

**TTT4275 - Estimering,  
deteksjon og klassifisering**  
**Prosjekt**  
**Gruppe 49**

**Sander Bakke**  
*sandebs@stud.ntnu.no*

**Kolbjørn Bølgen**  
*kolbjosk@stud.ntnu.no*

Vår 2024

---

## Sammendrag

Denne rapporten er et svar på et klassifiseringsprosjekt i faget TTT4275 på NTNU. Prosjektet inneholder to ulike klassifiseringsproblemer, som igjen består av to oppgaver. Det første problemet er et typisk klassifiseringsprosjekt, nemlig klassifisering av de tre underartene til Iris blomsten, *Setosa*, *Versicolor* og *Virginica*. For å gjøre dette er det konstruert en Lineært Diskriminerende Klassifikator (*LDK*) ved to ulike metoder; *MSE*-minimering og Matlab-funksjonen *fitdiscr*. Begge metoder fungerer og gir en *error rate* på 3.3%. Modellene er videre testet ved en annen fordeling av trenings-/testdata, noe som ved begge metoder ga 0% *error rate*.

Videre er *fitdiscr*-funksjonen brukt til å undersøke tyngden av påvirkningen til underartenes blad-egenskaper. Her er det ved å kun bruke ett av trekkene, *petal width*, oppnådd en *error rate* på 3.3%, som i de fleste tilfeller er akseptabelt lavt, og minimerer antall utregninger i prosesseringen betraktelig.

Det er funnet at bruk av *fitdiscr* er å foretrekke over *MSE*, da det minker analytisk arbeid og prosesseringstid, og gir lik *error rate* som *MSE*.

Det andre problemet tar for seg klassifisering av håndskrevne tall, hentet fra en MNIST database. Til dette er det brukt **NN**- og **K-NN** algoritmer både med og uten gruppering, basert på Euklidisk avstand. Den beste modellen her m.h.t. *error rate* og opptreningsstid, var en kmeans-grupperings algoritme bestående av 2048 grupper og med **K=1**, en ikke-veiledet maskinlæringsmodell. Denne ga en *error rate* på 2.84%, med opplæringstid på 36 sekunder, mens en veiledet modell hadde *error rate* på 3.09% og opplæringstid på rundt 17 minutter. Observasjoner viser at selv om gruppering reduserer opplæringstiden, fører en økning i **K** til dårligere ytelse.

Tallet 1 ble feilklassifisert minst og 8 ble feilklassifisert oftest, tallet 0 forekommer minst og tallet 9 forekommer oftest når det først skjer en feilklassifisering. Modellen for klassifisering av håndskrevne tall har forbedringspotensiale, da kun 2 av 5 tall som ble feilklassifisert er forståelig ved visuell analyse.

For begge klassifiseringsproblemer vil en utvidelse av treningsdataen kunne forbedre ytelsen, men potensielt på bekostning av opplæringstid avhengig av hvilken modell man bruker.

---

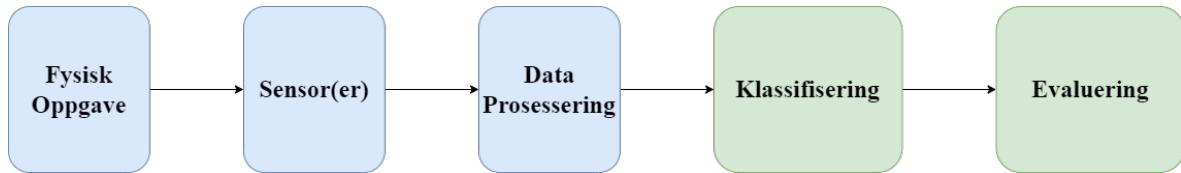
# Innhold

<b>1 Innledning</b>	<b>1</b>
<b>2 Maskinlæringsmetoder og Klassifisering</b>	<b>3</b>
2.1 Lineært Diskriminerende Klassifikator . . . . .	3
2.2 Tilpasning . . . . .	6
2.3 Malbasert Klassifikator . . . . .	6
2.4 Gruppering . . . . .	7
2.5 Evaluering . . . . .	9
<b>3 Oppgaver</b>	<b>11</b>
3.1 Iris Klassifisering . . . . .	11
3.2 Klassifisering av Håndskrevne Tall . . . . .	11
<b>4 Implementering og Resultater</b>	<b>13</b>
4.1 Iris Klassifisering . . . . .	13
4.2 Klassifisering av Håndskrevne Tall . . . . .	19
<b>5 Diskusjon</b>	<b>22</b>
<b>6 Konklusjon</b>	<b>24</b>
<b>Referanser</b>	<b>25</b>
<b>A Vedlegg</b>	<b>26</b>
A.1 Github . . . . .	26
A.2 Ekstra Figurer . . . . .	26

---

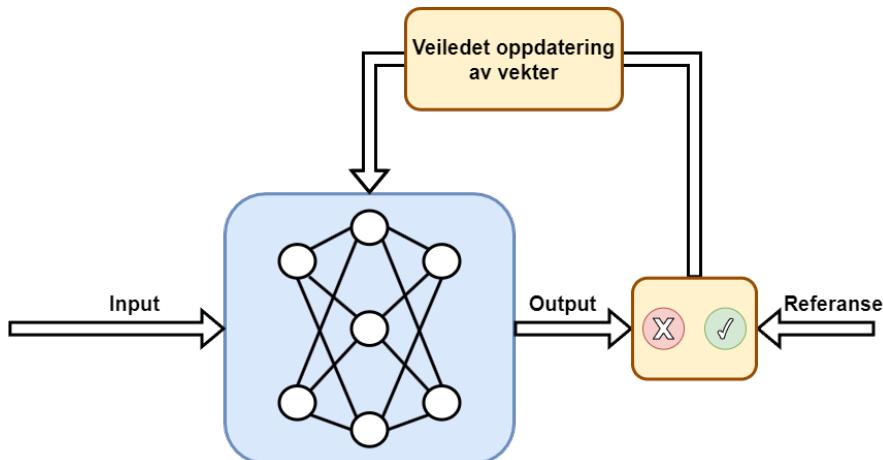
## 1 Innledning

Hensikten med dette prosjektet er å tilegne oss erfaring innen klassifisering og maskinlæring, samt oppnå en dyp forståelse for den underliggende teorien. I dette prosjektet tar vi for oss ett grunnleggende klassifiseringssystem, Figur 1, med fokus på klassifisering og evaluering. Systemet vil undergå både veiledet og ikke-veiledet maskinlæring, og vil ha som mål å redusere feilklassifiseringer, samtidig som modellene skal unngå unødvendig kompleksitet.



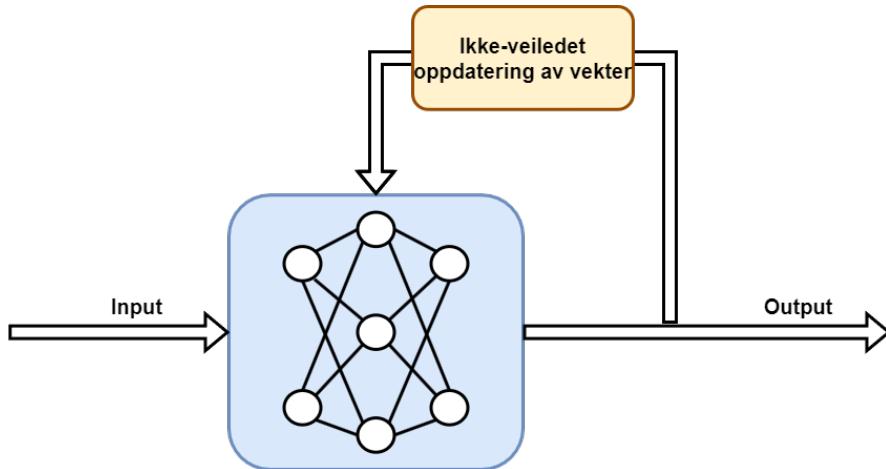
**Figur 1:** Grunnleggende klassifiseringssystem, basert på lignende figur [1][Fig. 3.3, s. 65].

Veiledet maskinlæringsmodeller vil i utgangspunktet baseres på både inputdata og tilhørende referanse (på engelsk *label*) eller klassifiseringer. Disse modellene jobber med å lære en funksjon (*vekter*) som kartlegger inputdata til ønskede utdata (*output*) basert på treningsdataene, som illustrert i Figur 2.



**Figur 2:** Veiledet maskinlæringsmodell, basert på lignende figur fra [4][Fig. 1.7, s. 31].

Ikke-veiledet maskinlæringsmodeller vil i utgangspunktet baseres kun på inputdata, uten noen form for referanse eller klassifiseringer, men heller et sett av «regler». Disse modellene jobber med å identifisere mønstre, grupperinger eller sammenhenger i dataene på egen hånd, som illustrert i Figur 3.



**Figur 3:** Ikke-veiledet maskinlæringsmodell, basert på lignende figur fra [4, Fig. 1.8, s. 32].

Begge disse variantene av maskinlæringsmodeller, har sine styrker og svakheter, samt anvendelser i dagens samfunn.

Veiledet maskinlæring har kraftig ytelse, gitt at man har tilstrekkelig mengde med data å trenne den opp på. Det blir brukt i bilde- og talegenkjenning og har vist seg å være svært effektivt i oppgaver som krever presis klassifisering eller prediksjon.

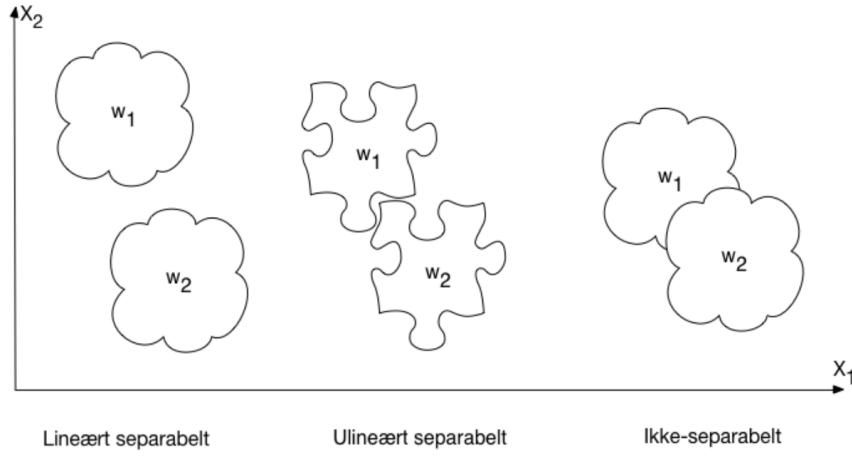
Ikke-veiledet maskinlæring vil være mer fleksibel, ettersom at det kan brukes ved små data-mengder. Disse er også utrolig gode til å finne uventede mønstre eller strukturer i data uten behov for å gi de et navn / etikett / *label*. Slike modeller kan brukes i alt fra markedsanalyse til kreftdiagnostikk.

