Optimization exam

Q1

Explain what is meant by a convex optimization problem and how to determine if an optimization problem is convex. You can use Exercise 1 connected to the lecture ”Convex optimization” as a guide for your explanation. (kursusgang 1-3?)

A **convex optimization problem** is a special class of optimization problems where the objective function and the feasible set (defined by constraints) have a structure that guarantees any **local minimum is also a global minimum**. This property makes convex problems particularly tractable and efficient to solve.

Exercise 1

Consider a transmitter at a location $\boldsymbol{x}_t = (-2, -4)$. It is required to find the optimal location to place a receiver $\boldsymbol{x}_r$ within a specified area of land such that the received signal's power is maximized. The received signal's power is inversely proportional to the distance between the transmitter and the receiver:

$$P(\boldsymbol{x}) \propto \frac{1}{d} \tag{1}$$

The allowable area to place the receiver is given by the following constraints in the euclidean plane:

$$c_1(\boldsymbol{x}) = -x_1^2 - (x_2 + 4)^2 + 16 \geq 0 \tag{2}$$
$$c_2(\boldsymbol{x}) = x_1 - x_2 - 6 \geq 0 \tag{3}$$

(a) Formulate the optimization problem (Write-up the cost function with the constraints). Is the **optimization problem** convex?

look if all constraints and loss functions are convex ie hessian positive semidefinite

form of constraint	condition fo convexity
f(x) <= 0	f(x) is convex
f(x) >= 0	f(x) is concave

```
syms x1 x2
f = (x1+2)^2 + (x2+4)^2;
hessian(f, [x1, x2])
```

```
ans =
```

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

```
%-x1^2 - (x2+4)^2 + 16 >= 0
% ==> x1^2 + (x2+4)^2  <= 16
```

```
c1 = x1^2 + (x2+4)^2;
hessian(c1, [x1, x2])
```

ans =

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

```
%x1 - x2 - 6 >= 0
% ==>  - x1 + x2 <= 6
c2 = -x1 + x2;
hessian(c2, [x1, x2])
```

ans =

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

(b) Consider now an additional constraint defined as:

$$c_3(\boldsymbol{x}) = x_1^2 + (x_2 + 6)^2 \geq 2. \tag{4}$$

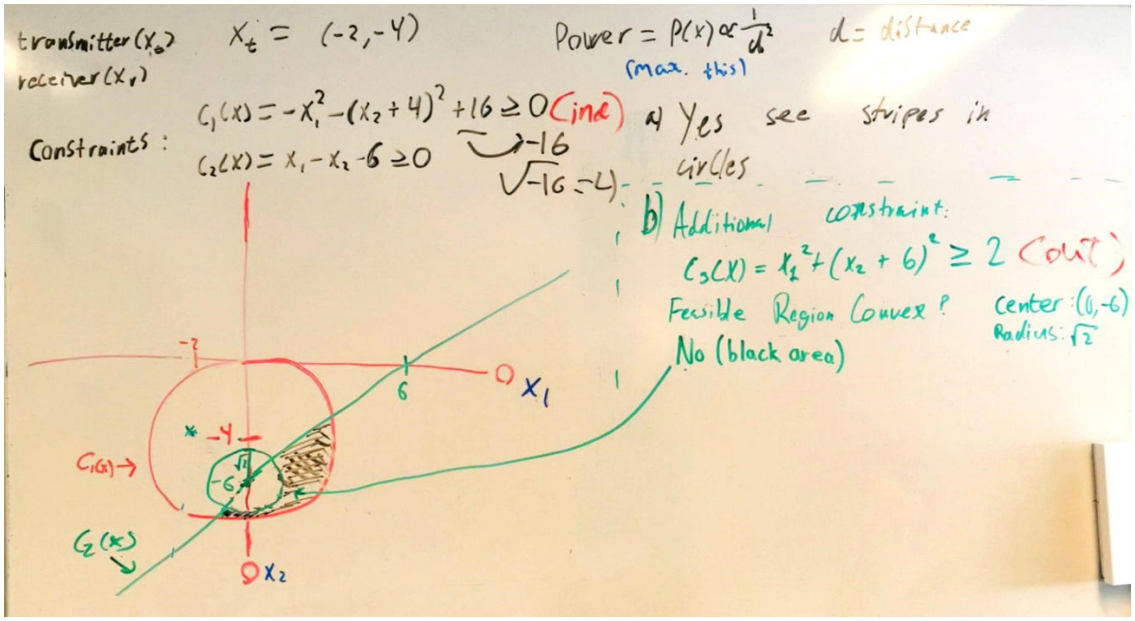
Is the **optimization problem** convex?

```
%x1^2 + (x2 + 6)^2 >= 2
%-x1^2 - (x2 + 6)^2 -2 <= 0
c3 = -x1^2 - (x2 + 6)^2 - 2;
hessian(c3, [x1, x2])
```

