
39 for i = 1:m
40     PRESS{m} = zeros(q, r);     % fills PRESS cells with zero matrices
41     Betas{m} = zeros(p, q, r); % fills Betas cells with zero matrices
42 end
43
44 % Augmentation of data
45
46 T2 = diff([speye(p); sparse(d2, p)], d2); % regul. type 2 transf. matrix
47
48
49 Y = [Y; zeros(p, m)]; % Y augmented
50 Z = [X; full(T2)];     % X augmented
51
52 % Transform of Z if regularization type 1 is > 0
53 if d1 > 0
54     T1 = diff([speye(p); sparse(d1, p)], d1); % regul. type 1 transf. matrix
55     Z  = Z/T1;                                % transformed Z
56 end
57
58 Xi = Z; % use Xi in loop to update data matrix more easily
59 for i = 1:r
60     Xi(n + 1:end, :) = sqrt(mus(i))*Z(n + 1:end, :);          % mu update
61     [U, S, V]        = svd(Xi, 'econ');                       % svd
62     s                = diag(S);                               % row of sv's
63     S2               = s.^2;                                  % row of squared sv's
64     D                = bsxfun(@plus, S2, lambdas);            % regularized sv's
65     H                = (U.^2)*bsxfun(@rdivide, S2, D) + 1/n; % leverage values
66     UYs              = bsxfun(@times, U'*Y, s);               % matrix prod in loop
67     for h = 1:m
68         Betas{h}(:, :, i) = V*bsxfun(@rdivide, UYs(:, h), D);   % reg coeffs
69         res = bsxfun(@minus, Y(:, h), Xi*Betas{h}(:, :, i));    % residuals
70         PRESS{h}(:, i) = sum(bsxfun(@rdivide, res, (1-H)).^2)'; % PRESS values
71     end
72 end
73 for i = 1:m
74     [~, idx]      = min(PRESS{i}(:));          % vector index of minimal PRESS value
75     [lidx, midx] = ind2sub([q, r], idx);      % matrix indices of minimal PRESS value
76     mu(i)         = mus(midx);                % mu value
77     lambda(i)     = lambdas(lidx);            % lambda value
78     Beta(:, i)    = Betas{i}(:, lidx, midx); % reg coeffs for lowest PRESS
79     if d1 > 0
```

```
80          Beta(:, i) = T1\Beta(:, i);          % retransformed reg coeffs
81      end
82  end
83  Beta = [mY - mX*Beta; Beta]; % reg coeffs with constant terms
```

A.1.15 PCR2LOOCV

```
1   %% LOOCV Principal component regression.
2
3   function [BetaLOOCV, PRESS] = PCR2LOOCV(X, Y)
4
5   % Function doing LOOCV multivariate PCR for considering the maximum
6   % possible number of PCs.
7
8   % ------------------------------------------------------------------------
9   %
10  % Inputs:
11  % X:        Data matrix with rows as individual measurements and colums
12  %           representing each variable.
13  % Y:        Corresponding column vector or matrix of response measurements.
14  %
15  % ------------------------------------------------------------------------
16  %
17  % Outputs:
18  % BetaLOOCV: The "winning" regression coefficients, including constant
19  %            term.
20  % PRESS:     Matrix of PRESS values. Each row represents a response
21  %            variable, each column a response variable
22  %
23  % ------------------------------------------------------------------------
24
25
26  [n, p]      = size(X);          % sample size
27  m           = size(Y, 2);       % # of responses
28  A           = min(n, p) - 2;    % max # of components
29  PRESS       = zeros(n - 2, m); % Init of PRESS
30  BetaLOOCV   = zeros(p + 1, m);      % Init of BetaLOOCV
31
32  % LOOCV
33  for i = 1:n
34      inds = setdiff(1:n, i);                          % nth sample left out
35      beta = pcr_int(X(inds, :), Y(inds, :), A); % fast pls for reduced data
36      for j = 1:A
37          res2 = (bsxfun(@minus, Y(i, :),[1, X(i, :)]*beta(:, :, j))).^2; % squared LOO res
38          PRESS(j, :) = PRESS(j, :) + res2;                    % adds to PRESS
39      end
```

```
40 end
41
42 [~, ncomp] = min(PRESS);  % # of components giving lowest PRESS for each response
43
44 Beta        = pcr_int(X, Y, max(ncomp)); % reg coeffs for full data set
45
46 for i = 1:m
47     BetaLOOCV(:, i) = Beta(:, i, ncomp(i)); % LOOCV reg coeffs
48 end
49
50
51 function Beta = pcr_int(X,Y,A)
52 %% Principal component regression with k components.
53
54 % Function for computing PCR based on A principal componets
55
56 % Written by U. G. Indahl, M. Ansnes has made the following adjustments:
57 % 1) Removed the option of finding the reg coeffs for just one choice of #
58 %    of components.
59 % 2) Expanded the function to work with multivariate response.
60
61 % Inits:
62 [n, p] = size(X);              % sample size
63 m      = size(Y, 2);           % # of responses
64 mX     = mean(X);              % means of X columns
65 X      = X - ones(n, 1)*mX;    % centering of X
66 Beta   = zeros(p + 1, A, m); % init of Beta
67
68 [U, S, V] = svds(X, A);        % extracting first A PCs
69 s = diag(S);                   % diagonalization of S
70 for k = 1:m
71     b = cumsum(bsxfun(@times, V, (s.\(U'*Y(:, k)))'),2); % reg coeffs for <= A PCs
72     Beta(:, :, k) = [mean(Y(:, k)) - mX*b; b];           % reg coeffs with const terms
73 end
74 Beta = permute(Beta, [1, 3, 2]); % rearrangement of dimensions for easier use of models
```

A.1.16 PLS2fastLOOCV

```
1 % Function that does leave one out cross validation on multi-response data
2 % analyzed with PLSR. Uses the function pls2fast as an internal function,
3 % to which I have made the following modifications:
4 %   1. Changed the names of the different dimensions.
5 %   2. Changed the beta-output so that the first index is a specific
6 %      regression coefficient, the second index gives the response
7 %      variable, and the third gives the number of PLS components used.
8 %-------------------------------------------------------------------------
```