Seksjon 2 gir en grundig beskrivelse av teknikker, metoder og nøkkelaspekter ved trening av både veiledede og ikke-veiledede maskinlæringsmodeller. I tillegg presenteres diverse problemstillinger i Seksjon 3, sammen med tilhørende løsninger og resultater i Seksjon 4. Disse resultatene vil bli ytterligere analysert og diskutert i Seksjon 5. Avslutningsvis vil de mest sentrale funnene og konklusjonene bli oppsummert i Seksjon 6.

## 2 Maskinlæringsmetoder og Klassifisering

Klassifisering går ut på å avgjøre hvilken klasse  $\omega$  et vilkårlig signal, måling eller annen type data  $x$  tilhører. Et system bestående av  $C$  antall klasser  $\omega_i$ ,  $i = 1, 2, \dots, C$ , vil avgjøre hvilken klasse dataen mest sannsynlig tilhører, noe som enten kan være en riktig eller feil avgjørelse, og man må dermed designe system som tar dette tilrette og estimerer systemytelsen [1, s. 65]. Dataen kan være fler- eller  $D$ -dimensjonal,  $x = [x_1, x_2, \dots, x_D]^T$ , og tilsvarende for klassene  $\omega_i = [\omega_{i1}, \omega_{i2}, \dots, \omega_{iD}]$ , noe som vil medføre økt kompleksitet og potensielt ytelse.

Avhengig av problemet kan man enten klassifisere dataen i lineære eller ikke-lineære klasser, samt separabel og ikke-separabel, som illustrert i Figur 4 for 2-dimensjonal data  $x = [x_1 \ x_2]^T$  og to klasser  $\omega_1$  og  $\omega_2$ . Hver av disse har forskjellig teoretisk optimal klassifikatorer.



**Figur 4:** Tre typer klassifisering, tatt fra [1, Fig. 3.4, s. 66].

### 2.1 Lineært Diskriminerende Klassifikator

For et lineært separabelt problem kan en lineær diskriminerende klassifikator (LDK) bli brukt, *linear discriminant classifier (LDC)* på engelsk, gitt sin simplisitet og akseptabel ytelse. I en slik klassifikator kan hver klasse  $\omega_i$  i en gruppe av klasser  $\omega_j$  bli beskrevet og avgjort av en diskriminerende funksjon  $g_i(x)$ , som beskrevet i likning (1), basert på [1, Eq. (3.6), s. 69].

$$x \in \omega_j \Leftrightarrow g_j(x) = \max_i \{g_i(x)\} \quad (1)$$

Den diskriminerende funksjonen  $g_i(x)$  er definert data og av den tilhørende klassen  $\omega_i$  samt dens medhørende avvik (offset)  $\omega_{io}$ , som beskrevet i likning (2), basert på [1, Eq. (3.7), s. 70].

$$g_i(x) = \omega_i^T x + \omega_{io} \quad (2)$$

Likning (2) kan forenkles til vektorform, som beskrevet i likning (4), med en vektermatrise  $\mathbf{W}$  og offset vektor  $\omega_o$  beskrevet i likning (3), som har dimensjoner  $C \times D$  og  $C \times 1$ .

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1D} \\ w_{21} & w_{22} & \cdots & w_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{C1} & w_{C2} & \cdots & w_{CD} \end{bmatrix} \quad \omega_o = \begin{bmatrix} \omega_{1o} \\ \omega_{2o} \\ \vdots \\ \omega_{Co} \end{bmatrix} \quad (3)$$

$$g(x) = \mathbf{W}x + \omega_o \quad (4)$$

For å trenne opp og senere bruke LDK-modellen trengs også en kostnadsfunksjon, som er med på å avgjøre når algoritmen gjør en feil, slik at den kan gjøre forbedringer underveis i opplæringen. For en linear klassifikator, kan *mean squared error (MSE)* kostnadsfunktionsen brukes, og ytterligere forenklinger kan gjøres på likning (4), ved å endre  $[\mathbf{W} \quad \omega_o] \rightarrow \mathbf{W}$  og  $[x^T \quad 1]^T \rightarrow x$  som vist i likning (5), slik at man ender opp med likning (6), [1, s. 77]. Her vil da vektormatrisen  $\mathbf{W}$  ha  $(D+1)$  kolonner, siden den inkluderer en offset vekt.

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1D} & \omega_{1o} \\ w_{21} & w_{22} & \cdots & w_{2D} & \omega_{2o} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{C1} & w_{C2} & \cdots & w_{CD} & \omega_{Co} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \\ 1 \end{bmatrix} \quad (5)$$

$$g(x) = \mathbf{W}x \quad (6)$$

- $w_{ij} > 0$  (Positiv vekt): Høyere verdi på trekk/egenskap  $j$ , gir høyere sannsynlighet for at datapunkt tilhører klassen  $i$ .
- $w_{ij} < 0$  (Negativ vekt): Høyere (negativ) verdi på trekk  $j$ , gir lavere sannsynlighet for at datapunkt tilhører klassen  $i$ .
- $w_{ij} = 0$  (Vekt på null): Egenskapen  $j$  har ingen påvirkning på sannsynligheten for at datapunkt tilhører klassen  $i$ .
- $|w_{ij}|$ : Størrelsen på innflytelsen til trekk  $j$  på sannsynligheten for at datapunkt tilhører klassen  $i$ , uten hensyn til retningen på innflytelsen.

For en visuell formidling av disse punktene, se Vedlegg A.2, Figur 22.

MSE kostnadsfunksjonen i vektorform er gitt i likning (7), hvor  $g_k = \mathbf{W}x_k$  er verdien gitt av ett datapunkt  $x_k$ , og  $t_k$  er en vektor som beskriver hvilken klasse som blir evaluert og består av de binære verdiene  $\{0, 1\}$ , basert på [1, Eq. (3.19), s. 77].

$$MSE = \frac{1}{2} \sum_{k=1}^N (g_k - t_k)^T (g_k - t_k) \quad (7)$$

For minimere  $MSE$  blir likning (7) derivert med hensyn på  $\mathbf{W}$  og satt lik 0. Siden  $g_k$  skal tilpasses  $t_k$  burde det ideelt sett bli brukt en "heaviside step function", men siden den ikke er kontinuerlig deriverbar, blir den istedenfor approksimert med en sigmoid-funksjon, som i likning (8), basert på [1, Eq. (3.20), s.77]

$$g_{ik} = \text{sigmoid}(x_{ik}) = \frac{1}{1 + e^{-z_{ik}}}, \quad (8)$$

der  $z_k = \mathbf{W}x_k$ , tidligere kalt  $g_k$ . Ved bruk av likning (8) i derivasjonen ender vi opp med likning (9).

$$\nabla_{\mathbf{W}} MSE = \sum_{k=1}^N \nabla_{g_k} MSE \nabla_{z_k} g_k \nabla_{\mathbf{W}} z_k \quad (9)$$

Forenklingene i likning (10), (11) og (12), tatt fra [1, s. 79], satt inn i likning (7), gir oss gradienten til  $MSE$  i likning (13).

$$\nabla_{g_k} MSE = g_k - t_k \quad (10)$$

$$\nabla_{z_k} g_k = g_k \circ (1 - g_k) \quad (11)$$

$$\nabla_{\mathbf{W}} z_k = x_k^T \quad (12)$$

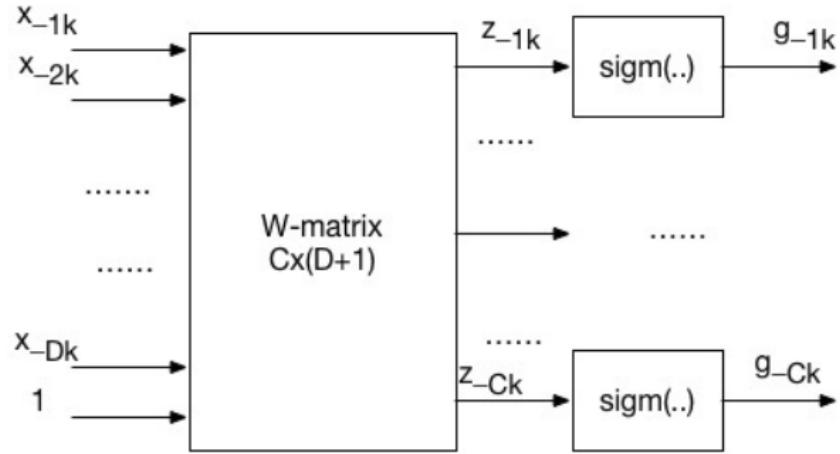
$$\nabla_{\mathbf{W}} MSE = \sum_{k=1}^N [(g_k - t_k) \circ g_k \circ (1 - g_k)] x_k^T \quad (13)$$

Målet er å kalkulere og oppdatere vektene,  $\mathbf{W}$ . For å minimere  $MSE$  må vi flytte  $\mathbf{W}$  i motsatt retning av gradienten frem til den konvergerer, gitt av likning (14), basert på [1, Eq. (3.23), s. 79]

$$\mathbf{W}(m) = \mathbf{W}(m-1) - \alpha \nabla_{\mathbf{W}} MSE \quad (14)$$

der  $m = 1, 2, 3, \dots$ , beskriver hvilken iterasjonen systemet er i, og  $\alpha$  er steg-faktoren som beskriver fluksringene mellom iterasjonene. Dersom størrelsen på  $\alpha$  er stor, blir fluksringene stor, og det blir da mindre nøyaktig. Dersom størrelsen på  $\alpha$  er for liten, kan det bli unødvendig mange iterasjoner før systemet konvergerer, [1, s. 80].

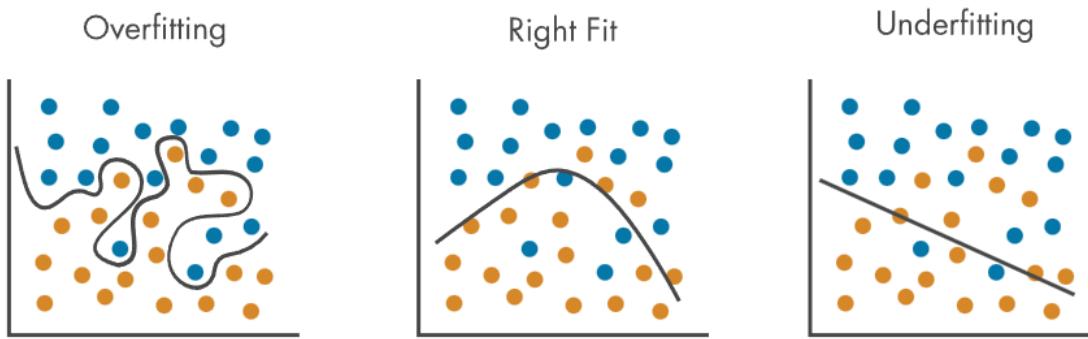
Strukturen til en lineær klassifikator er som illustrert i Figur 5.



