|  |  |
| --- | --- |
| **Ex No: 1a**  **Date: 12th August 2024** | **Practice Python** |

**Objective:** Develop a grading system that determines a student's eligibility for examinations and calculates their final grade based on tutorial, test, and examination marks.

**Description:**

This lab notebook is structured to simulate a grading system through a step-by-step approach:

* **Student Information Input**: The program collects a student number and various marks for tutorials, tests, and examinations.
* **Eligibility Check**: An average of tutorial and test marks is computed. If the average is below 40%, the student receives an "F" grade and is ineligible for the examination.
* **Final Grade Calculation**: If eligible, the final mark is calculated using a weighted formula: 25% tutorial, 25% test, and 50% exam. Based on the final mark, the student receives a grade (A to E).

**Model:**

No statistical or machine learning model is applied. The logic flow is entirely conditional, based on programmed decision-making steps:

* Averages and conditions check the eligibility for exams.
* The final grade uses fixed thresholds to categorize scores (e.g., 80–100 as "A," 70–79 as "B," etc.).

**Building the parts of the algorithm:**

1. Input Collection: Gathers student number, tutorial mark, and test mark as inputs.
2. Average Calculation: Computes the average of tutorial and test marks to determine eligibility.
3. Eligibility Check: If the average is below 40%, the student receives an "F" and cannot take the exam.
4. Exam Mark and Final Grade Calculation: If eligible, prompts for the exam mark and calculates a weighted final grade.
5. Grade Assignment: Assigns a final grade based on fixed thresholds.

**Key Observations:**

* This grading system is straightforward and effective for small-scale, manual input scenarios. However, it could be extended to support batch processing of multiple students and automated data input from files or databases.
* An observation is that the grading system may benefit from additional features like error handling for invalid inputs and user-friendly interfaces for broader usability.

**GitHub Link:** <https://github.com/adrocx/DeepLearning>

|  |  |
| --- | --- |
| **Ex No: 1b**  **Date: 12th August 2024** | **NumPy basics** |

**Objective:** Introduce foundational programming exercises with Python, focusing on numpy functions and mathematical operations, such as the sigmoid function, essential in machine learning.

**Description:**

This lab notebook serves as an introduction to numpy functions and basic mathematical operations, aimed at building familiarity with Python for scientific computing:

* **Function Definition**: A function, basic\_sigmoid, calculates the sigmoid of a given scalar input using both math.exp() and numpy.exp(). The sigmoid function, commonly used in machine learning, maps inputs to values between 0 and 1.
* **Importance of Numpy**: This exercise demonstrates the advantages of using numpy over math functions for handling larger arrays and matrices in Python, an essential skill for data scientists and machine learning practitioners.

**Model:**

The notebook includes a simple function model:

* **Sigmoid Function**: This function, common in logistic regression and neural networks, maps real-valued numbers to a bounded range between 0 and 1.
* The function model demonstrates how to implement the sigmoid function from scratch, an essential part of building neural networks in machine learning.

**Building the parts of the algorithm:**

1. **Function Setup**: Defines the basic\_sigmoid function.
2. **Mathematical Operation**: Uses the formula for the sigmoid function, initially with math.exp() and then with np.exp() for optimized computation.
3. **Output Validation**: Tests the function with sample input to ensure it works as expected.

**Key Observations:**

* The introductory exercises on numpy are essential for anyone transitioning into scientific computing or machine learning. Understanding numpy’s efficient handling of mathematical functions is fundamental for building models that involve matrix operations and data transformations.
* Observing the difference between math.exp() and np.exp() underscores the efficiency of numpy when dealing with large datasets, which is crucial in high-performance computing tasks within machine learning and data science.

**GitHub Link:** <https://github.com/adrocx/DeepLearning>