QCQT-QE9 Quantum Machine Learning 2024-2025

Prof. I. Mpoutalis - Topalidis Dimitrios (Registration number 60687)
2025 Questions in Supervised Machine Learning

Question 1

Describe the basic concepts of Supervised Learning and list the main categories of its application

Supervised machine learning is a core machine learning paradigm.
Its training set is a labeled dataset.
There are two categories of supervised machine learning applications, that are identified by the type of the labels:

1. **Classification**. In this case the labels are categorical variables ($y \in \mathbb{L}$) e.g. email spam detection, image recognition, disease diagnosis
2. **Regression**. In this case the labels are continuous variables ($y \in \mathbb{R}$) e.g. predicting house prices, forecasting stock prices, estimating temperature

The knowledge extracted from supervised learning is often utilized for prediction and recognition.
Some of the widely used algorithms for supervised learning are: k-Nearest Neighbours, Decision Trees, Naive Bayes, Logistic Regression and Support Vector Machines

Supervised machine learning assumes that there exists an unknown **target function f** that maps each data sample to its corresponding label.
It begins with a **hypothesis set H**, which contains several hypotheses **h** about the target function.
The goal of supervised machine learning is to find the best possible $h \in H$, which is called **final hypothesis g**, which approximates the target function f.
In order to find the final hypothesis g, a **learning algorithm A** is defined, which includes the objective function and the optimization methods.
The final hypothesis g, is used for future prediction.

Question 2

What is empirical risk minimization?

Suppose we have a training dataset with input-output pairs:

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Our goal with supervised machine learning is to estimate a model $h(x, \theta)$. Note that the model can be parametric, with θ being the parameters vector.
We define as **Loss function** $L\big(y, f(x, \theta)\big)$ a function that takes as input the real label and the prediction and produces a non-negative number, which we call **loss**, representing how much error we have made.

The average loss over the set of N training samples is called **empirical risk**:

$$E_{emp}(h, X, y) = \frac{1}{N} \sum_{i=1}^{N} L\big(y_i, f(x_i, \theta_i)\big)$$

The goal of supervised machine learning is to find a good parameter vector θ^* that minimizes the empirical risk.

Question 3

Describe the VC dimension and give typical examples.

We say that a function f **shatters** a set of data points, if for every labelling of those data points there exists a parameter vector θ such that the function f correctly classifies them and makes no error.

The **VC dimension** of a function f is the maximum number of points that are shuttered by f. It can be infinity.
The VC dimension is a concept of the Vapnik-Chervonenkis theory which provides a general measure of complexity and proves bounds on errors as a function of complexity.

For example, the VC dimension of the set of linear classifiers in 2D is 3, because for any 4 points in 2D plane, a linear classifier can not shatter all the combinations of the points.
On the other hand, a sine function can shatter any number of points with any assignment of label. Therefore its VC dimension is infinite.

Question 4

Describe the basic concepts of regression

Regression is a type of supervised learning used to model the relationship between a data sample and its corresponding label. In contrast to classification, the range of the possible label values is not discrete, but it can take any value in $\mathbb{R}$. Therefore, the goal with regression is to find a mapping f from one real-valued space to another.

The approximating function f is called **the regression function**

The performance of the regression function is usually evaluated using the residual sum of squares:

$$E = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

It is very common to search a **parametric function estimate** among a family of functions characterized by parameters. Then the optimization is performed on the parameter space. Function classes can be further refined to **linear** and **non-linear** estimates.

Question 5

Write a python program that repeats the Linear Regression example on page 15 of the slides. Use the data shown in the graph.

For this question I wrote a python program that performs Linear regression on the data that I visually extracted from the graph in page 15 of the supervised machine learning slides, which can be found here `https://eclass.duth.gr/modules/document/file.php/1031590/Qml_4-SupML.pdf`
You will find the respective jupyter notebook with all the code in the following link: `https://github.com/topalidisd-qcqt-duth/qe9-assignment3/blob/main/question5.ipynb`
In order to run the code you must have python installed in your system as well as the libraries **numpy**, **matplotlib**

```
import numpy as np
import matplotlib.pyplot as plt

x_data = np.array([0.01, 0.13, 0.18, 0.21, 0.3, 0.43, 0.48, 0.61,0.68, 0.79, 0.82, 0.85, 0.9,
    0.96])
y_data = np.array([0.25, 0.33, 0.35, 0.36, 0.38, 0.4, 0.41, 0.46,0.48, 0.54, 0.61, 0.62,
    0.7,0.72])
```

```
x_mean = np.mean(x_data)
y_mean = np.mean(y_data)

w = np.sum((x_data - x_mean) * (y_data - y_mean)) / np.sum((x_data - x_mean) ** 2)
b = y_mean - w * x_mean

x_fit = np.linspace(0, 1, 100)
y_fit = w * x_fit + b

plt.figure(figsize=(8, 5))
plt.plot(x_fit, y_fit, 'r-')
plt.plot(x_data, y_data, 'o',  mfc='none')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression example of page 15')
plt.xlim(0, 1)
plt.ylim(0, 1)

plt.show()
```