```
9  %
10 % Inputs:  X - data matrix with rows as measurements and one column for
11 %              each variable.
12 %          Y - matrix of measured responses, one column for each response
13 %              variable.
14 %
15 % Outputs: betaLOOCV - matrix of regression coefficients. Each row
16 %                       represents a set of coefficients for one response
17 %                       variable.
18 %          PRESS     - matrix of PRESS values. Each row represents the
19 %                       number of PLS components used, and each row
20 %                       represents a response variable.
21
22 function [betaLOOCV, PRESS] = PLS2fastLOOCV(X, Y)
23
24 % Inits
25
26 [n, p]      = size(X);          % sample size
27 m           = size(Y, 2);       % # of responses
28 A           = min(n, p) - 1;    % max # of components
29 PRESS       = zeros(n - 1, m); % Init of PRESS
30 betaLOOCV   = zeros(p, m);      % Init of BetaLOOCV
31
32 % LOOCV
33 for i = 1:n
34     inds = setdiff(1:n, i);                          % nth sample left out
35     beta = pls2fast_int(X(inds, :), Y(inds, :), A); % fast pls for reduced data
36     for j = 1:A
37         res2 = (bsxfun(@minus, Y(i, :), X(i, :)*beta(:, :, j))).^2; % squared LOO res
38         PRESS(j, :) = PRESS(j, :) + res2;                            % adds to PRESS
39     end
40 end
41
42 [~, ncomp] = min(PRESS);  % # of components giving lowest PRESS for each response
43
44 beta        = pls2fast_int(X, Y, max(ncomp)); % reg coeffs for full data set
45
46 for i = 1:m
47     betaLOOCV(:, i) = beta(:, i, ncomp(i)); % LOOCV reg coeffs
48 end
49
50 function beta = pls2fast_int(X, Y, A)
51 % ------------------------------------------------------------------------
52 % ---------------------- Ulf Indahl 11/02-2014 ---------------------------
53 % ------------------------------------------------------------------------
54 [n,p] = size(X);
55 %mX = mean(X); X = bsxfun(@minus, X, mX); % size & mean centering of X.
```

```
56  %mnY = mean(Y); Y = bsxfun(@minus, mY,Y);
57  m = size(Y,2);
58  beta = zeros(p,m,A);    % The regression coefficients
59  q = zeros(m,A);
60  T  = zeros(n,A);    % Orthonormal scores
61  W  = zeros(p,A);    % Orthonormal weights
62  % ------------------------ Solution of the PLS-problem ------------------------
63  for a = 1:A
64      [w,~,~] = svd(X'*Y,'econ'); w = w(:,1);
65      if a > 1
66          w = w-W(:,1:a-1)*(W(:,1:a-1)'*w); % Included for numerical stability.
67      end
68      W(:,a)= w/norm(w);
69      t = X*W(:,a);
70      if a > 1
71          t = t-T(:,1:a-1)*(T(:,1:a-1)'*t);
72      end
73      T(:,a) = t/norm(t);
74      q(:,a) = Y'*T(:,a);      % Regression coeffs (Y-loadings) for the orthogonal scores.
75      Y = Y - T(:,a)*q(:,a)'; % Y-residual for extraction of next component (Y-deflation)
76  end
77  % ---------------------------------------------------------------------------
78  % ---- Postprocessing to find regression coeffs & other key matrices ----
79  P    = X'*T; % X-loadings.
80  PtW  = triu(P'*W); % The W-coordinates of (the projected) P.
81  R    = W/PtW; % The "SIMPLS weights".
82
83  for i = 1:m,
84      beta(:,i,:) = cumsum(bsxfun(@times,R, q(i,:)),2);
85  end
```

A.2 Klassifiserings-script

A.2.1 RDA

```
1  function [Ghat, confmat, pcc, beta] = RDA(X, G, method)
2
3  % Function doing discriminant analysis by ordinary least squares
4  % regression, principal component regression, or partial least squares
5  % regression. Uses functions PCR2LOOCV and PLS2fastLOOCV.
6  %
7  % Inputs
8  % --------
9  % X:          Data matrix of n measurements (rows) and p predictor variables
10 %             (colums).
```

```
11 % G:          Group membership column vector.
12 % (method): Regression method. 'OLS' (Ordinary Least Squares), 'PCA'
13 %            (Principal Component Analysis), or 'PLS' (Partial Least
14 %            Squares). If not specified, function automatically chooses OLS
15 %            if n > p, elsewise PLS.
16 %
17 % Outputs
18 % ————————
19 % Ghat:     Model classifications of measurements in X.
20 % confmat: Confusion matrix.
21 % pcc:      Fraction of correct classifications.
22
23 Gd       = dummyvar(G); % dummy coding of G
24 [n, p]   = size(X);      % sample size
25 K        = max(G);       % # of classes
26 confmat = zeros(K);     % init of confusion matrix
27
28
29 if nargin == 2
30     if n > p
31         method = 'OLS'; % sets method to 'OLS' if no method argument and n > p
32     else
33         method = 'PLS'; % sets method to 'PLS' if no method argument and n =< p
34     end
35 end
36
37 if strcmp(method, 'OLS') == 1      % checks method
38     Ghat = X*((X'*X)\(X'*Gd));     % OLS regression predictions
39 elseif strcmp(method, 'PCA') == 1 % checks method
40     beta = PCR2LOOCV(X, Gd);       % PCR coefficients
41     Ghat = [ones(n, 1), X]*beta;  % PCR predictions
42 elseif strcmp(method, 'PLS') == 1 % checks method
43     beta = PLS2fastLOOCV(X, Gd);  % PLS regression coefficients
44     Ghat = X*beta;                 % PLSR predictions
45 end
46
47 [~, Ghat] = max(Ghat, [], 2);      % Model classifications
48 for i = 1:n
49     j = G(i);                          % actual class
50     k = Ghat(i);                       % model prediction
51     confmat(j, k) = confmat(j, k) + 1; % updates confmat
52 end
53 pcc = trace(confmat)/n; % fraction of correct classifications
```

A.2.2 LDA_Eu

```
function [Ghat, confmat, pcc] = LDA_Eu(X, G)

% Function doing linear discriminant analysis based on squared Euclidean
% distances.
%
% Inputs
% ------
% X: Data matrix of n measurements (rows) and p predictor variables
%    (columns).
% G: Group membership column vector.
%
% Outputs
% -------
% Ghat:    Model classifications of measurements in X.
% confmat: Confusion matrix.
% pcc:     Fraction of correct classifications.

