TTK4205 Mønstergjennskejnning
Prosjektoppgave 1

Anders Johannessen

Høst 2017

Kunnskap for en bedre verden

Introduksjon

Denen raporten er skrevet som et svar på Prosjektoppgave 1 i faget Unik4590 / 9590 / TTK4205 Mønstergjenkjenning ved UiO / NTNU. Oppgaven går ut på å implementere tre forskjellige klassifisatorer (nærmeste nabo, minimum feilrate og minste kvadraters metode) og trene/teste disse klassifisatorene på tre ulike datasett. Nærmeste nabo klassifisatoren skal, ut ifra feilrate, brukes til å finne den beste egenskapskombinasjon for hver dimensjon som videre brukes i minimum feilrate og minste kvadraters metode klassifisatorene. De tre klassifisatorene blir så evaluert basert på feilrate for hver av de beste egenskapskombinasjonene funnet av nærmeste nabo klassifisatoren.

Gjennomførelse

All programmvare er skrevet i pgrogrammeringspråket Python. Kode som er brukt er listet i appendix: Python scripts. Først ble dataene fra tekst filene importert i programmvaren slik at det kunne bearbeides av klassifisatorene. Dette ble gjøres av funksjonen `readData` i main.py (Listing: 1). Funksjonen sorterer dataene etter linjenummer hvor oddetallslinjene legges i en trenings-data matrise og partallslinjene legges i en test-data matrise. Som i data filene så forteller første linje i trenings- og test-data matrisene hvilken klasse objektet tilhører. I de påfølgene kolonnene blir egenskapsverdiene til objektet lagt inn. I test-data matrisen er det lagt inn en ekstra kolonne som klassifisatorene bruker til å lagre klassen den har kommet frem til at objektet tilhører.
Etter at dataene har blitt lagret ble funksjonen `constructDimentionMatrix` kalt. Denne funksjonen tar inn en antall dimensjoner for data settet ($n_{kolonner} - 1$ i data filene) og returnerer en matrise med alle mulige kombinasjoner av egenskaper for en gitt dimensjon ($dimensjon = 1, \ldots, n_{dimensjoner}$). For alle mulige egenskapkombinasjonene for alle dimensjoner kalles nærmeste nabo klassifisatoren.

2.1 Nærmeste nabo klassifikator

Nærmeste nabo klassifisatoren (Listing: 2) går gjennom alle egenskapskombinasjonene og beregner astanden mellom et objekt i test-data matrisen med alle objektene trenings-data matrisen:

$$d = ||x_{test} * dimensjon - x_{tren} * dimensjon||$$

Den minste avstanden lagres og det siste det siste elementet i test-data listen oppdateres til samme klasse som trenings-data objektet (det første kolonnen i trenings-data matrisen).
Etter at alle objektene i test-data matrisen er sammenlignet med alle objektene i trenings-data matrisen går funksjonen gjennom alle objektene i test-data matrisen og sammenligner egentlig klasse med tilegnet klasse.

$$feilrate = \frac{n_{feil}}{n_{objekter}}$$

Test-data matrisen, den beste kombinasjonen for hver dimensjon og tilhørende feilrate retuneres for videre bruk i programmet.

Den beste kombinasjonen for hver av dimensjonene brukes videre i minimum feilrate klassifisatoren og minste kvadraters metode klassifisatoren.

2.2 Minimum feilrate klassifisator

Minimum feilrate klassifisatoren (Listing: 3) begynner med å rekonstuere test-data og trenings-data matrisene slik at de kun inneholder de aktuelle egenskapene definert av de beste dimensjonene som ble funnet i nærmeste nabo klassifisatoren. Deretter går programmet gjennom alle nødvendige kalkulasjoner ($\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\Sigma}}_i$, W_i, $\boldsymbol{w}_i$ og w_{i0}) for å kunne regne ut diskriminantfunksjonene for hver av objektene i test-data matrisen:

$$g_i(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{W}_i \boldsymbol{x} + \boldsymbol{w}^T \boldsymbol{x} + w_{i0} \quad i = 1, 2$$

Dersom $g_1(\boldsymbol{x}) \geq g_2(\boldsymbol{x})$ tilegnes objektet til klasse 1, og dersom $g_1(\boldsymbol{x}) < g_2(\boldsymbol{x})$ tilegnes objektet til klasse 2. Feilraten kalkuleres på samme måte som i nærmeste nabo klassifisatoren. Funksjonen returnerer test-data matrisen.

2.3 Minste kvadraters metode

Minste kvadraters metode klassifisatoren (Listing: 4) starter med å rekonstruere test-data og trenings-data matrisene på samme måte som i minimum feilrate klassifisatoren. Så ved hjelp av hjelpefunksjonene `calculateY` og `calculateB` kalkuleres $\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1^T & \dots & \boldsymbol{y}_n^T \end{bmatrix}^T$ hvor $\boldsymbol{y} = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}^T$ og $\boldsymbol{b}$ som brukes i

$$\boldsymbol{a} = (\boldsymbol{Y}^T \boldsymbol{Y})^{-1} \boldsymbol{Y}^T \boldsymbol{b}$$

`leastSquareMethod` funksjonen finner nå diskriminantfunksjonen

$$g(\boldsymbol{x}) = \boldsymbol{a}^T \boldsymbol{y}$$

og dersom $g(\boldsymbol{x}) \geq 0$ tilegnes objektet klassen 1, og dersom $g(\boldsymbol{x}) < 0$ tilegnes objektet klassen 2. Feilraten kalkuleres på samme måte som i nærmeste nabo og minimum feilrate. Test-data matrisen retuneres ut av funksjonen.

