

# SWMAL gruppe 21- O[2]

Navn	StudieNummer	AUID
Mohamed Hashem Abodu	2019100895	Au656408

17/10/2023

<b>L03/SuperGruppe resume:</b>	<b>3</b>
Look at the big picture	3
Get the Data	3
Explore and visualise the data to gain insights	3
Prepare the data for machine learning algorithms	4
Select and train a model	4
Fine tune your model	5
Launch, monitor and maintain your system	5
Try it out	5
<b>L04/Dataanalyse.ipynb</b>	<b>5</b>
Qa) Description of dataset for O4 project	6
Conceptual Project Description:	6
Chosen Dataset:	6
Data Descriptions:	6
Use of Dataset:	6
Qb)Data Analysis of own dataset	6
Image visualisation:	6
Figure 1. The three patient's positions I need to classify.	7
Data Distribution:	7
Figure 2. Distribution of images across hospitals in the dataset	8
Figure 3. Label Distribution in images.	8
Considerations	8
<b>L04/pipelines.ipynb</b>	<b>9</b>
Qa) Creating Min/max scaler for the MLP	9
Figure 4. MLP + Linear regression score before scaling.	9
Qb) Scikit-learn Pipelines	11
Figure 6. The results of Sklearn MinMax scaler in the pipeline	11
Qc) Outliers and min-max scaler vs the standard scaler	12
Figure 7. The results of pipeline with StandardScaler	12
Qd) Modifying the MLP Parameter	13
Figure 8. The results of 3 different MLP model scores.	13
<b>L05/Linear_regression_1.ipynb</b>	<b>14</b>

Qa) Write a Python function that uses the closed-form to find $W^*$	14
Figure 9. The result of the test vector using close-form	14
Qb) Find the limits of the least-square method	15
Figure 10. LinalgError	15
<b>L05/gradient_decent.ipynb</b>	<b>15</b>
Qa) The Gradient Descent Method	15
Figure 11. Plots of the Gradient step with different $\eta$	16
Qb) The Stochastic Gradient Descent Method	17
Figure 12. The results for SGD	17
Qc) Adaptive learning rate for $\eta$	17
Qd) Mini-Batch Gradient Descent Method	18
Figure 13. The results of Mini-batch gradient descent.	18
Qe) Choosing a Gradient Descent Method	18
Figure 14. The results of Mini-batch GD	19
<b>L06/ Ann.ipynb</b>	<b>19</b>
Qa) Fitting the model	19
Figure 15. Plot the fit for the MLPRegressor model	20
Qb) Extract and print all coefficients	20
Figure 16. A drawing of the neural network.	20
Qc) Creating a mathematical formula for the network	21
Qd) Plot the $y_{\text{math}}$ and compare to $y_{\text{pred}}$ and $y_{\text{true}}$	21
Figure 17. Comparison between $y_{\text{true}}$ , $y_{\text{pred}}$ and $y_{\text{math}}$	21
Qe) Comparing the first and second half $y_{\text{math}}$	22
Figure 18. Comparison of the first two parts of $y_{\text{math}}$ and $y_{\text{math}}$	22
Qf) sinc-function	22
Figure 19. Comparison of 3 MLP models with different hidden layers and neurons	23

## L03/SuperGruppe resume:

In this exercise I will make a resume for the second chapter of the book “Hands-on machine Learning” in the following sections.

### Look at the big picture

This section of the chapter is about preparing the project, looking at the bigger picture in terms of thinking big, and to become a better datascientest/Machine learning expert.

The example in the book is to build a model of housing prices in California using the California census data, with metrics such as population, median income etc. The model should learn from the data and be able to predict the median housing price.

When starting a project assigned by a boss or for self amusement, framing the problem is essential, and we need to have a well defined checklist. Like a roadmap, that we can tweak either during the process of development or after deployment to fit our requirements.

In the housing price model, the model will be used as a component to train another model, which requires a supervised learning task since the labelled training examples were given, and a typical regression task, since the model needs to predict a value.

To conclude, this section introduces us to the path of designing a solution, more than just a system. It is a foundational step for an effective machine learning project to provide insights into project scope, problem framing and system design.

### Get the Data

I would describe this section as a guidance through setting up the workspace. Installing python and creating a dedicated workspace directory. Then installing models like Jupyter, NumPy, Pandas etc.

Then onto creating an isolated environment to maintain the project consistency. Also we were introduced to the Jupyter notebook, with instructions on using it.

Additionally, the section explains the process of downloading the data from an external source and loading it into our environment using pandas. It explains the importance of examining and understanding the dataset and creating a test set to ensure the right output results of the model.

### Explore and visualise the data to gain insights

The author here is drawing attention to the importance of examining the training set to gain insights over the dataset and visualising it. The author guides us through the visuals plotting a geographical data using scatterplots to reveal high density areas.

Thereafter, correlation analysis shows that the median income has a strong correlation with the median house value. Then we get to see the scatterplots to highlight the relationships for the correlation coefficients.

I was also introduced to the concept of experimenting with attribute combinations, and creating new attributes like rooms per household and number of rooms that show promising

correlations. Throughout this exploration process the author uncovered data patterns and quirks, providing valuable insights of the dataset before applying machine learning algorithms

## Prepare the data for machine learning algorithms

In this section the author digs deep into preparing the data for ML algorithms. This approach enables easy reproduction of transformations, builds a library for future projects, and makes live system data transformation possible.

The author delegates the process over 5 subsections that we are going to discuss:

- **Data Cleaning:** The author demonstrates here handling missing values, by using options like dropping districts, dropping the entire attribute or filling values with zero, mean or median. Or more efficiently using `SimpleImputer` from Scikit-learn's.
- **Handling text and categorical attributes:** Moving to the challenge of encoding text attributes, the author shifts focus to the use of *OrdinalEncoder* for numerical encoding. And then introduces the scikit-learn's dedicated *OneHotEncoder* class to convert categorical values into one-hot vectors.
- **Custom transformers :** The book introduces the concept of custom transformers by presenting the creation of *combinedAttributeAdder* class. This transformer adds additional attributes to the dataset based on existing ones.
- **Feature Scaling :** As the book explained the importance of transformers and transformations, it introduces us to the most important transformation that we need to apply to our data, that is Feature Scaling. It is needed when our data has numerical values that operate on different scales. The book further presents two methods that can help with this issue. First *min-max Scaling*, which shifts and rescaled the values so that they end up ranging from 0 to 1, so that everything gets proportionally adjusted. Secondly *Standardization*, which transforms the values to have a mean of 0 and a standard deviation of 1.
- **Transformation Pipelines :** this subsection is an introduction to the Pipeline class from Scikit-learn's library, whose purpose is to organise the transformations needed for our project. In the book's example a sequence involving imputation, augmentation and scaling. Then demonstrates how `COLUMN` transformer simplifies the process of applying transformations to different types of columns, and how to bring everything together using the `ColumnTransformer` and create a comprehensive preprocessing pipeline for both numerical and categorical columns.

## Select and train a model

After framing the problem, getting the data and exploring it, sampling a training set and test set, and after having the transformation pipelines ready, this section is a guidance to the next step of selecting a model, understanding its performance metrics, training the model, and exploring alternative models.

As in the last assignment, the author introduced the linear Regression Model, and then evaluated its performance using RMSE. Then hops on to DecisionTreesRegression to explore complex nonlinear relationships in the data.

The author then introduces the concept of cross-validation, a technique to assess a model generalisation performance. Also introducing the Scikit-learn's cross validation feature, that splits the data into a number of subset called folds, and then it trains and evaluates the model number of times, picking a different fold for evaluation every time and training it on the other remaining folds.

## Fine tune your model

Now after the author evaluated both the Linear regression model and the Decision tree model in the earlier section, he introduces a few ways to fine-tune the models. **Grid Search** is one of them, in using Scikit-learn's *GridSearchCv* to search for the best combination of hyperparameter values to experiment with. **Randomised Search** is another option. Instead of trying out all possible combinations, it evaluates a number of random combinations by selecting a random value for each hyperparameter.

Following that the author highlights the importance of ensemble methods, Random Forests in particular, in enhancing the model performance. Random Forests play a big role in getting better model accuracy and improving the prediction capabilities.

The next step will be to evaluate the system on a test set to ensure the performance is working as intended to.

## Launch, monitor and maintain your system

After examining the results and outputs of the model, launching the system is the next step, which shifts our focus to preparing our solution for production in this phase.