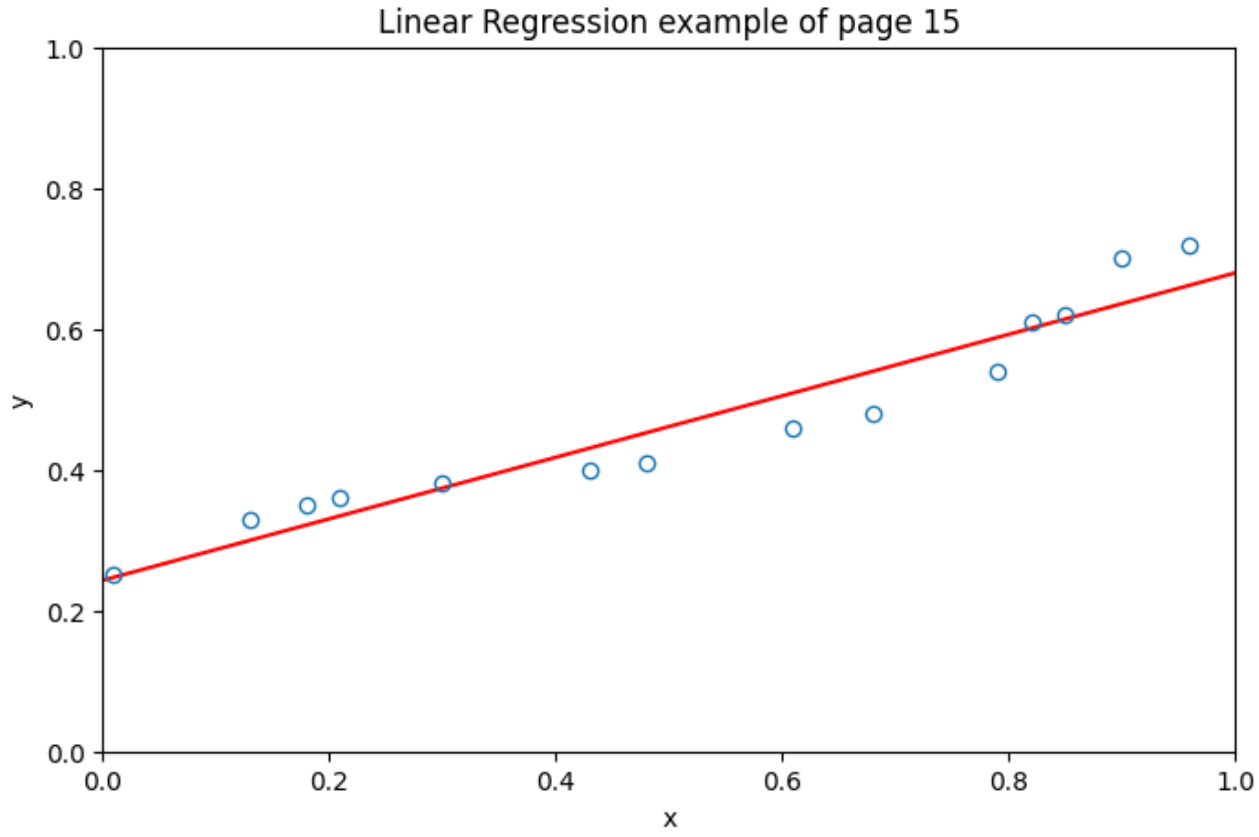

Figure 1: Result after replicating the Linear regression example of page 15 of the supervised machine learning slides

Question 6

Write a python program that solves a Linear Regression problem of multidimensional inputs. Use appropriate data.

For this question I wrote a python program that performs Linear regression on a dataset with **multidimensional**

inputs. Specifically, I chose a dataset with 3 dimensional inputs and scalar outputs.
Visualization for the results was not possible, so I just printed the weight and the bias vectors, as well as the predictions and the mean squared error.
You will find the respective jupyter notebook with all the code in the following link: `https://github.com/topalidisd-qcqt-duth/qe9-assignment3/blob/main/question6.ipynb`
In order to run the code you must have python installed in your system as well as the library **numpy**

```
import numpy as np

X = np.array([
    [1.0, 2.0, 3.0],
    [2.0, 1.0, 0.0],
    [3.0, 3.0, 1.0],
    [4.0, 5.0, 2.0],
    [5.0, 7.0, 3.0],
    [6.0, 8.0, 4.0],
    [7.0, 8.0, 3.0],
    [8.0, 9.0, 5.0]
])

y = np.array([10.0, 8.0, 12.0, 15.0, 18.0, 20.0, 22.0, 25.0])

ones = np.ones((X.shape[0], 1))
X_augmented = np.hstack((X, ones))
X_presudoinverse = np.linalg.inv(X_augmented.T @ X_augmented) @ X_augmented.T

w_augmented = X_presudoinverse@y

w = w_augmented[:-1]
b = w_augmented[-1]

y_pred = X @ w + b

print("Weights (w):", w)
print("Bias (b):", b)
print("Predictions:", y_pred)
print("Actual:", y)
print("Mean Squared Error:", np.mean((y - y_pred)**2))
```

```
Weights (w): [1.54088844 0.41595154 0.78344271]
Bias (b): 5.023725391216534
Predictions: [ 9.74684503  8.52145381 11.67768804 14.83392226 17.99015649 20.73043917
 21.48788491 25.0116103 ]
Actual: [10.  8. 12. 15. 18. 20. 22. 25.]
Mean Squared Error: 0.1579379101463908
```

Figure 2: Result of using Linear regression with multidimensional (3 dimensional) data.