ans =

$$\begin{pmatrix} -2 & 0 \\ 0 & -2 \end{pmatrix}$$

Fra lecture 2 exercise 1



notes for convex optimizations

- take the hessian and if it is positive semidefinite for all x in the domain it is convex

Q2

Explain the steepest (or gradient) descent algorithm and the Newton (Raphson) algorithm. You can use Exercise 1 connected to the lecture ”Gradient methods” as a guide for your explanation.

Exercise 1

In this exercise, you are provided with the Matlab function **unconstrained_opt.m**, which can be used to solve unconstrained optimization problems. This function allows you to choose between the steepest-descent and the Newton-Raphson methods for optimization.

(a) Open the function and follow the instructions provided to complete the missing parts of the code so that it functions correctly.

(b) Use the Matlab function to minimize $f(x) = \alpha x_1^2 + x_2^2$ for $\alpha = 1$ and $\alpha = 100$. Discuss the outcomes of these two cases.

(c) Verify that applying steepest-descent to find the regression coefficients for the accumulated CO_2 data (from the previous exercises) leads to failure, while the Newton-Raphson method succeeds.

(d) Explain why the steepest-descent method fails for the CO_2 least-squares problem.

> *Hint*: Consider the condition number of $A^\top A$ in your least-squares problem and recall its definition in terms of the eigenvalues. In Matlab, you can use the command **cond** to compute the conditioning number.

result of kappa is 7.8635e+06 is very bad beacuse much bigger than 1 leading to SD

As a high condition number means slow convergence or instability for SD.

r is 1.2717e-07 which is very small

r is inv(kappa)

with **condition number**

$$r = \frac{\min \rho(H_f(x_k))}{\max \rho(H_f(x_k))}.$$

Hence

- Fast convergence for $\min \rho(H_f(x_k)) \approx \max \rho(H_f(x_k))$ (implying $r \approx 1$).
- Slow convergence for $\min \rho(H_f(x_k)) \ll \max \rho(H_f(x_k))$ (implying $r \approx 0$).

from slide 14 lec 4

What does "the Hessian tells you how fast the gradient is changing" mean?

1. **Gradient** ∇f(x)\nabla f(x)**∇f(x) = slope or direction of steepest increase**At any point xxx, the gradient vector points in the direction where the function fff increases the fastest, and its length tells you how steep that increase is.
2. **Hessian** H(x)=∇2f(x)H(x) = \nabla^2 f(x)**H(x)=∇2f(x) = the rate of change of the gradient itself**The Hessian is a matrix made of second derivatives. Each element in the Hessian measures how one component of the gradient changes as you move in a certain direction.

Example:

- Imagine you're hiking on a hill represented by the function f(x).
- The **gradient** at your current spot tells you which way is uphill and how steep that uphill is.
- The **Hessian** tells you how the steepness (the gradient) changes if you take a step in different directions.

- If the Hessian is **positive and large** in a direction, it means the hill curves **upwards sharply** in that direction — the slope will increase quickly as you move.
- If the Hessian is **close to zero** in a direction, the slope is roughly constant — the hill is more flat there.
- If the Hessian has **mixed signs** (positive and negative eigenvalues), the function curves **up in some directions and down in others** (like a saddle).

