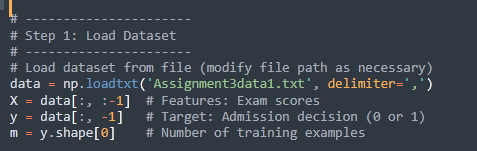
Assigment 3

The assigment focuses on using logistic regression to predict whether a student will be admitted to a university based on their scores from two exams. Logistic regression is a type of machine learning algorithm used for binary classification, where the goal is to determine the probability of one of two possible outcomes (admitted or not admitted). The dataset includes the exam scores of previous applicants and whether they were admitted, which is used to train the model. The project involves visualizing the data to understand the relationship between exam scores and admission decisions, building the logistic regression model to learn from the data, and optimizing the model to minimize errors in predictions. The project includes visualizing the decision boundary, predicting outcomes for new scores, and analyzing how the model behaves using plots like the sigmoid function and cost function.

I started by importing three libraries: numpy (imported as np) and matplotlib.pyplot (imported as plt). np is used for loading, storing, and manipulating data as arrays, while plt is used for visualizing the data through scatter plots, line plots, and other graphs. The minimize function from scipy.



This section of the code begins by loading the dataset from a file called Assignment3data1.txt. The dataset is in a comma-separated format, and the numpy function np.loadtxt is used to read the data into a structured array. The first two columns of the dataset, representing the exam scores, are extracted into a variable X, while the third column, which contains binary labels indicating whether the student was admitted (1) or not (0), is stored in the variable y. The total number of examples in the dataset is calculated using len(y) and stored in the variable m.



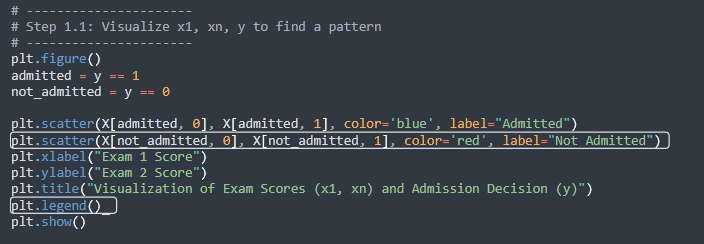
The code bellow help us to visualize the relationship between two features (exam 1 and exam 2 scores) and the target variable (admission decision).

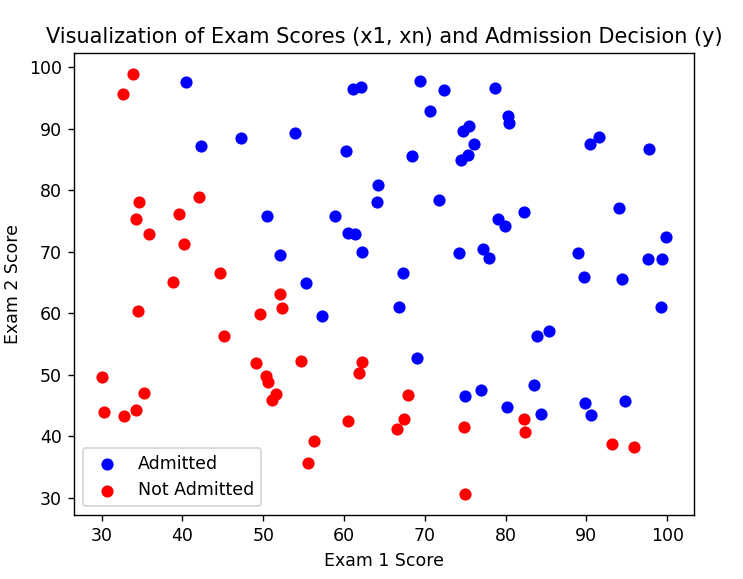
Separate the Data into Classes:

* **admitted = y == 1** filters the data points where the student was admitted (target = 1).
* **not\_admitted = y == 0** filters the data points where the student was not admitted (target = 0).

Scatter Plot for Each Class:

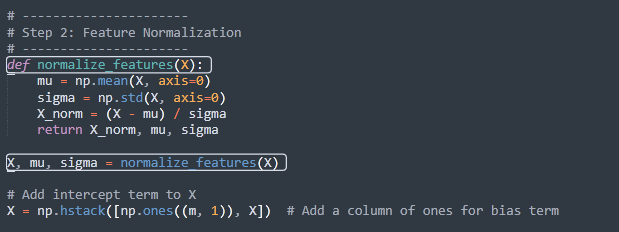
* **plt.scatter(X[admitted, 0], X[admitted, 1], ...)** plots the scores of students who were admitted in blue.
* **plt.scatter(X[not\_admitted, 0], X[not\_admitted, 1], ...)** plots the scores of students who were not admitted in red.





This step normalizes the features (input variables) and adds an intercept term to the feature matrix for logistic regression.