n       = size(X, 1);       % # of measurements
K       = max(G);           % # of classes
d2      = zeros(n, K);      % init of d2
confmat = zeros(K);         % init of confmat
Gd      = dummyvar(G);      % dummy coding of G
muG     = (Gd'*Gd)\Gd'*X;   % class centres

for i = 1:n
    for c = 1:K
        d2(i, c) = (X(i, :) - muG(c, :))*(X(i, :) - muG(c, :))'; % Eucl. dist ^2
    end
end

[~, Ghat] = min(d2, [], 2); % model predictions

for i = 1:n
    j = G(i);                           % actual class
    k = Ghat(i);                        % predicted class
    confmat(j, k) = confmat(j, k) + 1; % updates confmat
end

pcc = trace(confmat)/n; % fraction of correct classifications
```

A.2.3 LDA_M

```
function [Ghat, confmat, pcc] = LDA_M(X, G)

% Function doing linear discriminant analysis based on squared Mahalanobis
% distances.
%
% Inputs
% ------
% X: Data matrix of n measurements (rows) and p predictor variables
%    (columns).
% G: Group membership column vector.
%
% Outputs
% -------
% Ghat:    Model classifications of measurements in X.
% confmat: Confusion matrix.
% pcc:     Fraction of correct classifications.

n       = size(X, 1);      % # of measurements
K       = max(G);          % # of classes
d2      = zeros(n, K);     % init of d2
confmat = zeros(K);        % init of confusion matrix
Gd      = dummyvar(G);     % dummy coding of G
muG     = (Gd'*Gd)\Gd'*X;  % class means
Xc      = X - muG(G, :);   % class means centered data matrix
Sinv    = pinv(Xc'*Xc);    % scaled inverse of covariance matrix

for i = 1:n
    for c = 1:K
        d2(i, c) = (X(i, :) - muG(c, :))*Sinv*(X(i, :) - muG(c, :))'; % Mah dist ^2
    end
end

[~, Ghat] = min(d2, [], 2); % model predictions

for i = 1:n
    j = G(i);                          % actual class
    k = Ghat(i);                       % predicted class
    confmat(j, k) = confmat(j, k) + 1; % updates confmat
end

pcc = trace(confmat)/n; % fraction of correct classifications
```

A.2.4 TLDA_LOOCV

```
function [Ghat, GhatCV, d2CV, muG, W, ncc, pcc, confmat] = TLDA_LOOCV(X, G, lambdas, d)

% Function doing d-type Tikhonov regularized LDA of data matrix X and group
% membership vector G over an array of regularization parameter values
% lambdas. Does LOOCV and chooses the model that has the lowest sum of
% Mahalanobis distances from correct class centers from the models that has
% the maximum amount of correct classifications.
%
% This function is a modification of Ulf Indahl's LDA function of March
% 2015 for the MATH310 course at the Norwegian University of Life Sciences.
%
% Inputs
% ------
% X:       Sample matrix of n samples (rows) and p variables (columns).
%          Assumed to contain samples from all groups.
% G:       Vector containing numeric group membership for each sample in X
% lambdas: Array of regularization parameter values. Should be
%          log10-distributed around 1.
% d:       Regulariztaion type. Does Ridge regularization if set to 0,
%          otherwise Tikhonov regularization of nth order derivative if set
%          to n > 0.
%
% Outputs
% -------
% Ghat:    Class predictions for all lambda values
% GhatCV:  CV class predictions for all lambda values
% d2CV:    CV squared Mahalanobis distances
% muG:     Class centers
% W:       Sphering matrix for best choice of regularization parameter
%          lambda
% ncc:     # of correct CV classifications
% pcc:     Fraction of CV correct classifications
% confmat: Confusion matrix for CV predictions


% Initializations
[n, p]  = size(X);          % sample size
K       = max(G);           % # of classes
q       = length(lambdas); % # of regularization parameter values
d2      = zeros(n, K);      % init of squared Mahalanobis distances
d2CV    = zeros(n, K, q);  % init of squared CV Mahalanobis distances
Ghat    = zeros(n, q);      % init of model predictions
GhatCV  = zeros(n, q);      % init of CV model predictions
dist    = zeros(1, q);      % init of winning distances norms
confmat = zeros(K);         % init of confusion matrix
```

```
W       = cell(1, q);       % init of sphering matrices


% Regularization of data
if d > 0
    T = diff([speye(p); sparse(d, p)], d); % regularization matrix
    X = X/T;                               % T^{-1} transformed data
end

% Group centering and SVD
Gd          = dummyvar(G);                          % dummy coding of G
muG         = (Gd'*Gd)\Gd'*X;                       % class centres
Xc          = X - muG(G, :);                        % class centered matrix
[~, S, V] = svd(Xc, 'econ');                        % SVD
s           = diag(S);                              % undigaonalisation of S
D           = sqrt(bsxfun(@plus, s.^2, lambdas)); % updated SV's

for c = 1:K
    comp = (speye(p) - V(:, 1:n - K + c - 1)*V(:, 1:n - K + c - 1)')*muG(c, :)'; % orth comp.
    V(:, n - K + c) = comp/norm(comp); % updates V
end

% The LDA: Mahalanobis distances and group membership predictions
for k = 1:q
    W{k} = bsxfun(@rdivide, V, D(:, k)'); % sphering matrix
    for i = 1:n
        for c = 1:K
            d2(i, c) = (X(i, :) - muG(c, :))*W{k}*W{k}'*(X(i, :) - muG(c, :))'; % Mah dist^2
        end
    end
    [~, Ghat(:, k)] = min(d2, [], 2); % model predictions for each lambda
end

% LOOCV

for k = 1:q
    for i = 1:n
        inds            = setdiff(1:n, i);              % leaves out ith index
        Xred            = X(inds, :); Gred = G(inds);   % reduced data
        [Wred, muGred] = LDA_int(Xred, Gred, lambdas); % reduced W and means
        for c = 1:K
            d2CV(i, c, k) = norm((X(i, :) - muGred(c, :))*Wred{k}).^2; % CV Mah dist^2
        end
        d2CV(i, :, k) = d2CV(i, :, k)/sum(d2CV(i, :, k)); % Realtive distance
    end
    [windist, GhatCV(:, k)] = min(d2CV(:, :, k), [], 2); % winning dist's and pred's
    dist(k)                 = norm(windist); % norm of winning dist's
```