Quickie:

- Gradient = slope at a point (which way is uphill).
- Hessian = how that slope itself changes as you move around (how the hill bends).

notes

Gradient Descent:

An iterative optimization method that moves in the direction of the

negative gradient (steepest descent) of the function.

Update rule: x_{k+1} = x_k - alpha_k * grad_f(x_k)

- Simple and widely applicable.

- Slower convergence, especially near the minimum.

- Step size alpha_k must be chosen carefully.

Newton-Raphson Method:

Uses second-order (Hessian) information to find a stationary point

by approximating the function locally as a quadratic.

Update rule: x_{k+1} = x_k - inv(Hessian_f(x_k)) * grad_f(x_k)

- Faster (quadratic) convergence near the optimum.

- Computationally expensive due to Hessian calculation and inversion.

- Can fail if Hessian is not positive definite.

Feature	Gradient Descent	Newton's Method
Uses gradient	Yes	Yes
Uses Hessian	No	Yes
Convergence rate	Linear (slow near minimum)	Quadratic (fast near minimum)
Computational cost	Low (simple updates)	High (requires Hessian and matrix inversion)
Sensitivity	Sensitive to step size	May diverge if Hessian is not positive definite
Scalability	Good for large-scale problems	More suitable for small to medium-sized problems

Q3

Explain the Gauss-Newton method. You can use Exercise 3 connected to the lecture ”Gradient methods” as a guide for your explanation.

gauss newton for non lineaer least square

$$\min_x ||f(x)||^2 = \sum_{i=1}^{3} \left(||x - b_i||^2 - d_i^2 \right)^2$$

Exercise 3

A robot estimates its global position $x \in \mathbb{R}^2$ by measuring its distance to three fixed beacons (landmarks) located at known positions $b_1, b_2, b_3 \in \mathbb{R}^2$. These measurements are noisy, and our goal is to find the robot's actual position by solving

$$F(x) = \begin{bmatrix} \|x - b_1\|_2^2 - d_1^2 \\ \|x - b_2\|_2^2 - d_2^2 \\ \|x - b_3\|_2^2 - d_3^2 \end{bmatrix} = 0.$$

This problem can be tackled with the Gauss-Newton algorithm.

> *Hint*: For $x \in \mathbb{R}^n$, we have $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.

(a) How can this problem be reformulated as an optimization task?

(b) Follow the instructions in the function **Gauss_Newton.m** to fill in the missing parts of the code.

(c) In the script **Robot_loc_GN.m**, complete the missing code for the Jacobian $J_f(x)$ of $F(x)$.

(d) Run the **Robot_loc_GN.m** script and examine the results.

Golden Section Search (GSS) Analysis:

- GSS is used at each Gauss-Newton iteration to determine the optimal step size

along the direction d_k.

- The number of iterations required by GSS decreases over time:

- Initially: 30 iterations

- Then: 19, 6, and eventually 0 iterations

- This behavior suggests that the algorithm is approaching the minimum,

so the initial bracketing interval isn't shrinking much — GSS terminates quickly.

- The solution x (in 2D) appears to converge to:

x ≈ [1.0314, 1.0079]

- The final value of the cost function is:

f(x) = 0.0061

This small value indicates that the algorithm achieved a good fit (low residual error)

- Overall, the Gauss-Newton with GSS successfully converged within 9 iterations.

notes

% Gauss-Newton Method:

% An iterative method for solving nonlinear least squares problems of the form:

% min_x (1/2) ||r(x)||^2

% where r(x) is a vector of residuals.

% Update rule:

% x_{k+1} = x_k - (J(x_k)^T J(x_k))^(-1) J(x_k)^T r(x_k)

% where J(x_k) is the Jacobian matrix of r(x) evaluated at x_k.

% Advantages:

% - Efficient for nonlinear least squares problems.

% - Avoids computing the full Hessian matrix.

% Disadvantages:

% - Only applicable to least squares problems.

% - Can fail if J^T J is not invertible or if the residuals are highly nonlinear.

this is a least square problem and use gauss newton to solve it

$$\min_x ||Ax - b||^2$$

Q4

Explain how to solve convex optimization problems using the Karush-KuhnTucker condition. You can use Exercise 1 connected to the lecture ”Constrained Optimization II” as a guide for your explanation.