**Figur 5:** Lineær klassifikator, tatt fra [1, Fig. 3.8, s. 79].

## 2.2 Tilpasning

En av de mest utfordrende og viktige oppgavene innefor klassifisering, er å justere systemets kompleksitet. Det må være kompliest nok til å skille mellom klassene i systemet, men samtidig unngå at det gir dårlig klassifisering av nye mønster [3, Kap. 3.7.3]. Tre typiske tilpasninger er illustrert i Figur 6. Systemer hvor det er utilstrekkelig mengde med data vil det som oftest oppstå overtilpasset klassifisering, en måte å unngå dette er å redusere dimensjonaliteten, enten ved å endre på sensorerne fra Figur 1, eller ved å velge riktig delmengde av tilgjengelig data.

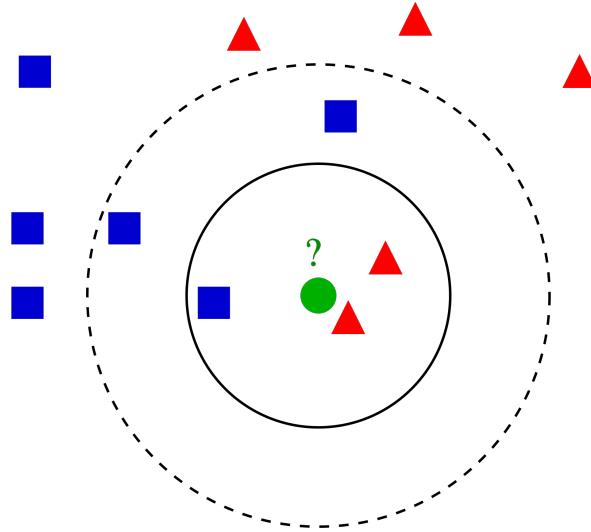


**Figur 6:** Overtilpasset, balansert tilpasset og undertilpasset klassifisering, tatt fra [5].

## 2.3 Malbasert Klassifikator

En malbasert klassifikator, *template based classifier* på engelsk, tar utgangspunkt i et input  $x$  og sammenligner det med andre referansepunkter, som har samme *form* som  $x$ , i et dataset [1, s. 70]. Det blir dermed avgjort hvilken klasse  $x$  tilhører ved å se på de nærmeste referanse-

punktene, og dermed anta at det tilhører den klassen som fremkommer først eller oftest av et gitt antall naboer. Dette er grunnlaget bak NN- (*Nearest Neighbour*) og **k**-NN algoritmen (*k-Nearest-Neighbours*). En enkel illustrasjon av k-NN algoritmen er illustrert i Figur 7, hvor den grønne sirkelen er uklassifisert data som enten tilhører rød trekant eller blå firkant klasse, og **k** beskriver antallet naboer man bruker som referanse. I situasjoner hvor majoriteten består av et likt antall naboer fra forskjellige klasser, vil valget av klasse være tilfeldig. Derfor kan det være hensiktsmessig å velge **k** som oddetall. Beste valg av **k** avhenger av dataen, men generelt vil en høy verdi redusere støy/forstyrrelser på klassifiseringen, men det gjør skillene mellom klassene mindre distinkte [6].



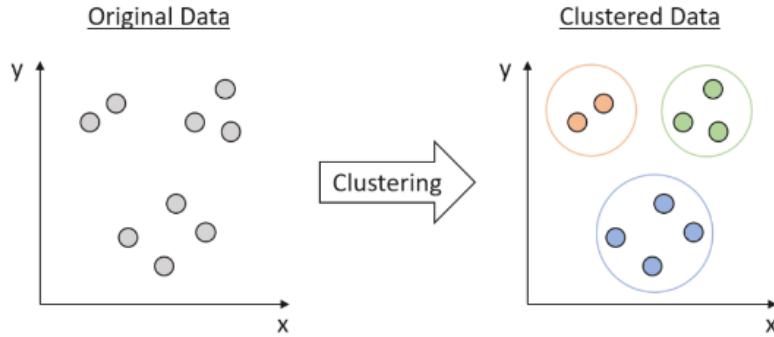
**Figur 7:** k-NN algoritme for  $k=3$  (sirkel med heltrukken linje) og  $k=5$  (sirkel med stiplet linje), tatt fra [6].

For å avgjøre og regne ut hvilke punkter som er «nærmest nabo», brukes Euklidisk avstand  $d(x, ref_{ik})$ , som beskrevet i likning (15), hvor  $ref_{ik}$  er referansepunktene med forventningsverdi  $\mu_{ik}$ , basert på [1, Eq. (3.9), s. 70].

$$d(x, ref_{ik}) = \sqrt{\sum_{k=1}^N (x - \mu_{ik})^2} \quad (15)$$

## 2.4 Gruppering

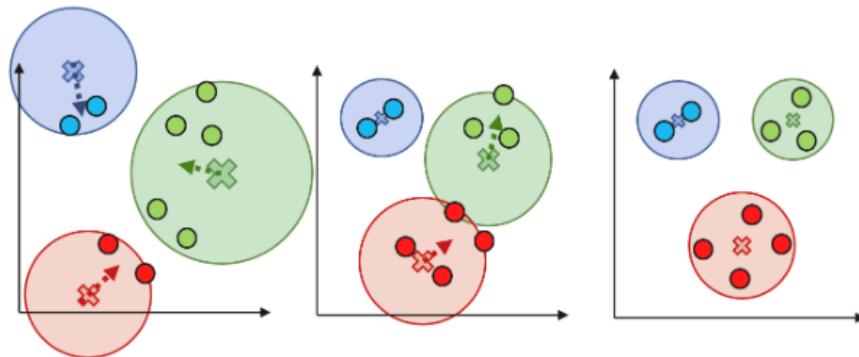
Gruppering, eller *clustering* på engelsk, er en form for ikke-veiledet maskinlæring, og innebærer at et system deler data inn i “naturlige grupper” basert på mønster [3, Kap. 1.4.2]. En enkel illustrasjon av gruppering av data er illustrert i Figur 8.



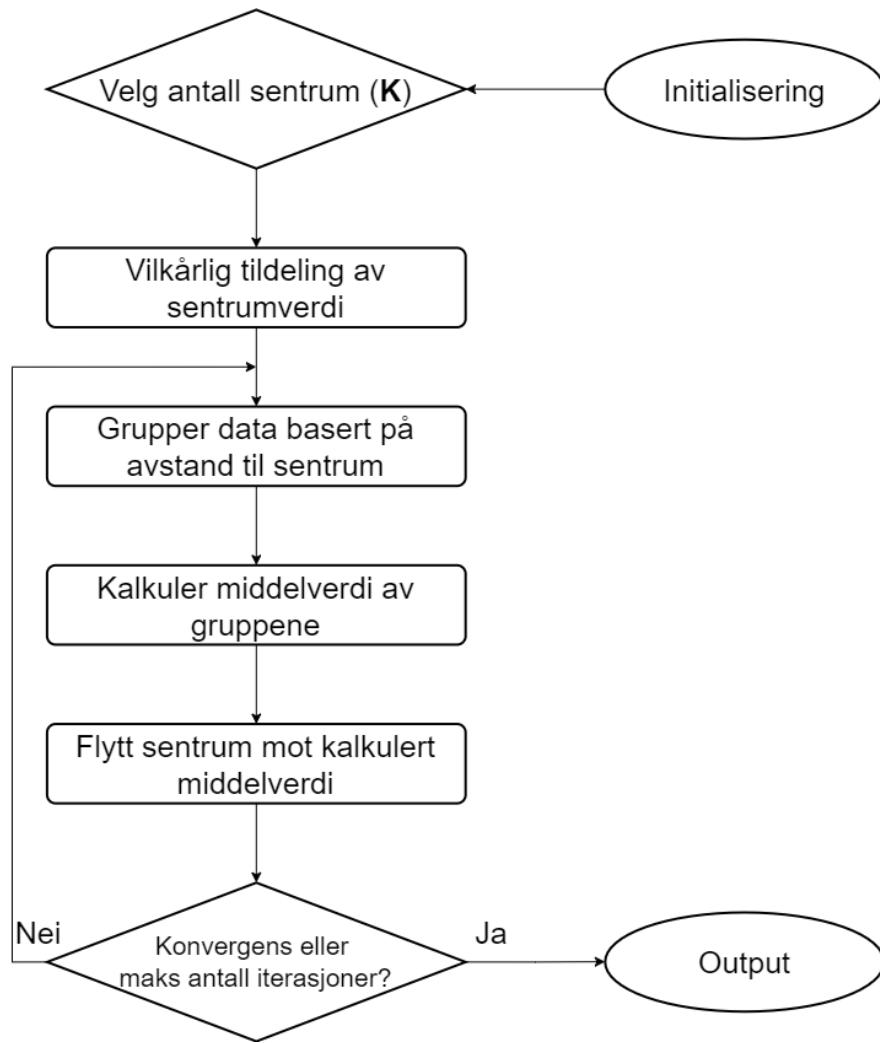
**Figur 8:** Gruppering av data, tatt fra [7].

### K-Means Gruppering

K-means gruppering er en algoritme som deler data inn i **K** distinkte grupper, hvor hver gruppe har et sentrum eller midtpunkt. Disse sentrene tildeles tilfeldige startverdier, og dataene gruppertes basert på hvilket sentrum de er nærmest. Etter grupperingen justeres sentrene mot gjennomsnittsverdien av alle punktene i deres respektive grupper. Denne justeringsprosessen gjentas gjennom et forhåndsbestemt antall iterasjoner, eller til endringene i sentrene blir ubetydelige, noe som indikerer konvergens [3, Kap. 10.4.3]. Denne prosessen er illustrert i Figur 9, og et tilhørende flytskjema er vist i Figur 10. I denne prosessen vil hvert datapunkt eller element tilhøre en bestemt gruppe, og overlapp mellom gruppene unngås.



**Figur 9:** K-means gruppering av data ( $K=3$ ), tatt fra [7]. Kryssene indikerer nåværende sentrum til en gruppe, og stiplet linjene med pilhoder indikerer hvor sentrum av gruppen vil forflytte seg til i neste iterasjon.