The book provides instructions to go through this step, involving connecting the system with actual data sources and implementing tests. To ensure good ongoing performance, we learn that we need to monitor the code with regular checks in how well the system is doing and alerts in case of drop in performance. The model can become less accurate overtime due to changes in data, thus why we also need to train the model with fresh data regularly, and it is preferred to have this process automated to maintain consistent performance.

## Try it out

This section mainly motivates us to apply what we learned so far. Creating a ML project involves a lot of data preparation, creating tools to monitor the system, automating the system training and understanding the whole process in general. Which is suggested and eased by the book instructions in this chapter as we went through it.

## L04/Dataanalyse.ipynb

In this exercise I will introduce my idea for the O4 assignment, which I will be working on continuously throughout the course.

## Qa) Description of dataset for O4 project

### Conceptual Project Description:

I have chosen to create an image classifier for the O4 project. The goal of this image classifier is to sort hospital's patient datasets efficiently. The classifier's job is detecting the position of the patient's body, whether they are holding their arms up, down or to the side, which will enhance the organisation of patient records and provide quicker access to relevant medical information.

### Chosen Dataset:

Since I'm taking this course beside my internship in AUH, I had the opportunity to work on this project and get access to the patients datasets from the DCPT(Danish Center for Particle Therapy) department. The dataset contains images of patients in various positions -arms up,down and side, where the patients have undergone breast cancer treatment using proton therapy. The images are intended to be studied for another machine learning project BCCT "Breast Cancer Conservative Treatment" which aims to predict results of the breast treatment and the expected outcome of the treatment. To ensure privacy, some parts of the images in the project documentation are going to be blurred.

### Data Descriptions:

The dataset include images of patients, each labelled either with randomization number or with patients number and the corresponding hand position. The dataset is around 36.000 samples from all 5 hospitals. Challenges that can happen from variation in lightning, poses and wrong labelled images, which causes uncertainty in future projects.

### Use of Dataset:

The dataset will be used for a classification task, to distinguish between the patient's body position. This classifier aims to automate the sorting of the hospital datasets. Also it will allow me to ease my work on the BCCT project in terms of organising data.

## Qb)Data Analysis of own dataset

### Image visualisation:

Let's take a look at what the image classes look like.



Figure 1. The three patient's positions I need to classify.

In this figure above I have an example on the three image classes, as was mentioned earlier the dataset contains various differences in lightning and patient poses, so i might consider dividing the work in smaller chunks and to test the performance.

### Data Distribution:

The dataset structure is complicated in a sense, it includes patient images from 5 hospitals, each hospital has many patients folders, and each folder contains 10-15 images of the patient in various positions and poses.

In Figure 2 below, I was able to visualise the distribution of patients' images across all hospitals. Where we can see that Aarhus hospital has the most of the patients' images with more than 16 thousand images, which are going to be challenging to go through. Though, it is worth mentioning that after examining the dataset for all hospitals, Aalborg dataset has the best image quality in terms of lightning, patients' position in the image and labelling.

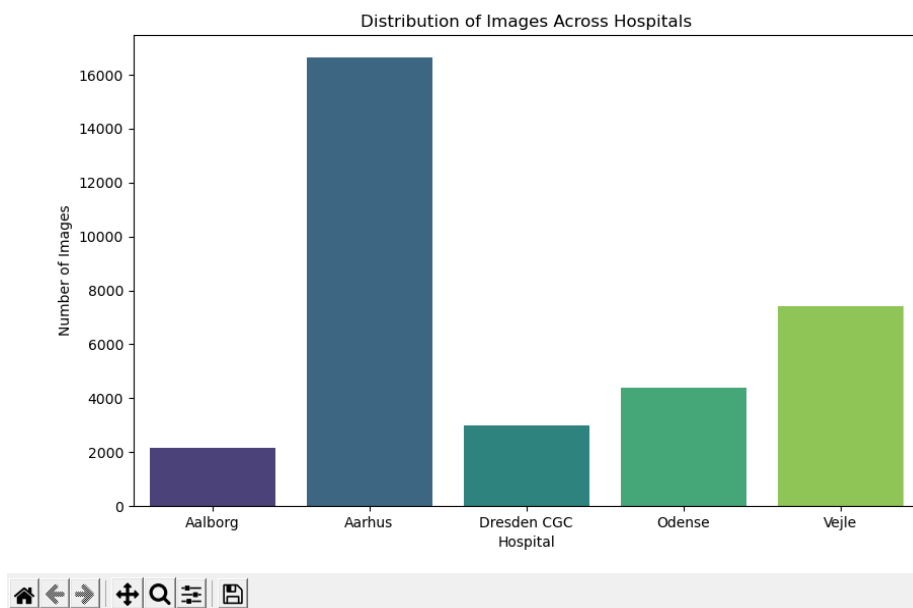


Figure 2. Distribution of images across hospitals in the dataset

In the figure below we can see the label distribution in the images for each hospital. I am taking Aalborg, Aarhus and Odense data to showcase the difference in label naming. As we can see the results are relatively similar in the 3 cases, in Aarhus, which is the largest dataset, we can see the 'other' label, which is when the image label isn't one of the 3 classes specified. Those obstacles can be challenging to organise the data and/or access for real-time use.

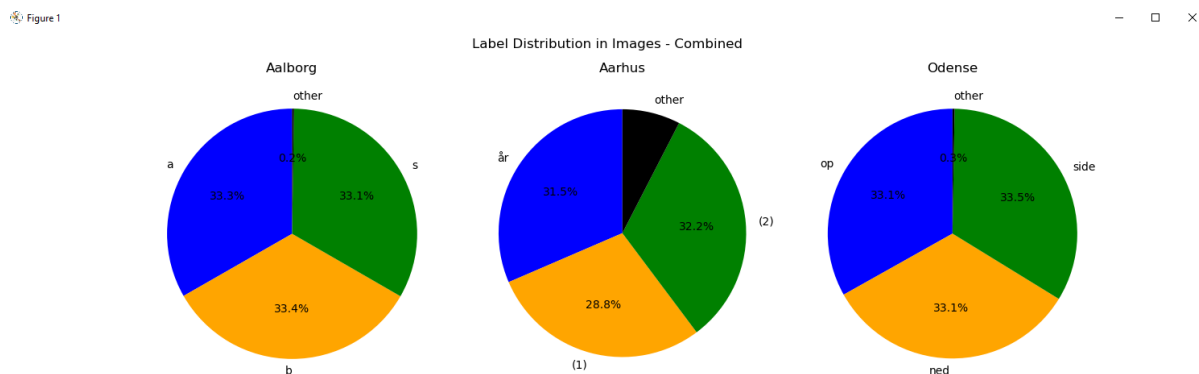


Figure 3. Label Distribution in images.

## Considerations

Like I mentioned earlier and proved in Figure 2, the dataset is huge. Considering working on smaller chunks might be beneficial. I would gather 30 samples from each hospital, for each position, so 150 arms up, 150 arms down and 150 side view. Then I would choose a classifier to work and train on my dataset. Most likely the Yolov8x-cls because it has the highest accuracy and with the most parameters. [[Ultralytics](#)]



## L04/pipelines.ipynb

This exercise provides an insight to the concept of pipelines in machine learning. We will be working with the pickle library and revisiting some of the problems that we faced in earlier exercises.

### Qa) Creating Min/max scaler for the MLP

In the exercise L01/into.ipynb we used MLP for QECD data and it produced a negative R2 score, meaning the model was worse than the mean value of y. We are going to revisit the dataset and the MLP model in this part to create a min-max scaler manually.