Resultatet fra alle klassifisatorene for den beste todimensjonale egenskapskombinasjonen plottes ved hjelp av hjelpefunksjonen `plotResults`.

Resultat

Tabell 1-4 viser resultatet av nærmeste nabo klassifisatoren for dimensjonene 1,...,4 til hver av datasettene. Vi kan se at for de beste éndimensjonal egenskapskombinasjonene er:

- [1, 0, 0, 0] med feilrate 0.24 for datasett 1
- [1, 0, 0] med feilrate 0.18 fro datasett 2
- [0, 1, 0, 0] med feilrate 0.31 for data set 3.

Table 1: Resultat ved kjøring av nærmeste nabo klassifisatoren for 1-dimensjonale kombinasjoner

<table>
<tr><th colspan="2">Datasett 1</th><th colspan="2">Datasett 2</th><th colspan="2">Datasett 3</th></tr>
<tr><th>Dimensjon</th><th>Feilrate</th><th>Dimensjon</th><th>Feilrate</th><th>Dimensjon</th><th>Feilrate</th></tr>
<tr><td>[1, 0, 0, 0]</td><td>0.24</td><td>[1, 0, 0]</td><td>0.18</td><td>[1, 0, 0, 0]</td><td>0.33</td></tr>
<tr><td>[0, 1, 0, 0]</td><td>0.36</td><td rowspan="2">[0, 1, 0]</td><td rowspan="2">0.28</td><td>[0, 1, 0, 0]</td><td>0.31</td></tr>
<tr><td>[0, 0, 1, 0]</td><td>0.43</td><td>[0, 0, 1, 0]</td><td>0.34</td></tr>
<tr><td>[0, 0, 0, 1]</td><td>0.39</td><td>[0, 0, 1]</td><td>0.49</td><td>[0, 0, 0, 1]</td><td>0.40</td></tr>
</table>

De beste todimensjonale egenskapskombinasjonene er:

- [1, 0, 0, 1] men feilrate 0.18 for datasett 1
- [1, 1, 0] med feilrate 0.01 for datasett 2
- [0, 1, 1, 0] med feilrate 0.10 for datasett 3

Table 2: Resultat ved kjøring av nærmeste nabo klassifisatoren for 2-dimensjonale kombinasjoner

<table>
<tr><th colspan="2">Datasett 1</th><th colspan="2">Datasett 2</th><th colspan="2">Datasett 3</th></tr>
<tr><th>Dimensjon</th><th>Feilrate</th><th>Dimensjon</th><th>Feilrate</th><th>Dimensjon</th><th>Feilrate</th></tr>
<tr><td>[1, 1, 0, 0]</td><td>0.18</td><td rowspan="2">[1, 1, 0]</td><td rowspan="2">0.01</td><td>[1, 1, 0, 0]</td><td>0.21</td></tr>
<tr><td>[1, 0, 1, 0]</td><td>0.19</td><td>[1, 0, 1, 0]</td><td>0.17</td></tr>
<tr><td>[1, 0, 0, 1]</td><td>0.17</td><td rowspan="2">[1, 0, 1]</td><td rowspan="2">0.19</td><td>[1, 0, 0, 1]</td><td>0.28</td></tr>
<tr><td>[0, 1, 1, 0]</td><td>0.32</td><td>[0, 1, 1, 0]</td><td>0.10</td></tr>
<tr><td>[0, 1, 0, 1]</td><td>0.23</td><td rowspan="2">[1, 0, 1]</td><td rowspan="2">0.29</td><td>[0, 1, 0, 1]</td><td>0.24</td></tr>
<tr><td>[0, 0, 1, 1]</td><td>0.30</td><td>[0, 0, 1, 1]</td><td>0.19</td></tr>
</table>

De beste tredimensjonale egenskapskombinasjonene er:

- [1, 1, 0, 1] med feilrate 0.10 for datasett 1
- [1, 1, 1] med feilrate 0.02 for datasett 2
- [0, 1, 1, 1] med feilrate 0.07 for datasett 3

For firedimensjonale egenskapskombinasjoner er det kun datasett 1 og 3 som er med. Dette er fordi datasett 2 har kun tre egenskaper per objekt. Det er også kun én mulig egenskapskombinasjon for hver av datasettene:

Table 3: Resultat ved kjøring av nærmeste nabo klassifisatoren for 3-dimensjonale kombinasjoner

Datasett 1		Datasett 2		Datasett 3	
Dimensjon	**Feilrate**	**Dimensjon**	**Feilrate**	**Dimensjon**	**Feilrate**
[1, 1, 1, 0]	0.15	[1, 1, 1]	0.02	[1, 1, 1, 0]	0.10
[1, 1, 0, 1]	0.10			[1, 1, 0, 1]	0.20
[1, 0, 1, 1]	0.13			[1, 0, 1, 1]	0.15
[0, 1, 1, 1]	0.21			[0, 1, 1, 1]	0.07