**Figur 10:** Flytskjema av K-means algoritmen.

## 2.5 Evaluering

Ved evaluering av klassifiseringsmodeller brukes en *confusion matrix*, som er en matrise som viser antall korrekte og antall feile prediksjoner modellen gjør. En slik matrise, bestående av tre forskjellige klasser, er illustrert i Figur 11, og en kvantitativ måling av modellens prestasjon, *estimated error rate (EER)*, er gitt av forholdet mellom antall feil klassifiseringer og totalt antall klassifiseringer, som beskrevet i likning (16), basert på [1, s. 90].

		Confusion Matrix		
		Klasse 1	Riktig Klassifisering	Feil Klassifisering
Sann Klasse	Klasse 1	Riktig Klassifisering	Feil Klassifisering	Feil Klassifisering
	Klasse 2	Feil Klassifisering	Riktig Klassifisering	Feil Klassifisering
	Klasse 3	Feil Klassifisering	Feil Klassifisering	Riktig Klassifisering
		Klasse 1	Klasse 2	Klasse 3
Prediktert Klasse				

**Figur 11:** *Confusion matrix* av tre vilkårlige klasser, hvor prediksjoner langs diagonalen (grønn) indikerer riktige klassifiseringer.

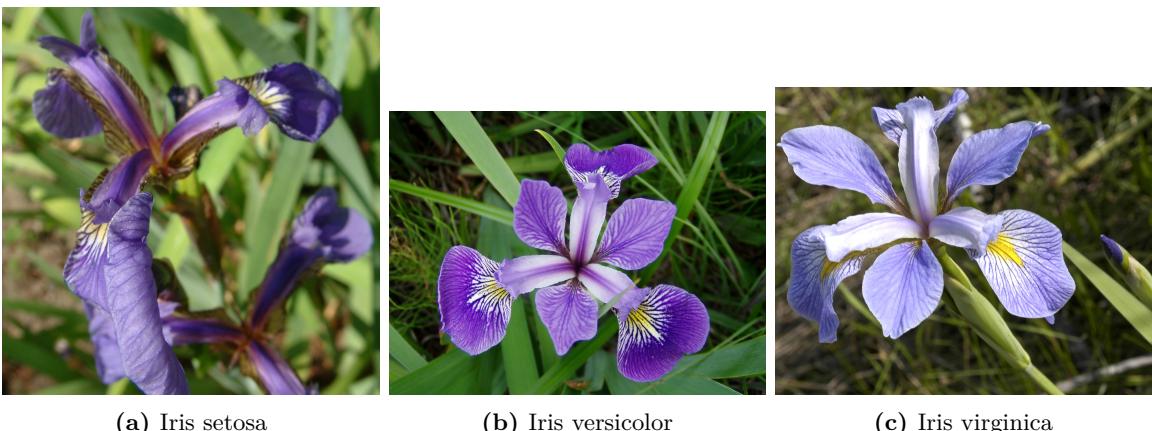
$$ERR = \frac{\text{antall feil klassifiseringer}}{\text{totalt antall klassifiseringer}} \quad (16)$$

### 3 Oppgaver

Denne rapporten tar for seg to ulike klassifiseringsproblem/oppgaver. I den oppgaven skal vi klassifisere ulike varianter av blomsten Iris, og i den andre skal vi klassifisere håndskrevne tall.

### 3.1 Iris Klassifisering

Iris blomsten har flere ulike underarter, og i denne oppgaven skal vi se på variantene *Setosa*, *Versicolor*, og *Virginica*, vist i Figur 12.

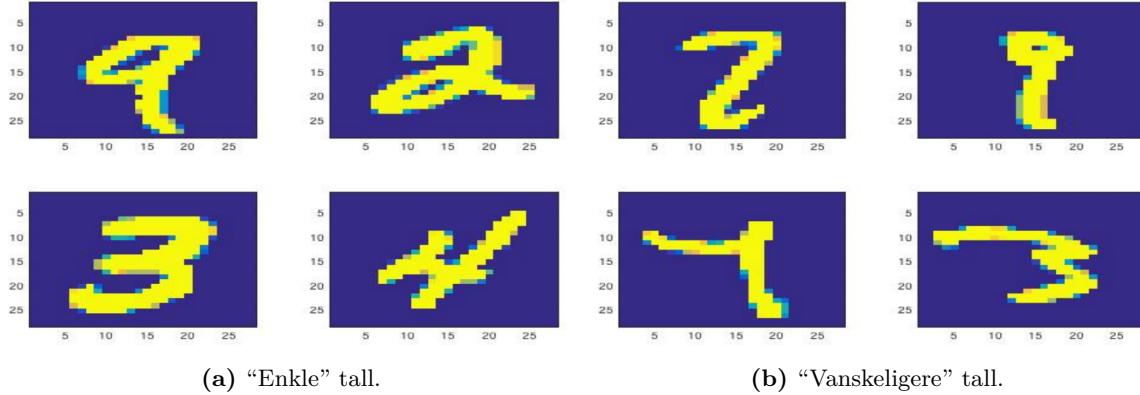


**Figur 12:** Bilder av Iris artene, hentet fra [8].

I denne oppgaven skal vi klassifisere de ulike underartene utifra Irisblomsten sine kjennetegn/egenskaper for lengde og bredde av bladtypene begerblad (*sepal*) og kronblad (*petal*), disse trekkene blir referert til som *sepal width (SW)*, *sepal length (SL)*, *petal width (PW)* og *petal length (PL)*. Gitt disse trekkene, anses dette klassifiseringsproblemet som lineært separabelt. Vi skal ta for oss en database med 50 eksempler på de ulike bladtypene tilknyttet blomstene til å utvikle en lineært diskriminerende klassifikator (LDK), forklart i seksjon 2.1, med fokus på design og evaluering av LDK-en. Videre skal vi analysere den relative tyngden av hver av de fire egenskapene med hensyn til separabilitet. Dette skal gjøres ved å fjerne trekk modellen trenes opp på, basert på overlapp i sannsynlighetstetthet, og se om dette gir bedre resultater.

### 3.2 Klassifisering av Håndskrevne Tall

I denne oppgaven skal vi klassifisere håndskrevne tall i sjiktet 0-9. Bildene er tatt fra en MNIST database, og har dimensjon 28x28 pixler i en 8-bit gråskala. Eksempler på henholdsvis "enkle" og "vankeligere" er vist i Figur



**Figur 13:** Eksempler på tall fra MNIS database. Hentet fra prosjektbeskrivelse, vedlagt i Vedlegg A.1.

Databasen består av 60000 treningseksempler og 10000 testeksempler, begge skrevet av 250 ulike personer. Det bør nevnes at bildene i databasen er allerede prosessert, slik at tallene er sentrert og skalert likt.

Målet er å klassifisere tallene korrekt, med en *error rate* på 1-10%. For å gjøre dette skal vi benytte oss av, og vurdere de malbaserte klassifikatorene NN og k-NN, diskutert i seksjon 2.3, samt gruppering, diskutert i seksjon 2.4. Vi skal vurdere de ulike metodene utifra oppnådd *confusion matrix*, *error rate*, og prosesseringstid.

## 4 Implementering og Resultater

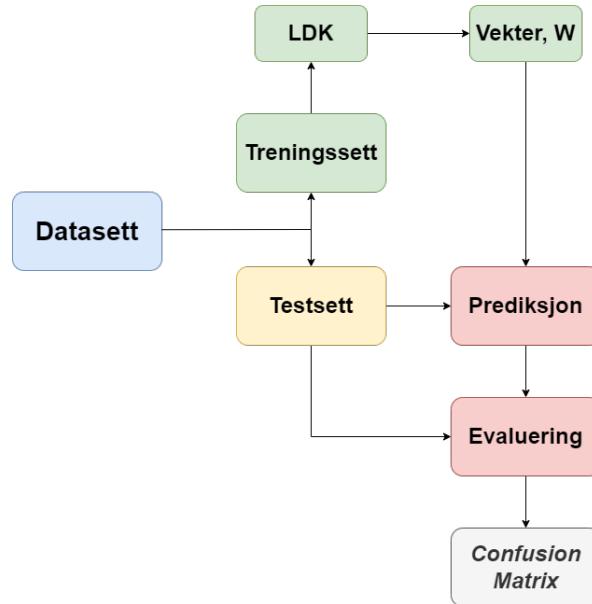
### 4.1 Iris Klassifisering

#### Implementering

For å klassifisere de ulike underartene av Iris, skal vi implementere en *LDK* og teste denne. Vi vil så undersøke hvordan ulik av input gitt som trenings- og testdata hvordan påvirker endelig resultat. Etter dette skal vi ta for oss de fire egenskapene om bladene til de tre underartene, og se om det er mulig å oppnå bedre resultater ved å fjerne overlappende egenskaper, da store overlapp vil gjøre det vanskeligere å skille klassene. Dette vil da også vise hvor mange egenskaper som faktisk trengs for å få en *god* klassifikator for Iris blomster.

Datasetssettet inneholder 150 samler av Iris blomsten, fordelt på de tre underartene. Hver av blomstene har fire egenskaper, som nevnt i seksjon 3, *sepal width (SW)*, *sepal length (SL)*, *petal width (PW)* og *petal length (PL)*.

En visuell forklaring av implementasjonen av klassifikator og tesing er illustrert i Figur 14.



**Figur 14:** Blokkskjema som beskriver prosessen bak veiledet maskinlæring av en klassifiseringsmodell (LDK), brukt til Iris klassifisering.

Før treningsprosessen av klassifikatoren starter, deles datasettet i ett treningssett og ett datasetssett. Generelt deles disse opp i henholdsvis 70-80% og 20-30% av datasettet. I denne implementasjonen derimot, vil vi bruke en 60/40 fordeling for å få hele tall. For den første testen tilsvarer dette de 30 første samplene til treningsdata, og de 20 siste tiltestdata, per klasse.

Generelt vil det bli brukt *MSE* som kostnadsfunksjon ved implementasjon av en *LDK*, og teste hvilke verdier av  $\alpha$  som gir laveste *error rate*. Altså kjøre et gitt antall iterasjoner av

treningen, slik  $MSE$  konvergerer mot en gitt lav verdi. Ved å bruke Matlab sin funksjon *fitdiscr* derimot, vil dette ikke være nødvendig. Denne vil automatisk trenre *LDK*-en til optimal ytelse, gitt treningssettet. Den kalkulerer først gjennomsnitt og kovariansmatriser, som brukes til å beregne diskriminantfunksjonene, funksjoner av egenskapene som gir den største forskjellen mellom klassene [9]. I denne delen av oppgaven vil vi teste begge metoder.

