Course:	**Maschinelles Lernen und künstliche Intelligenz/ Machine Learning and Artificial Intelligence**		
(Group) Member:	*Nr.*	*Name*	*Matrikel*
	1	**Miguel Angel Lopez Mejia**	**5197617**
	2		
	3		

Task Reporting Template

Please read the given Task carefully. You can find all necessary information in Moodle.

1. Definition

1.1. Introduction and Explanations

Task 5.1: Understand uploaded code example of Iris SLP

The goal is to familiarize yourself with reading, working, and understanding of pre-created solutions. Download the provided code from Moodle. Configure your environment to successfully run the **"SLP_Iris.py"** application. The code might not work out of the box for our dataset-file!

Libraries required:

1. NumPy
2. Pandas
3. Matplotlib
4. Sklearn

To ensure your understanding of the code, please complete the following questions:

a. Get the code running with our dataset.
b. Which activation function is being used? How is the Impact, if we now initialize the weights with ones and not with zeroes at Line46?

```
self._w = np.zeros(x.shape[1])
```

c. Review the following piece of code (around line 134). What is the code doing? Why is it needed?

```
# map the labels to a binary integer value
```

```
y = np.where(y == 'Iris-setosa', 1, -1)
```

d. What is the learning rate set for the SLP model? How is the Impact, if we now initialize the weights with ones and not with zeroes at Line46?
e. What is epoch and what is it set to?
f. Extend the code to use defined Number of Iterations as a variable and to print the Accuracy of your Algorithm. Therefore also investigate the impact of training and testing data ration!
   Some hints for using Libraries and where to define the Iterations variable:

```
from sklearn.model_selection import train_test_split #preparation of data (how many for training/testing)
from sklearn.metrics import accuracy_score # calculate easy the accuracy

# train the model
iterations=10
#consider iterations for classification#
```

1.2. Allowed Tools

- The script is going to be executed using ***Python version 3.13.2***
- ***Visual Studio Code*** is going to be used as IDE (Integrated Development Environment)
- **Libraries:** pandas, numpy, logging

2. Your Preparation/ Concept

2.1. Structure your Problem

- Task 5.1: Understand uploaded code example of Iris SLP
- Task 5.2: Improving “SLP_Iris.py” to classify between all 3 Iris classes with all 4 features (e.g. as MLP)
- Task 5.3 Test Your NN on other Datasets

Flows:

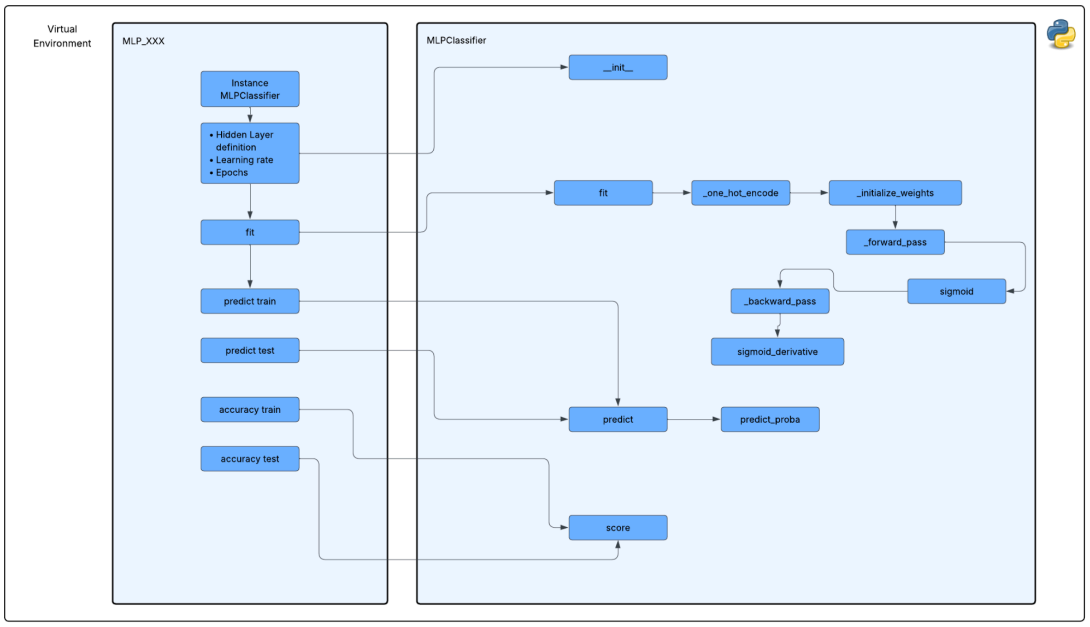


Figure 1: Iris analysis diagram

Libraries:

- **numpy** is a Python library providing a multidimensional array object, derived objects like masked arrays and matrices, and tools for fast array operations, including math, logic, shape manipulation, sorting, I/O, Fourier transforms, linear algebra, statistics, and random simulation.
- **Seaborn** is a Python data visualization library built on top of Matplotlib. It focuses on making it easier to create attractive and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations with just a few lines of code and provides a high-level interface for drawing various types of statistical plots.

2.2. Your Implementation

Virtual Environment

- In order to run the script, a virtual environment must be created, open a new terminal and create it as it is shown:

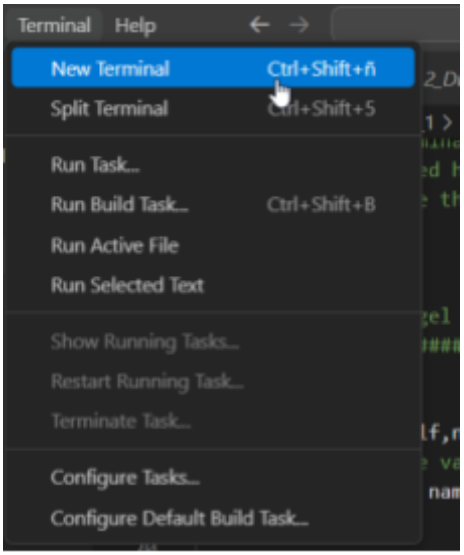


Figure 2: New terminal

- Execute the following command in the console in order to create a new virtual environment: ***python -m venv venv*** once the virtual environment is created, it will be displayed in the explorer section:

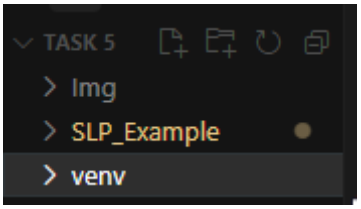


Figure 3: Virtual environment

- In order to activate the virtual environment, execute this script in the console: ***venv/scripts/activate*** .