Table 4: Resultat ved kjøring av nærmeste nabo klassifisatoren for 4-dimensjonale kombinasjoner

Datasett 1		Datasett 2		Datasett 3	
Dimensjon	**Feilrate**	**Dimensjon**	**Feilrate**	**Dimensjon**	**Feilrate**
[1, 1, 1, 1]	0.09			[1, 1, 1, 1]	0.10

- [1, 1, 1, 1] med feilrate 0.09 for datasett 1
- [1, 1, 1, 1] med feilrate 0.10 for datasett 3

Tabell 5 og Tabell 6 viser resultatet av kjøring av henholdsvis minimum feilrate klassifisatoren og minste kvadraters metode klassifisatoren for de beste egenskapskombinasjonene fra nærmeste nabo klassifisatoren for hver av datasettene.

Table 5: Resultat ved kjøring av minimum feilrate klassifisatoren for beste dimensjon kombinasjoner

Datasett 1		Datasett 2		Datasett 3	
Dimensjon	**Feilrate**	**Dimensjon**	**Feilrate**	**Dimensjon**	**Feilrate**
[1, 0, 0, 0]	0.19	[1, 0, 0]	0.11	[0, 1, 0, 0]	0.23
[1, 0, 0, 1]	0.11	[1, 1, 0]	0.02	[0, 1, 1, 0]	0.20
[1, 1, 0, 1]	0.10			[0, 1, 1, 1]	0.13
[1, 1, 1, 1]	0.08	[1, 1, 1]	0.02	[1, 1, 1, 1]	0.07

Table 6: Resultat ved kjøring av minste kvadraters metode klassifisatoren for beste dimensjon kombinasjoner

Datasett 1		Datasett 2		Datasett 3	
Dimensjon	**Feilrate**	**Dimensjon**	**Feilrate**	**Dimensjon**	**Feilrate**
[1, 0, 0, 0]	0.15	[1, 0, 0]	0.11	[0, 1, 0, 0]	0.33
[1, 0, 0, 1]	0.11	[1, 1, 0]	0.13	[0, 1, 1, 0]	0.23
[1, 1, 0, 1]	0.11			[0, 1, 1, 1]	0.14
[1, 1, 1, 1]	0.09	[1, 1, 1]	0.13	[1, 1, 1, 1]	0.10

Den beste klassifisatoren for hver av egenskapskombinasjonen var for datasett 1:

- [1, 0, 0, 0] med feilrate 0.15 fra minste kvadraters metode
- [1, 0, 0, 1] med feilrate 0.11 fra minste kvadraters metode
- [1, 1, 0, 1] med feilrate 0.10 fra både nærmeste nabo og minimum feilrate
- [1, 1, 1, 1] med feilrate 0.08 fra minimum feilrate

For datasett 2:

- [1, 0, 0] med feilrate 0.11 fra minimum feilrate
- [1, 1, 0] med feilrate 0.01 fra nærmeste nabo
- [1, 1, 1] med feilrate 0.02 fra både nærmeste nabo og minimum feilrate

For datasett 3:

- [0, 1, 0, 0] med feilrate 0.23 fra minimum feilrate
- [0, 1, 1, 0] med feilrate 0.10 fra nærmeste nabo
- [0, 1, 1, 1] med feilrate 0.07 nærmeste nabo
- [1, 1, 1, 1] med feilrate 0.07 fra minimum feilrate

For hvert datasett ble dem beste todimensjonale egenskapskombinasjonen plottet. Figur 1-3 viser hendoldsvis resultatet av nærmeste nabo, minimum feilrate og minste kvadrater metode fra datasett 1, figur 4-6 fra datasett 2 og figur 7-9 fra datasett 3.

Avsluttende spørsmål

1. Den asymptotiske feilraten til nærmeste nabo regelen er gitt av

$$P^* \leq P \leq P^*(2 - \frac{c}{c-1}P^*)$$

hvor P^* er den optimale feilraten og c er antall klasser. I denne situasjonen er $c = 2$, noe som gir oss

$$P^* \leq P \leq P^*(2 - 2P^*)$$

Dette er da en relativ lav asymptotisk feilrate, noe som gjør nærmeste nabo klassifisatoren til en fornuftig klassifikator til å finne gunstige egenskapskombinasjoner.