Treningssettet vil altså bli brukt til å konstruere to *LDK*-er, med bruk av  $MSE$  og Matlab-funksjonen *fitdiscr*, som gir basis for vektingen  $\mathbf{W}$ . Før testingen starter, er det hensiktsmessig å finne et optimalt antall iterasjoner, samt steglengde  $\alpha$ . Dette er gjort ved å plotte antall iterasjoner mot  $MSE$  for ulike verdier av  $\alpha$ , og vises i Figur 15. Her viser Figur 15a fem ulike jevnt spredte verdier for  $\alpha$ , der det ved visuell analyse er funnet best resultat for en steglengde på  $\alpha = \frac{1}{100}$ . Basert på dette er det plottet på nytt for steglengde  $\alpha = \frac{1}{100}$ , samt for to nye nærtliggende verdier,  $\frac{1}{99}$  og  $\frac{1}{100}$ , i Figur 15b. Figur 15c viser det samme plottet, men forstørret. Basert på de to siste plottene kan en visuelt avgjøre hvilken  $\alpha$  som gir laveste konvergert  $MSE$ , og når denne konvergerer. Her er det konkludert med at  $\alpha = \frac{1}{99}$  gir best resultat, og at for denne verdien konvergerer  $MSE$  mot laveste verdi fra ca. 1230 iterasjoner, og det er disse tallene som vil bli brukt videre.

Vektene for denne fordelingen av datasettet er gitt av  $\mathbf{W}_{01}$  i likning (19) for bruk av  $MSE$ , og  $\mathbf{W}_1$  i likning (17) for bruk av *fitdiscr*.

Vektene blir deretter, som illustrert i Figur 14, brukt til prediksjon av testsettet, altså *LDK*-ene produsert av de første 30 samplene testes på de 20 siste samplene fra datasettet. Deretter sammenliknes prediksjonen med de faktiske klassene i testdataen for å produsere en *confusion matrix*. For denne fordelingen er matrisen og *error rate* gitt av likning (16), vist i Figur 16 for bruk av  $MSE$ , og Figur 17a for *fitdiscr*.

For å se på betydningen av å hvilke samples som ligger i henholdsvis test- og datasettene, skal vi nå utføre nøyaktig samme prosess, men nå bruke de 30 *siste* samplene til trening, og de 20 siste til testing. Dette gir vektene  $\mathbf{W}_{02}$  i likning (18) for  $MSE$ , og  $\mathbf{W}_2$  i likning (20) for *fitdiscr*. Tilhørende *confusion matrix* med *error rate* er vist i henholdsvis Figur 16b og Figur 17b.

For å undersøke påvirkningen av de ulike egenskapene *SW*, *SL*, *PW* og *PL*, skal vi starte med produsere histogram for hver av egenskapene og se på overlappene. Her brukes de 30 første samplene til trening, og de 20 siste til testing. I denne delen av oppgaven skal vi benytte oss av Matlab-funksjonen *fitdiscr*, da denne har en kortere prosesseringstid.

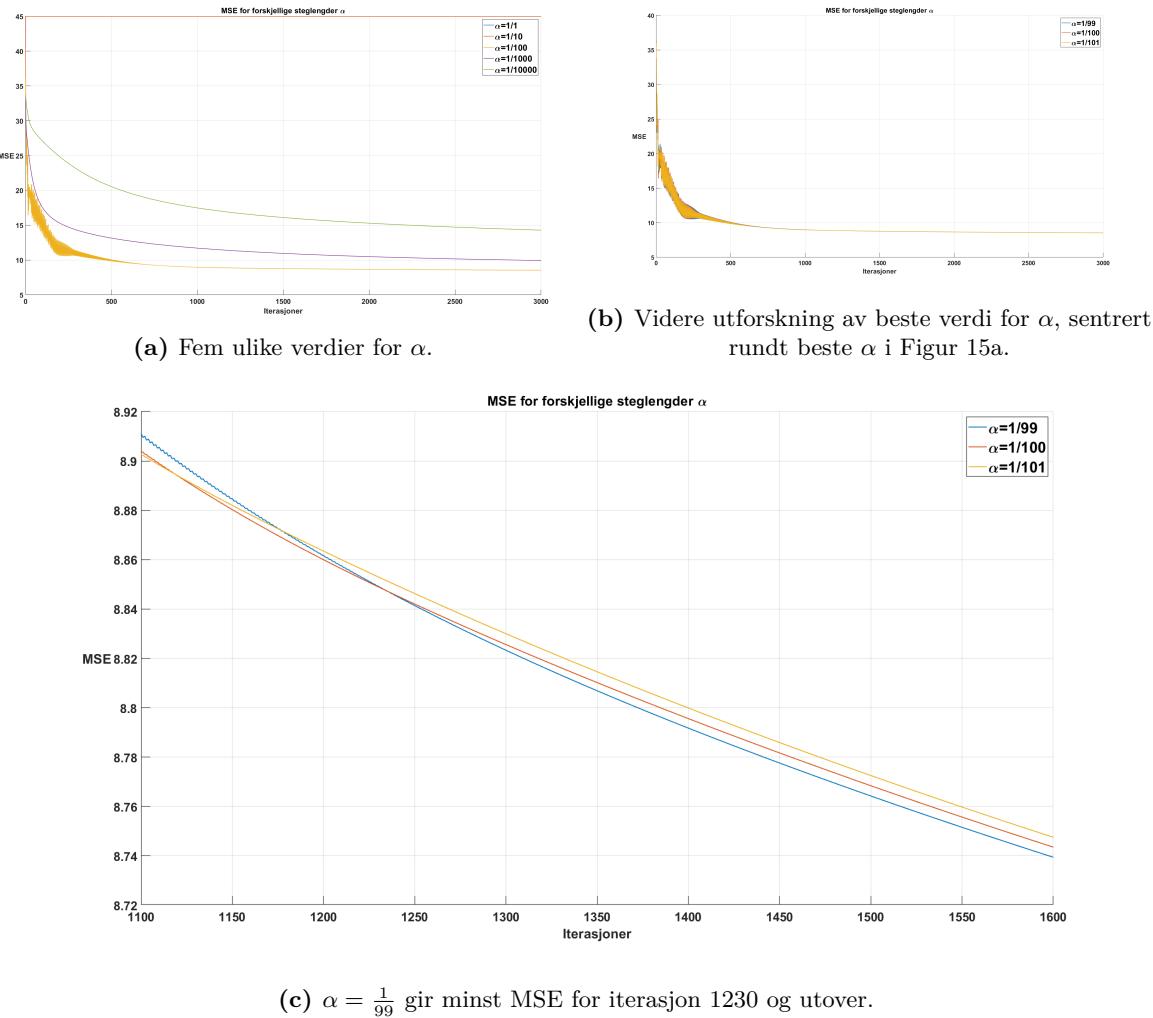
Basert på datasettene for de ulike egenskapene produseres histogrammene i Figur 18, der en også kan se *PDF*-ene (sannsynlighetstetthetene) til de ulike egenskapene for å letttere kunne visuelt analysere overlappene. Middelverdiene for egenskapene er gitt av likningene (21), når det er opprettet ved de 30 første samplene. Overlappet er funnet matematisk i Matlab, og verifisert visuelt, til å være størst for trekket *SL* (*sepal length*). For å undersøke effekten gjentas stegene beskrevet over for å utvikle en ny *LDK*, men nå uten trekket *SL*. Dette resulterte i vektene  $\mathbf{W}_3$  i likning (22), og *confusion matrix* i Figur 19a.

Denne prosessen er gjentatt to ganger, der det er funnet at de trekkene som overlapper mest etter *SL* er henholdsvis *SW* (*sepal width*) og *PL* (*petal length*). Resultatene fra å fjerne først *SW*, og så *PL*, kan sees av henholdsvis vektene  $\mathbf{W}_4$  og  $\mathbf{W}_5$  i likning (23) og (24), samt *confusion matrix*-ene med *error ratene* i Figur 19b og 19c.

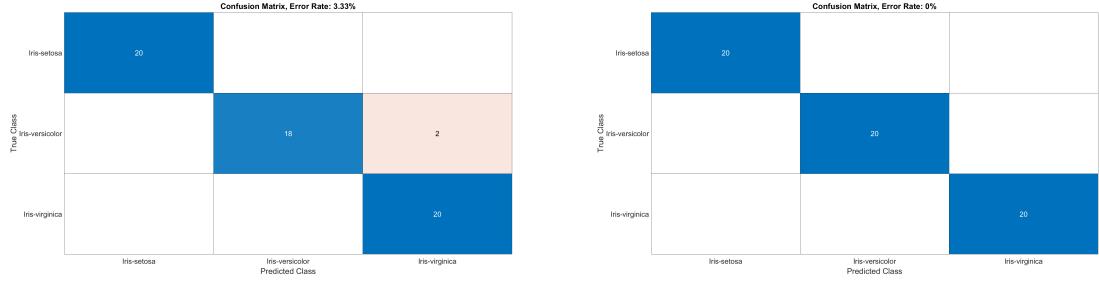
For enkel sammenlikning er en sammenheng mellom vektene  $\mathbf{W}_n$ , sampler brukt til trening, hvilke trekk som er inkludert i modellen og *error rate*, vist i Tabell 1.

Fullstendig kode for implementasjon for klassifiseringen av Iris blomstens underarter ved bruk av *MSE* og Matlab-funksjonen *fitdiscr* er henholdsvis gitt i Vedlegg A.1, **IrisTask.mlx** og **IrisMSE.mlx**.

## Resultater



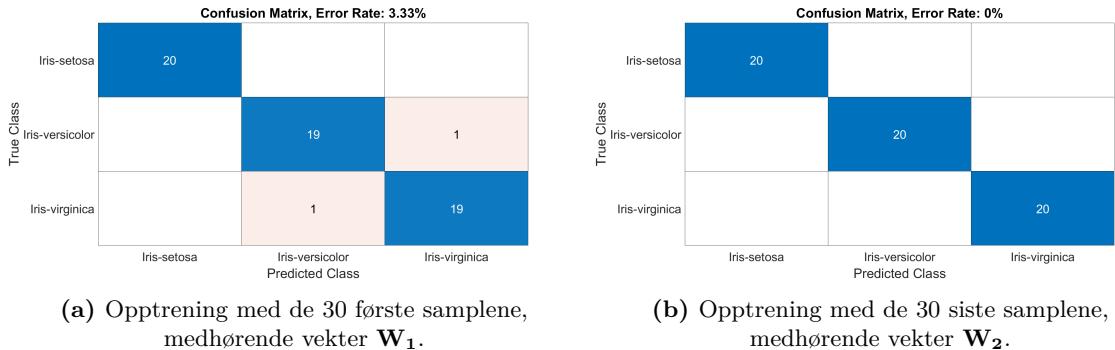
**Figur 15:** Testing av ulike verdier av  $\alpha$  for å finne den som minimerer *MSE* best.



(a) Opptrening med de 30 første samplene,  
medhørende vekter  $\mathbf{W}_{01}$ .