```
93      ncc                         = sum(G == GhatCV(:, k));
94  end
95
96  pcc = ncc/n;
97  inds = find(ncc == max(ncc));
98  ranking = pcc(inds) - dist(inds);
99  [~, idx] = max(ranking);
100 GhatCV = GhatCV(:, inds(idx));
101 W = W{inds(idx)};
102
103 for i = 1:n
104     j = G(i);                           % actual class membership
105     k = GhatCV(i);                      % CV predicted class membership
106     confmat(j, k) = confmat(j, k) + 1; % updates confmat
107 end
108
109 ncc = trace(confmat); % # of correct classifications
110 pcc = ncc/n;          % fraction of correct classifications
111
112
113
114
115 function [Wred, muGred] = LDA_int(X, G, lambdas)
116 [n, p] = size(X);
117 K = max(G);
118 q = length(lambdas);
119 Gd = dummyvar(G);
120 Wred = cell(q, 1);
121 muGred = (Gd'*Gd)\Gd'*X;
122 Xc = X - muGred(G, :);
123 [~, S, V] = svd(Xc, 'econ');
124 s = diag(S);
125 D = sqrt(bsxfun(@plus, s.^2, lambdas));
126 for c = 1:K
127     comp = (speye(p) - V(:, 1:n - K + c - 1)*V(:, 1:n - K + c - 1)')*muGred(c, :)';
128     V(:, n - K + c) = comp/norm(comp);
129 end
130 for k = 1:q
131     Wred{k} = bsxfun(@rdivide, V, D(:, k)');
132 end
```

A.2.5 TLDAfastLOOCV

```
1  function [muG, W, Ghat, GhatCV, dist,...
2      ncc_all, pcc_all, lambda, ncc, pcc, confmat] = TLDAfastLOOCV(X, G, lambdas, d)
3  % Function doing Tikhonov regularized LDA of sample matrix X and group
4  % membership vector G over an array of regularization parameter values
5  % lambdas.
6  %
7  % This function is an modification of Ulf Indahl's LDA function of March
8  % 2015 for the MATH310 Course at the Norwegian University of Life Sciences.
9  %
10 % Inputs:
11 % ------------------------------------------------------------------------
12 % X       - Sample matrix with each row as individual samples, each column
13 %           represents a variable. X is assumed to contain measurements
14 %           for all groups.
15 % G       - Vector containing numeric group memberships for each sample in
16 %           X.
17 % lambdas - Array of regularization parameter values. Generally
18 %           log10-distributed.
19 % d       - Type of regularization. For non-negative integer values, does
20 %           the corresponding derivative regularization. Gives the zeroth
21 %           (Ridge) derivative regularization if = 0.
22 %
23 % Outputs:
24 % ------------------------------------------------------------------------
25 % Ghat    - Predicted group membership based on Mahalanobis distances from
26 %           group means.
27 % GhatCV  - LOOCV predicted group memberships based on Mahalanobis
28 %           distances from group means.
29 % dist    - Vector containing the norm of relative Mahalanobis distances
30 %           for each lambda.
31 % ncc_all - # of correct LOOCV classifications for each lambda.
32 % pcc_all - % of correct LOOCV classifications for each lambda.
33 % lambda  - The "winning" lambda value based on difference in % correct
34 %           LOOCV classifications and dist value.
35 % ncc     - # of correct LOOCV classifications for winning lambda
36 % pcc     - % of correct LOOCV classifications for winning lambda
37 % confmat - Confusion matrix for the analysis. Rows are actual group
38 %           memberships, columns are LOOCV predicted.
39 % ------------------------------------------------------------------------
40
41 % Initializations
42
43 [n, p]  = size(X);          % Size of data matrix
44 K       = max(G);           % # of groups
45 q       = length(lambdas); % # of lambda values
```

```
46  d2       = zeros(n, K);      % Init of squared Mahalanobis distances
47  d2CV     = zeros(n, K);      % Init of squared LOOCV Mahalanobis distances
48  Ghat     = zeros(n, q);      % Init of Ghat
49  GhatCV   = zeros(n, q);      % Init of GhatCV
50  dist     = zeros(1, q);      % Init of dist
51  ncc_all  = zeros(1, q);      % Init of ncc_all
52  confmat  = zeros(K);         % Init of confmat
53  W        = cell(q, 1);       % Init of W
54
55  % Regularization of data
56  if d > 0
57      T = diff([speye(p);sparse(d, p)], d); % Regularization matrix
58      X = X/T;                              % T^{-1}-transformed data
59  end
60
61  % Group centering of data and SVD
62  Gd          = dummyvar(G);                % Dummy coding of group memberships
63  nG          = sum(Gd)';                   % # of values for each group
64  muG         = (Gd'*Gd)\Gd'*X;             % Group centres
65  Xc          = X - muG(G, :);              % Data centred wrt group memberships
66  [~, S, V] = svd(Xc, 'econ');              % Economical SVD of group centred data
67  s = diag(S);                              % Diagonalization of S
68  D = sqrt(bsxfun(@plus, s.^2, lambdas)); % Matrix containing lambda-corrected SVs
69
70  if K > 2
71      D = sparse(D); % Making D sparse if more than 2 groups
72  end
73
74  % The LDA: Mahalanobis distances and group membership predictions
75  for k = 1:q
76      W{k} = V/diag(D(:, k)); % Sphering matrix
77      for i = 1:n
78          for c = 1:K
79              d2(i, c) = norm((X(i, :) - muG(c, :))*W{k})^2; % Squared Mahalanobis distances
80          end
81      end
82      [~, Ghat(:, k)] = min(d2, [], 2); % Predicted group memberships
83      % LOOCV squared Mahalanobis distances (Ripley, p. 100 [*]):
84      for i = 1:n
85          g  = G(i);      % Group membership for jth sample
86          nc = nG(g);     % # of samples belonging to group c
87          xc = Xc(i, :); % jth sample group centred wrt correct group
88          for c = 1:K
89              if c == g
90                  % Ripley's Mahalanobis distance to correct group mean:
91                  d2CV(i, g) = d2(i, g)*(nc/(nc - 1))^2/...
92                  (1 - (nc/(nc - 1))*d2(i, g));
```