MLP and Linreg results before scaling :

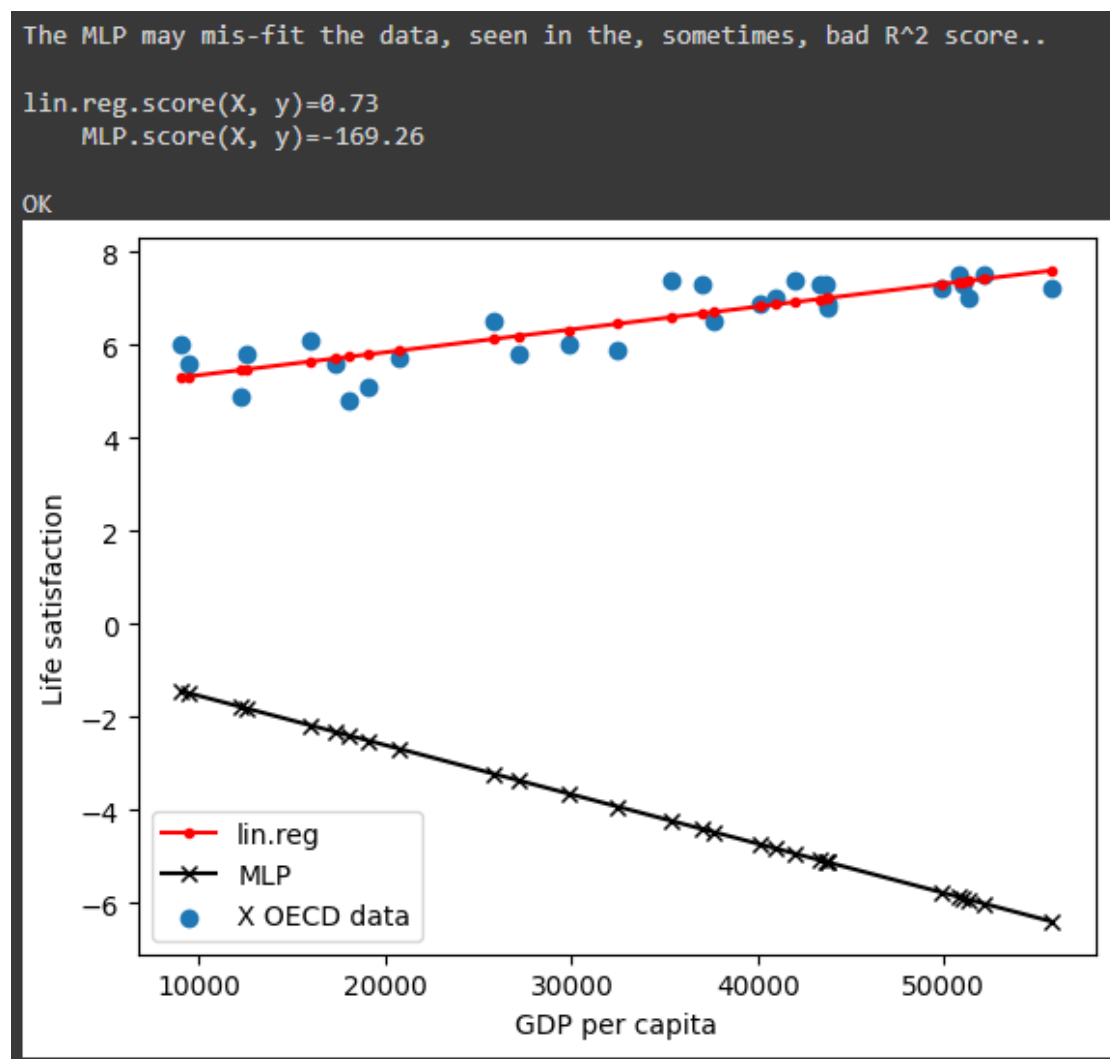


Figure 4. MLP + Linear regression score before scaling.

And after some manual scaling :

```
# TODO: add your code here..
X_min = X.min(axis=0)
X_max = X.max(axis=0)
X_scaled = (X - X_min) / (X_max - X_min)
```

```

#retrain the MLP model
scaled_mlp = MLPRegressor(hidden_layer_sizes=(10, ),
                           solver='adam',
                           activation='relu',
                           tol=1E-5,
                           max_iter=100000,
                           verbose=False)
scaled_mlp.fit(X_scaled,y)

#check R2 score
y_pred_mlp_scaled = mlp.predict(X_scaled)
r2_mlp = r2_score(y,y_pred_mlp_scaled)

linreg_scaled = LinearRegression()

linreg_scaled.fit(X_scaled, y)

print(f" R2 score after scaling : {r2_mlp}")
PlotModels(linreg_scaled, scaled_mlp, X_scaled, y, "lin.reg", "MLP")

```

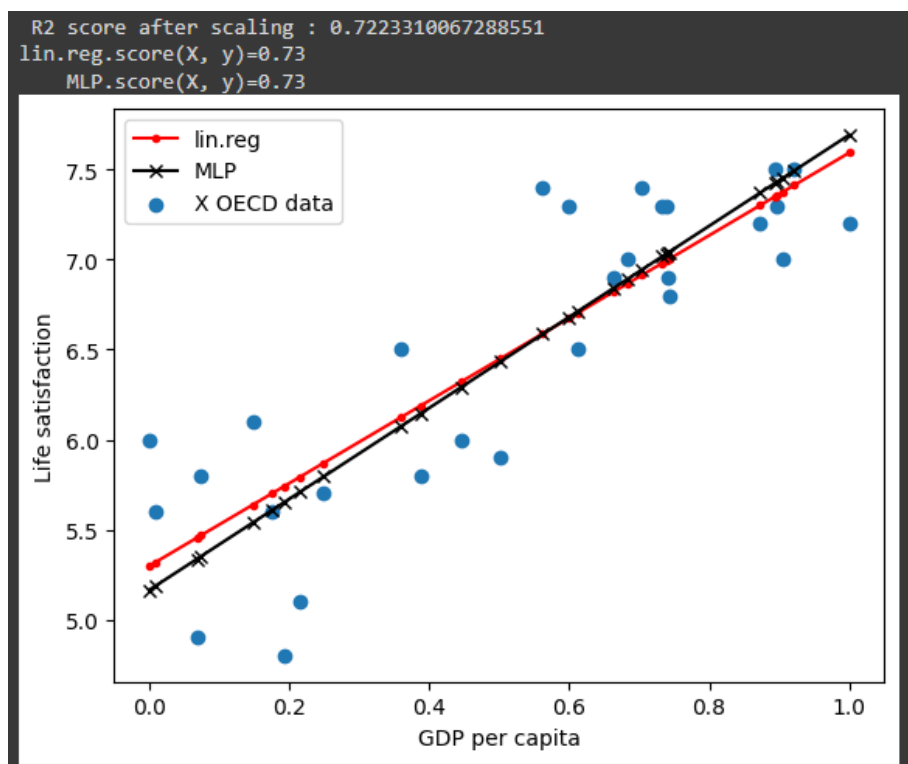


Figure 5. MIP and Linear regression scores after.

We can see in the figure above that the results are much better after scaling X on a range of [0;1] , which indicates a successful scaling of X. and now the MLP model is performing similarly to linear regression model.

## Qb) Scikit-learn Pipelines

In this section I'm going to rescale again, but with the help of sklearn's MinMaxScaler and put the MLP model and the scaler into a construction called pipeline from sklearn.pipeline.Pipeline.

```
# TODO: add your code here..
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

#pipeline with scaler and MLP
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('mlp', MLPRegressor(hidden_layer_sizes=(10, ),
        solver='adam', activation='relu',
        tol=1E-5, max_iter=100000,
        verbose=False))
])

pipeline.fit(X,y)
y_pred_pl = pipeline.predict(X)
r2_pipeline = r2_score(y,y_pred_pl)
print(f"R2 score after scaling with sklearn.minmaxscaler : {r2_pipeline}")
PlotModels(linreg, pipeline, X, y, "line.reg", "MLP")
```

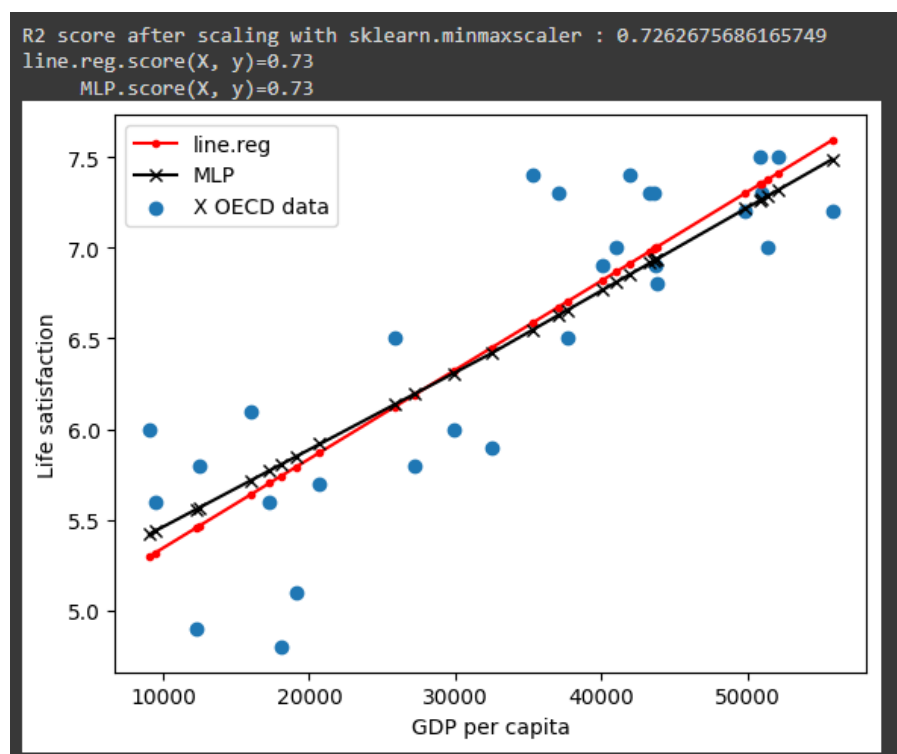


Figure 6. The results of Sklearn MinMax scaler in the pipeline

As we can see in the figure above we got similar results for the R2 score after scaling with Sklearn's MinMaxScaler, while putting all the work in the pipeline also makes it more comprehensible.