(b) Opptrening med de 30 siste samplene,  
medhørende vekter  $\mathbf{W}_{02}$ .

**Figur 16:** *Confusion matrix(er)* av LDK-modeller opptrent og testet på forskjellige sampler ved bruk av *MSE* kostnadsfunksjon.



(a) Opptrening med de 30 første samplene,  
medhørende vekter  $\mathbf{W}_1$ .

(b) Opptrening med de 30 siste samplene,  
medhørende vekter  $\mathbf{W}_2$ .

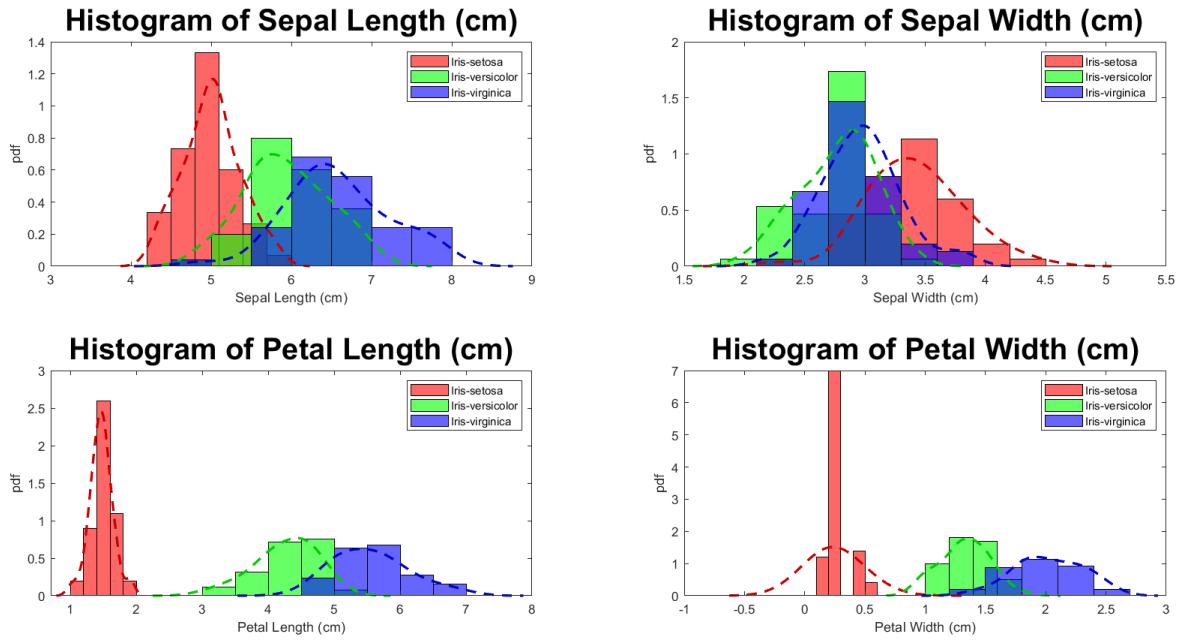
**Figur 17:** *Confusion matrix(er)* av LDK-modeller opptrent og testet på forskjellige sampler med Matlab-funksjonen *fitdiscr*.

$$\mathbf{W}_{01} = \begin{bmatrix} 0.450 & 1.785 & -2.651 & -1.231 & 0.321 \\ 1.475 & -3.070 & -0.269 & -1.036 & 2.160 \\ -3.270 & -2.817 & 4.799 & 4.346 & -2.338 \end{bmatrix} \quad (17)$$

$$\mathbf{W}_{02} = \begin{bmatrix} 0.524 & 1.788 & -2.814 & -1.323 & 0.328 \\ 0.019 & -2.358 & 1.904 & -4.070 & 4.052 \\ -2.282 & -3.323 & 3.892 & 4.793 & -3.072 \end{bmatrix} \quad (18)$$

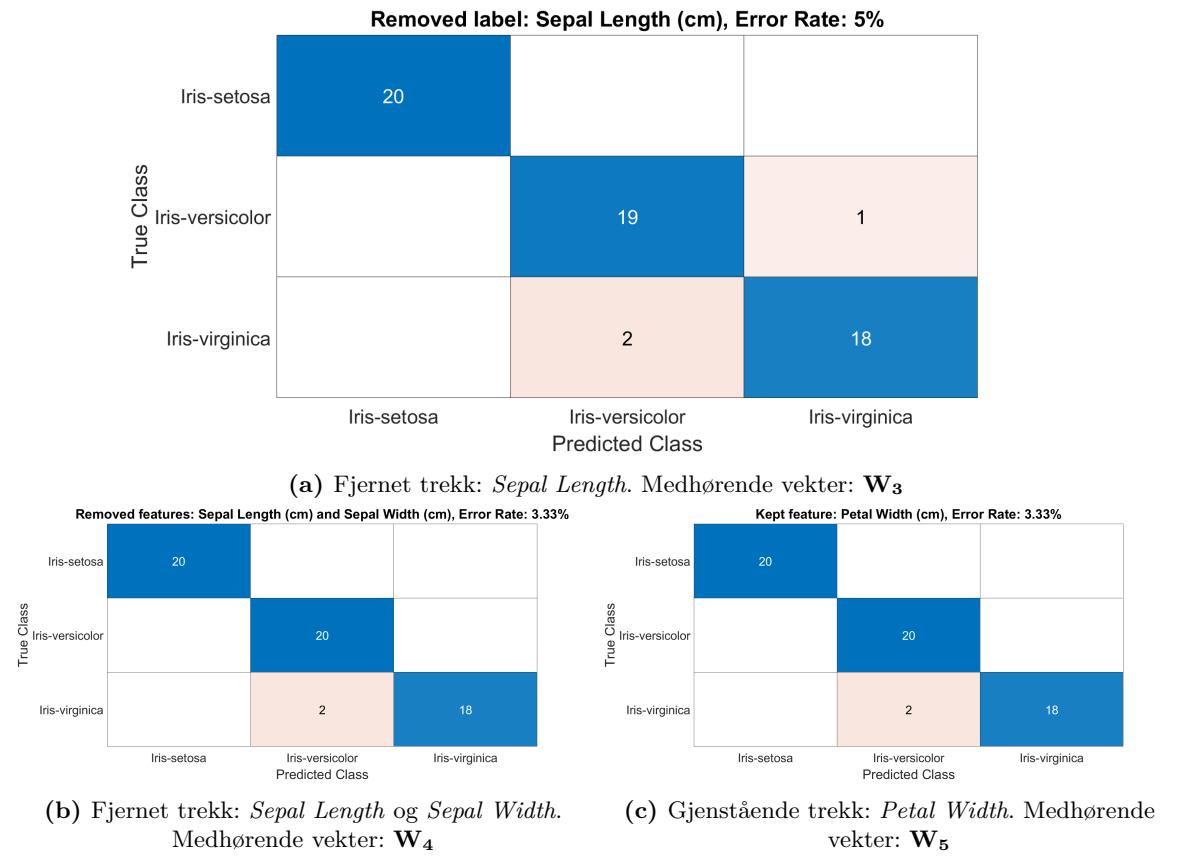
$$\mathbf{W}_1 = \begin{bmatrix} 12.313 & 44.476 & -39.399 & -78.247 & -3.852 \\ -0.744 & -16.276 & 8.996 & 16.474 & 41.856 \\ -11.568 & -28.200 & 30.404 & 61.773 & -38.004 \end{bmatrix} \quad (19)$$

$$\mathbf{W}_2 = \begin{bmatrix} 18.479 & 36.507 & -51.975 & -51.885 & 0.320 \\ -6.330 & -14.268 & 16.304 & 6.801 & 49.256 \\ -12.149 & -22.240 & 35.671 & 45.084 & -49.575 \end{bmatrix} \quad (20)$$



**Figur 18:** Histogram / sannsynlighetstetthet (pdf) av de forskjellige trekkene til blomstene.  
Middelverdier til hver av trekkene (i cm) er lagt i en matrise  $\mu$ , likning (21).

$$\mu = \begin{bmatrix} SL & SW & PL & PW \\ 5.0267 & 3.4500 & 1.4733 & 0.2467 \\ 6.0700 & 2.7900 & 4.3333 & 1.3533 \\ 6.5833 & 2.9333 & 5.6033 & 2.0067 \end{bmatrix} \begin{array}{l} Setosa \\ Versicolor \\ Virginica \end{array} \quad (21)$$



**Figur 19:** Confusion matrix(er) til opprette modeller ved reduserte antall trekk. Opprettet på de 30 første samplene.

$$\mathbf{W}_3 = \begin{bmatrix} 52.838 & -28.057 & -87.273 & 11.570 \\ -16.781 & 8.310 & 17.020 & 39.763 \\ -36.057 & 19.747 & 70.254 & -51.333 \end{bmatrix} \quad (22)$$

$$\mathbf{W}_4 = \begin{bmatrix} -23.019 & -44.554 & 120.127 \\ 6.710 & 3.452 & -4.591 \\ 16.309 & 41.102 & -115.536 \end{bmatrix} \quad (23)$$

$$\mathbf{W}_5 = \begin{bmatrix} -71.502 & 71.542 \\ 11.307 & 5.294 \\ 60.195 & -76.837 \end{bmatrix} \quad (24)$$

**Tabell 1:** *Error rate* til opprente LDK-modeller, med de forskjellige trekkene og hvilke samples de er trent på.

<b>W<sub>n</sub></b>	<b>Sampler</b>	<b>Trekk</b>	<b>Error Rate</b>
<b>W<sub>01</sub></b>	1-30	Alle	3.33%
<b>W<sub>02</sub></b>	21-50	Alle	0.00%
<b>W<sub>1</sub></b>	1-30	Alle	3.33%
<b>W<sub>2</sub></b>	21-50	Alle	0.00%
<b>W<sub>3</sub></b>	1-30	<i>SW &amp; PL &amp; PW</i>	5.00%
<b>W<sub>4</sub></b>	1-30	<i>PL &amp; PW</i>	3.33%
<b>W<sub>5</sub></b>	1-30	<i>PW</i>	3.33%

$$ERR_{T_{snitt}} = \sum_n^M \frac{ERR_{T_n}}{M} \quad (25)$$

$$ERR_{T_{snitt,MSE}} \stackrel{M=02}{\stackrel{n=01}{=}} 1.665\% \qquad \qquad ERR_{T_{snitt,fitdiscr}} \stackrel{M=5}{\stackrel{n=1}{=}} 2.998\%$$

$$ERR_{T_{snitt,totalt}} \stackrel{M=7}{\stackrel{n=01}{=}} 2.167\%$$

## 4.2 Klassifisering av Håndskrevne Tall

### Implementering

Først designes en **NN**-klassifikator ved hjelp av Euklidisk avstand som beskrevet i likning (15), i funksjonen `classify_nn` som er vedlagt i Vedlegg A.1. Denne funksjonen tar inn treningsdata, treningsetiketter og testdata som input. For hvert element itestdataene beregner den avstanden til alle elementene i treningsdataen, identifiserer nærmeste nabo, og gir tilsvarende etikett til testpunktet. Dette gjentas for alle testpunktene, og funksjonen returnerer de predikerte etikettene for alle testpunktene. Deretter sammenlignes disse predikerte etikettene med de sanne etikettene, og resultatet vises i en *confusion matrix* i Figur 20. Dette blir gjort for forskjellige treningsmengder, og sammenlignet i Tabell 2. Det dårligste resultatet er vedlagt i Vedlegg A.2, med rad- og kolonne oppsummeringer.