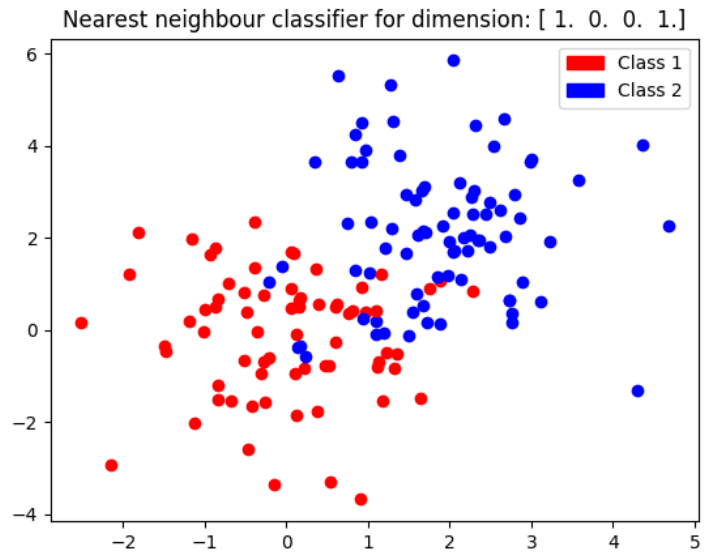

Figure 1: Nærmeste nabo klassifisering for dimensjonen: [1, 0, 0, 1]

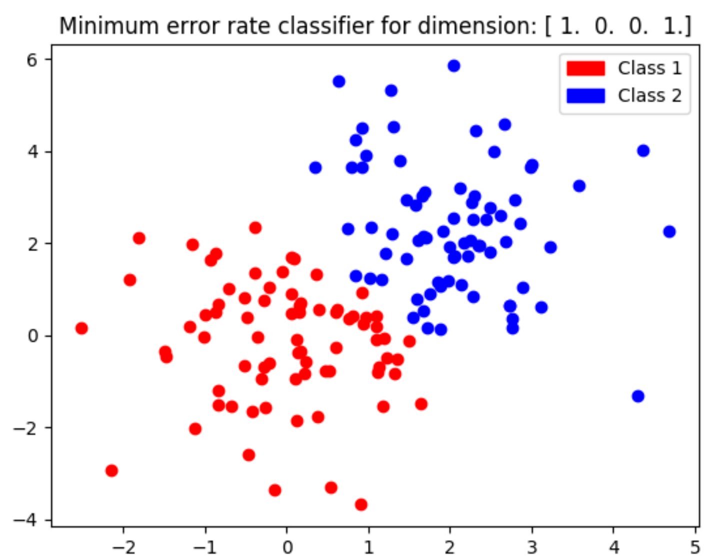

Figure 2: Minimum feilrate klassifisering for dimensjonen: [1, 0, 0, 1]

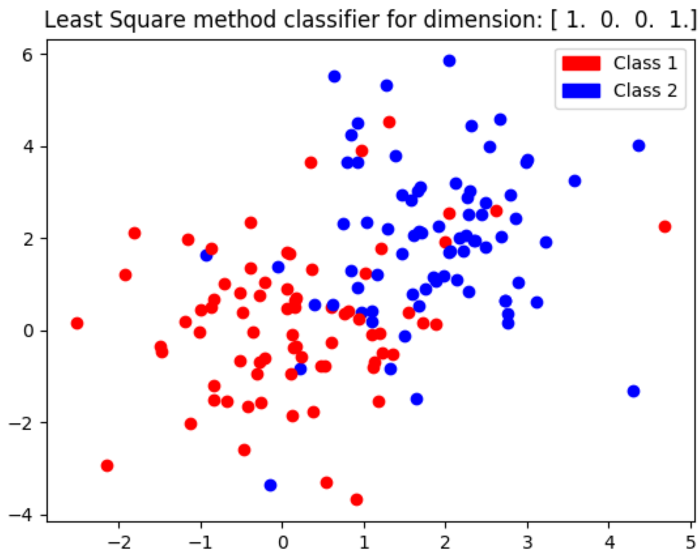

Figure 3: Minste kvadraters metode klassifisering for dimensjonen: [1, 0, 0, 1]

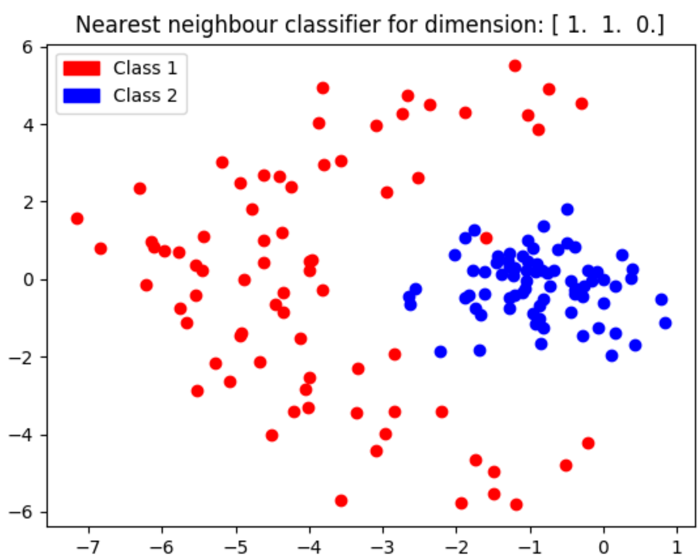

Figure 4: Nærmeste nabo klassifisering for dimensjonen: [1, 1, 0]

