# Statistical Methods in Artificial Intelligence

Assignment 3

**Deadline : 11 October 2024 11:55 P.M**

Instructor: Prof Ravi Kiran Sarvadevabhatla

September 19, 2024

## 1 General Instructions

- Your assignment must be implemented in Python.
- While you're allowed to use LLM services for assistance, you must explicitly declare in comments the prompts you used and indicate which parts of the code were generated with the help of LLM services.
- Plagiarism will only be taken into consideration for code that is not generated by LLM services. Any code generated with the assistance of LLM services should be considered as a resource, similar to using a textbook or online tutorial.
- The difficulty of your viva or assessment will be determined by the percentage of code in your assignment that is not attributed to LLM services. If during the viva you are unable to explain any part of the code, that code will be considered as plagiarized.
- Clearly label and organize your code, including comments that explain the purpose of each section and key steps in your implementation.
- Properly document your code and include explanations for any non-trivial algorithms or techniques you employ.
- Ensure that your files are well-structured, with headings, subheadings, and explanations as necessary.
- Your assignment will be evaluated not only based on correctness but also on the quality of code, the clarity of explanations, and the extent to which you've understood and applied the concepts covered in the course.
- Make sure to test your code thoroughly before submission to avoid any runtime errors or unexpected behavior.

- Submit your assignment sticking to the format mentioned below. Not doing so will result in penalization.

```
├── assignments
│   ├── 1
│   │   └── a1.py
│   ├── 2
│   │   └── a2.py
│   ├── 3
│   │   ├── a3.py
│   │   ├── figures
│   │   │   └── README.md
│   │   └── README.md
│   ├── 4
│   │   └── a4.py
│   └── 5
│       └── a5.py
├── data
│   ├── external
│   │   ├── linreg.csv
│   │   ├── regularisation.csv
│   │   ├── spotify-2
│   │   │   ├── test.csv
│   │   │   ├── train.csv
│   │   │   └── validate.csv
│   │   └── spotify.csv
│   ├── interim
│   │   ├── 1
│   │   ├── 2
│   │   ├── 3
│   │   ├── 4
│   │   ├── 5
│   │   └── README.md
│   └── README.md
├── models
│   ├── MLP
│   │   └── MLP.py
│   ├── AutoEncoders
│   │   └── AutoEncoders.py
│   └── README.md
├── performance-measures
│   └── README.md
└── README.md
```

  - The `data/external` folder contains the datasets being provided to you. You should use the `data/interim` folder to store data that has

been transformed and that is in use.
  - There are folders for each assignment inside the `data/interim` directory, and make sure to store data specific to each assignment accordingly.
  - Store implementations of the models in the `models` folder.
    * Contrary to what was mentioned in the tutorial, you are supposed to make `fit` and `predict` routines inside the specific model class instead of making separate train, test and eval files. These routines **should** be present in the specific model class and have the naming convention mentioned.
  - For particular tasks related to the assignments, do them inside the folder for that specific assignment and import classes and data from the other files.
  - `performance-measures/` should contain the generalized implementations of all evaluation metrics, in a way such that they can be used for any model as and when needed.
  - Each assignment folder has a `README.md` which you will have to modify to make the final report with all your observations, analyses, and graphs.
  - The figures folder has been moved into the assignment-specific directory. You should save all the generated plots, animations, etc inside that folder and then include them in the report.
  - Mention the references used for each assignment in the report.

- The deadline will not be extended.
- MOSS will be run on all submissions along with checking against online resources.
- We are aware of how easy it is to write code now in the presence of LLM services, but we strongly encourage you to write the code yourself.
- We are aware of the possibility of submitting the assignment late in GitHub classrooms using various hacks. Note that we will have measures in place accordingly and anyone caught attempting to do the same will be given a straight zero in the assignment.
- You are not allowed to use external libraries except for standard ones such as numpy and matplotlib unless specified in the question or consulted by a TA.
- Vectorize the code as much as possible as that will also be evaluated along with the results and code.
- The total marks of this assignment is 250. There is 210 marks for the questions and 40 marks for miscellaneous(File structure(10), report(10), viva(10), code(10))

# 2 Multi Layer Perceptron Classification

[Marks : 90, Estimated Days: 7-8 days]

In this part, you are required to implement an MLP class for classification task from scratch using numpy, pandas and experiment with various activation functions and optimization techniques, evaluate the model's performance. You have to train this model on Wine Quality Dataset (WineQT.csv) to classify a wine's quality based on the values of its various contents.

**Note:** You will be asked to create an MLP class for regression later in the assignment. While you can create separate classes for classification and regression tasks, you will receive a **bonus** (Further details are provided further in the assignment) if you can combine them into a single class.