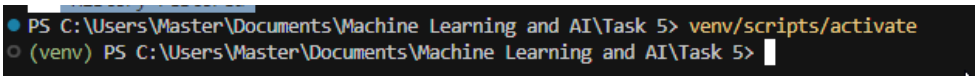


Figure 4: activation of venv

Task 5.1: Understand uploaded code example of Iris SLP

The goal is to familiarize yourself with reading, working, and understanding of pre-created solutions. Download the provided code from Moodle. Configure your environment to successfully run the **“SLP_Iris.py”** application. The code might not work out of the box for our dataset-file!

Libraries required:

5. NumPy
6. Pandas
7. Matplotlib
8. Sklearn

To ensure your understanding of the code, please complete the following questions:

g. Get the code running with our dataset.

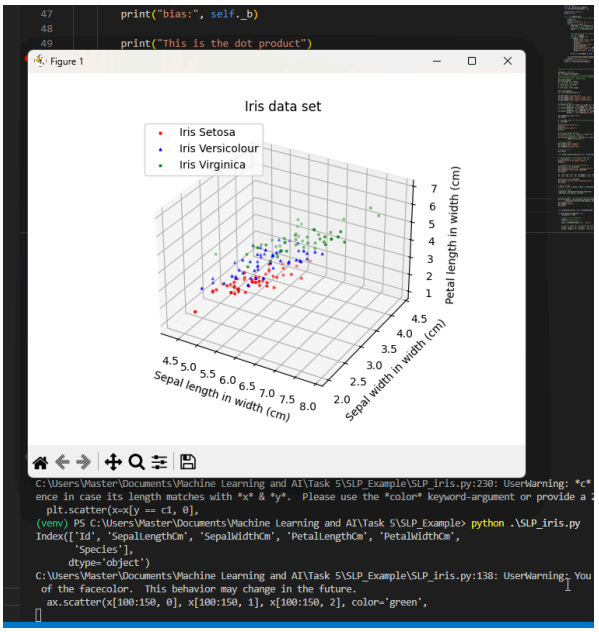


Figure 5: Code running in local environment

h. Which activation function is being used? How is the Impact, if we now initialize the weights with ones and not with zeroes at Line46?

```
self._w = np.zeros(x.shape[1])
```

1. Which activation function is being used?
   The activation function is a binary step function:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b \geq 0 \\ -1 & \text{if } w \cdot x + b < 0 \end{cases}$$

Figure 6: Binary step representation

2. How is the Impact, if we now initialize the weights with ones and not with zeroes at Line46?
   We can see that with 10 iterations and a weight = 0 the errors drop to 0 after 3 iterations

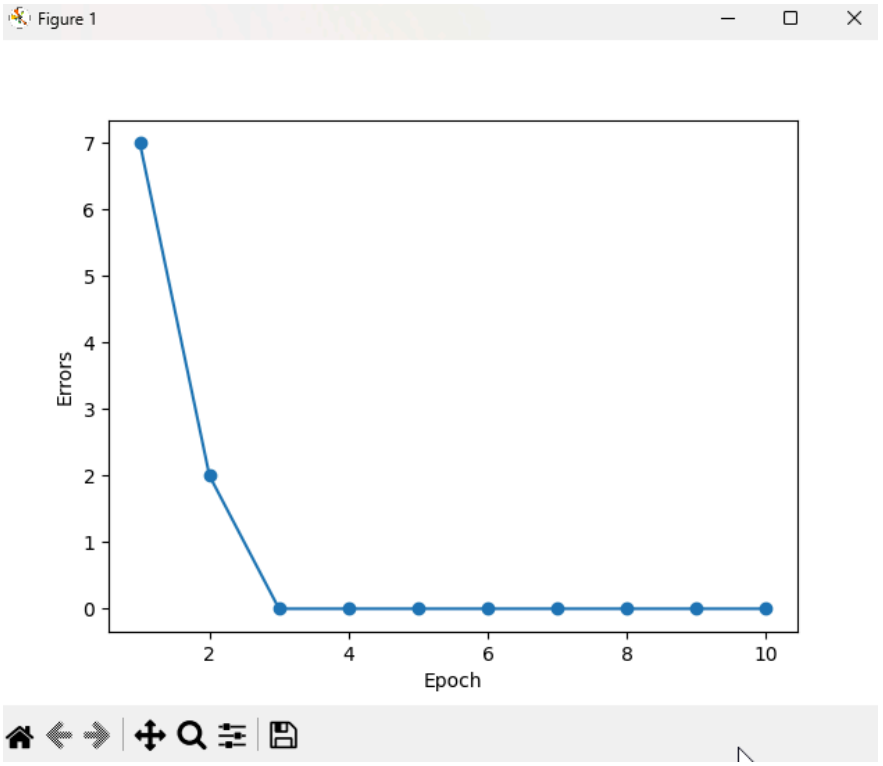


Figure 7: Number of errors with weight = 0

But if we change the weight to start with 1, the number of iterations increments to 12 to reach the 0 errors.