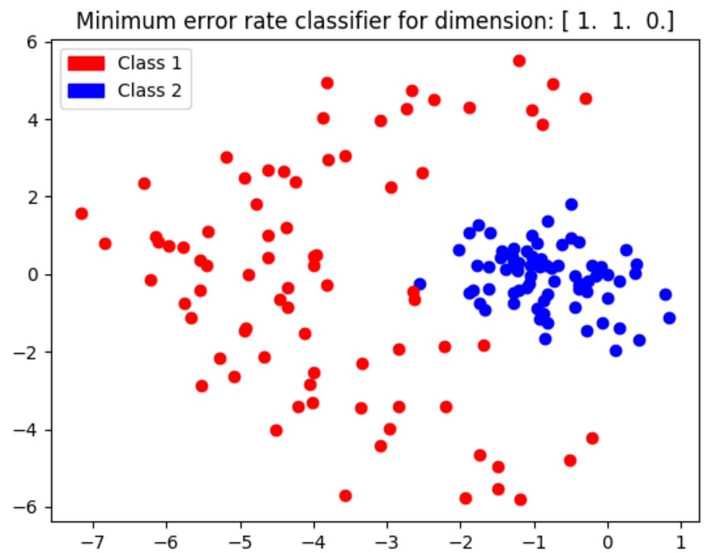

Figure 5: Minimum feilrate klassifisering for dimensjonen: [1, 1, 0]

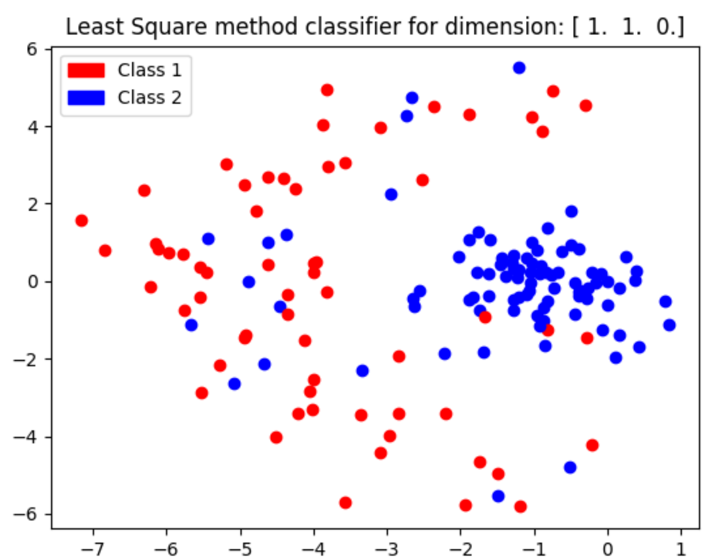

Figure 6: Minste kvadraters metode klassifisering for dimensjonen: [1, 1, 0]

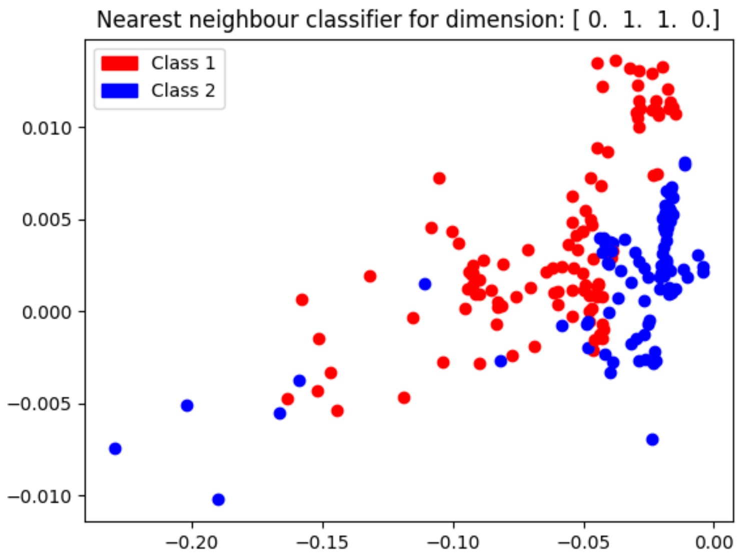

Figure 7: Nærmeste nabo klassifisering for dimensjonen: [0, 1, 1, 0]

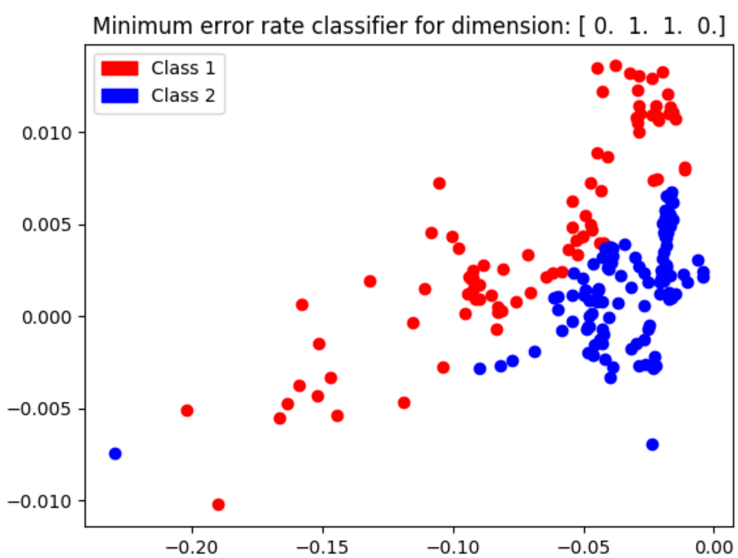

Figure 8: Minimum feilrate klassifisering for dimensjonen: [0, 1, 1, 0]