Question 7

Why is Nonlinear Regression needed? What categories of this do you know?

As we said before, in regression, the models we use have the general form:

$$y = f(x)$$

In linear regression we have $f(x) = wx + b$ where **w** and **b** contain the parameters that have to be fit to the data. Very often though, linear models are not sufficient to capture the real-world phenomena. In such cases, **nonlinear models** are used and **nonlinear regression** is necessary.

In nonlinear regression, $f(x)$ can be any function. Ideally, the form of the model would be matched exactly to the underlying phenomenon. If we do not know much about the underlying nature of the process we typically turn to a few models in machine learning that are widely-used and quite effective for many problems.

The **basis functions representation** assumes the general form:

$$f(x) = \sum_k w_k b_k(x)$$

where $b_k(x)$ are the **basis functions**
Two common choices of basis functions are:

1. **Polynomials**. A simple basis for Polynomials are **monomials**, i.e. $b_0(x) = 1$, $b_1(x) = x$, $b_2(x) = x^2$, $b_3(x) = x^3$, ...
2. **Radial basis functions (RBF)**. A common 1D example are gaussians of the form $b_k(x) = e^{-(x-c_k)^2/2\sigma^2}$ where c_k is the "center" of the basis function and σ^2 represents its "width".

Note that the parameters w_k should still be estimated. For this, we can use again least-squares regression by minimizing the sum of squared residual error between model predictions and the training data outputs:

$$E(w) = \sum_i (y_i - f(x_i))^2 = \sum_i \left(y_i - \sum_k w_k b_k(x_i)\right)$$

Question 8

Mention approaches used for choosing hyperparameters in common basis functions

We presented how we can estimate the parameters w_k, but the parameters of the basis functions, which are also called **hyperparameters**, must also be somehow estimated or determined. For example, let's take the case of Radial basis functions (RBF) and specifically the gaussians we presented earlier. The hyperparameters in this case are the center and the width of the basis function. Some common heuristics to determine the basis centers are:

- Place the centers uniformly spaced in the region containing the data.
- Place one center at each data point.
- Cluster the data, and use one center for each cluster.

Some common heuristics to determine the basis widths are:

- Manually try different values of the width and pick the best by trial-and-error.
- Use the average squared distances (or median distances) to neighboring centers, scaled by a constant, to be the width.

Question 9

What do you know about the overfitting phenomenon? In what ways is it overcome?

Overfitting appears when we fit the training data extremely well but we obtain a model that produces very poor predictions on future test data whenever the test inputs differ from the training input.
This can happen for several reasons, such as:

- The problem is insufficiently constrained.
- When the model is so powerful that it can fit the data and also the random noise in the data.
- The posterior probability distribution of the unknowns is insufficiently peaked to pick a single estimate.

A successful approach to overcome overfitting is to add **prior knowledge** to the estimation problem. A very common way to achieve is, is to assume that the underlying function is likely to be smooth. This in turn means that the underlying function has small derivatives and reduces the model's complexity.
One way to add smoothness is to parameterize the model in a smooth way but this limits the expressiveness of the model.

In order to overcome this we can add **regularization**. Regularization is implemented by adding an extra term that favors smooth models to the objective function:

$$E(w) = ||y - Bw||^2 + \lambda||w||^2$$

The "smoothness-term" $\lambda||w||^2$ essentially penalizes non-smoothness caused by rapid changes in $f(x)$ that come from large weights. For this reason it is called **weight decay**.

An alternative approach is to chose a "smoothness-term" term of the form $\lambda||B_1 w||^2$ where $B_1 w = \frac{d}{dx}(Bw)$
In this case we penalize solutions that produce output predictions with large first order derivatives in respect to the inputs.

Question 10

Write a python program that repeats the example on page 29 of the slides.

For this question I wrote a python program that performs Linear regression on the data that I visually extracted from the graph in page 29 of the supervised machine learning slides, which can be found here `https://eclass.duth.gr/modules/document/file.php/1031590/Qml_4-SupML.pdf`
You will find the respective jupyter notebook with all the code in the following link: `https://github.com/topalidisd-qcqt-duth/qe9-assignment3/blob/main/question10.ipynb`
In order to run the code you must have python installed in your system as well as the libraries **numpy**, **matplotlib**

Specifically, in this example we perform **nonlinear regression** for a given dataset, using the polynomial function:

$$f(x) = w_6 x^6 + w_5 x^5 + w_4 x^4 + w_3 x^3 + w_2 x^2 + w_1 x + w_0$$

which resembles to the monomials basis function we have presented before, with degree 6.
We try first the case of nonlinear regression without regularization, using the regular least squares method. **(blue line)**
Then we try the case of nonlinear regression with regularization implemented with the weight decay technique **(red line)**
Finally we try the case of regularization, in which we add a term that penalizes solutions that produce output predictions with large first order derivatives **(green line)**
The example of the lecture notes uses $\lambda = 0.5$ for the regularization. We replicate the example for this case, but we also try for other values $0 < \lambda < 1$
We also plot the combined results.

```
import numpy as np
import matplotlib.pyplot as plt

x_data = np.array([0.2, 0.5, 2, 4, 5, 6, 7, 7.4, 8, 9, 9.5, 10])
y_data = np.array([0.1, -5, 3, 8, 5, 6.5, 15, 14, 9.2, 8.5,9.5, 9.8])
degree = 6
```

```
B = np.vstack([x_data**k for k in range(degree + 1)]).T
x_fit = np.linspace(0, 10, 500)
B_fit = np.vstack([x_fit**k for k in range(degree + 1)]).T

def regular_least_squares():
    B_pseudoinverse = np.linalg.inv(B.T @ B) @ B.T
    w_regular_least_squares = B_pseudoinverse @ y_data
    return  B_fit @ w_regular_least_squares

def weight_decay_regularization(lambda_):
    w_decay_regularization = np.linalg.inv(B.T @ B + lambda_ * np.eye(degree + 1)) @ B.T @ y_data
    return B_fit @ w_decay_regularization

def first_derivative_regularization(lambda_):
    B1 = np.vstack([k * x_data**(k - 1) if k > 0 else np.zeros_like(x_data) for k in range(degree
    + 1)]).T
    w_first_derivative_regularization = np.linalg.inv(B.T @ B + lambda_ * B1.T @ B1) @ B.T @
    y_data
    return B_fit @ w_first_derivative_regularization

def example_of_page_29(lambda_):
    y_fit_regular_least_squares = regular_least_squares()
    y_fit_weight_decay_regularization = weight_decay_regularization(lambda_)
    y_fit_first_derivative_regularization = first_derivative_regularization(lambda_)

    plt.figure(figsize=(8, 5))
    plt.plot(x_fit, y_fit_regular_least_squares, 'b-')
    plt.plot(x_fit, y_fit_weight_decay_regularization, 'r-')
    plt.plot(x_fit, y_fit_first_derivative_regularization, 'g-',)
    plt.plot(x_data, y_data, 'o', mfc='none', color='r', label='Data')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(r'Nonlinear regression examples of page 29 - $\lambda=' + str(lambda_) + '$')
    plt.xlim(0, 10)
    plt.ylim(-10, 16)
    plt.show()

example_of_page_29(0.5)
example_of_page_29(0.01)
example_of_page_29(0.1)
example_of_page_29(0.75)
example_of_page_29(0.9)
```

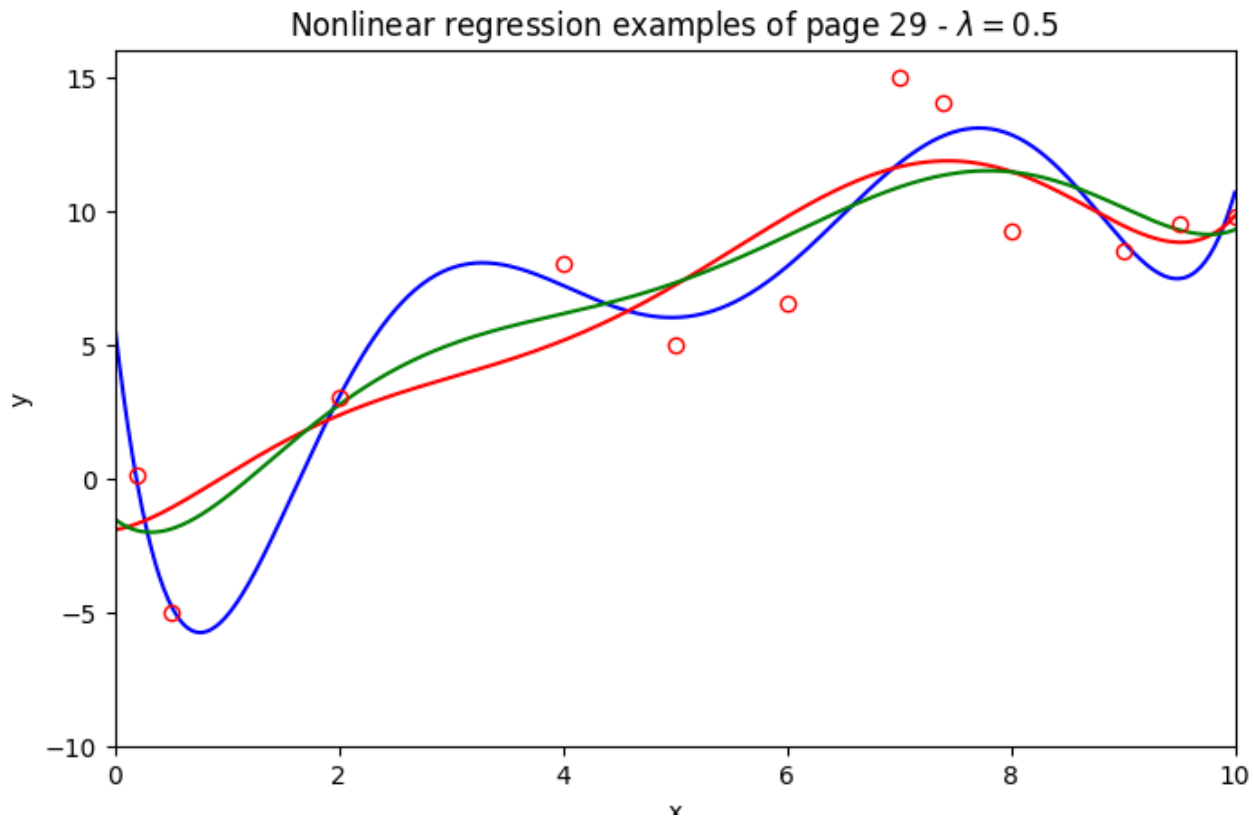

Figure 3: Result after replicating the exact nonlinear regression example of page 29 of the supervised machine learning slides, with $\lambda = 0.5$