Exercise 1

Two voltage sources connected in series with voltages x_1 [v] and x_2 [v] respectively are required to provide a constant voltage $V_L = 1$ [v] for a load. The cost for using the first one is x_1^2. The voltage of the first one is upper bounded by 0.5 [v]. The cost of using the second one is $2x_2^2$. The voltage of the second one is not upper bounded.

(a) Find the candidate solutions using the KKT conditions.

Case 3: x1 = 0 ⇒ mu2 ≥ 0 (active), x2 = 1 from equality constraint

Stationarity conditions:

From ∂L/∂x1: 0 + lambda + mu1 - mu2 = 0 → (A)

From ∂L/∂x2: 4*x2 + lambda = 0 → 4 + lambda = 0 ⇒ lambda = -4 → (B)

Plug (B) into (A): -4 + mu1 - mu2 = 0 ⇒ mu1 = 4 + mu2 ⇒ mu1 ≥ 4

Complementary slackness:

x1 = 0 ⇒ mu2 ≥ 0 (OK)

x2 = 1 > 0 ⇒ mu3 = 0

x1 = 0 ≤ 0.5 ⇒ mu1 ≥ 0 (OK)

All KKT conditions are satisfied.

Cost: f = x1^2 + 2*x2^2 = 0 + 2*1^2 = 2

Compare to Case 2 (x1 = 0.5, x2 = 0.5): f = 0.25 + 0.5 = 0.75

⇒ This case is feasible but not optimal (higher cost)

% Final Answers:

(a) Optimal solution from KKT:

x1 = 0.5, x2 = 0.5

Lagrange multipliers: lambda = -2, mu1 = 1, mu2 = 0, mu3 = 0

(b) Are the KKT conditions sufficient for this problem? Justify your answer.

The objective function f(x1, x2) = x1^2 + 2*x2^2 is strictly convex.

The constraints are all linear (affine): x1 + x2 = 1, x1 ≤ 0.5, x1 ≥ 0, x2 ≥ 0.

Therefore, the entire optimization problem is convex.

If the optimization problem is convex_a then the KKT necessary conditions are

also sufficient (for a global minimizer). slide 10 lec constrained opti II

Q5

Name a parametric method and a nonparametric method with an explanation on the major differences between them. You can use Exercise 1 connected to the lecture ”Parametric and Nonparametric Methods” as a guide to your explanation

Two broad families of models:

- **Parametric:** Models have fixed, finite set of parameters θ (e.g., mean and variance of a Gaussian).
- **Nonparametric:** The parameters' complexity of the models grows with data (e.g., histograms, kernel density, k-NN).

nonparametric does not mean
NONE-parametric (we still have parameters).

Parametric Methods

- Assume a specific form or function with fixed parameters θ (e.g., Gaussian distribution with mean and variance, linear model with slope and intercept).
- A **fixed number of parameters** θ, independent of the dataset size.
- Can be more robust to memory limitations.
- Often faster once trained, but can be less flexible if the assumed form is poor.

Propose a family of distributions/models with a **fixed set of parameters** θ.

Estimate θ from data using:

- Maximum Likelihood Estimation (MLE)
- Maximum A Posteriori (MAP)
- The posterior's mean.
- Least-Squares.

Examples:

- Linear/Logistic Regression (coefficients as parameters).
- Gaussian distribution (mean, variance).

non parametric

No fixed number of parameters: the complexity can grow with N.

Typically store or reference the training data for inference.

Examples:

- Histograms (fixed bin widths).
- Kernel Density Estimation (KDE).
- k-Nearest Neighbors (k-NN).

Hyperparameters are the parameters we do not estimate. They are chosen by domain knowledge or other means.

Exercise 1

In this exercise, you will use the Breast Cancer Wisconsin data set to build a classifier for breast tumor diagnosis based on features extracted from a digitized image of a fine needle aspirate of a breast mass (more information can be found Here. The code for the exercise in Matlab is Classifier_NBC_KNN.m .

(a) Read through the file Classifier_NBC_KNN.m and fill in the missing parts.