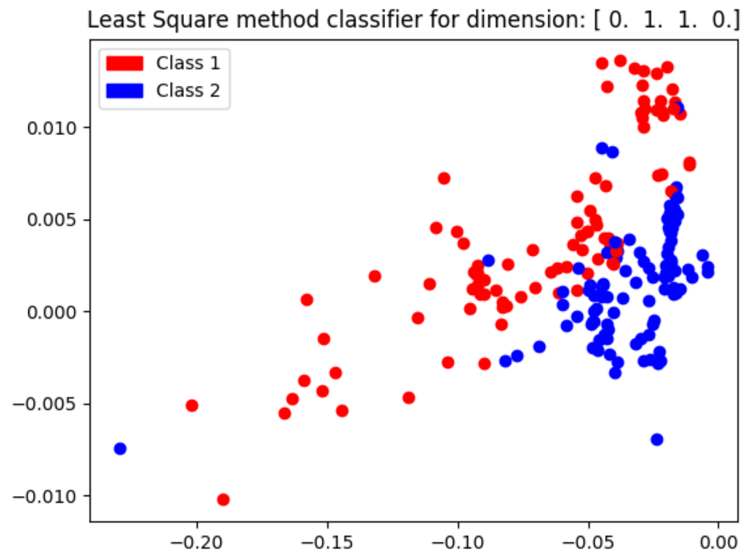

Figure 9: Minste kvadraters metode klassifisering for dimensjonen: [0, 1, 1, 0]

2. Nærmeste nabo klassifisatoren sammenligner alle test objektene med alle treningsobjektene. Når du i praktiske situasjoner ofte har større datasett krever denne klassifisatoren ekstremt mye regnekapasitet sammenlignet med f.eks minimum feilrate klassifisatoren som bruker en diskriminantfunksjon til å klassifisere et objekt. Et større datasett krever også større krav til lagringsplass.

3. Å bruke samme datasett til både trening å evaluering er en dårlig idé fordi da trenes klassifisatoren kun opp på en type data. Klassifisatoren vil da presetere bra på den test med samme type data, men årestere dårligere på data fra et annet set. Du vil altså få en lavere feilrate enn den faktiske feilraten ved evaluering på samme datasett.

4. At minste kvadraters metode klassifisatoren presterer dårligere enn nærmeste nabo og minste feilrate klassifisatorene er tydelig fremstilt i figur 6 sammenlignet med figur 4 og figur 5. Grunnen til dette kan være at datasett 2 ikke er lineært separabelt. Dette kan vi se fra figurene da det ikke er mulig å trekke en rett linje som skiller de to klassene.