## Qc) Outliers and min-max scaler vs the standard scaler

MinMaxScaler and outliers don't go along very well, because if there are any extreme outliers in the dataset, it will affect the scaling, and that's because MinMaxScaler scales the data to a [0;1] range, based on the minimum and maximum values in the dataset.

As explained in [Feature Scaling](#) earlier on the difference between minMaxScaler and StandardScaler, the standardScaler does not care about the extreme outliers during scaling, because it standardises the data by centering it around the mean and scaling it by the standard deviation. Let's take a look at how it would perform on the chosen dataset.

```
# TODO: research the problem here..
from sklearn.preprocessing import StandardScaler

standard_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('mlp', MLPRegressor(hidden_layer_sizes=(10, ),
        solver='adam', activation='relu',
        tol=1E-5, max_iter=100000,
        verbose=False))
])

standard_pipeline.fit(X,y)
y_pred_st = standard_pipeline.predict(X)
r2_standard = r2_score(y,y_pred_st)
print(f"R2 score after scaling with sklearn StandardScaler : {r2_standard}")
PlotModels(linreg, standard_pipeline, X, y, "line.reg", "MLP")
```

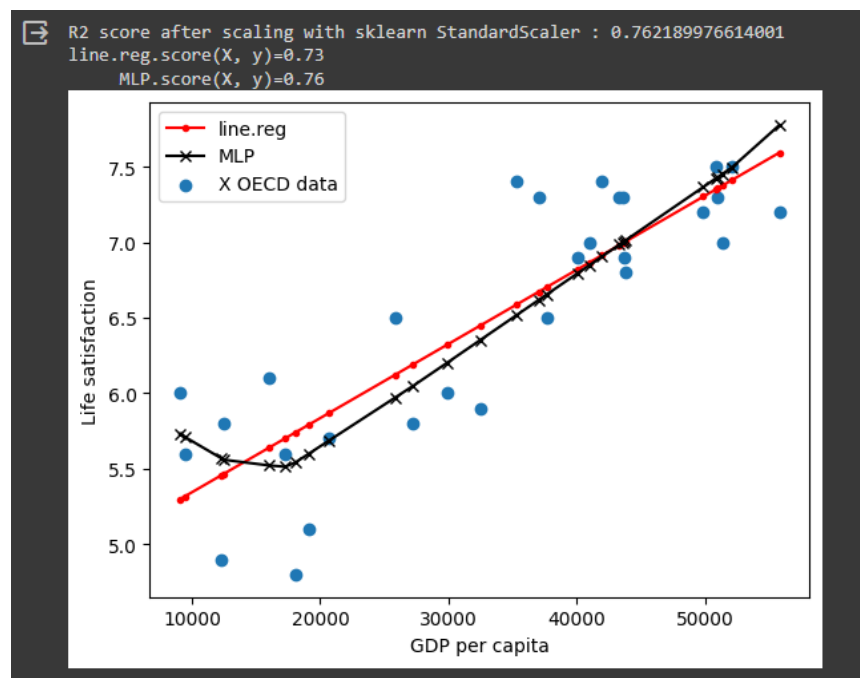


Figure 7. The results of pipeline with StandardScaler

We can see from the figure above a slight improvement in the MLP model performance on the dataset, which means standardising the features helped the MLP model to fit a bit better.

However, it could also mean that the dataset may not have extreme outliers that would affect the scaling.

## Qd) Modifying the MLP Parameter

In this section I am going to try some of the MLP hyperparameters, i'm going to explore how few neurons on the MLP model can produce sensible output. Also I will be trying other activation functions for the MLP model in different pipelines and printing their output.

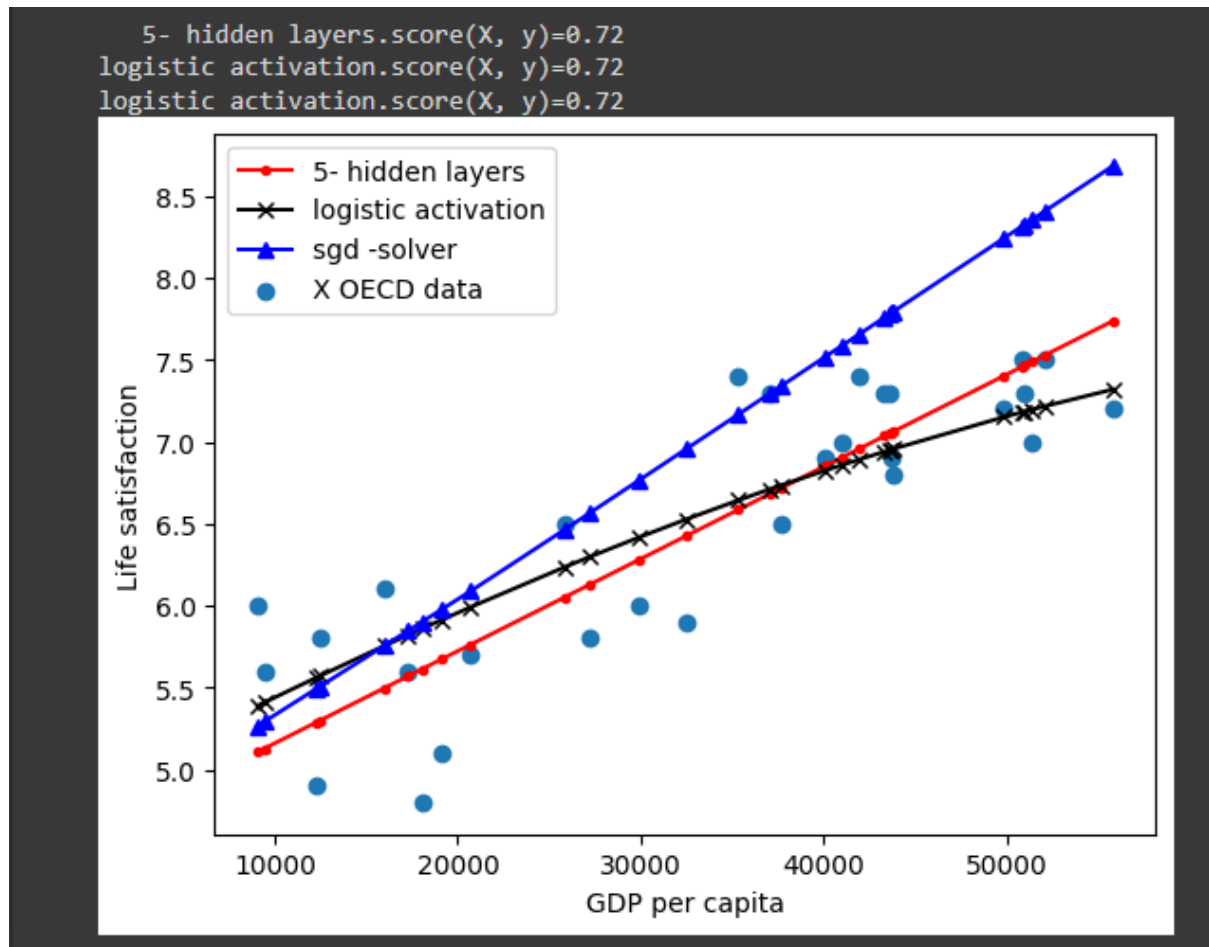


Figure 8. The results of 3 different MLP model scores.