```
93                else
94                    xk = X(i, :) - muG(c, :); % jth sample group centered wrt incorrect group
95                    % Ripley's Mahalanobis distance to incorrect group mean:
96                    d2CV(i, c) = d2(i, c)*(1 + ((xc*W{k})*...
97                    (xk*W{k})')^2/...
98                    (((nc - 1)/nc - d2(i, g))*d2(i, c)));
99                end
100           end
101           d2CV(i, :) = d2CV(i, :)/sum(d2CV(i, :)); % Realtive distance
102       end
103       [d2CVw, GhatCV(:, k)] = min(d2CV, [], 2); % Winning d2CV distance + predictions
104       dist(k) = norm(d2CVw);                    % Norm of relative LOOCV Mahalanobis distance
105       ncc_all(k) = sum(G == GhatCV(:, k));      % # of correct LOOCV predictions
106   end
107
108   pcc_all  = ncc_all/n;                  % % of correct LOOCV predictions
109   ncc      = max(ncc_all);               % Max # of correct LOOCV predictions
110   pcc      = ncc/n;                      % Max % of correct LOOCV predictions
111   inds     = find(ncc_all == ncc);       % Lambda indices with max correct predictions
112   ranking  = pcc_all(inds) - dist(inds); % Ranking parameter
113   [~, idx] = max(ranking);               % Index of inds with highest ranking
114   lambda   = lambdas(inds(idx));         % winning lambda
115
116   % Confusion matrix
117
118   for k = 1:n
119       i = G(k);                          % Actual group membership for kth sample
120       j = GhatCV(k, inds(idx));          % Predicted group membership
121       confmat(i,j) = confmat(i,j)+1; % Updates confmat accordingly
122   end
123
124   W = W{inds(idx)};
125
126   % References:
127
128   % [*] Ripley, B. D.: "Pattern Recognition and Neural Networks". Cambridge
129   %     University Press, 1996.
```

A.2.6 QDA

```
function [Ghat, pclass, a, covi, muG, confmat, pcc] = QDA(X, G)

% Function doing quadratic discriminant analysis on a data matrix X with
% corresponding class membership vector G.
%
% Inputs
% ------
% X: Data matrix consisting of n measurements (rows) and p variable
%    (columns)
% G: Column vector of class memberships for each row in X.
%
% Outputs
% -------
% Ghat:    Predicted class memberships.
% plcass:  Predicted class membership probabilities.
% a:       Coefficients for calculating probabilities.
%          = (2*pi)^(p/2)*sqrt(det(Cov)), where Cov is the class covariance
%          matrix. Needed for prediction.
% covi:    Inverses of covariance matrices for each class. Needed for
%          prediction.
% muG:     Class centres. Needed for prediction.
% confmat: Confusion matrix.
% pcc:     Fraction of correct classifications

% Initializations
[n, p]  = size(X);          % sample size
K       = max(G);           % # of classes
Gd      = dummyvar(G);      % dummy coding of G
nG      = sum(Gd);          % # of measurements for each class
muG     = (Gd'*Gd)\Gd'*X;   % class centres
pclass  = zeros(n, K);      % init of predicted class membership probabilities
a       = zeros(K, 1);      % init of a
d2      = zeros(n, K);      % init of squared Mahalanobis distances matrix
confmat = zeros(K);         % init of confusion matrix
covi    = cell(K, 1);       % init of covi


for c = 1:K
    Xc      = X(G(:) == c, :);              % data belonging to class c
    Xc      = Xc - ones(nG(c), 1)*muG(c, :); % centered data for class c
    Cov     = 1/(nG(c) - 1)*(Xc'*Xc);       % estimated cov matrix for class c
    a(c)    = (2*pi)^(p/2)*sqrt(det(Cov));  % coefficient needed for probabilities
    covi{c} = pinv(Cov);                    % (pseudo)inverse of cov matrix
end
```

```
46 for i = 1:n
47     for c = 1:K
48         xbar          = X(i, :) - muG(c, :);        % class centered data point
49         d2(i, c)      = xbar*covi{c}*xbar';         % squared Mahalanobis distance
50         pclass(i, c) = 1/a(c)*exp(-(d2(i, c))/2); % class membership probability
51     end
52 end
53 pclass     = bsxfun(@rdivide, pclass, sum(pclass, 2)); % making sums of probs = 1
54 [~, Ghat] = max(pclass, [], 2);                        % class predictions
55
56 for i = 1:n
57     j = G(i);                              % actual class membership
58     k = Ghat(i);                           % model predicted class membership
59     confmat(j, k) = confmat(j, k) + 1; % updates covariance matrix
60 end
61
62 pcc = trace(confmat)/n; % fraction of correct classifications
```

A.2.7 logQDAsvd

```
1  function [Ghat, pclass, a, Ws, muG, confmat, pcc] = logQDAsvd(X, G)
2
3  % Function doing quadratic discriminant analysis on a data matrix X with
4  % corresponding class membership vector G.
5  %
6  % Inputs
7  % ------
8  % X: Data matrix consisting of n measurements (rows) and p variable
9  %    (columns)
10 % G: Column vector of class memberships for each row in X.
11 %
12 % Outputs
13 % -------
14 % Ghat:    Predicted class memberships.
15 % plcass:  Predicted class membership probabilities.
16 % a:       Coefficients for calculating probabilities.
17 %          = ((nG(c) - 1)/(2*pi))^(p/2)*prod(diag(S))^(-1), where S is the
18 %          contains the singular values for class c. Needed for prediction.
19 % Ws:      Sphering matrices for each class. Needed for prediction.
20 % muG:     Class centres. Needed for prediction.
21 % confmat: Confusion matrix.
22 % pcc:     Fraction of correct classifications
23
24 % Initializations
25 [n, p]    = size(X);          % sample size
26 K          = max(G);           % # of classes
```