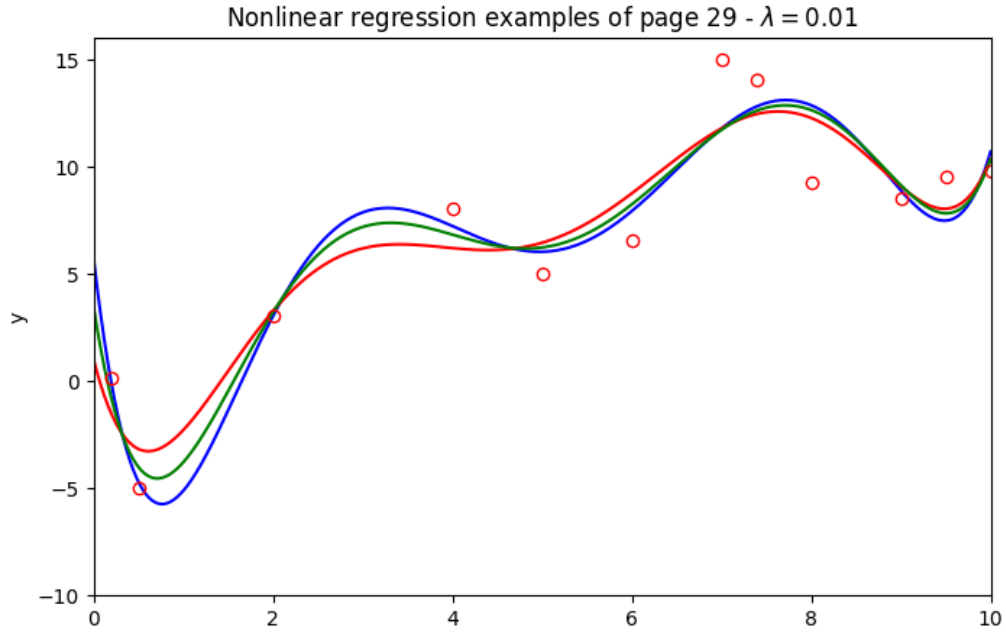

Figure 4: Result after replicating the nonlinear regression example of page 29 of the supervised machine learning slides, with $\lambda = 0.01$

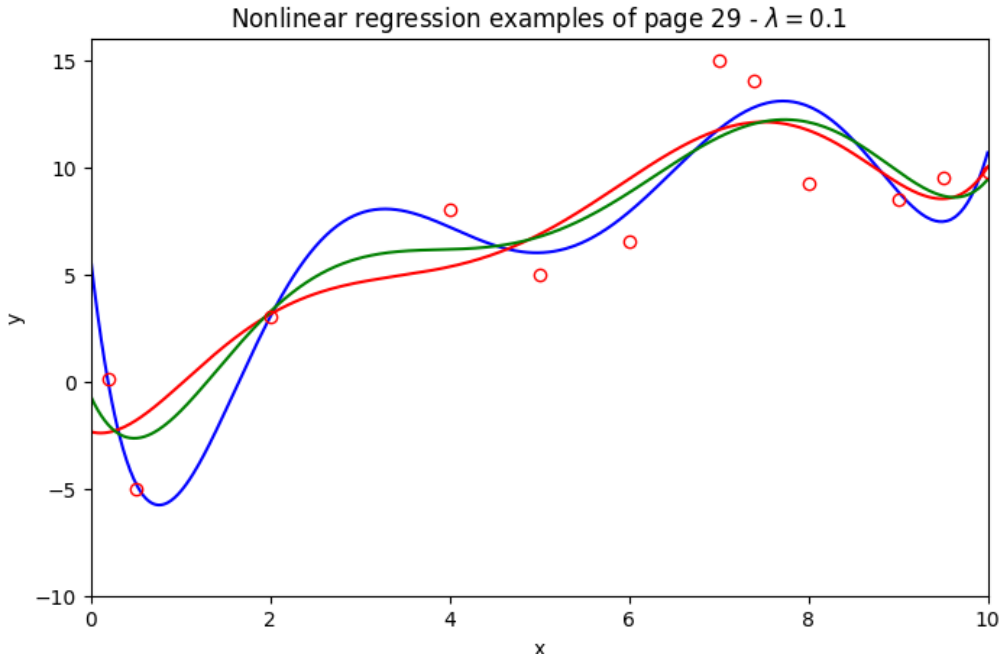

Figure 5: Result after replicating the nonlinear regression example of page 29 of the supervised machine learning slides, with $\lambda = 0.1$