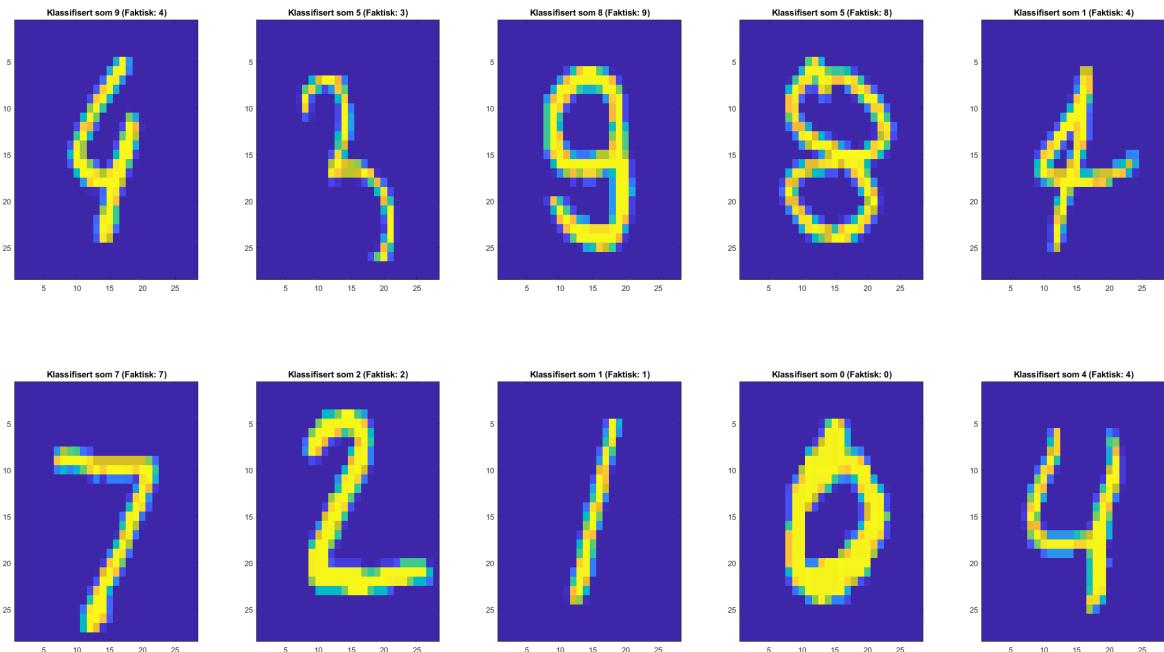
Noen av de feil klassifiserte og riktige klassifiserte håndskrevne tallene, ved bruk av NN-algoritmen, er illustrert i Figur 21. Total kode `NumberTask1.mlx` er vedlagt i Vedlegg A.1. I tillegg designes en tilsvarende algoritme for **K-NN** (se funksjonen `knn_classifier` vedlagt i Vedlegg A.1). Merk at denne kan brukes som **NN**-algoritme ved å sette **K**=1.

Deretter brukes den innebygde matlab-funksjonen `kmeans` til å dele treningssettet inn i mindre grupper, som deretter blir brukt som referanse til **K-NN**-algoritmen. Dette blir gjort for forskjellige verdier av **K** og forskjellige antall av grupper, relevante resultater er lagt til i Tabell 3. Total kode `NumberTask2.mlx` er vedlagt i Vedlegg A.1.

## Resultater



**Figur 20:** Confusion matrix av håndskrevne tall, ved bruk av NN-algoritme.



**Figur 21:** Eksempler på tall som ble feil klassifisert (rad 1) og riktig klassifisert (rad 2).  
NN-algoritme uten gruppering ved bruk av en treningsmengde på 60 000.

**Tabell 2:** Opprenningstid og *error rate* av NN-algoritme ved forskjellige treningsmengder.

Treningsmengde	Error Rate	Tid [MM:SS]
60 000	3.09%	17:25
10 000	5.37%	03:19
2 500	8.64%	00:48
1 000	13.1%	00:18

**Tabell 3:** Opprenningstid og *error rate* av forskjellige k-NN med og uten gruppering ved bruk av en treningsmengde på 60 000.

k-NN	Gruppering	Antall Grupper	Error Rate	Tid [MM:SS]
1-NN	✗	✗	3.09%	17:25
1-NN	✓	64	4.86%	00:06
3-NN	✓	64	5.58%	00:05
5-NN	✓	64	6.10%	00:05
7-NN	✓	64	6.49%	00:06
9-NN	✓	64	6.78%	00:06
1-NN	✓	16	6.69%	00:05
1-NN	✓	32	5.36%	00:05
1-NN	✓	128	3.86%	00:05
1-NN	✓	256	3.54%	00:07
1-NN	✓	512	3.22%	00:11
1-NN	✓	1024	3.09%	00:18
1-NN	✓	2048	2.84%	00:36
1-NN	✓	4096	3.27%	01:15

## 5 Diskusjon

I denne seksjonen skal vi diskutere hvorvidt resultatene presentert i seksjon 4 er tilstrekkelig valide. Vi vil først ta for oss resultatene for Iris klassifiseringen, for så å diskutere resultatene tilknyttet klassifiseringen av de håndskrevne tallene fra MNIST.

### Iris Klassifisering

Fra Tabell 1 kan en se at modellen med lavest *error rate*,  $ERR_{T_2} = 0\%$ , og dermed den mest nøyaktige, er den som er trent på de 30 siste samplene i datasettet, og inkluderer alle egenskapene. Dette gjelder for både bruk av *MSE* og Matlab-funksjonen *fitdiscr*. Det bør likevel presiseres at alle andre tester har gitt akseptabelt lav *error rate*, med snitt på  $ERR_{T_{snitt}, MSE} = 1.665\%$  for *MSE*,  $ERR_{T_{snitt}, fitdiscr} = 2.998\%$  for *fitdiscr*, og totalt snitt for begge metodene på  $ERR_{T_{snitt}, totalt} = 2.167\%$ . Det bør merkes her at snittet for bruk av *MSE* er lavest, men her inngår det kun to tester, og dermed senker det totale snittet.

Fordelen med å bruke færre av egenskapene i modellen er at det langt færre utregninger, og dermed en kortere prosesseringstid. Hvis dette er noe en hadde kjennskap til ved innhenting av data, ville dette også vært en langt kortere prosess, og spart mye arbeidstid. Her bør en også merke seg at ved å kun benytte ett trekk i modellen, *PW*, er det oppnådd en *error rate* på 3.3%, som ofte er ansett akseptabelt.

For å videre undersøke forskjellene mellom å benytte de 30 første eller siste samplene fra datasettet til trening, kunne en ha testet klassifikatoren på treningssettene, og regne ut en kombinert *error rate*. Videre kunne en ha stokket om på samplene i datasettet, da det vil kunne forbedre modellen ved å minimere overtilpasning, diskutert i seksjon 2.2.

For videre forbedring av klassifikatorene, vil det også være gunstig å utvide datasettet med flere samples. Det bør også merkes at de største overlappene, og der modellene gjør flest feil, er mellom underartene *Versicolor* og *Virginica*. Ved å øke antall samples i datasettet, særlig for disse to underartene, vil mest sannsynlig modellen operere med lavere *error rate*. Selv om dette vil kunne forbedre resultatet, opererer klassifikatorene allerede med en lav “nok” *error rate*.

Det er verdt å diskutere forskjellen mellom de to metodene med *MSE* og Matlab-funksjonen *fitdiscr*. Først og fremst krever *MSE* mer analytisk arbeid, bl.a. utforskning av optimal steg-lengde  $\alpha$  og antall iterasjoner for minimering av *MSE*, der *fitdiscr* finner optimal modell, gitt treningsdataen, automatisk. Dermed vil bruk av *fitdiscr* spare en del arbeidstid, i tillegg til at denne metoden, som nevnt, har en noe raskere prosesseringstid. Derimot vil bruk av *MSE* gi mulighet til å variere flere parametere i utregningen når en analytisk finner de optimale verdiene. Det er mulig å legge inn ekstra parametere i *fitdiscr* også, f.eks. for å minmre kryssvalideringstap, spesifisere kostanden for feilklassifisering, de forutgående sannsynlighetene for hver klasse, eller observasjonsklassene [9]. Her er det derfor dømt unødvendig å gå videre med *MSE*-metoden i andre delen av oppgaven (undersøkelse av trekkenes påvirkning), da resultatene var forholdsvis like ved bruk av *fitdiscr* i første del, men det vil kunne være et forbedringspotensiale for videre utvikling ved å utforske bruken av flere parametre.

## Klassifisering av Håndskrevne Tall

Figur 23 viser oppsummering av radene til høyre i figuren, noe som gir oss en forståelse av hvor godt hver klasse er blitt klassifisert. Fra denne ser vi at tallet 8 er det som blir feilklassifisert oftest, og tallet 1 blir feil klassifisert færrest ganger. I tillegg viser den en kolonne oppsummering, som forteller oss hvor ofte en klasse har blitt forvekslet med en annen klasse. Her ser vi at 9 er det tallet som blir predikert oftest når det skjer en feilklassifisering, og 0 er den som forkommer minst når prediksjonen er feil. Fra matrisen i seg selv, observerer vi også at 4 blir veldig ofte klassifisert som tallet 9, og tilsvarende mellom 3 og 5. For den beste modellen med lavest *error rate* i Figur 24, ser vi at 4 og 9 blir sjeldnere forvekslet, samtidig som at tallet 1 fortsatt er den som blir feilklassifisert færrest ganger.

Med utgangspunkt i de feilklassifiserte tallene i Figur 21, er det forståelig at modellen gjør disse feilene, ihvertfall de i rad 1, kolonne 2 og 5. De resterende feilklassifiserte tallene, i rad 1, kolonne 1, 3 og 4, ville man tro var litt lettere å klassifisere riktig.