Python scripts

A.1 Hoved script

Listing 1: main.py

```python
import numpy as np
import itertools
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from nearestNeighbour import nearestNeighbour
from minimumErrorRate import minimumErrorRate
from leastSquareMethod import leastSquareMethod



def readData(file_name):
    file = open(file_name, 'r')
    lines = file.readlines()

    N_ROWS = len(lines)//2
    N_COLLUMS = len(lines[0].split())

    training_set = np.zeros([N_ROWS, N_COLLUMS])
    test_set = np.zeros([N_ROWS, N_COLLUMS + 1])

    for i in range(N_ROWS):
        for j in range(N_COLLUMS):
            training_set[i][j] = float(lines[i*2].split()[j])
            test_set[i][j] = float(lines[i*2 + 1].split()[j])

    return N_ROWS, N_COLLUMS, training_set, test_set



# Returns a matrix with all possible dimension combinations
def constructDimentionMatrix(dim, N_DIM):
    for i in range(N_DIM):
        combinations =
        ↪ list(itertools.combinations(list(range(N_DIM)), dim +
        ↪ 1))

        dimension_matrix = np.zeros([len(combinations), N_DIM])

        row = 0
        for indexes in combinations:
```

```
            for col in indexes:
                dimension_matrix[row][col] = 1
            row += 1

        return dimension_matrix



def plotResults(best_error_rate_dimension, training_set, test_set, N_ROWS, N_COLLUMS, data_file):
    data_file,_ = data_file.split(".")
    name = "results/" + data_file


    dimension = [best_error_rate_dimension[1]]

    result_NN,_,_ = nearestNeighbour(training_set, test_set, N_ROWS, dimension)

    x_test = np.zeros([N_ROWS, np.count_nonzero(dimension[0] == 1) + 2])
    x_test[:, 0] = result_NN[:, 0]
    x_test[:, -1] = result_NN[:, -1]
    index = 1
    for i in range(len(dimension[0])):
        if dimension[0][i] == 1:
            for j in range(N_ROWS):
                x_test[j][index] = result_NN[j][i + 1]
            index += 1

    result_MER = minimumErrorRate(training_set, test_set, N_ROWS, dimension)
    result_LSM = leastSquareMethod(training_set, test_set, N_ROWS, N_COLLUMS, dimension)


    plt.figure(1)
    for row in x_test:
        if row[0] == 1:
            plt.plot(row[1], row[2], 'ro')
        else:
            plt.plot(row[1], row[2], 'bo')
    plt.legend(handles=[mpatches.Patch(color='red', label='Class 1'), mpatches.Patch(color='blue', label='Class 2')])
    t1 = "Original plot before classifying for dimension: "
    t2 = ''.join(str(i) for i in dimension)
```

```
    plt.title(t1 + t2)
    plt.savefig(name + " Original plot.png")

    plt.figure(2)
    for row in x_test:
        if row[-1] == 1:
            plt.plot(row[1],row[2], 'ro')
        else:
            plt.plot(row[1],row[2], 'bo')
    plt.legend(handles=[mpatches.Patch(color='red', label='Class
    ↪  1'), mpatches.Patch(color='blue', label='Class 2')])
    t1 = "Nearest neighbour classifier for dimension: "
    t2 = ''.join(str(i) for i in dimension)
    plt.title(t1 + t2)
    plt.savefig(name + " NN plot.png")

    plt.figure(3)
    for row in result_MER:
        if row[-1] == 1:
            plt.plot(row[1],row[2], 'ro')
        else:
            plt.plot(row[1],row[2], 'bo')
    plt.legend(handles=[mpatches.Patch(color='red', label='Class
    ↪  1'), mpatches.Patch(color='blue', label='Class 2')])
    t1 = "Minimum error rate classifier for dimension: "
    t2 = ''.join(str(i) for i in dimension)
    plt.title(t1 + t2)
    plt.savefig(name + " MER plot.png")

    plt.figure(4)
    for row in result_LSM:
        if row[-1] == 1:
            plt.plot(row[1],row[2], 'ro')
        else:
            plt.plot(row[1],row[2], 'bo')
    plt.legend(handles=[mpatches.Patch(color='red', label='Class
    ↪  1'), mpatches.Patch(color='blue', label='Class 2')])
    t1 = "Least Square method classifier for dimension: "
    t2 = ''.join(str(i) for i in dimension)
    plt.title(t1 + t2)
    plt.savefig(name + " LSM plot.png")

    plt.show()
```

```
if __name__ == "__main__":
    data_file = "ds-1.txt"
    #data_file = "ds-2.txt"
    #data_file = "ds-3.txt"
    N_ROWS, N_COLLUMS, training_set, test_set = readData(data_file)

    best_error_rate_dimension = []
    for current_dimension in range(N_COLLUMS - 1):
        dimension_matrix = constructDimentionMatrix(current_dimension, N_COLLUMS - 1)
        result_NN, min_error_rate, min_error_rate_dimension = nearestNeighbour(training_set, test_set, N_ROWS, dimension_matrix)
        best_error_rate_dimension.append(min_error_rate_dimension)
        print("Best error rate: ", "%.2f" % min_error_rate, " for dimension: ", min_error_rate_dimension)

    minimumErrorRate(training_set, test_set, N_ROWS, best_error_rate_dimension)
    leastSquareMethod(training_set, test_set, N_ROWS, N_COLLUMS, best_error_rate_dimension)

    plotResults(best_error_rate_dimension, training_set, test_set, N_ROWS, N_COLLUMS, data_file)
```

A.2 Nærmeste nabo klassifisator script

[h!t]

Listing 2: nearestNeighbour.py

```
import numpy as np



def nearestNeighbour(training_set, test_set, N_ROWS, dimension_matrix):
    print("\n# ========== Running: Nearest neighbour classifier ========== #")
    min_error_rate = float('inf')

    for dimension in dimension_matrix:
        error = 0
```

```python

        for test_row in range(N_ROWS):
            min_dinstance = float('inf')

            for training_row in range(N_ROWS):
                distance = abs(np.linalg.norm(test_set[test_row][1:-1]*dimension - training_set[training_row][1:]*dimension))
                if distance < min_dinstance:
                    min_dinstance = distance
                    test_set[test_row][-1] = training_set[training_row][0]


            # Estimating error rate
            if test_set[test_row][-1] != test_set[test_row][0]:
                error += 1
        error_rate = error/N_ROWS

        if error_rate < min_error_rate:
            min_error_rate = error_rate
            min_error_rate_dimension = dimension

        print("Minimum error rate: ", "%.2f" % error_rate, " for dimension: ", dimension)

    return test_set, min_error_rate, min_error_rate_dimension
```

A.3 Minimum feilrate klassifikator script

[h!t]