As we can see in the figure above the R2 score for each model and then we have a plot for each model with the different hyperparameters. We can conclude that after changing in the hyperparameters, we still receive a high output using the standardised MLP model. The R2 score is consistent for both models and we can see from the plot that they are performing similarly, which indicates that in this dataset, changing the hyperparameters does not have much impact on the performance of the model.

## L05/Linear\_regression\_1.ipynb

In this exercise I will be training a linear regressor. I will be working on finding the optimal weights for the model to minimise the sum of squares error. I will introduce the closed-form solution as instructed in the exercise to provide us with an analytical way to calculate the weights that best fits the data.

### Qa) Write a Python function that uses the closed-form to find $W^*$

To implement a function that uses the closed form, we need to use a lot of linear algebra equations, luckily the numpy library is very useful in this case. After some research in the book "Hands on machine learning", I found `numpy.linalg` which provides efficient low level implementations of standard linear algebra algorithms. [\[numpy\]](#) We will start with adding the bias term to X with `np.c_` which translates slice objects to concatenation along the second axis. [\[numpy.c\]](#) then using the `np.linalg.inv()` to compute the inverse of the matrix. [\[numpy.linalg.inv\(\)\]](#). Then calculating the optimal weights by multiplying the inverse matrix by the transposed matrix `X_bias` and then `y`.

```
# TODO: Qa...
def W_star(X,y):
    #adding the bias term with np.c_
    X_bias = np.c_[np.ones(X.shape[0]), X]

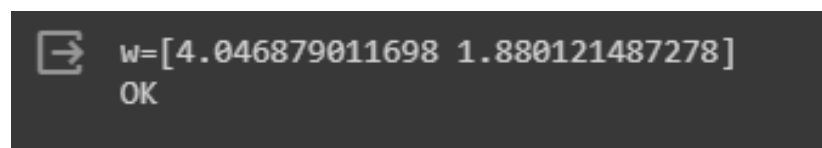
    # the inverse of the matrix using numpy.linalg.inv()
    inv = np.linalg.inv(X_bias.T @ X_bias)
    w_star = inv @ X_bias.T @ y

    return w_star

W = W_star(X,y)
# TEST VECTOR:
w_expected = np.array([4.046879011698, 1.880121487278])
itmalutils.PrintMatrix(w, label="w=", precision=12)
itmalutils.AssertInRange(w, w_expected, eps=1E-9)

print("OK")
```

And the results are :



```
➞ w=[4.046879011698 1.880121487278]
OK
```

Figure 9. The result of the test vector using close-form

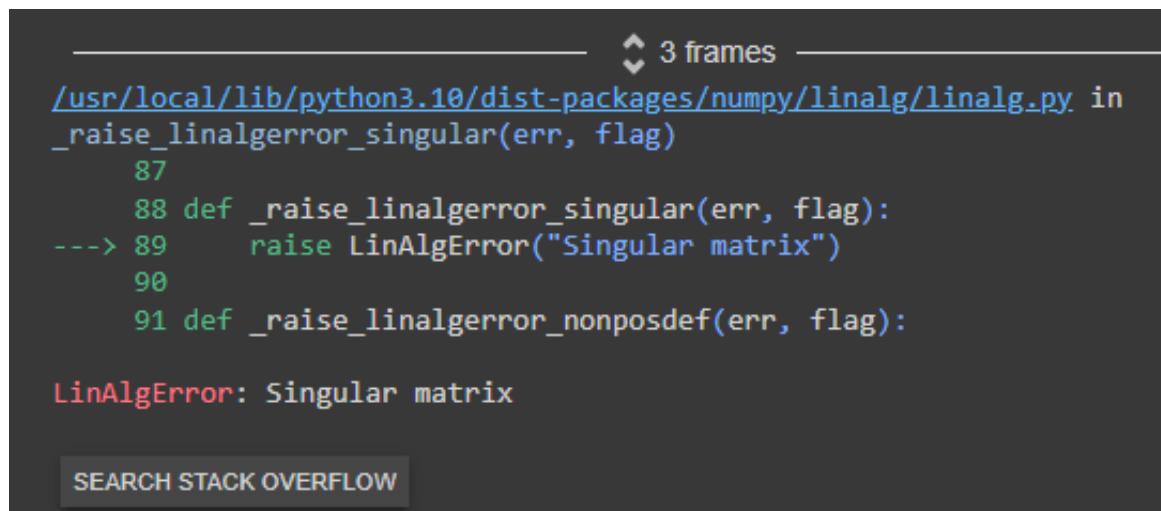
Hereby confirming that I found the least-square solution for X and y.

## Qb) Find the limits of the least-square method

In this part of the exercise I need to find the least-square optimal value for  $w$ . Using new  $X$  and  $y$  as inputs .

The problem with matrix inversion is that a singular matrix does not have an inverse. Because some of the singular matrix values are zeros, hence invert with have undefined values [\[forum\]](#).

In the figure below is the output whenever i try to find the least-square optimal values for the given matrices,without changing  $W\_star()$ :



```
3 frames
/usr/local/lib/python3.10/dist-packages/numpy/linalg/linalg.py in
_raise_linalgerror_singular(err, flag)
87
88 def _raise_linalgerror_singular(err, flag):
--> 89     raise LinAlgError("Singular matrix")
90
91 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix

SEARCH STACK OVERFLOW
```

Figure 10. LinAlgError

As we can see the output is an error from the numpy's linear algebra functions because it was not possible to compute the inverse of the singular matrix.

## L05/gradient\_decent.ipynb

In this exercise i will explore the gradient Descent methods and training, by finding the optimal solution, explaining the Gradient Descent algorithm and examine the plots

## Qa) The Gradient Descent Method

Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimise a cost function. [HOML].

To find the optimal solution, as discussed in the book, the gradient of the cost function is computed with respect to each model parameter  $\theta_j$ , which means a calculation of how much the cost function will change with a small change in  $\theta_j$ . To do that, the provided Equation 4-5 in the book computes the partial derivative of the cost function with respect to parameter  $\theta_j$  denoted as  $\frac{\partial}{\partial \theta} MSE(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T x_i - y_i) x_{ij}$ , to calculate how much each parameter

contributes to the error for each datapoint, then averages this contribution over all the data points.

Then we need a vector equation to contain all the partial derivatives, and that is where Equation 4-6 comes in handy, because it calculates all the partial derivatives at the same time, and the Batch Gradient Descent uses this vector in every step over the full training set. Now the gradient descent vector pointing uphill we need to downhill which is what equation 4-7:  $\theta^{(next\ step)} = \theta - \eta \nabla_{\theta} MSE(\theta)$  do, by subtracting  $\nabla_{\theta} MSE(\theta)$  from  $\theta$  and multiplying the gradient vector by  $\eta$  which is the learning rate to determine the size of the step.

In the code provided in the exercises we can see that eta “learning rate” is 0.1 to control the size of steps taken during each iteration. In `gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)` we can see the implementation of equation 4-6 for the gradient vector. Then the code updates the random parameter with `theta = theta - eta * gradients`, which is the implementation of the equation 4-7 in the book, in each iteration.

Let's take a look on what role does eta play and what will happened if we change it:

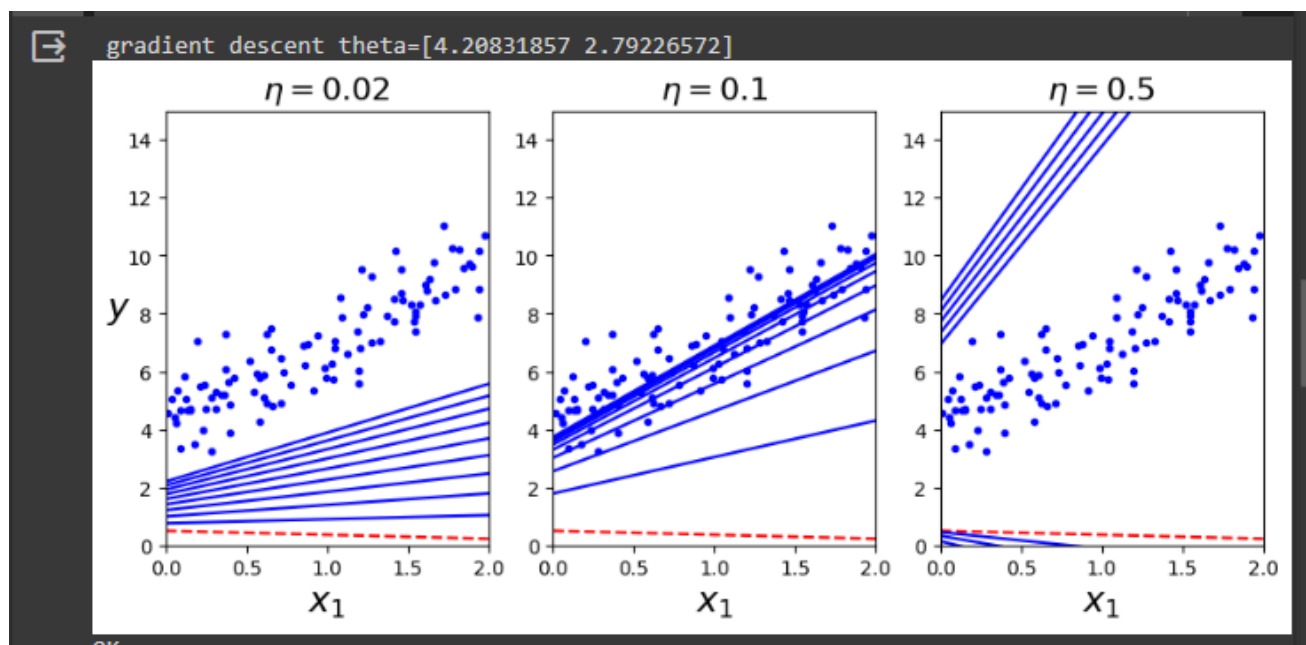


Figure 11. Plots of the Gradient step with different  $\eta$

We can see in the first plot on the left that eta = 0.02, which is a very small learning rate, therefore the algorithm will take a lot of iterations to reach a solution. In the middle plot with eta = 0.1 looks more balanced. It converges relatively fast without being too slow. In the last plot with eta = 0.5 which is a high learning rate, we can see some convergence in the beginning but then it doesn't settle into a stable solution, which can lead to failure to converge (divergence). But experimenting with the learning rate can lead us eventually to a more balanced algorithm in terms of convergence speed and stability.



## Qb) The Stochastic Gradient Descent Method

The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large. [HOML]

So the book introduces us to the stochastic Gradient Descent, which picks a random instance in the training set at every step and calculates the gradients based on that instance. This approach makes the algorithm much faster because of the little data to manipulate at every iteration.

"np.random.randint()" is a function that returns random integers from low to high. [NumPy], and in this setting is responsible for randomly selecting an index in each iteration. Let's take a look at how the SGD model performed.

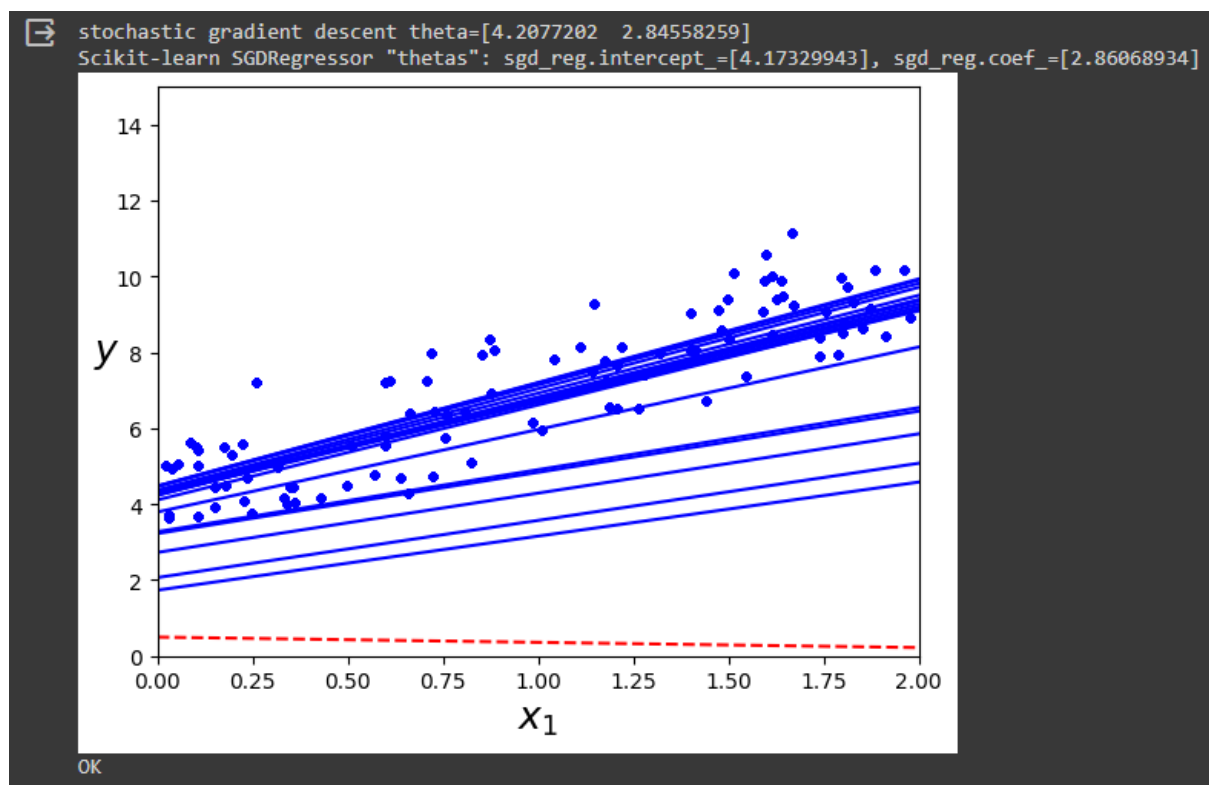


Figure 12. The results for SGD

We can see in the figure above the results of the SGD model which are similar to the normal Gradient Descent In figure 11 with  $\eta = 0.1$ , but we can also see the improvement in converging to an optimal solution.

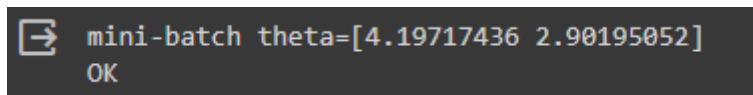
## Qc) Adaptive learning rate for $\eta$

There is a way to manipulate the learning rate at each iteration during the training process. It's called `learning_schedule()`, and it is not limited to the SDG model, it can be applicable on other models as well. It has the formula " $t_0 / (t + t_1)$ " which basically decreases the learning rate over time.

The effect of this method is that it allows the algorithm to respond to the evolving dynamics of the optimization process, and it can balance the tradeoff between rapid convergence in the initial phase and fine tune the model during the process.

## Qd) Mini-Batch Gradient Descent Method

Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.[\[MLM\]](#). Let's explain a bit further, we can say is the compromise between SGD and GD, because it strike the balance of BGD and the efficiency of SGD. and the main differences between mini-batch GD and GD is that GD stops at the minimum datapoint, while the mini-batch continue to explore( as well as SGD). Mini batch also offers faster convergence than BGD and less erratic updates than SGD, the voice between the three depends mostly on the size of the database. As we can see in the figure below the mini batch achieved the best result in terms of precision and efficiency



```
mini-batch theta=[4.19717436 2.90195052]
OK
```

Figure 13. The results of Mini-batch gradient descent.

## Qe) Choosing a Gradient Descent Method

As we can see here in the figure 14 below we have  $\theta$ ,  $\theta_1$  for each variant of the gradient descent, and we can see that the SGD(red) algorithm shows a more erratic trajectory with frequent updates based on the individual instances. We can also see BGD(blue) model has a smoother/more stable trajectory because it process the entire set at each iteration, and finally we can see the Mini-batch GD(green) that has a bit of both, it has the smoothness from the BGD and the erratic trajectory from SGD as expected. In the following table i will explain the pros and cons for each variant of the Gradient Descent:

<b>Model</b>	<b>Pros</b>	<b>Cons</b>	<b>When to use</b>
<b>BGD</b>	Most precise Direct convergence	Computationally expensive Slow converge	Small to moderate datasets.
<b>SGD</b>	Fast convergence Can escape from local minima	Erratic convergence path Noisy updates	Large datasets.
<b>mini-Batch GD</b>	Most balanced	Variability in updates	Need for balance between efficiency and precision. Moderate to large datasets.