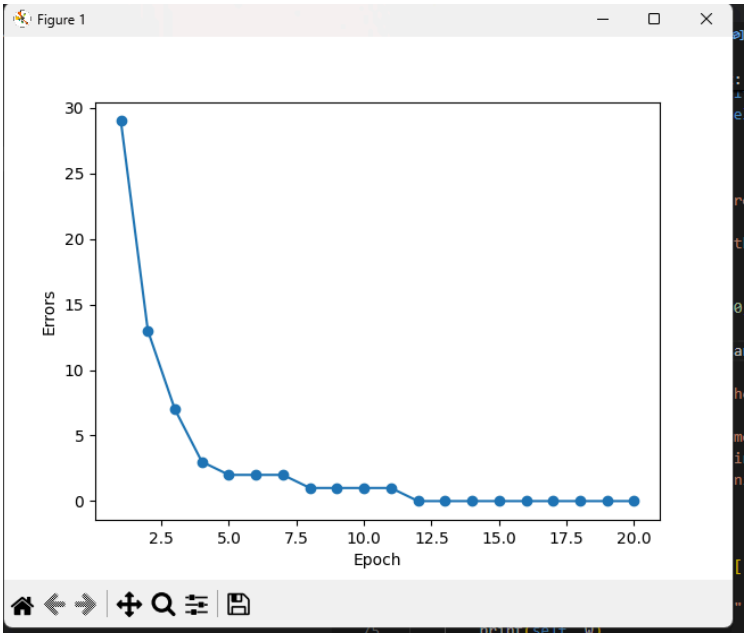


Figure 8: Number of errors with weight = 1

The reason is that the perceptron starts with no bias towards any particular decision and the initial prediction only depends on the bias term, allowing the perceptron to learn the appropriate weight based purely on the training data. With 1, perceptron already starts with a strong initial prediction and the decision boundary is already heavily based on this particular decision and the model needs to unlearn these initial strong biases before it can find the correct decision boundary.

i. Review the following piece of code (around line 134). What is the code doing? Why is it needed?

```
# map the labels to a binary integer value
y = np.where(y == 'Iris-setosa', 1, -1)
```

- This line is creating an array, and is using the existing value of ‘y’ which contains two values, 'Iris-setosa' and 'Iris-versicolor’, is using the numpy library, to assign values to the new array depending on the value ‘y’.
- If the value of the iteration in place is = 'Iris-setosa', the value of the new ‘y’, in that position, will be 1, otherwise, it will be -1.

```
['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor']
These are the mapped values:
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1]
```

Figure 9: Representation of the mapping of ‘y’ vector

j. What is the learning rate set for the SLP model? How is the Impact, if we now initialize the weights with ones and not with zeroes at Line46?
   - The learning rate is 0.01, therefore, the model makes small adjustments to the weights during each update, which means that the model will learn more slowly but potentially more stably.
   - If we increase the learning rate to 0.05 and start the weight on 1, the number of iterations is reduced from 12 to 4.

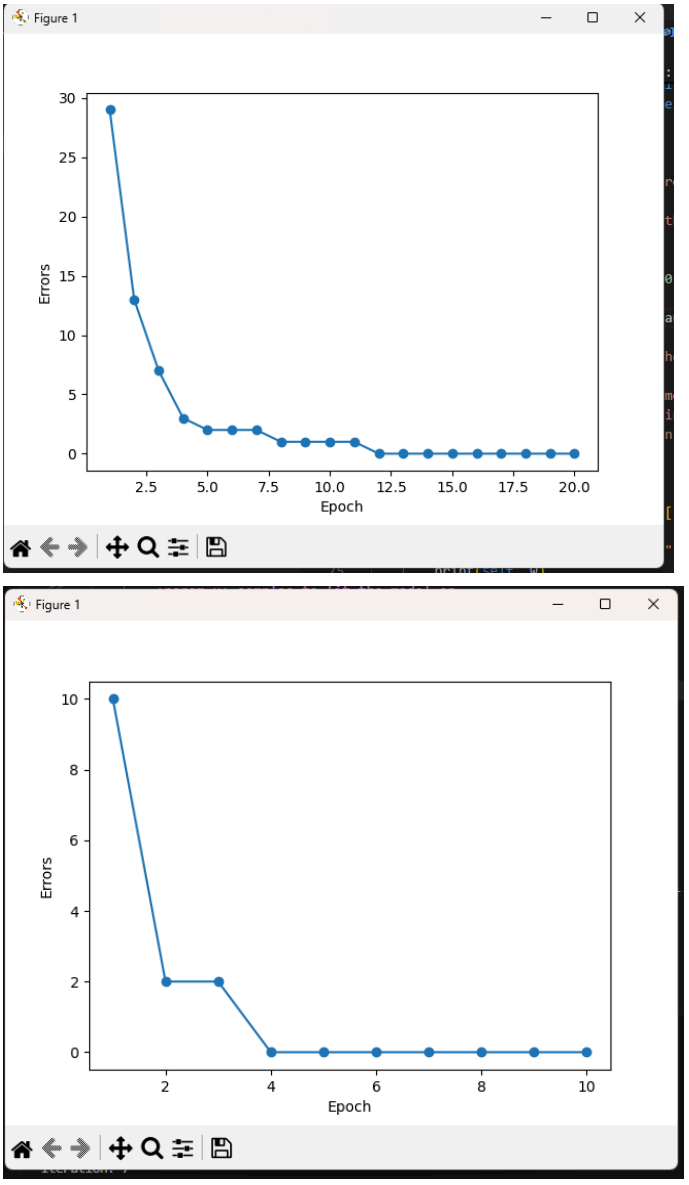


Figure 10: Comparison of errors with distinct learning rate and weight

- The learning rate controls how big of a step the model takes in the direction of the correction
- With 0.05, each update is 5 times larger than with 0.01
- This means the model can make bigger corrections when it makes mistakes
- The larger learning rate (0.05) allows the model to make bigger corrections
- The initial weights of 1 happened to push the decision boundary in a generally correct direction
- Together, these allowed the model to:
    - Make larger steps towards the correct decision boundary
    - Start from a position that was accidentally more favorable for this specific dataset

k. What is epoch and what is it set to?
   i. It represents one complete pass through the entire training dataset, basically the loop 'for' and is set to 10.

```
def fit(self, x: np.array, y: np.array, n_iter=10):
    """
    fit the Perceptron model on the training data

    :param x: samples to fit the model on
    :param y: labels of the training samples
    :param n_iter: number of training iterations
    """
```

Figure 11: Definition of the epoch

l. Extend the code to use defined Number of Iterations as a variable and to print the Accuracy of your Algorithm. Therefore also investigate the impact of training and testing data ration!
   i. The approach to achieve this point is:
      1. Split the training data
         a. We can use train_test_split and indicate the percentage of the data we want to use for training, the method returns 4 results, the input training data, the output training data, the input test data and the output test data: x_train, x_test, y_train, y_test

```
# split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
                                                    random_state=0)
```

