Linear Regression

In the `LinearRegressionImp.py` file, you will implement the Linear Regression algorithm and test its performance using the diabetes dataset. You do not need to download the dataset as it is included in the `scikit-learn` library. For evaluation, we provide you the test function `test_LR()` (**Note: keep this function as it is in your submission**). This function loads the diabetes dataset, runs your implementation and outputs the mean-squared-error on the test set (for each coding approach). Follow the below instructions to get started.

Part 1a: Linear Regression using NumPy and Python

You will be implementing in `LinearRegressionImp.py` the exact computation of the solution for linear regression using the `NumPy` library functions via the least squares method. You will be computing the parameters of a linear plane that best fits the training dataset. Specifically, you will be implementing three functions which are detailed in the following:

- Function 1: `def fit_LinRegr(X_train, y_train)`
  - Inputs: `X_train, y_train`
    The first input `X_train` represents the matrix of input features that belongs to $\mathbb{R}^{NXd}$ where N is the total number of training samples and d is the dimension of each input feature vector. The second input `y_train` is an N dimensional vector where the i^{th} component represents the output observed in the training set for the i^{th} row in `X_train` matrix which corresponds to the i^{th} input feature. Each element in `y_train` takes a value in $\mathbb{R}$.
  - Output: `w`
    The output of this function is the vector `w` that represents the coefficients of the line computed using the least square method that best fits the training data points. The dimensions of this vector is $d + 1$ as the offset term is accounted in the computation.
  - Function implementation considerations:
    This function computes the parameters `w` of a linear plane which best fits the training dataset. Useful functions in `NumPy` for implementing this function are: `shape` (for identifying the number of rows and columns in a matrix), `hstack` (to add an additional column to the front of the original input matrix), `ones` (for setting the first column of the input feature matrix to 1), `dot` function to take the dot product of two vectors of the same size, `transpose` function for taking the transpose of a vector, and `linalg.inv` function for finding the inverse of a square matrix.
- Function 2: `def mse(X, y, w)`
  - Inputs: `X`, `y` and `w`
    The inputs to this function, `X` and `y`, are defined in a manner similar to `X_train` and `y_train` in Function 1. `w` represents the coefficients of a linear plane and is of $d + 1$ dimensions.

- Output: `avgError`
  The output of this function is the mean squared error introduced by the linear plane defined by `w`.
- Function implementation considerations:
  You will use the `pred` function that you will implement in Function 3 to find the output of the linear plane defined by `w`. Functions from `NumPy` that will be useful are: `shape` (for identifying the number of rows and columns in a matrix), `hstack` (to add an additional column to the front of the original input matrix), `ones` (for setting the first column of the input feature matrix to 1), and `dot` function to take the dot product of two vectors of the same size.

- Function 3: `def pred(X_i,w)`
  - Inputs: `X_i`, `w`
    The first input is `X_i` which is the feature vector of $d + 1$ dimensions of the i^{th} test datapoint. `w` is defined in a manner similar to Function 2.
  - Output: Predicted value
    The output predicted by the linear regression model defined by `w` for the input datapoint `X_i` is computed. This output can take values in the Real space $\mathbb{R}$.
  - Function implementation considerations:
    The `dot` product function in `NumPy` will be useful for this function implementation.

For this implementation, we also provide the test function `subtestFn()`. This function loads a toy dataset, runs your NumPy implementation and return a message indicating whether your solution works when `X_train` is not a full-column rank matrix, i.e. the input features are not linearly independent.

Answer the following question(s), write and save your answer in a separate `PA1_qa.pdf` file. Remember to submit this file together with your code.

- When we input a singular matrix, the function `linalg.inv` often returns an error message. In your `fit_LinRegr(X_train, y_train)` implementation, is your input to the function `linalg.inv` a singular matrix? Explain why. (2 marks)
- As you are using `linalg.inv` for matrix inversion, report the output message when running the function `subtestFn()`. We note that inputting a singular matrix to `linalg.inv` sometimes does not yield an error due to numerical issue. (1 marks)
- Replace the function `linalg.inv` with `linalg.pinv`, you should get the model's weight and the `"NO ERROR"` message after running the function `subtestFn()`. Explain the difference between `linalg.inv` and `linalg.pinv`, and report the model's weight. (2 marks)

The following is the mark breakdown for Part 1a:

- Test file successfully runs all three implemented functions: 8 marks

- Outputs of all four functions are close to the expected output: 12 marks
- Code content is organized well and annotated with useful comments: 5 marks
- Questions are answered correctly: 5 marks

Part 1b: Linear Regressions using scikit-learn

In this part, you will use the `scikit-learn` library to train the linear regression model. You will then compare the performance of your implementation in Part 1a with the one available in the `scikit-learn` library. You will implement one function in this part in the `LinearRegressionImp.py` file. You can refer to the `scikit-learn` demo covered in the lecture to aid you with this completing this part.

- Function: `def test_SciKit(X_train, X_test, Y_train, Y_test)`
  - Inputs: `X_train, X_test, Y_train, Y_test`
    The first input `X_train` represents the matrix of input features that belongs to $\mathbb{R}^{NXd}$ where N is the total number of training samples and d is the dimension of each input feature vector. The second input `X_test` represents the matrix of input features that belongs to $\mathbb{R}^{MXd}$ where M is the total number of testing samples and d is the dimension of each input feature vector. The third input `Y_train` is an N dimensional vector where the i^{th} component represents the output observed in the training set for the i^{th} row in `X_train` matrix which corresponds to the i^{th} input feature. The similar counterpart to `X_test` is `Y_test`.
  - Output: `error`
    This function will output the mean squared error on the test set, which is obtained from the `mean_squared_error` function imported from `sklearn.metrics` library to report the performance of the model fitted using the linear regression algorithm available in the `sklearn.metrics` library.
  - Function implementation considerations:
    `LinearRegression` and `mean_squared_error` functions imported from `sklearn.lin`
    `sklearn.metrics` will be utilized to fit the linear classifier using the linear regression algorithm and evaluate the performance of this algorithm. As presented in the `scikit-learn` demo in the lecture, you will initiate an object of the `LinearRegression` type, you will run the `fit` function to train the model, you will use the `predict` function to perform predictions using the trained algorithm and finally you will use the `mean_squared_error` function to compute the mean squared error of the trained model.

How close is the performance of your implementation in comparison to the existing modules in the `scikit-learn` library? Place this comment at the end of the code file.

The following is the mark breakdown for Part 1b:

- Test file successfully runs implemented function: 6 marks

- Output is close to the expected output from the test file: 8 marks
- Code content is organized well and annotated with comments: 6 marks