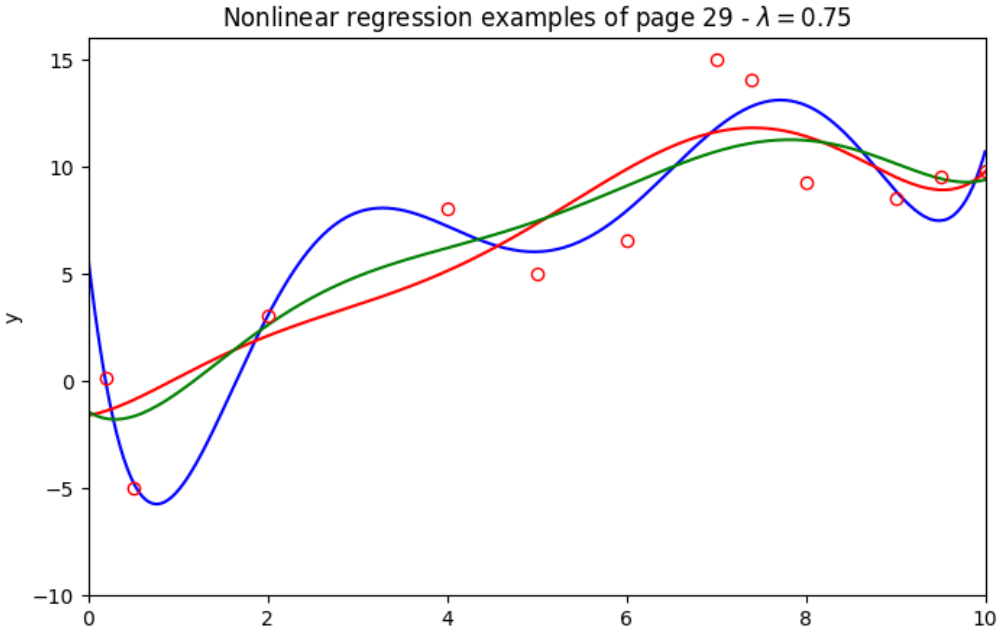

Figure 6: Result after replicating the nonlinear regression example of page 29 of the supervised machine learning slides, with $\lambda = 0.75$

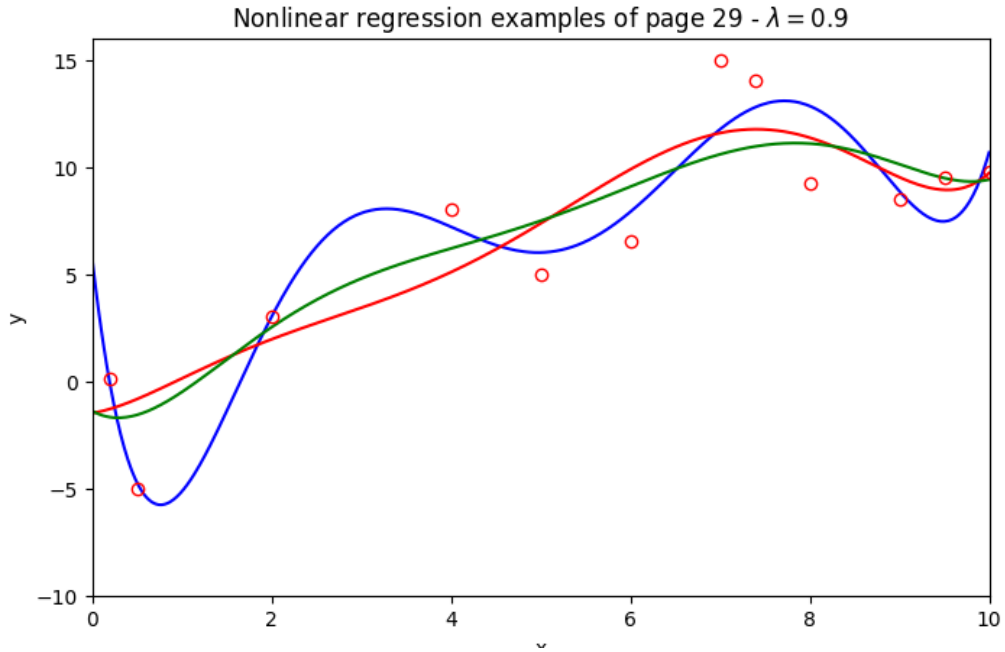

Figure 7: Result after replicating the nonlinear regression example of page 29 of the supervised machine learning slides, with $\lambda = 0.9$

Question 11

Write a program in python that implements the K-NN algorithm and the weighted average K-NN algorithm. Use appropriate data.