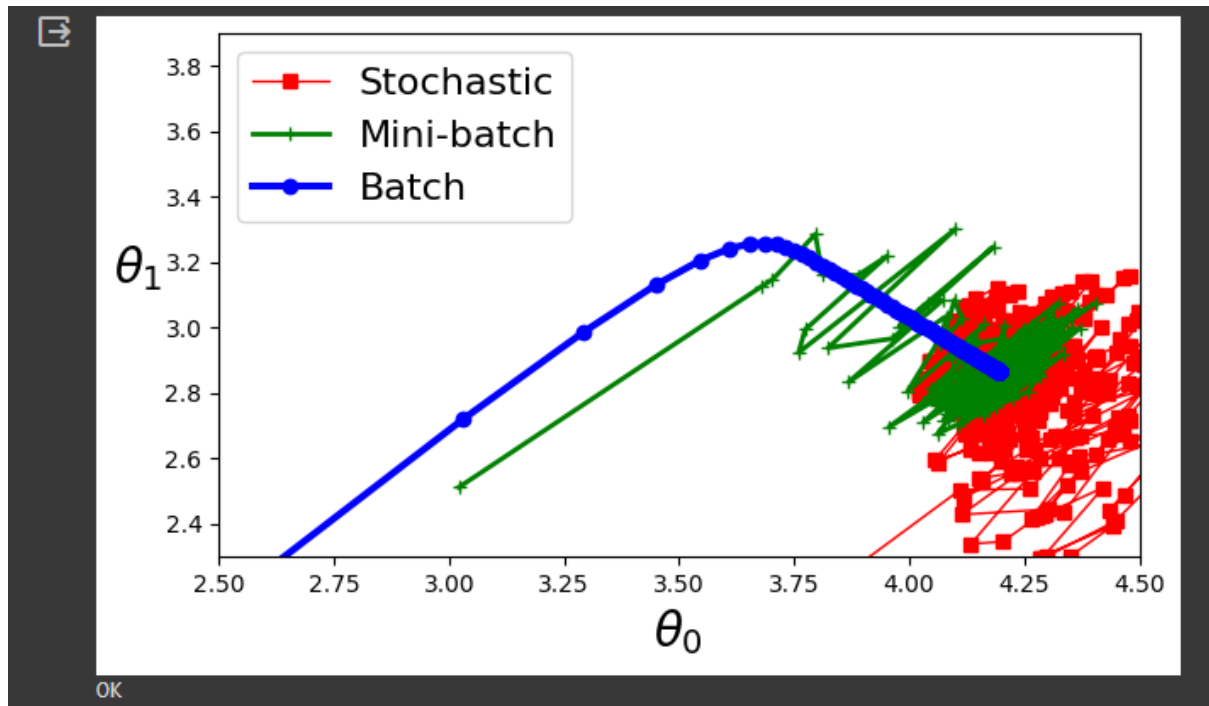


Figure 14. The results of Mini-batch GD

## L06/ Ann.ipynb

In this exercise I will dive into the Artificial Neural Networks, a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. [\[AWS\]](#) I will use MLPRegressor in the exercise with synthetic data, and train the model with a very simple ANN with only two neurons.

### Qa) Fitting the model

As instructed in the exercise we will fit the MLPRegressor model on the generated data to plot a graph of  $y_{\text{true}}$  and  $y_{\text{pred}}$ , also I will extract the network weights and bias coefficients just like earlier exercises.

```
#Fitting the model
mlp.fit(X, y_true)
y_pred = mlp.predict(X)
#extracting coefficients
weights = mlp.coefs_
biases = mlp.intercepts_
```

As we can see in the figure 15 below,  $y_{\text{true}}$  and  $y_{\text{pred}}$  are identical, which means the model has captured the root pattern and can generate predictions that are matching the true values.

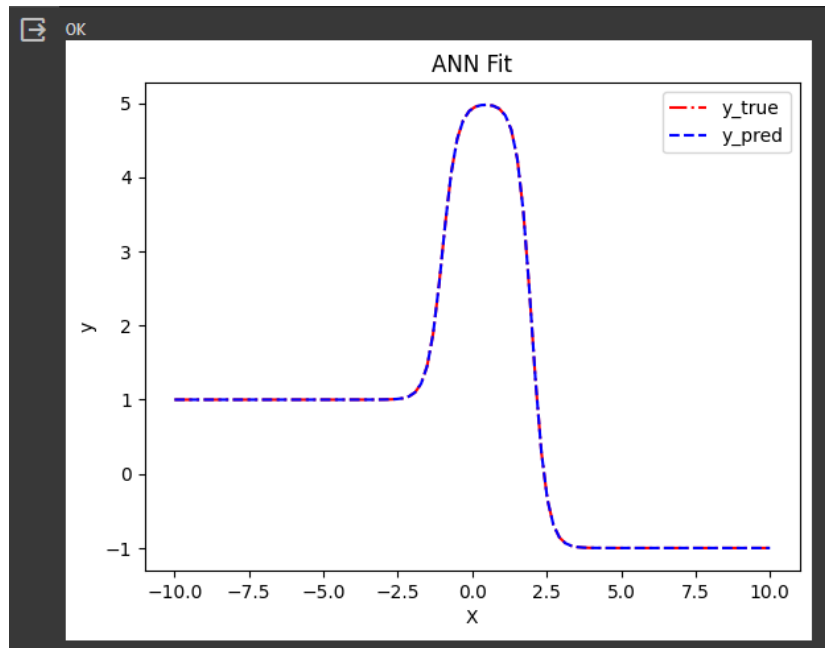


Figure 15. Plot the fit for the MLPRegressor model

## Qb) Extract and print all coefficients

We can see in the figure below a drawing for the neural network, it has an input layer, a hidden layer with two neurons and an output layer with “tanh” activation function.

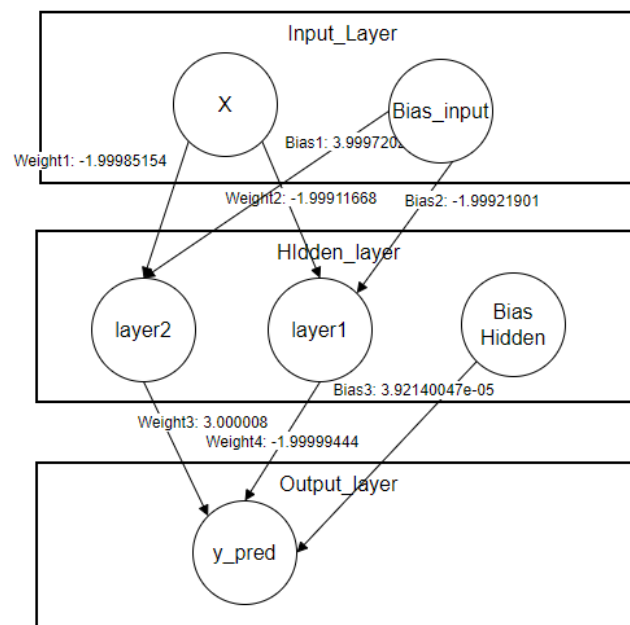


Figure 16. A drawing of the neural network.

After extracting the seven weights from the MLP attributes, I added them to the drawing of the ANN in figure 16.

### Qc) Creating a mathematical formula for the network

```
# TODO: create formula..  
neur1_output = np.tanh(-1.999 * x + 3.999)  
neur2_output = np.tanh(-1.999* x - 1.999)  
bias3 = 3.92140047e-05  
y_math = 0.3 * neur1_output - 0.3* neur2_output + bias3
```

### Qd) Plot the y\_math and compare to y\_pred and y\_true

After plotting the result for y\_math, we can see in the figure below the behaviour of the monotonic tanh activation function where the network made a general approximation for the data