```
27 Gd          = dummyvar(G);      % dummy coding of G
28 nG          = sum(Gd);          % # of measurements for each class
29 muG         = (Gd'*Gd)\Gd'*X;   % class centres
30 logpclass = zeros(n, K);       % init of predicted class membership probabilities
31 a           = zeros(K, 1);      % init of a
32 d2          = zeros(n, K);      % init of squared Mahalanobis distances matrix
33 confmat     = zeros(K);         % init of confusion matrix
34 Ws          = cell(K, 1);       % init of Ws
35
36 for c = 1:K
37     Xc       = X(G(:) == c, :);                % data belonging to class c
38     Xc       = Xc - ones(nG(c), 1)*muG(c, :); % centered data for class c
39     [~, S, V] = svd(Xc, 'econ');               % SVD of data in class c
40     Ws{c} = sqrt(nG(c) - 1)*V/S;               % sphering matrix
41     a(c) = p/2*log((nG(c) - 1)/(2*pi)) - sum(log(diag(S))); % coeff for pred
42 end
43
44 for i = 1:n
45     for c = 1:K
46         xbar          = X(i, :) - muG(c, :);      % class centered data point
47         d2(i, c)      = norm(xbar*Ws{c}).^2;      % squared Mahalanobis distance
48         logpclass(i, c) = a(c) - (d2(i, c))/2;  % log class membership probability
49     end
50 end
51
52 pclass     = exp(logpclass);                              % getting positive values
53 pclass     = bsxfun(@rdivide, pclass, sum(pclass, 2)); % making sums of probs = 1
54 [~, Ghat] = max(pclass, [], 2);                          % class predictions
55
56 for i = 1:n
57     j = G(i);                           % actual class membership
58     k = Ghat(i);                        % model predicted class membership
59     confmat(j, k) = confmat(j, k) + 1; % updates covariance matrix
60 end
61
62 pcc = trace(confmat)/n; % fraction of correct classifications
```

A.2.8 TQDAfastLOOCV

```
1  function [pclass, pclassCV, GhatCV, a, W, muG, confmat, pcc, lambda] = TQDAfastLOOCV(X, G, lambdas, d)
2  % Attempt of fast LOOCV Tikhonov regularized QDA.
3
4  % Inputs
5  % ------
6  % X:       Data matrix containing n measurements (rows) of p variables
7  %          (columns).
8  % G:       Corresponding class membership vector.
9  % lambdas: Array of log distributed regularization parameter values.
10 %          lambdas = 0 gives no regularization.
11 % d:       Order of differentiation. d = 0 gives Ridge regularization
12
13 % Outputs
14 % -------
15 % pclass:   model probabilities for all measurements over all lambdas
16 % pclassCV: LOOCV probabilities for all measurements over all lambdas
17 % GhatCV:   LOOCV predictions for all measurements over all lambdas
18 % a:        coefficient for calculating log probabilites, needed for
19 %          prediction
20 % W:        winning sphering matrices, needed for prediction
21 % muG:      class centres, needed for prediction
22 % confmat:  LOOCV confusion matrix
23 % pcc:      fractions of correct classifications for all lambdas
24 % lambda:   winning lambda value
25
26 % Initializations
27 [n, p]      = size(X);          % sample size
28 q           = length(lambdas); % # of lambda values
29 K           = max(G);          % # of classes
30 Gd          = dummyvar(G);     % dummy coding of G
31 nG          = sum(Gd);         % # of measurements for each class
32 muG         = (Gd'*Gd)\Gd'*X;  % class centres
33 GhatCV      = zeros(n, q);     % init of LOOCV predictions
34 pclass      = zeros(n, K, q);  % init of probabilities tensor
35 pclassCV    = zeros(n, K, q);  % init of LOOCV probabilities tensor
36 logpclass   = zeros(n, K);     % init of predicted class membership probabilities
37 logpclassCV = zeros(n, K, q);  % init of LOOCV log probabilities
38 pclassCVw   = zeros(q, 1);     % init of winning probabilities
39 ncc         = zeros(q, 1);     % init of ncc
40 a           = zeros(K, q);     % init of a
41 d2          = zeros(n, K);     % init of squared Mahalanobis distances matrix
42 confmat     = zeros(K);        % init of confusion matrix
43 Ws          = cell(K, q);      % init of Ws
44 W           = cell(K, 1);      % init of winning W matrices
45
```

```
46 if d > 0
47     T = diff([speye(p);sparse(d, p)], d); % regularization matrix
48     X = X/T;                              % T^{-1}-transformed data
49 end
50
51 for c = 1:K
52     Xc      = X(G(:) == c, :);             % data belonging to class c
53     Xc      = Xc - ones(nG(c), 1)*muG(c, :); % centered data for class c
54     [~, S, V] = svd(Xc, 'econ');           % SVD of data in class c
55     dimS = length(diag(S));                % dimension of S
56     s = diag(S);                           % vecorization
57     S2 = s.^2;                             % squared s
58     D2 = bsxfun(@plus, S2, lambdas);       % lambda updated squared sv's
59     trD2 = sum(D2); trS2 = sum(S2);        % traces for re-adjusting total variance
60     D2 = bsxfun(@times, D2, trS2./trD2);   % D2 has the same total variance as S2
61     D = sqrt(D2);                          % new sv's
62     a(c, :) = dimS/2*log(nG(c) - 1) - p/2*log(2*pi) - sum(log(D)); % coeff for pred
63     for k = 1:q
64         Ws{c, k} = sqrt(nG(c) - 1)*bsxfun(@rdivide, V, D(:, k)');  % sphering matrix
65     end
66 end
67 for k = 1:q
68     for i = 1:n
69         for c = 1:K
70             xbar          = X(i, :) - muG(c, :);     % class centered data point
71             d2(i, c)      = norm(xbar*Ws{c, k}).^2;  % squared Mahalanobis distance
72             logpclass(i, c) = a(c) - (d2(i, c))/2;   % log class membership probability
73         end
74     end
75     pcl = exp(logpclass);                                  % probabilities
76     pclass(:, :, k) = bsxfun(@rdivide, pcl, sum(pcl, 2)); % probs now sum to 1
77
78     logpclassCV(:, :, k) = logpclass; % CV probs only change for group of omitted point
79     for i = 1:n
80         g = G(i);                                                    % group of X(i, :)
81         const = abs(1 - nG(g)/(nG(g) - 1)^2*d2(i, g));               % constant for LOOCV upd.
82         aCV = a(g, k) - p/2*(log((nG(g) - 1)/(nG(g) - 2)) + log(const)); % coeff for probs
83         d2CV = (d2(i, g)*nG(g)^2*(nG(g) - 2)/(nG(g) - 1)^3) / const;     % upd. Mah. dist
84         logpclassCV(i, g, k) = aCV - 0.5*d2CV;                       % CV prob
85     end
86     pcl = exp(logpclassCV(:, :, k));                          % probabilities
87     pclassCV(:, :, k) = bsxfun(@rdivide, pcl, sum(pcl, 2)); % probs now sum to 1
88     [pclw, GhatCV(:, k)] = max(pclassCV(:, :, k), [], 2);   % winning probs & preds
89     pclassCVw(k) = norm(pclw);                              % secondary discr criterium
90     ncc(k) = sum(G == GhatCV(:, k));                        % # of correct classifications
91 end
92 pcc      = ncc/n;                        % fraction of correct classifications
```

