ECE421: Introduction to Machine Learning — Fall 2024

Assignment 1: Pocket Algorithm and Linear Regression

Due Date: Thursday, September 26, 11:59 PM

Objectives

In this assignment, you will be implementing the following algorithms: **Pocket Algorithm** and **Linear Regression**; using two different methods of coding approaches.

- In the first approach, you will implement these algorithms using `Python` and functions in the `NumPy` library only.
- In the second approach, you will use `scikit-learn` to gauge how well your initial implementation using `NumPy` functions fares in comparison to off-the-shelf modules available in `scikit-learn`.
- You will also be asked to answer several questions related to your implementations.

To avoid any potential installation issue, you are encouraged to develop your solution using Google Colab notebooks.

Requirements

In your implementations, please use the function prototype provided (*i.e.* name of the function, inputs and outputs) in the detailed instructions presented in the remainder of this document. We will be testing your code using a test function that which evokes the provided function prototype. If our testing file is unable to recognize the function prototype you have implemented, you can lose significant portion of your marks. In the assignment folder, the following files are included in the `starter_code` folder:

- `PerceptronImp.py`
- `LinearRegressionImp.py`

These files contain the test function and an outline of the functions that you will be implementing. You also need to submit a separate `PA1_qa.pdf` file that answer questions related to your implementations.

1 Pocket Algorithm

In the `PerceptronImp.py` file, you will implement the Pocket Algorithm to classify two different classes in the `IRIS` 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_Part1()`. This function loads the `IRIS` dataset, runs your implementation and outputs the confusion matrix on the test set (for each coding approach). Follow the below instructions to get started.
[**NOTE:** keep the function `test_Part1()` as it is in your submission.]

1.1 Pocket Algorithm Using `NumPy`

You will be implementing the Pocket Algorithm using the `NumPy` library functions in the `PerceptronImp.py` file. You will be computing parameters of a linear plane that best separates input features belonging to two classes. Specifically, you will be implementing the three functions listed below. For more instructions and hints, see the comments in the `PerceptronImp.py` file.

- `fit_Perceptron(X_train, y_train)`
  Function implementation considerations: This function computes the parameters `w` of a linear plane which separates the input features from the training set into two classes specified by the training dataset. As the pocket algorithm is used, you will set the maximum number of epochs (the maximum number of passes over the training data) to 5000. Useful functions in `NumPy` for implementing this function are:

  - `zeros`: for initializing the weight vector with 0s.
  - `shape`: for identifying the number of rows and columns in a matrix.
  - `hstack`: for adding 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`: for taking the dot product of two vectors of the same size.

  You will also use the function `errorPer` that you will implement next to compute the average number of misclassifications for the plane you are currently considering with respect to the training dataset.

- `errorPer(X_train, y_train, w)`
  Function implementation considerations: This function finds the ratio of missclassified points. Note that a point on the hyperplane must be considered as a missclasification.

- `confMatrix(X_train, y_train, w)`
  Function implementation considerations: This function will populate a two-by-two matrix. Using the zero-index, the (0, 0) position represents a count of the total number of points correctly classified to be class -1 (True Negative). The (0, 1) position contains a count of total number of points that are in class -1 but are classified to be class $+1$ by the classifier (False Positive). The (1, 0) position contains a count of total number of points that are in class $+1$ but are classified to be class -1 by the classifier (False Negative). The (1, 1) position represents a count of the total number of points correctly classified to be class $+1$ (True Positive). Refer to the table below.

		PREDICTION	
		-1	+1
ACTUAL	-1	True Negative	False Positive
	+1	False Negative	True Positive

  You may find the function `pred` implemented in `perceptronImp.py` useful.

The following is the mark breakdown for Part 1.1:

(i) Test file successfully runs all four implemented functions: 8 marks

(ii) Outputs of all four functions are close to the expected output: 12 marks

(iii) Code content is organized well and annotated with useful comments: 10 marks

1.2 Pocket Algorithm Using `scikit-learn`

In this part, you will use the `scikit-learn` library to train the binary linear classification model. You will then compare the performance of your implementation in Part 1.1 with the one available in the `scikit-learn` library. You will implement one function in this part in the `PerceptronImp.py` file. You can refer to the Perceptron class and function reference by scikit-learn, available here, to familiarize yourself with the implementation of this linear model and its parameters. Specifically, you will be implementing the function listed below. For more instructions and hints, see the comments in the `PerceptronImp.py` file.