## 2.1 Dataset Analysis and Preprocessing (5 marks)

1. Describe the dataset using mean, standard deviation, min, and max values for all attributes. (1)
2. Draw a graph that shows the distribution of the various labels across the entire dataset. You are allowed to use standard libraries like Matplotlib. (2)
3. Normalise and standarize the data. Make sure to handle the missing or inconsistent data values if necessary. You can use sklearn for this. (2)

## 2.2 Model Building from Scratch (25 marks)

**Build an MLP classifier class with the following specifications:**

1. Create a class where you can modify and access the learning rate, activation function, optimizers, number of hidden layers and neurons per layer, batch size, number of epochs. (5)
2. Implement methods for forward propagation, backpropagation, training, early stopping, as well as methods for fitting and predicting data. Adhere to the standard naming convention of **fit** and **predict**. (5)
3. Different activation functions introduce non-linearity to the model and affect the learning process. Implement the Sigmoid, Tanh, and ReLU activation functions and make them easily interchangeable within your MLP framework. Also include a Linear layer for comparison purposes. (5)
4. Optimization techniques dictate how the neural network updates its weights during training. Implement methods for the Stochastic Gradient Descent

(SGD), Batch Gradient Descent, and Mini-Batch Gradient Descent algorithms from scratch, ensuring that they can be employed within your MLP architecture. (5)

5. Unit tests for gradient checking can help you debug your code and validate the correctness of your model by verifying the computed gradients. Implement a method to evaluate the accuracy of these gradients. (5)

**Note:** Before proceeding to hyperparameter tuning, we strongly recommend verifying your gradients using the gradient check method you implemented.

## 2.3 Model Training & Hyperparameter Tuning using W&B (15 marks)

Effective hyperparameter tuning can significantly enhance a model's performance. Integrate Weights & Biases (W&B) to log and track your model's metrics. By leveraging W&B and your validation set, experiment with hyperparameters such as learning rate, epochs, hidden layer neurons, activation functions, and optimization techniques. Use W&B to monitor loss during training and to record the effects of different activation functions and optimizers on model performance.

- Log your scores - loss and accuracy on validation set and train set using W&B.
- Report metrics: accuracy, f-1 score, precision, and recall.

Tune your model on various hyperparameters, such as learning rate, activation functions, optimizers, and hidden layer neurons.

1. Plot the trend of accuracy scores with change in these hyperparameters using W&B. (5)
2. Generate a table listing all the hyperparameters tested and their corresponding metrics mentioned above. (5)
3. Report the parameters for the best model that you get (for the various values you trained the model on). (5)

You should store the best model as it'll be used for later tasks.

## 2.4 Evaluating Single-label Classification Model (5 marks)

1. To assess the performance of the best model identified through hyperparameter tuning, evaluate it on the test set and report the metrics as outlined in section 2.2.

## 2.5 Analyzing Hyperparameters Effects (15 marks)

In this section, we will analyze the impact of non-linearity, learning rate, and batch size on the convergence of the model using a fixed network architecture. Using the best-performing model identified in section 2.2, conduct the following experiments:

1. **Effect of Non-linearity:** Select four activation functions and vary only these functions while keeping other hyperparameters constant. Plot the loss versus the number of epochs for each activation function on a single graph. (5)
2. **Effect of Learning Rate:** Choose four learning rates and vary only these rates while keeping other hyperparameters constant. Plot the loss versus the number of epochs for each learning rate on a single graph. (5)
3. **Effect of Batch Size:** Select four batch sizes and vary only these sizes while keeping other hyperparameters constant and the optimiser as mini-batch. Plot the loss versus the number of epochs for each batch size on a single graph. (5)

Finally, report your observations regarding the influence of each factor on the model's convergence.

## 2.6 Multi-Label Classification (15 marks)

For this part, you will be training and testing your model on Multi-label dataset: "advertisement.csv"

1. Create a separate MLP class specifically designed for handling multi-label classification tasks. (5)
2. Repeat the exercises in section 2.2 using the new multi-label MLP class. For evaluation, employ accuracy, recall, F1-score, precision, and Hamming loss as metrics, which are commonly used for multi-label classification tasks. (5)
3. To assess the performance of the best model identified through hyper-parameter tuning, evaluate it on the test set and report the metrics as outlined above. (5)

## 2.7 Analysis (10 marks)

Let us try to see what our classifier is good at classifying and what it is not.

1. For every datapoint in the dataset, observe which class it was classified into. Do you notice any classes which the model did really well at classifying and any class which it did bad at classifying. If so, why?

# 3 Multilayer Perceptron Regression

[Marks : 70, Estimated Days: 6-7 days]

In this task, you will implement a Multi-layer Perceptron (MLP) for regression from scratch, and integrate Weights & Biases (W&B) for tracking and tuning. Using the Boston Housing dataset, you have to predict housing prices while following standard machine learning practices. In this dataset, the column MEDV gives the median value of owner-occupied homes in $1000's.