(b) Discuss the results and the confusion matrix for each classifier.

Confusion Matrix

A confusion matrix is a table that summarizes the performance of a classification model.

For a binary classifier, it is structured as:

Actual \Predicted	Positive	Negative
Positive	#TP	#FN
Negative	#FP	#TN

Key Components:

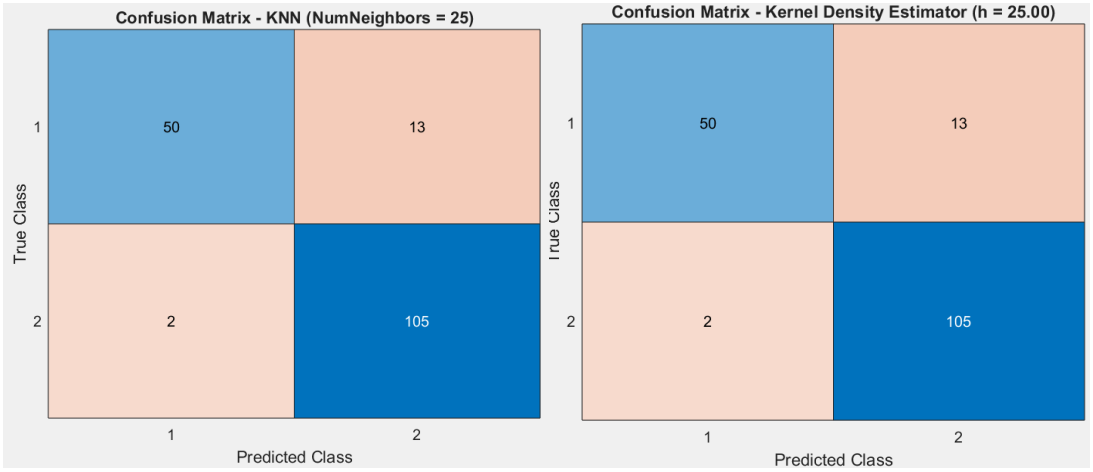
- **TP (True Positives)**: Correctly predicted positive cases.
- **FP (False Positives)**: Negative cases incorrectly predicted as positive.
- **TN (True Negatives)**: Correctly predicted negative cases.
- **FN (False Negatives)**: Positive cases incorrectly predicted as negative.
- It can be also normalized by the total number of predictions.

$$\text{Sensitivity} = \frac{\#\ \text{TP}}{\#\ \text{TP}+\#\text{FP}}$$

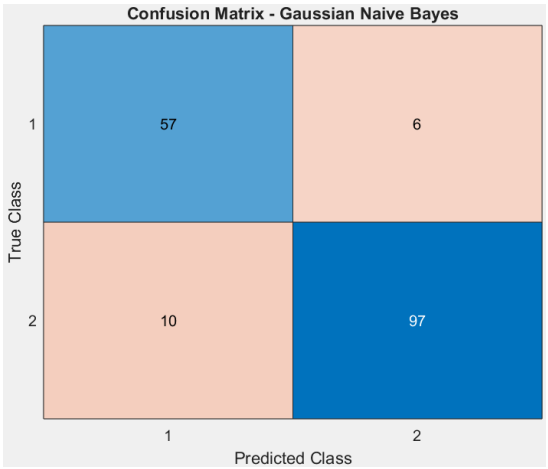
$$\text{Specificity} = \frac{\#\ \text{TN}}{\#\ \text{TN}+\#\text{FN}}$$

For multi-class classification, the matrix generalizes to a $C \times C$ table.