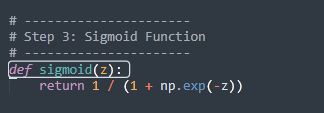
* The function **normalize\_features(X)** adjusts each feature so that it has a mean of 0 and a standard deviation of 1. This is important to ensure that the gradient descent algorithm works efficiently. Feature normalization ensures that all input features contribute equally to the model, making optimization faster and more stable
  + mu = np.mean(X, axis=0) calculates the mean of each feature.
  + sigma = np.std(X, axis=0) calculates the standard deviation of each feature.
  + X\_norm = (X - mu) / sigma scales each feature by subtracting the mean and dividing by the standard deviation.
* The function returns the normalized feature matrix (X\_norm) along with the mean (mu) and standard deviation (sigma) for future use.
* X, mu, sigma = normalize\_features(X) applies the normalization to the original feature matrix. Now, the features are scaled and ready for use in the model.
* X = np.hstack([np.ones((m, 1)), X]) adds a column of ones to the feature matrix. This is needed for the intercept term (or bias) in logistic regression. The bias allows the model to fit data that does not pass through the origin.



The sigmoid function is used to calculate probabilities in logistic regression by mapping input values to a range between 0 and 1. This is essential for predicting probabilities. The function converts the linear combination of features and weights into a probability value. This allows logistic regression to predict whether a data point belongs to one class or another.

* z: The input value(s), which can be a number, vector, or matrix.
* np.exp(-z): Calculates the exponential of -z.
* 1 / (1 + ...): The sigmoid formula ensures the output is between 0 and 1.

For very large positive z, the sigmoid value is close to 1. For very large negative z, the sigmoid value is close to 0. When z = 0, the sigmoid value is 0.5, representing equal probability.



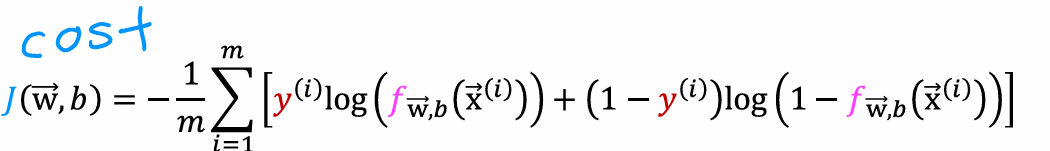
The compute\_cost function provided is designed to calculate the cost (or loss) for a regression model, but it currently implements the **mean squared error (MSE)** cost function. This cost function is typically used for **linear regression**, not logistic regression.

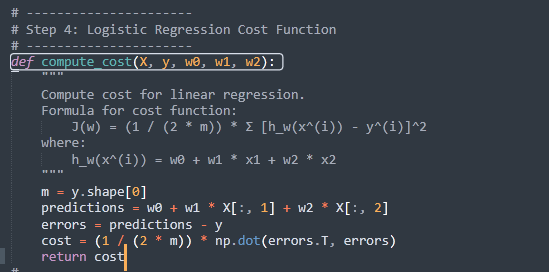
Here’s a detailed explanation of each part of the function:

The function compute\_cost calculates the **mean squared error** for a regression model based on the difference between predicted values (h\_w(x)) and actual target values (y).

* X: The input feature matrix where each row represents a training example, and each column represents a feature. Here, it includes an intercept column (bias term).
* y: The target vector containing the actual values (e.g., admission decisions in this case).
* w0, w1, w2: The model parameters (weights) that correspond to the intercept term and the coefficients for the features.

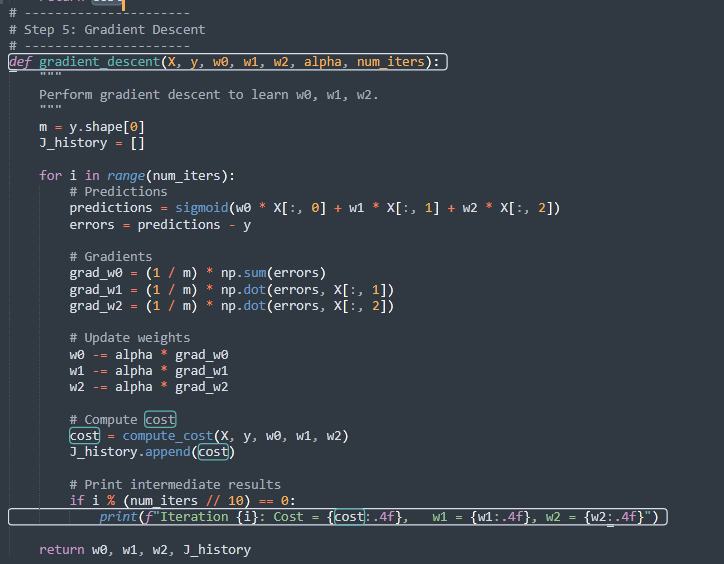
First, it computes the number of training examples (m) from the target variable (y). Then, it calculates the model's predictions by taking the dot product of the feature matrix (X) and the weights (w). The difference between these predictions and the actual values (y) is computed as the error. Using the error, it calculates the cost by summing the squared errors, dividing by 2m, which scales the result to represent the average squared error. Finally, it returns the cost, which quantifies the model's performance and is used to guide improvements during gradient descent.