      2. Train the model
         a. By creating the instance of the Perceptron, we can train the model, indicating the learning rate, then, in the fit method, indicate the number of epoch or iterations, this step will set the weight and bias to the most suitable values to classify the dataset.

```
n_iterations = 15  # We can adjust this value as needed

classifier = Perceptron(learning_rate=0.05)
classifier.fit(x_train, y_train, n_iter=n_iterations)
```

3. Calculate the accuracy
    a. Once the weights and bias are optimized for the training data, we can create a method that compares the accuracy of the model by calling the 'predict' function.

```
def calculate_accuracy(X, y_true, model):
    predictions = model.predict(X)
    accuracy = round(np.mean(predictions == y_true),4)*100
    return accuracy
```

```
Performance Metrics:
Number of Iterations (Epochs): 15
Training Accuracy: 100.00%
Test Accuracy: 100.00%
```

Figure 12: Results of the accuracy and Iterations

Regarding the impact of training and testing data ration, I added this functionality to the script to train the model with a distinct number of test sizes.

```
# TRAINING MODEL WITH DIFFERENT DATASETS SIZES
########################################################################
test_sizes = [0.1, 0.25, 0.4, 0.5]
n_iterations = 15  # You can adjust this value as needed
learning_rate=0.05

# Dictionary to store results
results = []
for test_size in test_sizes:
    # split the data
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=0)
    classifier = Perceptron(learning_rate)
    classifier.fit(x_train, y_train, n_iter=n_iterations)
    # Calculate accuracy for both training and test sets
    train_accuracy = calculate_accuracy(x_train, y_train, classifier)
    test_accuracy = calculate_accuracy(x_test, y_test, classifier)
    # Store results
    results.append({
        'test_size': test_size,
        'train_size': 1 - test_size,
        'train_samples': len(x_train),
        'test_samples': len(x_test),
        'train_accuracy': train_accuracy,
        'test_accuracy': test_accuracy,
        'errors': classifier.misclassified_samples[-1]  # Final epoch errors
    })
# Print results in a formatted table
print("\nImpact of Train-Test Split Ratios:")
print("-" * 80)
print(f"{'Train-Test Ratio':15} {'Train Samples':13} {'Test Samples':12} {'Train Acc%':10} {'Test Acc%':10} {'Final
Errors':12}")
print("-" * 80)
```

```
for r in results:
    ratio = f"{int(r['train_size']*100)}-{int(r['test_size']*100)}"
    print(f"{ratio:15} {r['train_samples']:<13} {r['test_samples']:<12} {r['train_accuracy']:10.2f}
{r['test_accuracy']:10.2f} {r['errors']:12}")
```

```
Problems  Output  Debug Console  Terminal  Ports
Train-Test Ratio Train Samples Test Samples Train Acc% Test Acc%  Final Errors
--------------------------------------------------------------------------------
90-10           75            25               100.00     100.00            0
75-25           75            25               100.00     100.00            0
60-40           75            25               100.00     100.00            0
50-50           75            25               100.00     100.00            0
```

Figure 13: Results of the model accuracy with distinct Ratios

Task 5.2: Improving "SLP_Iris.py" to classify between all 3 Iris classes with all 4 features (e.g. as MLP)

Currently the **"SLP_Iris.py"** application is a single layer perceptron with one output (classification). It also only works with 3 out of the 4 features. We want to classify between all three Iris classes using all the features. Thus, update the application that all 4 features are used to classify between all three classes.

In other words, change the SLP structure to allow for all features of the Iris dataset and all the classes. You could create a MLP or leave it as a SLP with multiple neurons in one layer. Anyway, in following your NN code is just named as MLP.

Judge the Accuracy of your MLP Neural Network (by training and testing data) and compare against your rule-classificator from task 4. Please also document your Accuracy result to Moodle-Task5-Result-xls.

Therefore, create following files:

- Class "**MLPClassifier.py**"
- Iris test file "**MLP_Iris.py**". This file will load your iris dataset and "import MLPClassifier as classifier" to train and test with Iris data. The Class/Method should be called this way:

```
classifier.fit(x_train, y_train, classes, iterations)

x_train - training feature-sets [each set has 4 features]
y_train - training label set [Numeric, used Labels are 1, 2,3]
classes - Number of classes [Numeric, in our case 3]
iterations - Number of Iterations for training
To feed the test data into your Model the Class/ Method should be called this way:
y_class_label = classifier.predictlabel(x_test)

x_test - testing feature-sets [each set has 4 features]
y_class_label - labeled output data [Numeric, used Labels are 1, 2,3]
```

From existing code:

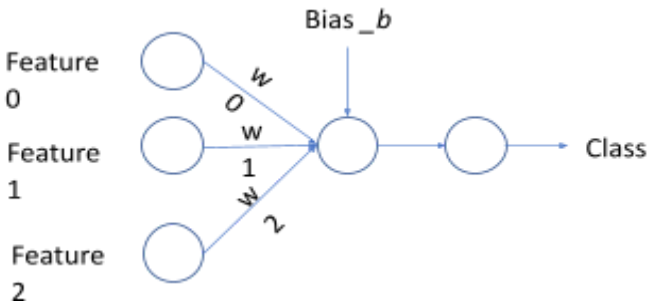


Figure 14: SLP Diagram

To your new code (where we have one perceptron per class and the output layer aggregates the class label. “Output Layer” could be rule or neuron based.):

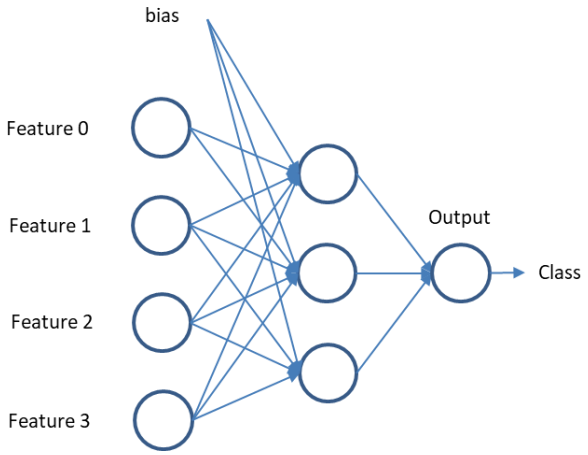


Figure 15: MLP Diagram

Results:

These are the results of the MLP script that I created:

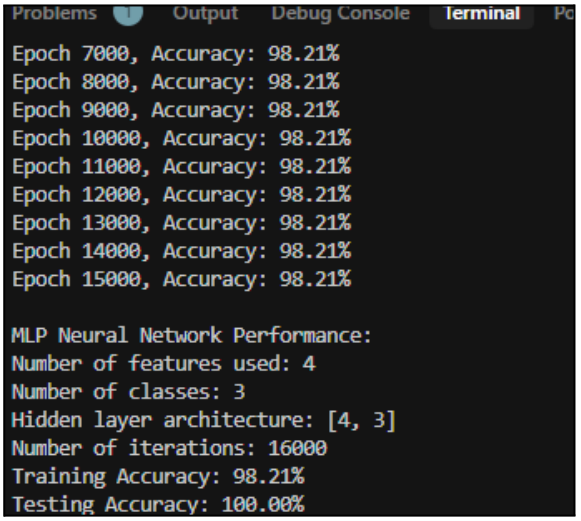


Figure 16: MLP Results in Iris Dataset

- The neuronal network contains 4 hidden neurons and 3 output layers:
- Includes a logic that verifies if the accuracy in the current epoch is less or equal than the previous one in order to avoid overfeeding.

Task 5.3 Test Your NN on other Datasets

Goal is to test our ML-Modell “MLPClassifier” with different datasets. Therefore, your code has to be programmed in a generalized way.

Please download two datasets – Prof. Twieg announces in lecture.

of the Most Popular Data Sets at: UCI Machine Learning Repository
https://archive.ics.uci.edu/ml/index.php

Feed the Dataset to your MLP Neural Network Model to train and test:

- create a new py-file to train/test each dataset you like to work with and name it accordingly eg MLP_Wine.py and MLP_Cars.py

- import/use the “MLPClassifier.py” class from Task 5.2

- Evaluate and compare the accuracy of your MLP NN against other researchers and judge (just make e.g. a google search for papers. Please also document your result to Moodle-Task5-Result-xls.

[Sidenote: It might be necessary to optimize and generalize your NN-Model-Class/Methods in “MLPClassifier.py” during Task 5.2 and Task5.3. If you optimize/ change during Task 5.3 please check again, that Iris dataset still works!]

Finally Upload in Moodle and also document your result to Moodle-Task5-Result-xls:

Results

Bean dataset

Step 1: Cleaning data:

A. Check dataset info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13611 entries, 0 to 13610
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             13611 non-null  int64
 1   Perimeter        13611 non-null  float64
 2   MajorAxisLength  13611 non-null  float64
 3   MinorAxisLength  13611 non-null  float64
 4   AspectRation     13611 non-null  float64
 5   Eccentricity     13611 non-null  float64
 6   ConvexArea       13611 non-null  int64
 7   EquivDiameter    13611 non-null  float64
 8   Extent           13611 non-null  float64
 9   Solidity         13611 non-null  float64
 10  roundness        13611 non-null  float64
 11  Compactness      13611 non-null  float64
 12  ShapeFactor1     13611 non-null  float64
 13  ShapeFactor2     13611 non-null  float64
 14  ShapeFactor3     13611 non-null  float64
 15  ShapeFactor4     13611 non-null  float64
 16  Class            13611 non-null  object
dtypes: float64(14), int64(2), object(1)
memory usage: 1.8+ MB
```

Figure 17: Bean dataset - information

B. Check for null values in the dataset:

```
Area             0
Perimeter        0
MajorAxisLength  0
MinorAxisLength  0
AspectRation     0
Eccentricity     0
ConvexArea       0
EquivDiameter    0
Extent           0
Solidity         0
roundness        0
Compactness      0
ShapeFactor1     0
ShapeFactor2     0
ShapeFactor3     0
ShapeFactor4     0
Class            0
```

Figure 18: Bean dataset - Count of Null values

C. Check labels or classes:

```
Unique classes: ['BARBUNYA' 'BOMBAY' 'CALI' 'DERMASON' 'HOROZ' 'SEKER' 'SIRA']
```

Figure 19: Bean dataset - Labels

D. Identify features with high correlation:

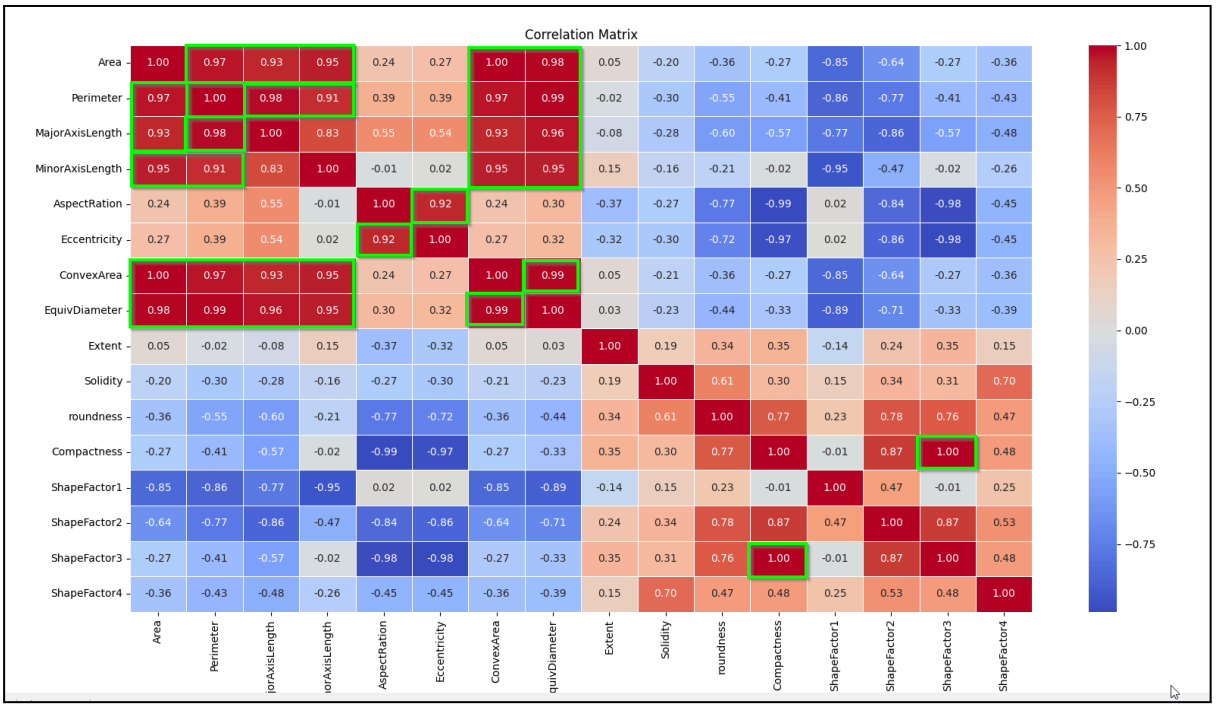


Figure 20: Bean dataset - Features correlation