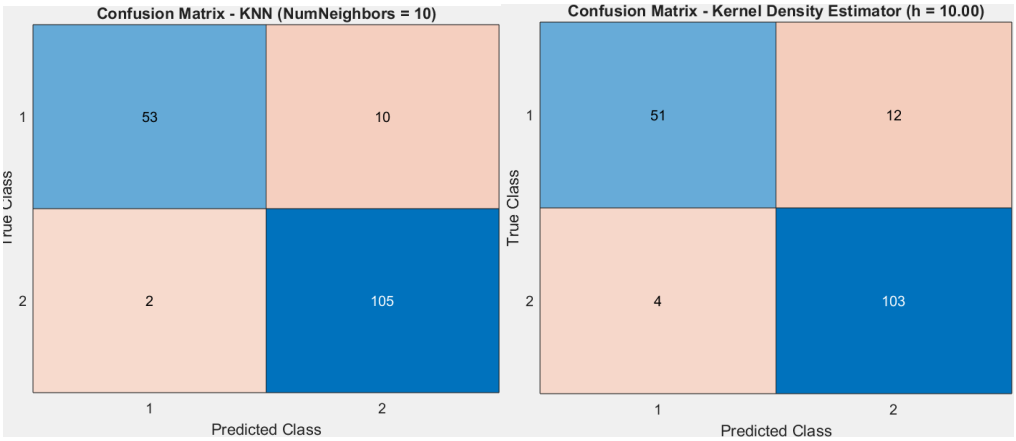
non-parametric

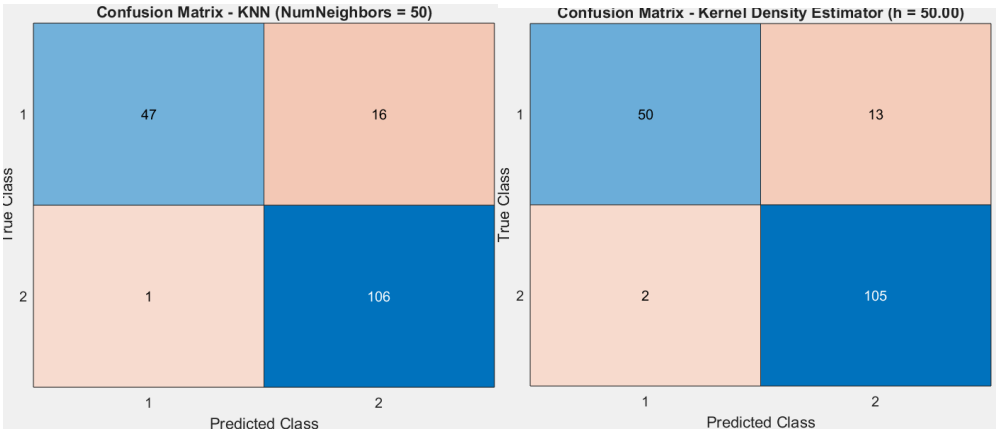


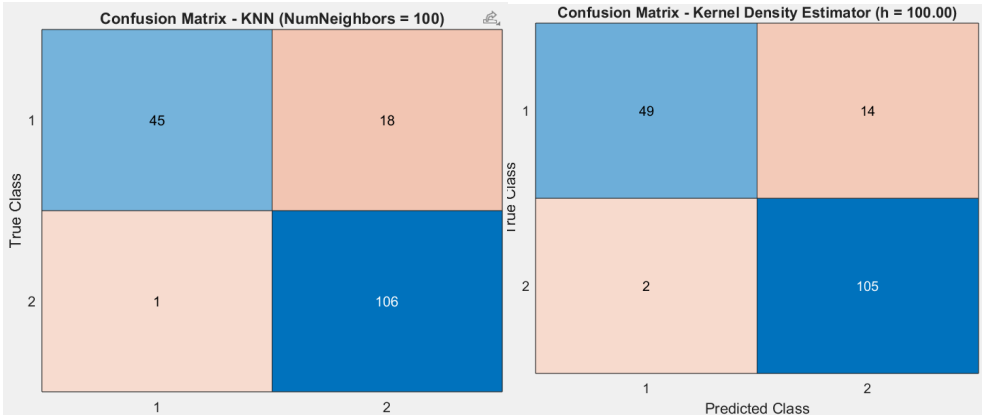
parametric

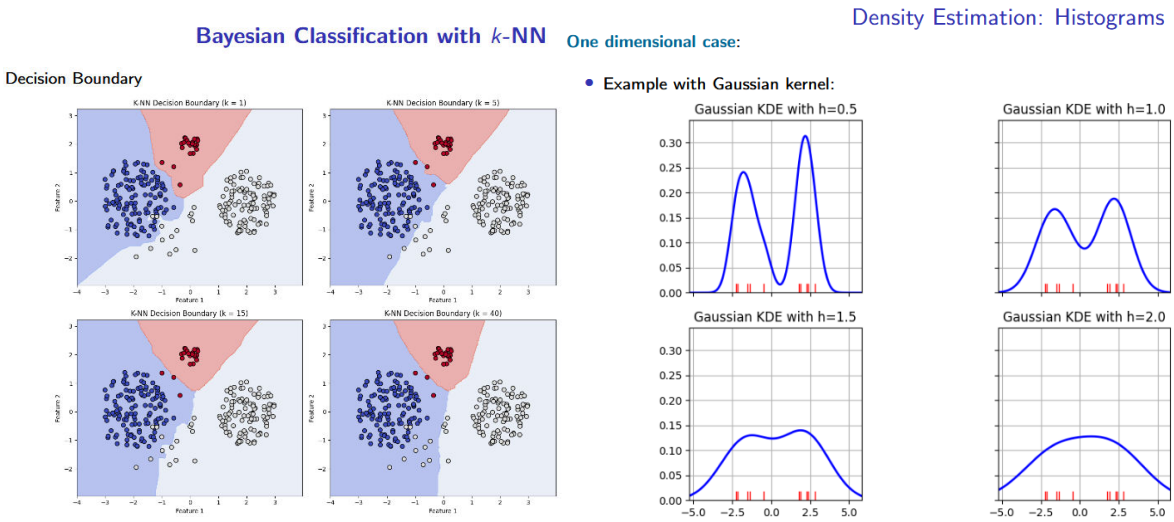


(c) Experiment with the width parameter for the kernel density estimator and the number of neighbors for the k-nearest neighbor.









KNN (K-Nearest Neighbors):

- **Small K (few neighbors):**
- High model complexity.
- Sensitive to noise.
- **Risk of overfitting**.
- **Large K (many neighbors):**
- Smoother, more general decisions.
- **Risk of underfitting** if K is too large.

KDE (Kernel Density Estimation):

- **Small bandwidth (narrow kernel):**
- Sharp, highly localized density estimates.
- Can model fine detail, but also **prone to overfitting**.
- **Large bandwidth (wide kernel):**
- Smooths over more data.
- Can **underfit** and miss structure in the data.

notes

Feature	Parametric (e.g., Linear Regression)	Nonparametric (e.g., k-NN)

Assumptions	Strong (fixed form, specific distribution)	Minimal (data-driven, flexible form)
Complexity	Fixed number of parameters	Grows with training data
Training time	Fast (solve for parameters once)	Often slower (may involve storing all data)
Flexibility	Less flexible, may underfit	More flexible, can overfit without tuning

Q6

Explain the general model of linear regression with different possible regularizations. You can use Exercise 1 connected to the lecture ”Linear Regression” as a guide to your explanation.

Exercise 1

In this exercise, we will revisit the problem of performing linear regression to fit the measured concentration of CO_2 in parts per million in Hawaii over the time span between 1974 to 2020. The function which will be fitted in this exercise is of this structure.

$$g(t) = w_0 + w_1\phi_1(t) + w_2\phi_2(t) + ... + w_n\phi_n(t) \quad (1)$$

where $\phi_i(t)$ is a base function and w_i is a linear coefficient for $g(t)$. The base functions are known and the coefficients will be solved for using the least squares method but with different regularizers.

We will fit the function to the first half of the data and then see what happens when we extrapolate it into the future. Further, we will take a look at what can happen if the function is overfitted.

(a) Using the basis functions you chose earlier, solve for the coefficients with ridge regression and lasso regression for different regularization weights.

(b) Create 100 different basis functions of your own choice and fit a new set of coefficients to the data in **data_fit** with the different regularization options. What happens now when you extrapolate the function into the future?