```
93  inds      = find(ncc == max(ncc));        % lambda indices for max corr class
94  ranking   = pcc(inds) + pclassCVw(inds); % ranking parameter to be maximized
95  [~, idx] = max(ranking);                  % inds index for best lambda
96  win       = inds(idx);                    % winning lambda index
97
98  for i = 1:n
99      j = G(i);                            % actual class membership
100     k = GhatCV(i);                       % model predicted class membership
101     confmat(j, k) = confmat(j, k) + 1; % updates covariance matrix
102 end
103
104 a = a(:, win);          % winning a's
105 lambda = lambdas(win); % winning lambda
106 for c = 1:K
107     W{c} = Ws{c, win}; % winning Ws matrices
108 end
```

A.2.9 class_numbers_svd

```
1   function [Ghat, V, pcc, confmat, Ghat_test, pcc_test, confmat_test] = class_numbers_svd(Xtrain, Gtrain,
2
3   % Function classifying handwritten digits based on SVD bases. Uses the
4   % same number of bases for all digits and chooses this number based on
5   % correct classifications of training data. Predicts a test set if one is
6   % given.
7   %
8   % Inputs
9   % ------
10  % Xtrain:  Matrix of n images (rows of stacked transposed pixel columns)
11  %          each consisting of p pixels/variables (colums)
12  % Gtrain:  Corresponding column vector of class memberships
13  % vs:      Maximum number of basis images considered
14  % (Xtest): Test data
15  % (Gtest): Test data class memberships
16  %
17  %
18  % Outputs
19  % -------
20  % Ghat:
21  % V:
22  % pcc:          Fraction of correct classifications for training data
23  % (Ghat_test): Test data prediction (if test data provided)
24  % (pcc_test):  Fraction of correct classifications of test data (if given)
25
26  % Initializations
27  [n, p]    = size(Xtrain);   % sample size
```

```
28 K          = max(Gtrain);     % # of classes
29 V          = cell(K, 1);      % init of collection of basis vector matrices
30 ncc        = zeros(vs, 1);     % init of correct classifications vector
31 res_sums2 = zeros(n, K, vs); % init of squared residuals tensor
32 Ghat       = zeros(n, vs);     % init of training data predictions matrix
33 confmat    = zeros(K);
34
35 for c = 1:K
36     A = Xtrain(Gtrain == c, :);     % data points for class c
37     [~, ~, V{c}] = svd(A, 'econ'); % basis image vector matrix
38     for j = 1:vs
39         comp = eye(p) - V{c}(:, 1:j)*V{c}(:, 1:j)'; % orth. comp of basis
40         res = Xtrain*comp';                          % residuals
41         res_sums2(:, c, j) = sum(res.^2, 2);         % sum of res.^2 over n
42     end
43 end
44
45 for j = 1:vs
46     [~, Ghat(:, j)] = min(res_sums2(:, :, j), [], 2); % training data pred's
47     ncc(j)          = sum(Gtrain == Ghat(:, j));       % # of corr. classifications
48 end
49
50 [~, bvs] = max(ncc); % max # of correct classifications & # of basis vectors needed
51 pcc        = ncc/n;     % fractions of correct classifications
52
53 Ghat = Ghat(:, bvs);
54
55
56 for c = 1:K
57     V{c} = V{c}(:, 1:bvs); % V matrices updated to contain bvs column vectors
58 end
59
60 for i = 1:n
61     j = Gtrain(i);
62     k = Ghat(i);
63     confmat(j, k) = confmat(j, k) + 1;
64 end
65
66
67 if nargin > 3 % i.e. if test set is given
68     confmat_test = zeros(K);
69     n          = size(Xtest, 1); % # of samples in test set
70     res_sums2 = zeros(n, K);     % new init of res_sums2
71     for c = 1:K
72         comp             = eye(p) - V{c}*V{c}'; % orthogonal complement of V{c}
73         res              = Xtest*comp';         % residuals
74         res_sums2(:, c) = sum(res.^2, 2);      % sum of squared residuals
```

```
75      end
76      [~, Ghat_test] = min(res_sums2, [], 2);      % test set predictions
77      pcc_test       = sum(Ghat_test == Gtest)/n; % fraction of corr. class. for test data
78      for i = 1:n
79          j = Gtest(i);
80          k = Ghat_test(i);
81          confmat_test(j, k) = confmat_test(j, k) + 1;
82      end
83  end
```

A.2.10 class_rand_cont_Ridge

```
1   function [Beta, pcc, confmat, Ghat_test, pcc_test] = class_rand_cont_Ridge(Xtrain, Gtrain, Rdim, lambdas
2
3   % Function doing image classifications of a data matrix X consisting of n
4   % vectorized images (trasposed stacked pixel columns) as rows and a
5   % corresponding class membership column vector G. Also predicts a test set
6   % if one is given. Adds Rdim randomly generated features and does a full
7   % LOOCV multivariate regression classification. Does a Ridge regularization
8   % if a lambda array is given. lambda = 0 gives no regularization. Predicts
9   % test set if given. Calls the functions TregsMulti and (or)
10  % OLS_fastLOOCV_SVD.
11  %
12  % Inputs
13  % ------
14  % Xtrain:  Data matrix consisting of n vectorized images (transposed
15  %          stacked pixel columns) of p pixels/variables
16  % Gtrain:  Corresponding class membership column vector
17  % Rdim:    # of randomly generatet features added
18  % lambdas: Array of perferrably log-distributed regularization parameter
19  %          values.
20  % (Xtest): Optional test set data matrix
21  % (Gtest): Optional test set class membership vector
22  %
23  % Outputs
24  % -------
25  % GhatCV:      LOOCV predictions of training data
26  % pcc:         Fraction of correct classifications for LOOCV model (and
27  %              test data)
28  % confmat:     Confusion matrices for LOOCV model of training data and test
29  %              data predictions if applicable
30  % Ghat_testCV: Predictions of test data
31
32  % Initializations
33  p     = size(Xtrain, 2);      % sample size
34  K          = max(Gtrain);       % # of classes
```