## 3.1 Data Preprocessing (5 marks)

1. Describe the dataset using mean, standard deviation, min, and max values for all attributes. (1)
2. Draw a graph that shows the distribution of the various labels across the entire dataset. You are allowed to use standard libraries like Matplotlib. (1)
3. Partition the dataset into train, validation, and test sets (1)
4. Normalise and standarize the data. Make sure to handle the missing or inconsistent data values if necessary. (2)

## 3.2 MLP Regression Implementation from Scratch (20 marks)

In this part, you are required to implement MLP regression from scratch using numpy, pandas and experiment with various activation functions and optimization techniques, and evaluate the model's performance.

1. Create a class where you can modify and access the learning rate, activation function, optimisers, number of hidden layers and neurons. (4)
2. Implement methods for forward propagation, backpropagation, training, early stopping, as well as methods for fitting and predicting data. Adhere to the standard naming convention of **fit** and **predict**. (4)
3. Implement the Sigmoid, Tanh, and ReLU activation functions and make them easily interchangeable within your MLP framework. (4)
4. Implement methods for the Stochastic Gradient Descent (SGD), Batch Gradient Descent, and Mini-Batch Gradient Descent algorithms from scratch, ensuring that they can be employed within your MLP architecture. (4)
5. Unit tests for gradient checking can help you debug your code and validate the correctness of your model by verifying the computed gradients. Implement a method to evaluate the accuracy of these gradients. (4)

### 3.3 Model Training & Hyperparameter Tuning using W&B (15 marks)

- Log your scores - loss (Mean Squared Error) on the validation set using W&B.
- Report metrics: MSE, RMSE, R-squared.

Tune your model on various hyperparameters, such as learning rate, activation functions, optimizers, and hidden layer neurons.

1. Plot the trend of accuracy scores with change in these hyperparameters using W&B. (5)
2. Generate a table listing all the hyperparameters tested and their corresponding metrics mentioned above. (5)
3. Report the parameters for the best model that you get (for the various values you trained the model on). (5)

### 3.4 Evaluating Model (5 marks)

1. To assess the performance of the best model identified through hyperparameter tuning, evaluate it on the test set and report the metrics as outlined in section 3.3. (MSE, MAE).

### 3.5 Mean Squared Error vs Binary Cross Entropy (15 marks)

In this part, we will perform binary classification on the Pima Indians Diabetes dataset (diabetes.csv), using MSE and BCE loss function in the final layer and observe their effects.

1. Use the MLP class to model a simple logistic regression model which takes input as the features of the data and maps it to one neuron followed by a sigmoid activation function. Make 2 models, one that uses BCE loss and the other that uses MSE loss. (5)
2. Plot the loss vs epochs for both models in two different plots. (5)
3. Note down the observations and differences that you see in both loss functions. What do you see in the convergence plots? (5)

### 3.6 Analysis (10 marks)

Let us try to see what our model is performs well on.

1. For every datapoint in the test dataset, observe the MSE Loss. Do you notice if there is a pattern in the datapoints for which it gives a high MSE Loss or a low MSE Loss. If so, why?

### 3.7 [BONUS: 15 marks]

Bonus sections give you marks to compensate for other places in the assignment where you have not performed well. If however you attain full marks, the bonus quetion will not reward you extra marks.

1. Identify common parts in the MLP classification and regression task and attempt to generalise both processes so that you can write one class which takes care of both classification and regression together.

# 4 AutoEncoders

[Marks : 50, Estimated Days: 4-5 days]

We have been performing some kind of classification task in every assignment up until now. Let us play with another way to classify. We will be using the same dataset as that in assignment 1

## 4.1 AutoEncoder implementation from scratch (15 marks)

In this part, we will make an autoencoder class. Make sure the dimension that your dataset is reduced to using the encoder is the same as the optimal dimensions you reduced it to using PCA in assignment 2.

1. Make an AutoEncoder class which calls the MLP regression class to make neural network model which takes in an input vector, reduces to n-dimensions and reconstructs it back to the original vector.
2. The class must have the following 3 methods:
   - The initialisation method which initialises the model. (5)
   - The fit method which trains the model. (5)
   - The get_latent method which will return the reduced dataset. (5)

## 4.2 Train the autoencoder (10 marks)

1. Now, using the dataset from assignment 1, train the auto encoder using the forward and backward pass methods of the class.

## 4.3 AutoEncoder + KNN (15 marks)

1. Use the trained model on the dataset to obtain the output of the encoder which will be the reduced dataset. Apply the KNN model from assignment 1 on the reduced dataset and return the validation F1 score, accuracy, precision, and recall. (5)
2. Compare these metrics with those obtained in Assignment 1 and Assignment 2 and provide an analysis of any differences or improvements. (10)

## 4.4 MLP classification (10 marks)

Now let us see how well the MLP classifier does.

1. With your own choice of hyperparameters, train and classify the dataset using the MLP class. Return the validation F1 score, accuracy, precision, and recall and compare it with the previous results.

---

Good luck with the assignment!