Helpful tips:

- Note, for the ridge regression, use the closed-form solution. For the Lasso regression, then either use CVX (`https://cvxr.com/cvx/doc/install.html#`) or Matlab's Problem-Based Optimization Workflow (`https://se.mathworks.com/help/optim/ug/optim.problemdef.optimizationproblem.html`).
- The original lasso problem is written as

$$\min_{w \in \mathbb{R}^M} \quad \frac{1}{2}\|\Phi(\boldsymbol{x})w - \boldsymbol{y}\|_2^2 + \lambda \|w\|_1.$$

  Since the L_1 norm is not differentiable at zero, we introduce auxiliary variables s_i so that $s_i \geq |w_i|$ for $i = 0, \ldots, M-1$. This allows us to rewrite the L_1 term as $\sum_{i=0}^{M-1} s_i$ with linear constraints. The reformulated optimization problem becomes

$$\begin{aligned} \min_{w \in \mathbb{R}^M,\, s \in \mathbb{R}^M} \quad & \frac{1}{2}\|\Phi(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \lambda \sum_{i=0}^{M-1} s_i, \\ \text{s.t.} \quad & -s_i \leq w_i \leq s_i, \quad i = 0, \ldots, M-1, \\ & s_i \geq 0, \quad i = 1, \ldots, M-1. \end{aligned}$$

notes

Model	Objective Function	Effect
Linear Regression	∥Xβ−y∥2| X\beta - y |^2	Fits data, risk of overfitting
Ridge (L2)	∥Xβ−y∥2+λ∥β∥2| X\beta - y |^2 + \lambda | \beta |^2	Shrinks coefficients
Lasso (L1)	∥Xβ−y∥2+λ∥β∥1| X\beta - y |^2 + \lambda | \beta |_1	Feature selection (sparse)
Elastic Net	∥Xβ−y∥2+λ1∥β∥1+λ2∥β∥2| X\beta - y |^2 + \lambda_1 | \beta |_1 + \lambda_2 | \beta |^2	Combines L1 and L2

Q7

Explain the general model of logistic regression and the importance of feature selection. You may use Exercise 1 connected to the lecture ”Linear Classification” as a guide to your explanation.

Q8

Explain the general model of multilayer perceptrons and their expressive power. Mention an algorithm to train them. You can use Exercise 2 from the lecture connected to ”Introduction to Neural Networks” as a guide to your explanation.

Exercise 1

Consider the simple feedforward neural network in figure 1. The hidden layer's activation function is the rectified linear unit function (ReLU), and the activation function for the output layer is the sigmoid function. For a data point y_n, assume that we use the cross-entropy loss for the output $o \in [0, 1]$ (o represents the probability $\mathrm{P}(1 \mid x, w)$):

$$L_n(w) = -y_n \ln(o) - (1 - y_n) \ln(1 - o)$$

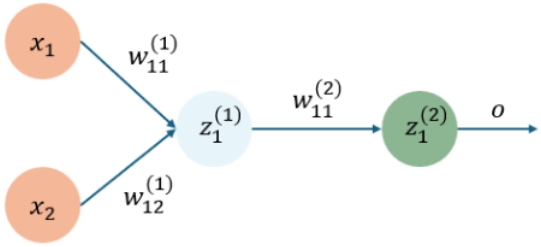


Figure 1: Simple feedforward neural network

(a) Write the gradients of the loss $L_n(w)$ with respect to the weights in the networks using back-propagation.

(b) Choose one training example (pick an $x \in \mathbb{R}^2$ and a $y \in \{1, 0\}$) and perform one update step for the weights with a suitable step size.

Q9

Explain the general concept of Principal Component Analysis (PCA) and how it can be computed using the Singular Value Decomposition (SVD). You can use Exercise 1 connected to the lecture "PCA and SVD" as a guide to your explanation.

Exercise 1

In this exercise, you will use the Fisher Iris dataset to perform Principal Component Analysis (PCA) via Singular Value Decomposition (SVD). The dataset comprises 150 samples across three species (setosa, versicolor, virginica), with four features each (sepal length, sepal width, petal length, petal width). You are provided with the MATLAB file:

`PCA_SVD_IRIS_ex.m`

The file aims to implement PCA using SVD.

(a) Read through the file and fill in the missing parts.

(b) Discuss the results.

(extra-c) Compare the results with Matlab's own PCA function.