Listing 3: minimumErrorRate.py

```python
import numpy as np



def minimumErrorRate(training_set, test_set, N_ROWS, best_error_rate_dimension):
    print("\n# ========== Running: Minimum error rate classifier ========= #")

    error_rate_storage = []

    for dimension in best_error_rate_dimension:
```

```
        # Restructuring data sets
        x_training = np.zeros([N_ROWS, np.count_nonzero(dimension == 1) + 1])
        x_training[:, 0] = training_set[:, 0]
        x_test = np.zeros([N_ROWS, np.count_nonzero(dimension == 1) + 2])
        x_test[:, 0] = training_set[:, 0]
        index = 1
        for i in range(len(dimension)):
            if dimension[i] == 1:
                for j in range(N_ROWS):
                    x_training[j][index] = training_set[j][i + 1]
                    x_test[j][index] = test_set[j][i + 1]
                index += 1
        x_training = [x_training[x_training[:, 0] == 1], x_training[x_training[:, 0] == 2]]


        # Calculate mean
        mu = [x_training[0][:, 1:].mean(axis=0), x_training[1][:, 1:].mean(axis=0)]

        # Calculate covariance
        Sigma = [np.cov(x_training[0][:, 1:], rowvar=False), np.cov(x_training[1][:, 1:], rowvar=False)]

        # W_i = -1/2*Sigma_i^-1      i = 1,...,c
        W = [-np.linalg.inv(np.atleast_2d(Sigma[0]))/2, -np.linalg.inv(np.atleast_2d(Sigma[1]))/2]

        # w = Sigma_i^-1*mu_i    i = 1,...,c
        w = [np.matmul(np.linalg.inv(np.atleast_2d(Sigma[0])), mu[0]),
             np.matmul(np.linalg.inv(np.atleast_2d(Sigma[1])), mu[1])]

        # w_0 = -1/2*mu_i^T*Sigma_i^-1*mu_i - 1/2*ln(|Sigma_i|) + ln(P(omega_i))     i = 1,...,c
        w_0 = [-np.matmul(np.matmul(np.transpose(mu[0]), np.linalg.inv(np.atleast_2d(Sigma[0]))), mu[0]) / 2
               - np.log(np.linalg.det(np.atleast_2d(Sigma[0]))) / 2 + np.log(0.5),
               -np.matmul(np.matmul(np.transpose(mu[1]), np.linalg.inv(np.atleast_2d(Sigma[1]))), mu[1]) / 2
```

```
                    - np.log(np.linalg.det(np.atleast_2d(Sigma[1]))) / 2 + np.log(0.5)]

        for j in range(N_ROWS):
            # g_i = x^T*W_i*x + w^T*x + w_0_i    i = 1,...,c
            g_1 = np.matmul(np.matmul(np.transpose(x_test[j][1:-1]), W[0]), x_test[j][1:-1]) + \
                  np.matmul(np.transpose(w[0]), x_test[j][1:-1]) + w_0[0]
            g_2 = np.matmul(np.matmul(np.transpose(x_test[j][1:-1]), W[1]), x_test[j][1:-1]) + \
                  np.matmul(np.transpose(w[1]), x_test[j][1:-1]) + w_0[1]

            if (g_1 - g_2) >= 0:
                x_test[j][-1] = 1
            else:
                x_test[j][-1] = 2


        # Esimating error rate
        error = 0
        for i in range(N_ROWS):
            if x_test[i][-1] != x_test[i][0]:
                error += 1
        error_rate = error / N_ROWS
        error_rate_storage.append(error_rate)

        print("Error rate: ", "%.2f" % error_rate, " for dimension: ", dimension)

    return x_test
```

A.4 Minste kvadraters metode script

[h!t]

Listing 4: leastSquareMethod.py

```
import numpy as np



def calculateY(x_training, N_ROWS):
```

```python
    Y = np.zeros([N_ROWS, len(x_training[0])])
    for i in range(len(Y)):
        Y[i][0] = 1
        Y[i][1:] = x_training[i][1:]
    return Y



def calculateB(x_training, N_ROWS):
    b = np.zeros(N_ROWS)
    for i in range(len(x_training)):
        if x_training[i][0] == 1:
            b[i] = 1
        else:
            b[i] = -1
    return b



def leastSquareMethod(training_set, test_set, N_ROWS, N_COLLUMS, best_error_rate_dimension):
    print("\n# ========== Running: Least square method classifier ======== # ")

    error_rate_storage = []

    for dimension in best_error_rate_dimension:
        # Restructuring data sets
        x_training = np.zeros([N_ROWS, np.count_nonzero(dimension == 1) + 1])
        x_training[:, 0] = training_set[:, 0]
        x_test = np.zeros([N_ROWS, np.count_nonzero(dimension == 1) + 2])
        x_test[:, 0] = training_set[:, 0]
        index = 1
        for i in range(len(dimension)):
            if dimension[i] == 1:
                for j in range(N_ROWS):
                    x_training[j][index] = training_set[j][i + 1]
                    x_test[j][index] = test_set[j][i + 1]
                index += 1


        # Create b
        b = calculateB(x_training, N_ROWS)

```

```python
        # Create Y
        Y = calculateY(x_training, N_ROWS)

        # a = (Y^T*Y)^-1*Y*b^T
        a = np.matmul(np.matmul(np.linalg.inv(np.matmul(np.transpose(Y), Y)), np.transpose(Y)), b)

        for i in range(N_ROWS):
            # g = a^T*y
            g = np.matmul(np.transpose(a), Y[i])

            if g >= 0:
                x_test[i][-1] = 1
            else:
                x_test[i][-1] = 2


        # Esimating error rate
        error = 0
        for i in range(N_ROWS):
            if x_test[i][-1] != x_test[i][0]:
                error += 1
        error_rate = error/N_ROWS
        error_rate_storage.append(error_rate)

        print("Error rate: ", "%.2f" % error_rate, " for dimension: ", dimension)

    return x_test
```