- `(X_train, X_test, y_train, y_test)`
  Function implementation considerations: `Perceptron` and `confusion_matrix` functions imported from `sklearn.linear_model` and `sklearn.metrics` will be utilized to fit the linear classifer using the Perceptron learning algorithm and evaluate the performance of this algorithm. First, initiate an object of the `Perceptron` type. Then, run the `fit` function to train the classifier. Next, use the `predict` function to perform predictions using the trained algorithm. Finally, use the `confusion_matrix` function to identify how many points have been classified correctly and incorrectly in the test dataset. **Refer to the questions below** in order to set the parameters for your `Perceptron` model.

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

1.2.a) Refer to the documentation, what is the functionality of the `tol` parameter in the `Perceptron` class? (2 marks)

1.2.b) If we set `max_iter=5000` and `tol=1e-3` (the rest as default), does this guarantee that the algorithm will pass over the training data 5000 times? If not, ensure that the algorithm will pass over the training data 5000 times? (2 marks)

1.2.c) How can we set the weights of the model to a certain value? (2 marks)

1.2.d) How close is the performance (through confusion matrix) of your `NumPy` implementation in comparison to the existing modules in the `scikit-learn` library? (2 marks)

The following is the mark breakdown for Part 1.2:

(i) Test file successfully runs implemented function: 4 marks

(ii) Output is close to the expected output from the test file: 5 marks

(iii) Code content is organized well and annotated with comments: 3 marks

(iv) Questions are answered correctly: 8 marks

2 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_Part2()`. 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.
[**NOTE:** keep the function `test_Part2()` as it is in your submission.]

2.1 Linear Regression Using `NumPy`

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 the two functions which are detailed in the following. For more instructions and hints, see the comments in the `LinearRegressionImp.py` file.

- `fit_LinRegr(X_train, y_train)`
  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`: for adding 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`: for taking the dot product of two vectors of the same size.
  - `transpose`: for taking the transpose of a vector.
  - `linalg.inv`: for finding the inverse of a square matrix.
- `mse(X, y, w)`
  Function implementation considerations: This function finds the mean squared error introduced by the linear plane defined by `w`. You can use the `pred` function that is implemented for you to find the prediction by the classifier defined by `w`. Moreover, Useful functions in `NumPy` for implementing this function are `shape`, `hstack`, `ones`, and `dot`.

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.

2.1.a) 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)

2.1.b) 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)

2.1.c) 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 2.1:

(i) Test file successfully runs the two implemented functions: 8 marks

(ii) Outputs of all the functions are close to the expected output: 12 marks

(iii) Code content is organized well and annotated with useful comments: 5 marks

(iv) Questions are answered correctly: 5 marks

2.2 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 2.1 with the one available in the `scikit-learn` library. You will implement one function in this part in the `LinearRegressionImp.py` file. Specifically, you will be implementing the function which is detailed in the following. For more instructions and hints, see the comments in the `LinearRegressionImp.py` file.

- `test_SciKit(X_train, X_test, y_train, y_test)`
  Function implementation considerations: 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 fitted `LinearRegression` model using the linear regression algorithm available in the `sklearn.linear_model` library. First, initiate an object of the `LinearRegression` type. Next, run the fit function to train the model. Then, use the predict function to perform predictions using the trained algorithm. Finally, 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 your answer as a comment at the end of the code file.

The following is the mark breakdown for Part 2.2:

(i) Test file successfully runs implemented function: 6 marks

(ii) Output is close to the expected output from the test file: 8 marks

(iii) Code content is organized well and annotated with comments: 6 marks

3 Discussion

Please answer the following short questions so we can improve future assignments.

3.a) How much time did you spend on each part of this assignment? (optional)

3.b) Any additional feedback? (optional)

4 Turning It In

You need to submit your version of the following files:

- PerceptronImp.py
- LinearRegressionImp.py
- PA1_qa.pdf that answer questions related to the implementations.
- The cover file (this file) with their name and student ID filled.

Please pack them into a single folder, compress into a .zip file and name it as PA1.zip.