```
# region Result of the correlation matrix
    # correlation_1 :Area - Perimeter
    # correlation_2 :Area - MajorAxisLength
    # correlation_3 :Area - MinorAxisLength
    # correlation_4 :Area - ConvexArea
    # correlation_5 :Area - EquivDiameter
    # correlation_6 :Perimeter - Area
    # correlation_7 :Perimeter - MajorAxisLength
    # correlation_8 :Perimeter - MinorAxisLength
    # correlation_9 :Perimeter - ConvexArea
    # correlation_10 :Perimeter - EquivDiameter
    # correlation_11 : MajorAxisLength - Area
    # correlation_12 : MajorAxisLength - Perimeter
    # correlation_13 : MajorAxisLength - ConvexArea
    # correlation_14 : MajorAxisLength - EquivDiameter
    # correlation_15 : MinorAxisLength - Area
    # correlation_16 : MinorAxisLength - Perimeter
    # correlation_17 : MinorAxisLength - ConvexArea
    # correlation_18 : MinorAxisLength - EquivDiameter
```

```
        # correlation_19 : ConvexArea - Area
        # correlation_20 : ConvexArea - Perimeter
        # correlation_21 : ConvexArea - MajorAxisLength
        # correlation_22 : ConvexArea - MinorAxisLength
        # correlation_23 : ConvexArea - EquivDiameter
        # correlation_24 : EquivDiameter - Area
        # correlation_25 : EquivDiameter - Perimeter
        # correlation_26 : EquivDiameter - MajorAxisLength
        # correlation_27 : EquivDiameter - MinorAxisLength
        # correlation_28 : EquivDiameter - ConvexArea
    # endregion
```

E. Map classes/labels to numeric representation:

```
label_map = {'BARBUNYA': 1, 'BOMBAY': 2, 'CALI': 3, 'DERMASON': 4, 'HOROZ': 5, 'SEKER': 6, 'SIRA': 7}
y = np.array([label_map[label] for label in y])
```

F. Create train and test data:

```
79
80  # Split the data
81  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
82  print(X_train.shape)
83  print(X_test.shape)
84  print(y_train.shape)
85  print(y_test.shape)
86  """
```

```
Problems 1  Output  Debug Console  Terminal  Ports
(venv) PS C:\Users\Master\Documents\Machine Learning and AI\Task 5\SLP_Example\MLP_Model> python .\MLP_Bean.py
(10208, 6)
(3403, 6)
(10208,)
(3403,)
(venv) PS C:\Users\Master\Documents\Machine Learning and AI\Task 5\SLP_Example\MLP_Model>
```

Figure 21: Bean dataset - Create train and test datasets

Analysis of results:

- **Results with layers [36,18,7] and only 6 features:**

```
MLP Neural Network Performance:
Number of features used: 6
Number of classes: 7
Hidden layer architecture: [36, 18, 7]
Number of iterations: 50000
Training Accuracy: 91.36%
Testing Accuracy: 91.18%
```

Figure 22: Bean dataset - Results with [36,18,7] and only 6 features

- **Results with layers [36,18,7] and only ALL features (16):**

```
MLP Neural Network Performance:
Number of features used: 16
Number of classes: 7
Hidden layer architecture: [36, 18, 7]
Number of iterations: 50000
Training Accuracy: 93.63%
Testing Accuracy: 93.18%
```

Figure 23: Bean dataset - Results with [36,18,7] and only 16 features

Comparison:

Source 1:

Abstract:
The technological explosion has paved the way for agriculture to flourish exponentially thus contributing to better yield of crops through the aid of machine learning, the Internet of things, mechanical systems in agriculture. In our research work, we have investigated various types of dry beans followed by a deep neural network based approach to classify the beans automatically. The results shows that our approach had an accuracy of 93.44%, and an F-1 score of 94.57%, with the dataset that consisted of 7 varieties of dry beans. Our results, which performed substantially better in comparison to traditional machine learning approaches aided us to devise further research scopes in the field of agricultural machine learning.

Reference 1: K. K. Patil and S. R. Kulkarni, "Classification of Dry Beans Using Machine Learning and Transfer Learning," 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 2022, pp. 1-6, doi: 10.1109/ICAAIC53929.2022.9689905.

Source 2:

The XGB classifier preferably outperformed than other classifiers with balanced and imbalanced distribution of dry beans within each class. It has acquired accuracy (ACC) 93.0% and 95.4% in imbalanced and balanced classes respectively. In case of balanced dataset, after application of ADASYN algorithm both KNN and RF techniques also performed well regarding the Classification Accuracy (ACC), Sensitivity (SE), Specificity (SP) and Cohen's kappa coefficient (Kappa) etc. The most important attributes for classifying the dry beans were found ShapeFactor2, Minor Axis Length, and ShapeFactor1 along with EquivDiameter, Roundness and ConvexArea.

Reference 2: K. K. Patil and S. R. Kulkarni, "Classification of Dry Beans Using Machine Learning and Transfer Learning," Artificial Intelligence in Agriculture, vol. 7, pp. 1-12, 2023, doi: 10.1016/j.aiia.2023.01.001.

Heart dataset

Step 1: Cleaning data:

A. Check dataset info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Figure 24: Heart dataset - Dataframe information

B. Check for null values in the dataset:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Figure 25: Heart dataset - Count of null values

C. Check labels or classes:

```
Unique classes: [0 1]
```

Figure 26: Heart dataset - Check labels

D. Identify features with high correlation:

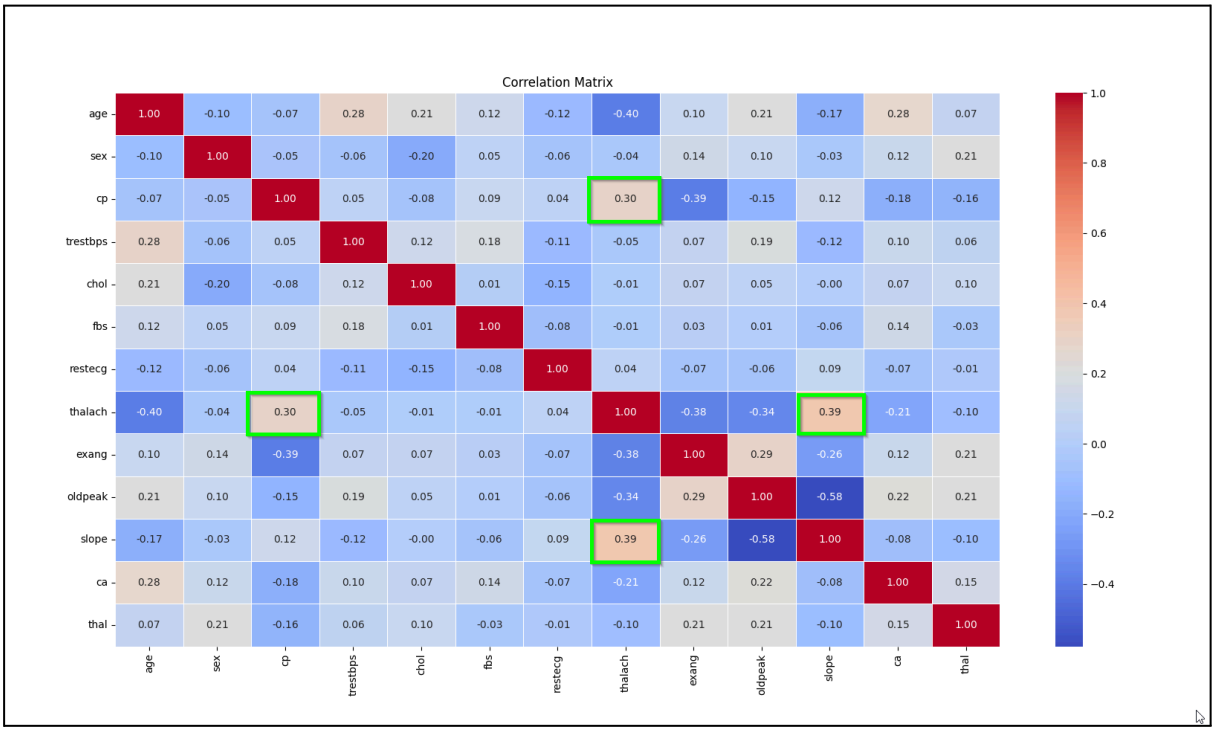


Figure 27: Heart dataset - Correlation of features

E. We can see that there is no correlation between the features, in this case we are going to use all the features.

F. Create train and test data:

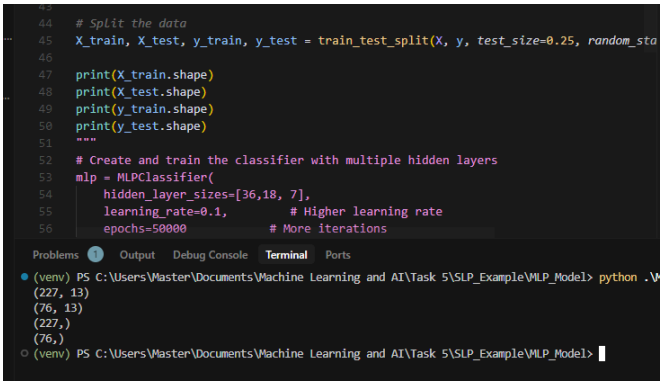


Figure 28: Heart dataset - Creation of test and train datasets

Analysis of results:

- **Results with layers [26,2] ALL the features:**

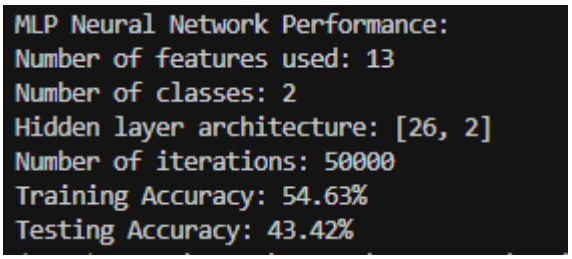


Figure 28: Heart dataset - Accuracy with [26,2] layers and all the features

- **Results with layers [52, 26, 13] ALL the features:**

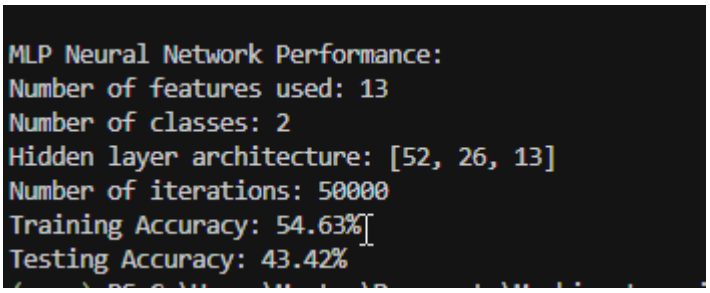
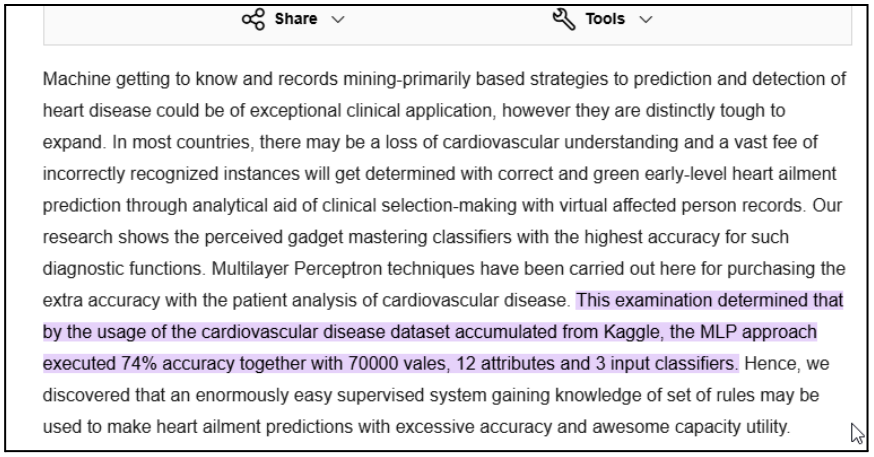


Figure 28: Heart dataset - Accuracy with [52,26,13] layers and all the features

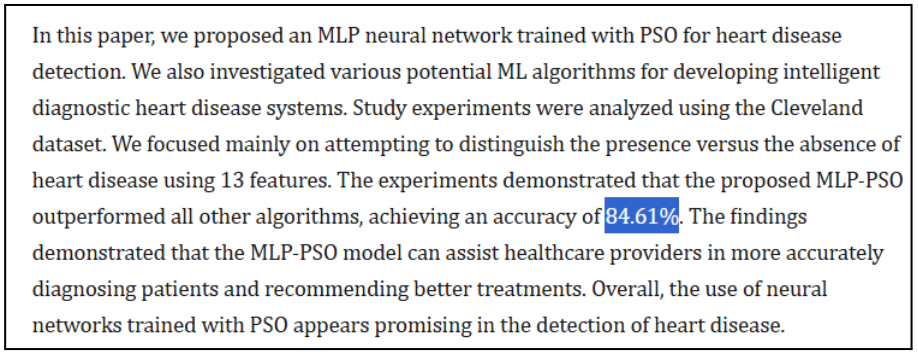
Comparison:

Source 1:



Reference 3: Kumar, A., Kumar, P., & Kumar, Y. (2023). Prediction of cardiovascular disease using MLP. AIP Conference Proceedings, 2495(1), 020059.

Source 2:



Reference 4: Al Bataineh, A., & Manacek, S. (2022). MLP-PSO Hybrid Algorithm for Heart Disease Prediction. Journal of Personalized Medicine, 12(8), 1208. https://doi.org/10.3390/jpm12081208

Conclusion

There are several points to mention in this conclusion:

1. The activation function plays a critical part in the SLP and MLP models. Even though using a step function was helpful to classify two classes, it didn't work for classifying more than two classes. Therefore, I used the sigmoid activation function, which allows for generating values between 0 and 1. It's worth mentioning that this is a non-linear function.
2. Including more neurons helps achieve better performance when classifying the data; however, more hidden layers do not necessarily mean that the accuracy will be better.
3. Preprocessing the data is a highly important step before calling the MLP model. We need to ensure that there are no NULL values in the dataset, identify the features and classes/labels, normalize the features, and identify the features with more correlation and those that can help us classify the data more accurately.