For this question I wrote a python program that implements the **KNN algorithm** and the **weighted average KNN algorithm**, using appropriate data.
For the sample regression, I used $K = 3$ for both versions of the algorithm.
For the weighted average KNN algorithm specifically, I needed to define the parameter σ as well. This parameter controls the degree of smoothing performed by the algorithm. To showcase this, I visualize the result for $\sigma = 0.5$ and $\sigma = 0.1$
You will find the respective jupyter notebook with all the code in the following link: `https://github.com/topalidisd-qcqt-duth/qe9-assignment3/blob/main/question11.ipynb`
In order to run the code you must have python installed in your system as well as the libraries **numpy**, **matplotlib**

```
import numpy as np
import matplotlib.pyplot as plt

x_data = np.array([0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5,
                   5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10])
y_data = x_data**2
X_fit = np.linspace(0, 10, 200)

def knn(xi, x, y, K=3):
    distances = np.abs(x - xi)**2
    nearest_indices = np.argsort(distances)[:K]
    return np.mean(y[nearest_indices])

def weighted_average_knn(xi, x, y, K=3, sigma=0.5):
    distances = np.abs(x - xi)**2
    nearest_indices = np.argsort(distances)[:K]
    w = np.exp(-distances[nearest_indices]**2 / (2 * sigma**2))
    return np.sum(w * y[nearest_indices]) / np.sum(w) if np.sum(w) > 0 else 0
```

```
y_fit_knn = np.array([knn(x, x_data, y_data, K=3) for x in X_fit])
y_fit_weighted_average_knn_0_5 = np.array([weighted_average_knn(x, x_data, y_data, K=3, sigma=0.5)
    for x in X_fit])
y_fit_weighted_average_knn_0_1 = np.array([weighted_average_knn(x, x_data, y_data, K=3, sigma=0.1)
    for x in X_fit])

plt.figure(figsize=(12, 6))
plt.scatter(x_data, y_data, color='blue')
plt.plot(X_fit, y_fit_knn, color='red', linestyle='--')
plt.title("KNN Regression - K=3")
plt.xlabel("X")
plt.ylabel("y")
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.scatter(x_data, y_data, color='blue')
plt.plot(X_fit, y_fit_weighted_average_knn_0_5, color='green', linestyle='--')
plt.title(r"Weighted average KNN Regression - K=3, $\sigma=0.5$")
plt.xlabel("X")
plt.ylabel("y")
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.scatter(x_data, y_data, color='blue')
plt.plot(X_fit, y_fit_weighted_average_knn_0_1, color='green', linestyle='--')
plt.title(r"Weighted average KNN Regression - K=3, $\sigma=0.1$")
plt.xlabel("X")
plt.ylabel("y")
plt.grid(True)
plt.show()
```