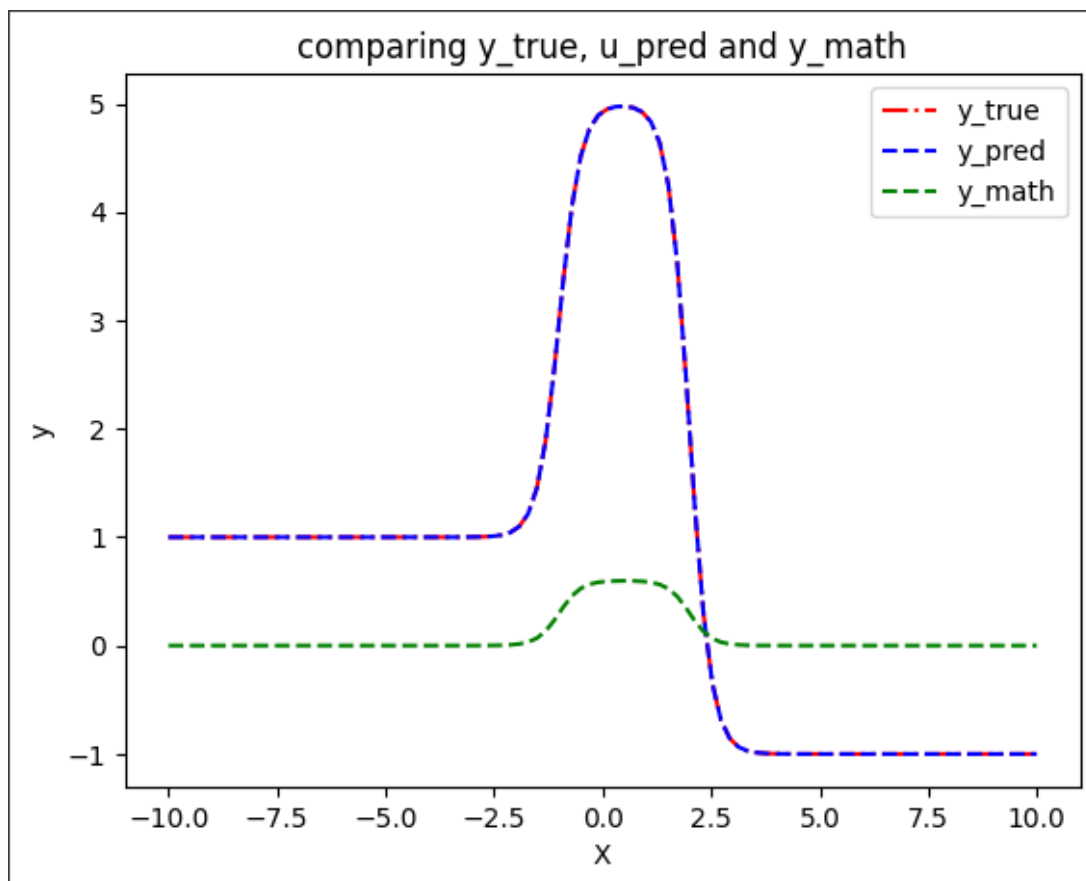


Figure 17. Comparison between y\_true, y\_pred and y\_math

## Qe) Comparing the first and second half $y_{\text{math}}$

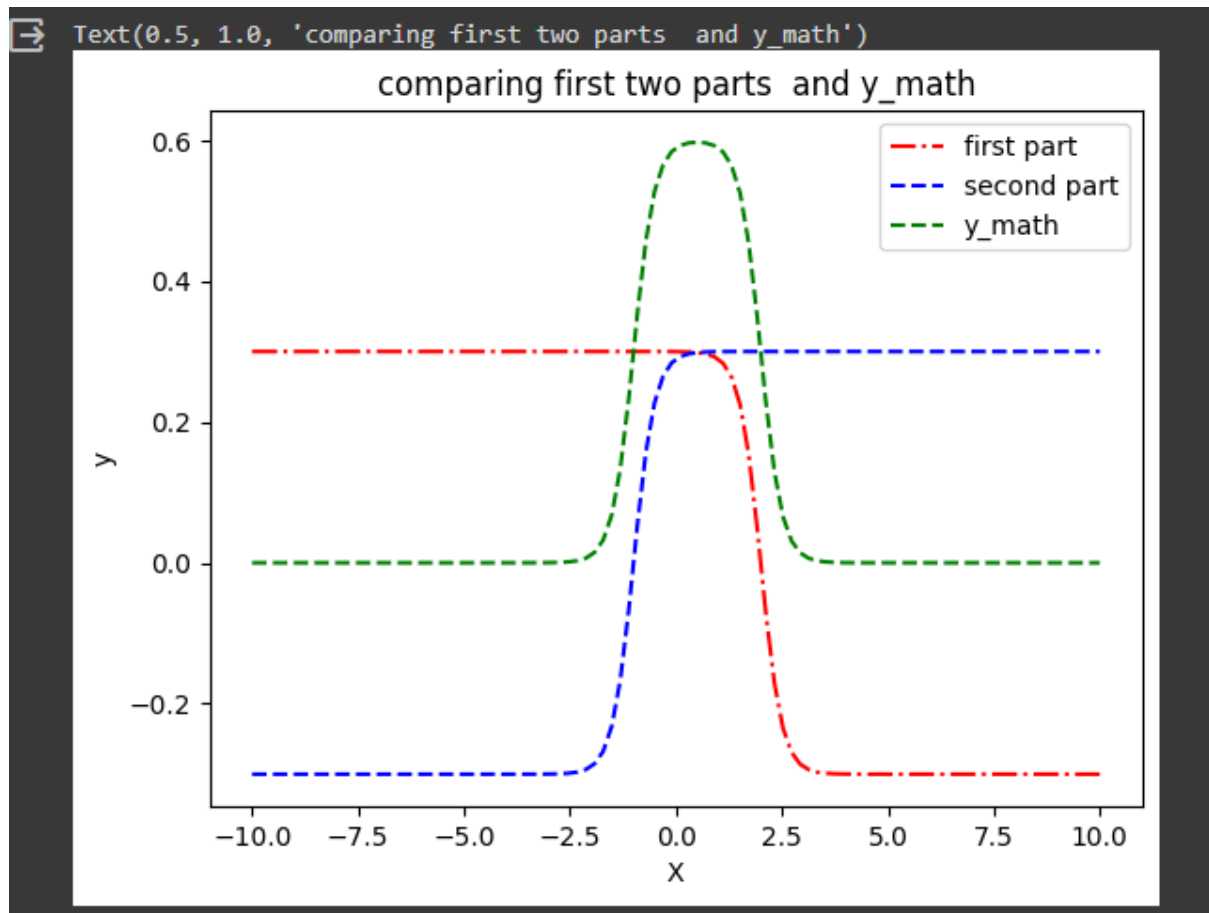


Figure 18. Comparison of the first two parts of  $y_{\text{math}}$  and  $y_{\text{math}}$

As we can see in the figure above, the two parts of  $y_{\text{math}}$  are similar to the behaviour of a monotonic tanh activation function. Each neuron specialises in different parts of the data. First neuron focuses on the initial segment (features) and the second neuron focuses on the next segment of  $y_{\text{math}}$ , adapting the network to various shapes and nuances in the data. This combination also enhances the power of the network, allowing it to be a general approximator for a variety of functions.

## Qf) sinc-function

To experiment with the behaviour of the network with different hidden layers and neurons, the change in the code is very simple. This is what i did to create 3 different models with different hidden layers and neurons:

```
hidden_layer_sizes = [10]
hidden_layer_sizes = [50]
hidden_layer_sizes =(30,70)
```

In the figure below we can see the plot for the model I created, and we can see that all of the 3 models are exhibiting a similar behaviour to the ground truth, which means my neural networks have successfully learned to approximate the pattern of the data generated with sinc-function, despite changing the layers and the neurons.

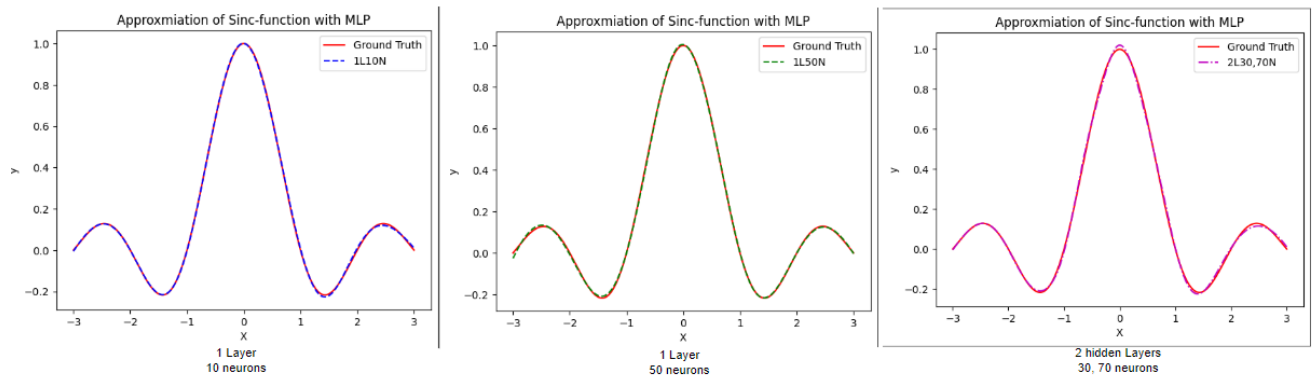


Figure 19. Comparison of 3 MLP models with different hidden layers and neurons