4. The learning rate can improve the performance of the model. By having a higher number, the model can make bigger adjustments to the weights, which can lead to classifying the data faster. Nevertheless, taking big steps in the adjustments can lead to "not converging" results; a lower rate has the potential to deliver better accuracy.
5. The derivative of the sigmoid function is used by the neural network to learn; it calculates how much to adjust the weights during learning. The formula $x*(1-x)$ is used because when x is closer to 0, the result is small, which means small adjustments. When x is around 0.5, the result is larger, leading to bigger adjustments.
6. When creating the weights matrix, it's crucial to consider their arrangement, as there are multiplications between input, hidden layers, and output layers. The number of columns of the left matrix must match the number of rows of the right matrix, resulting in a matrix that has the same number of rows as the left matrix and the same number of columns as the right matrix. E.g.
    a. $matrix\ a\ (1x4)\ *\ matrix\ b\ (4x4)\ =\ matrix\ c\ (1x4)$

These are the final result of each dataset:

Dataset	Epoch count	Layers	Features	Train Accuracy (%)	Test Accuracy (%)
Iris	15,000	[4,3]	4	98.21	100
Bean	50000	[36,18,7]	6	91.36	91.18
Bean	20,000	[36,18,7]	16	93.63	93.18
Heart	50000	[26,2]	12	54.63	43.42
Heart	50000	[56,26,13]	12	54.63	43.42

Github Repository

MiguelAnhalt/machinel_learning_ai_seminar_task_4

References, extra reading links, and resources

- Dua, D., & Graff, C. (2017). *UCI Machine Learning Repository* http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.
- NIST/SEMATECH. (n.d.). *Box Plot*. In *Engineering Statistics Handbook*. Retrieved May 2, 2025, from https://www.itl.nist.gov/div898/handbook/eda/section3/boxplot.htm
- GeeksforGeeks. (2023, November 20). *Iris Dataset*. https://www.geeksforgeeks.org/iris-dataset/
- W3Schools. (n.d.). *Python Classes and Objects*. https://www.w3schools.com/python/python_classes.asp
- GeeksforGeeks. (2024, October 22). *Activation functions*. Retrieved from https://www.geeksforgeeks.org/activation-functions/
- K. K. Patil and S. R. Kulkarni, "Classification of Dry Beans Using Machine Learning and Transfer Learning," 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 2022, pp. 1-6, doi: 10.1109/ICAAIC53929.2022.9689905.
- Kumar, A., Kumar, P., & Kumar, Y. (2023). Prediction of cardiovascular disease using MLP. AIP Conference Proceedings, 2495(1), 020059.