Fra Tabell 2, observerer vi at en høyere treningsmengde gir lavere *error rate* på bekostning av lengre opptreningstid, som forventet. Merk at dette ble gjort på en kraftig PC med 32GB ram, og disse tidene blir betydelig høyere ved bruk dårligere hardware.

Opptreningstiden synker betraktelig ved implementering av gruppering, som observert i Tabell 3. Her vil **1-NN**-algoritmen gi lavest *error rate* og bruke cirka like lang tid som de med høyere verdi av **K**. Totalt sett får vi lavest *error rate* når vi bruker 2048 grupper, med bare en liten økning i opptreningstid, men fortsatt betydelig lavere enn uten bruk av gruppering. Til tross for hva man hadde forventet, vil en økning i **K** gi dårlige ytelese utifra Tabell 3, ihvertfall for denne problemstillingen.

Ved veiledet maskinlæring vil vi også få identiske resultater hver gang det blir repitet, mens ved ikke-veiledet maskinlæring ser vi at *error rate* og feilene som blir gjort endrer seg mellom hver repetisjon.

## 6 Konklusjon

### Iris Klassifisering

For å klassifisere de tre underartene av Iris blomsten, *Setosa*, *Versicolor* og *Virginica*, er det konstruert en Lineært Diskriminerende Klassifikator, der det er oppnådd en gjennomsnittlig *error rate* av alle testene utført på  $ERR_{T_{snitt}} = 2.998\%$ . Beste resultat, med  $ERR_T = 0\%$ , ble realisert ved å inkludere alle trekkene, *SW*, *SL*, *PW* og *PL*, og benytte de 30 siste samplene i datasettet til trening av modulen, og de 20 første til testing. Det er likevel verdt å merke seg at det er oppnådd en *error rate* på 3.33% ved å kun bruke et av trekkene, *PW*, med de 30 første samplene til trening, og de 20 siste til testing, som ofte er ansett et akseptabelt resultat.

### Klassifisering av Håndskrevne Tall

Ved klassifisering av håndskrevne tall ble det brukt **NN**- og **K-NN** algoritmer både med og uten gruppering, basert på Euklidisk avstand. Her fant vi at den beste modellen, basert på *error rate* og opptreningstid, var en kmeans-gruppering algoritme som besto av 2048 grupper, med **K=1** (vi så kun på nærmeste nabo). Dette er en ikke-veiledet maskinlæringsmodell som ga oss en *error rate* på 2.84% med en opplæringstid på 36 sekunder. Sammenlignet med den veilede modellen som har en *error rate* på 3.09% og opplæringstid på over 17 minutter, er dette en betydelig forbedring. Som teoretisk forventet ble også *error rate* og prestasjonen på klassifiseringsmodellene bedre ved opptrening av større datamengder. Dette er på bekostning av tid, men ved implementering av grupperingsalgoritmen så vi at denne gikk, noe som viser at denne algoritmen er effektiv ved bruk av store datasett. Imidlertid, selv om gruppering reduserte opptreningstiden, observerte vi at en økning i **K** medførte dårligere ytelse. Dette antyder at for dette spesifikke problemet, at det er mest effektivt å bare bruke den nærmeste naboen som referanse.

Videre, ved analyse rundt de feilklassifiserte tallene, observerte vi at tallet 1 ble feilklassifisert færrest ganger og tallet 0 er det som forekommer minst når det skjer en feilklassifisering. Utifra de feilklassifiserte tallene som modellen gjorde, illustrert i Figur 21, er det tydelig at det er stort rom for forbedring, her var det kun 2 av 5 tall vi kunne forstå hvorfor ble feilklassifisert.

## Referanser

- [1] Myrvoll, T. A. & Werner, S. & Johnsen, M. H. (ingen dato). *TTT4275 Estimering, deteksjon og klassifisering - Kompendium*, Institutt for elektroniske systemer - NTNU.
- [2] Kay, S. M. (1993). *Fundamentals of Statistical Signal Processing: Estimation Theory*, USA, Prentice-Hall, Inc., University of Rhode Island.
- [3] Duda, R. O. & Hart, P. E. & Stork, D. G. (November 2000). *Pattern Classification*, Second Edition, John Wiley & Sons, Inc.
- [4] López, O. A. M. & López, A. M. & Crossa, J. (2022). *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, Springer.
- [5] MathWorks, *Overfitting*, 07.04.2024
- [6] Wikipedia, *k-nearest neighbors algorithm*, 07.03.2024
- [7] Trevor Amestoy, *Clustering basics and a demonstration in clustering infrastructure pathways*, 16.03.2022
- [8] Wikipedia, *Iris flower data set*, 18.04.2024
- [9] MathWorks, *fittediscr*, 18.04.2024

## A Vedlegg

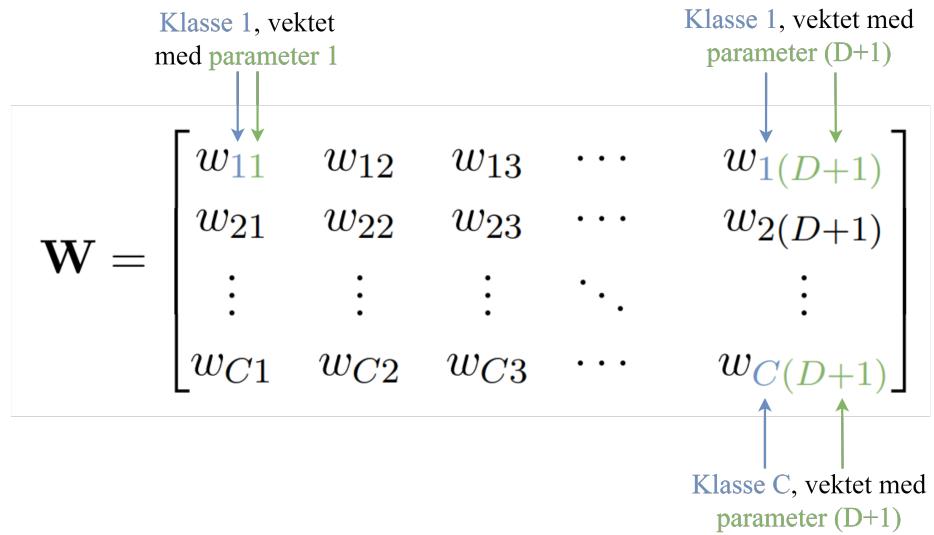
### A.1 Github

# Kode og data.

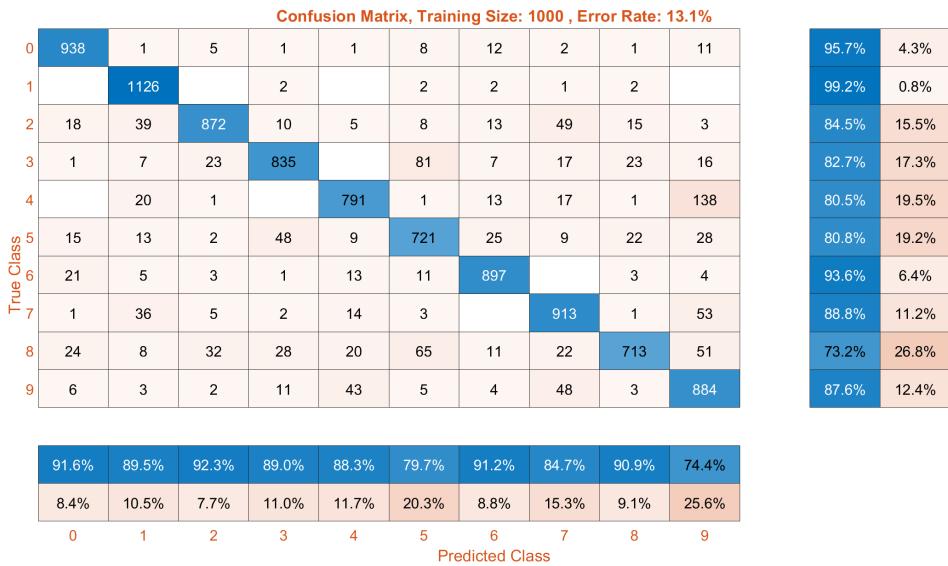
### A.2 Ekstra Figurer

$$\mathbf{W} = \begin{bmatrix} \text{Klasse 1, vektet med parameter 1} & & & & \text{Klasse 1, vektet med parameter } (D+1) \\ \downarrow & & & & \downarrow \\ w_{11} & w_{12} & w_{13} & \cdots & w_{1(D+1)} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2(D+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{C1} & w_{C2} & w_{C3} & \cdots & w_{C(D+1)} \end{bmatrix}$$

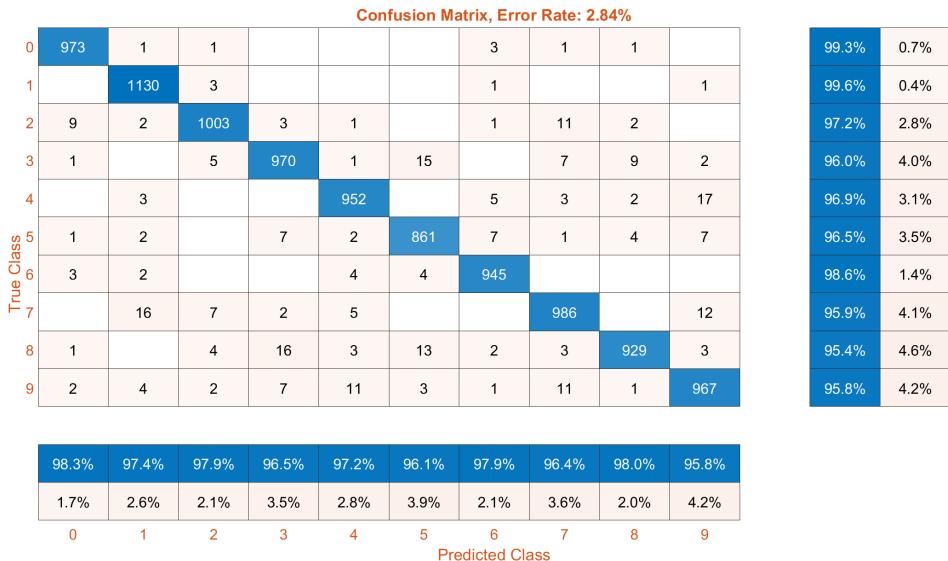
Klasse C, vektet med parameter (D+1)



**Figur 22:** Intuitiv forklaring av vektingsmatrise som inkluderer offset-vekting.



**Figur 23:** Confusion matrix av håndskrevne tall, ved bruk av NN-algoritme med en treningsmengde på 1000. Høyest error rate av alle tester.



**Figur 24:** Confusion matrix av håndskrevne tall, ved bruk av 1-NN-algoritme med gruppering bestående 2048 grupper. Lavest error rate av alle tester.