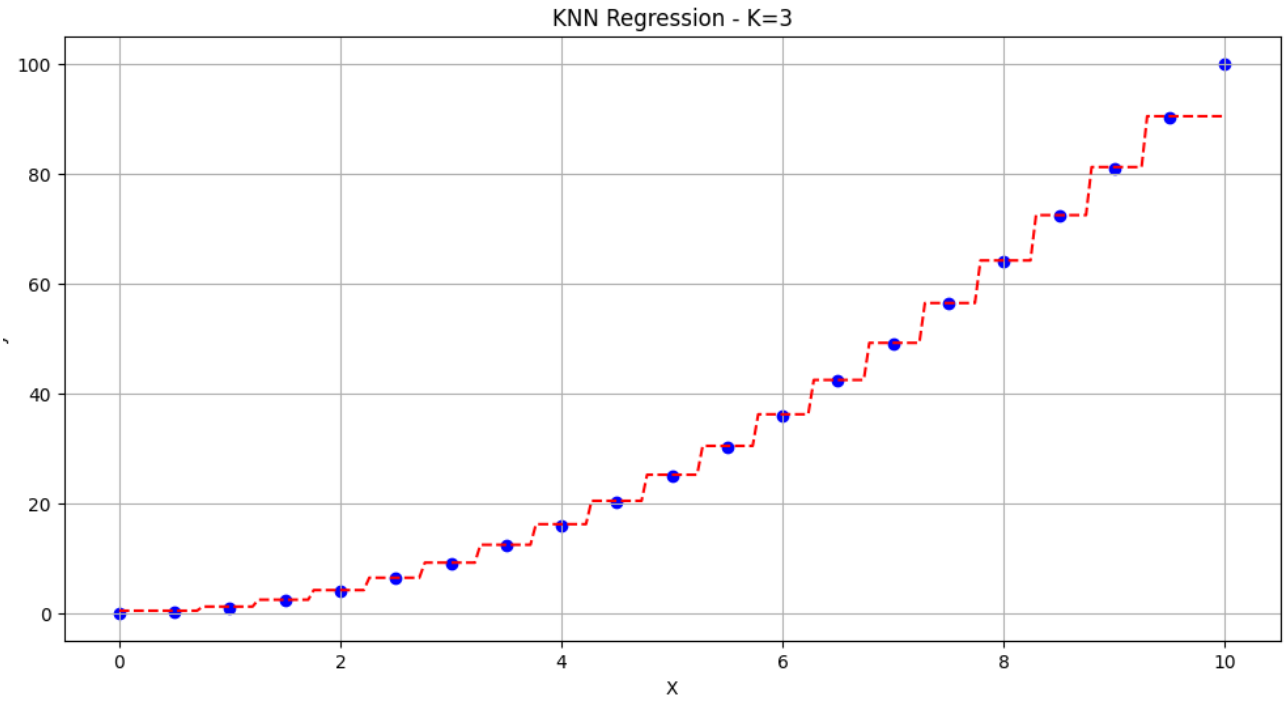

Figure 8: Result of the regular KNN algorithm with $K = 3$

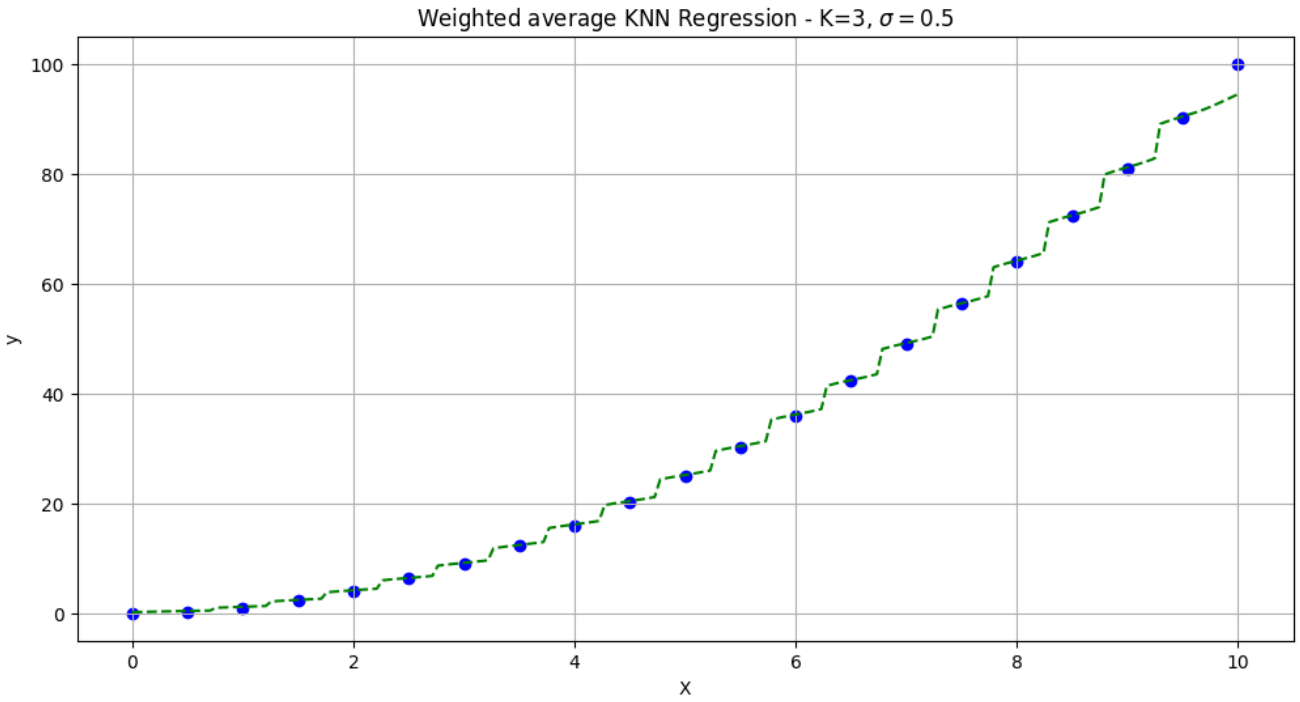

Figure 9: Result of the regular weighted average KNN algorithm with $K = 3$ and $\sigma = 0.5$

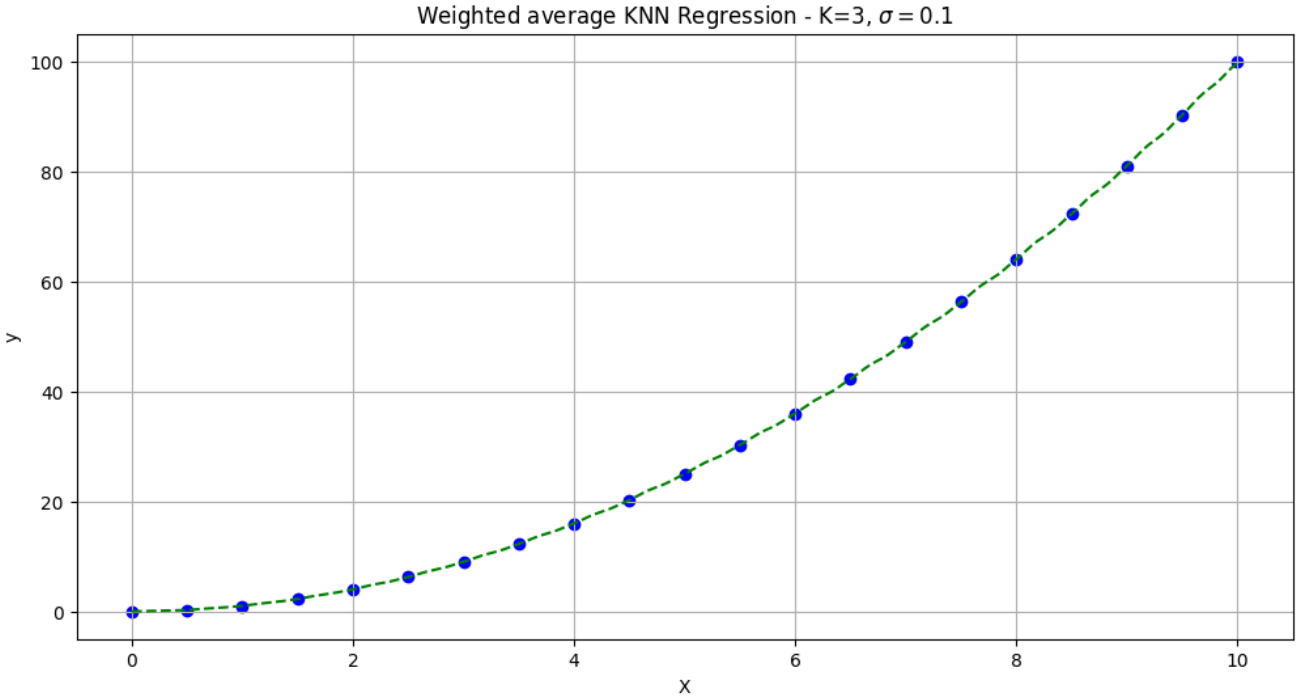

Figure 10: Result of the regular weighted average KNN algorithm with $K = 3$ and $\sigma = 0.1$