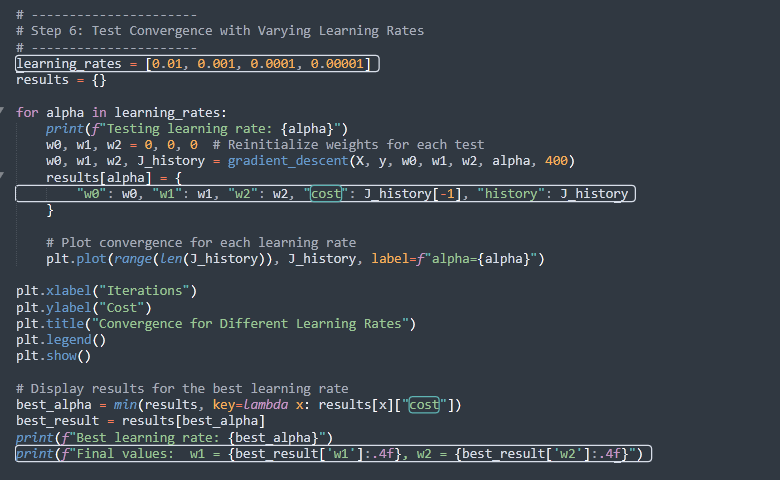
The gradient\_descent function performs optimization of weights (w0, w1, w2) for logistic regression by minimizing the cost function iteratively. It takes the feature matrix X, target vector y, initial weights, learning rate alpha, and the number of iterations num\_iters as inputs. The function initializes the number of training examples (m) and a list to store the cost history. In each iteration, it calculates predictions using the sigmoid function, computes the errors as the difference between predictions and actual target values, and derives the gradients for each weight. The gradients are computed by taking the average partial derivative of the cost function with respect to each weight. These gradients are then used to update the weights by subtracting the product of the learning rate and the respective gradient.

The cost is calculated at each iteration using the compute\_cost function, and its value is stored for tracking convergence. Periodically, the function prints the current iteration, cost, and updated weights for monitoring progress. After all iterations, the function returns the optimized weights and the cost history. This approach ensures the weights are adjusted iteratively to minimize the cost and improve the model's predictions.

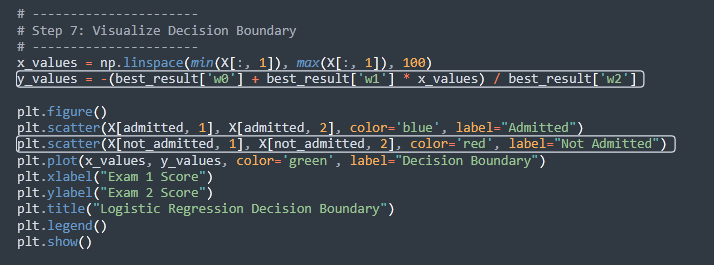


This block of code tests the convergence of the gradient descent algorithm with varying learning rates. It initializes a list of learning rates (learning\_rates) and iterates through each value to train the model using the gradient\_descent function. For each learning rate, the weights (w0, w1, w2) are reinitialized, and the gradient descent is executed for a fixed number of iterations (400). The results, including the final weights, cost, and cost history, are stored in a dictionary (results) with the learning rate as the key.

The convergence is visualized by plotting the cost values over iterations for each learning rate, making it easier to analyze the effect of the learning rate on the optimization process. The best learning rate is determined by finding the key in the results dictionary with the lowest final cost. Finally, the best learning rate and the corresponding optimized weights are printed for reference, providing insight into the most effective learning rate for minimizing the cost function.



This block of code visualizes the logistic regression decision boundary by plotting the data points and the boundary line that separates the two classes. It calculates x\_values as a range of Exam 1 scores and y\_values using the decision boundary equation based on the optimized weights. The scatter plot shows "Admitted" and "Not Admitted" points in blue and red, while the decision boundary is plotted as a green line. The plot is labeled, includes a legend, and is displayed to show how well the model distinguishes between the classes.



In the last step the code predicts the probability of admission for a given set of exam scores using the optimized logistic regression model. The function predict\_probability normalizes the input features based on the training data's mean (mu) and standard deviation (sigma) and adds a bias term. The probability is calculated using the sigmoid function applied to the weighted sum of the features and the model's optimized weights.

A exam scores interval between [50, 100] is choosen to demonstrate the prediction. The resulting probability is compared to the threshold of 0.5 to classify the admission decision as "Accepted" or "Rejected." The predicted probability and decision are printed, summarizing the likelihood of admission for the given scores.