```
35
36
37
38 R           = randi([0, 1], p, Rdim); % random matrix of 0 and 1 entries
39 R(R == 0)   = -1;                     % setting zero entries in R to -1
40 XR          = Xtrain*R;               % rows of 1000 randomly generated features
41 XR(XR < 0)  = 0;                      % setting negative feature vaules to 0
42 Xaug        = [Xtrain, XR];           % augmented data matrix
43
44
45 [Beta, ~, pcc, confmat] = TregsMultiClass(Xaug, Gtrain, lambdas, 0, 0); % Ridge reg coeffs
46
47 if nargin == 6
48     n                = size(Xtest, 1);                % # of images in test set
49     confmat_test     = zeros(K);                      % init of confmat_test
50     XR               = Xtest*R;                       % rows of randomly gen. feat's
51     XR(XR < 0)       = 0;                             % setting negative entries = 0
52     GhatCVs          = [ones(n, 1), Xtest, XR]*Beta; % test set reg predictions
53     [~, Ghat_test] = max(GhatCVs, [], 2);           % test set classifications
54     for i = 1:n
55         j                  = Gtest(i);               % actual class membership
56         k                  = Ghat_test(i);           % predicted class membership
57         confmat_test(j, k) = confmat_test(j, k) + 1; % updates confmat_test
58     end
59     pcc_test     = trace(confmat_test)/n; % fractions of correct classifications
60     confmat = {confmat, confmat_test};       % confusion matrices
61 end
```

Bibliografi

[1] Adams, D.: *The Hitch Hiker's Guide to the Galaxy.* Harmony Books, New York, 1980.

[2] Allen, D. M.: *The Relationship between Variable Selection and Data Augmentation and a Method for Prediction.* Technometrics, 16:125 – 127, 1974.

[3] Bickel, P. J. og Doksum, K. A.: *Mathematical Statistics: Basic Ideas and Selected Topics*, bind 1. Holden-Day, California, 1977.

[4] Boyd, S. og Vandenberghe, L.: *Vectors, Matrices, and Least Squares - Rough Draft (December 9th, 2015).* Lastet ned fra `http://stanford.edu/class/ee103/mma.pdf` 17.04.2015.

[5] Brown, P. J.: *Wavelength selection in multicomponent near-infrared calibration.* Journal of Chemometrics, 6(3):151–161, mai/juni 1992.

[6] Databank, The FDA NCI Clinical Proteomics Program: *Ovarian Cancer.* `https://home.ccr.cancer.gov/ncifdaproteomics/ppatterns.asp`.

[7] Efron, B.; Hastie, T.; Johnstone, I. og Tibshirani, R.: *Least Angle Regression.* The Annals of Statistics, 32(2):407 – 499, 2004.

[8] Eldén, Lars: *Matrix Methods in Data Mining and Pattern Recognition.* Society for Industrial and Applied Mathematics, 2007.

[9] Fisher, R. A.: *The Use of Multiple Measurements in Taxonomic Problems.* Annals of Eugenics, 7(2):179–188, 1936.

[10] Golub, G. H.; Heath, M. og Wahba, G.: *Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter.* Technometrics, 21(2):215 – 223, mai 1979.

[11] Hansen, P. C.; Pereyra, V. og Scherer G.: *Least Squares Data Fitting with Applications*, side 28. The John Hopkins University Press, Baltimore, 2013.

[12] Hastie, T.; Tibshirani, R og Friedman, J.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, andre utgave, 2009.

[13] Hull, J. J.: *A Database for Handwritten Text Recognition Research.* IEEE PAMI, 16(5):550–554, 1994.

[14] Indahl, U. G.: *Some comments on variants of linear least squares regression methods: PCR, Rigde Regression and PLS.* Notes 11 - MATH310 vår 2015 NMBU.

[15] Indahl, U.; Sahni, N. S.; Kirkhus, B. og Næs, T.: *Multivariate strategies for classification based on NIR-spectra – with application to mayonnaise.* Chemometrics and Intelligent Laboratory Systems, 49:19 – 31, 1999.

[16] Kalivas, J. H.: *Two Data Sets of Near Infrared Spectra.* Chemometrics and Intelligent Laboratory systems, 37:255 – 259, 1997.

[17] Lay, D.: *Linear Algebra and Its Applications.* Pearson, 2014.

[18] LeCun, Y.; Cortes, C. og Burges, C.: *The MNIST handwritten digits database.* `http://yann.lecun.com/exdb/mnist/`, lest 20.04.2016.

[19] Mahalanobis, P. C.: *On the Generalized Distance in Statistics.* Proceedings of the National Institute of Sciences of India, 2(1):49–55, 1936.

[20] Marks, S. og Dunn, O. J.: *Discriminant functions when covariance matrices are unequal.* Journal of the American Statistical Association, (69):555 – 599, 194.

[21] Mathworks: *ridge.* `http://se.mathworks.com/help/stats/ridge.html`.

[22] Næs, T.; Tomic, O.; Afseth, N. K.; Segtnan, V. og Måge, I.: *Multi-block regression based on combinations of orthogonalisation, PLS-regression and canonical correlation analysis.* Chemometrics and Intelligent Laboratory Systems, 124:32 – 42, Mai 2013.

[23] Osborne, B. G.; Fearn, T.; Miller, A. R. og Douglas, S.: *Application of near infrared reflectance spectroscopy to the compositional analysis of biscuits and biscuit doughs*. Journal of the Science of Food and Agriculture, 35(1):99 – 105, januar 1984.

[24] Ripley, B. D.: *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[25] Wold, S.; Sjöström, L. og Eriksson, L.: *PLS-regression: a basic tool of chemometrics*. Chemometrics and Intelligent Laboratory Systems, (58):109 – 130, 2001.

[26] Woodbury, M. A.: *Inverting Modified Matrices*. Memorandum Rept. 42, Statistical Research Group, Princeton University, Princeton, NJ, 1950.

Norges miljø- og
biovitenskapelige
universitet

Postboks 5003
NO-1432 Ås
67 23 00 00
www.nmbu.no