INF 552 Programming Assignment 4

Linear Classification and Regression

4472-6221-33 Yo Shuan Liu

I. Introduction

I complete this homework in Python3. My code can be executed from terminal, and there is no absolute path for input files. User only needs to follow below instructions to execute this script:

1) Put all the input .txt file needed (classification.txt, linear-regression.txt) and this code **under the same directory**.
2) Change directory to this folder and execute below line in terminal:
   python LinearClassification_Regression.py
3) After printing out all the output, a message will show up in the terminal and ask user to **press [enter]** in order to end the execution and close all the output graph windows.

Since the data structure used in each algorithm in this assignment are the same, I will introduce it only once in the Data Structure session.

In this report, I will introduce my implementations of perceptron algorithm and pocket algorithm together, since the ideals are quite similar. Then I will introduce the logistic regression and finally the linear regression implementation. Under each part, I will cover:

1) Result
2) Challenges I Face
3) Software Familiarization

Then I will list out all the code level optimization I have done in this assignment. Finally, I will briefly introduce three interesting applications regarding linear classification, logistic regression and linear regression.

II. Data Structure

Besides columns representing independent variables in input datasets, I add a column of ones for intercept estimation, therefore, all of the data points are stored as a N*(d+1) matrix, I named it as **X** in my python code, where N is the total number of rows in dataset and d is the original number of dimension. The actual classification is stored in another N*1 array, which is named as **y** in my python code. Output weights are arrays with length (d+1).

III. Implementation of Perceptron Algorithm and Pocket Algorithm

Implementation

I run linear regression on binary classification input data first, and then use the weight output by linear regression as initial weight of my PLA and pocket algorithm.

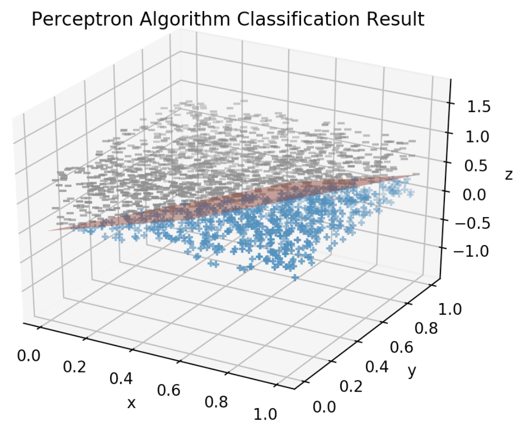
Result

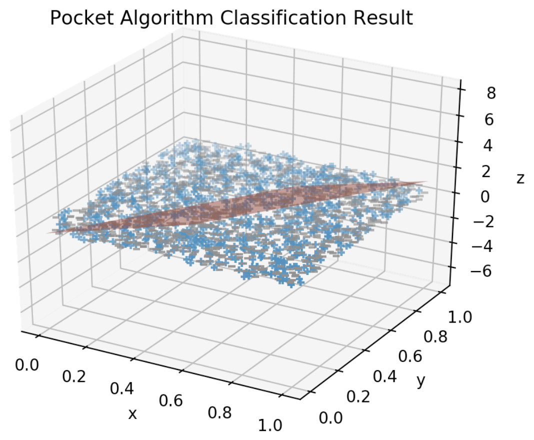
- Perceptron Algorithm
  Accuracy: 100.00%
  Number of Iteration: 538
  Weight: [-0.08021714 69.62813721 -55.77353248 -41.6615246]*
- Pocket Algorithm
  Accuracy: 53.30%
  Number of Iteration: 7000 (as request)
  Weight: [-0.01572516 -3.83583945   3.2871186   0.50330905]*

* coefficients for x0, x1, x2, x3

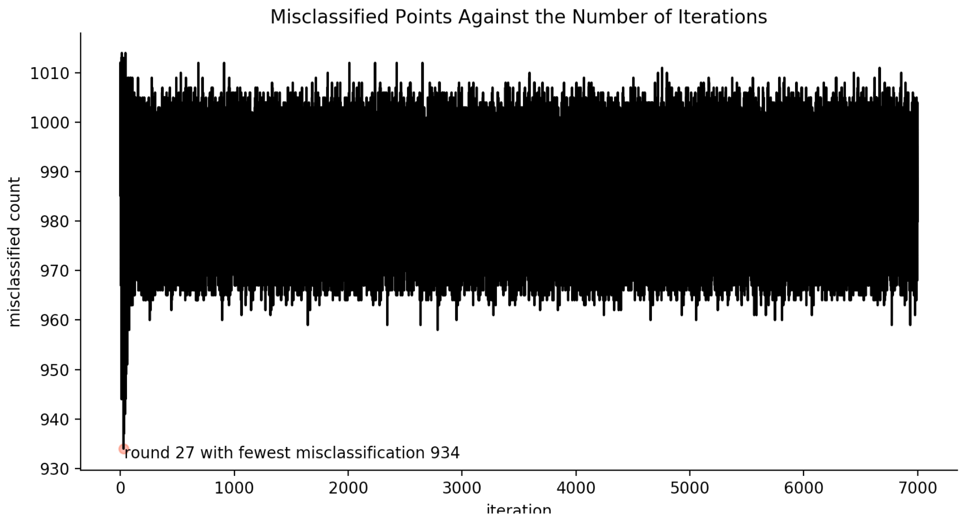
Visualization of data points and decision boundary:
- Data points with positive label are marked as blue '+', data points with negative label are marked as grey '-'.
- Weight is used as the normal vector of the hyperplane, which is seen as the decision boundary.

We can observe that the data for running perceptron algorithm are linear separable, therefore, we could obtain the result after finite iterations (538 iterations in my case), and an accuracy of 100%. However, the data used by pocket algorithm can't be separated by any hyperplane without any transformation. This is also the reason why we choose to use pocket algorithm to obtain the best result among all iterations.





Relationship between the number of misclassified points and the number of iterations of pocket algorithm is shown as below. This graph illustrates that the accuracy is not decreasing through iterations. Therefore, it is possible that the best result is obtained at an early stage which let the later iterations seem useless.



<u>Challenges I Face: show visualization result without blocking the execution of python script</u>
I tried to enable the script to be executed in terminal, as well as visualizing the data points and result. However, I found out that showing graph would block the execution of the script. In other words, the execution wouldn't continue until I closed the graph window.

I did some research online and found out that lots of people are facing the same problem. Finally, the problem was solved, and I also added a line of code at the very bottom which prevents all the output windows close after the execution ends. User only need to press enter whenever they want to exit the execution process.

<u>Software Familiarization: sklearn.linear_model.**Perceptron**</u>
Sklearn's perceptron library allows user to decide if the intercept should be estimated. Also, training data will be shuffled after each epoch if the parameter "shuffle" is set to True.
My implementation don't include these functions, instead I estimate the intercept no mater the data is already centered or not. Which might cost additional time when estimating intercept is not necessary.

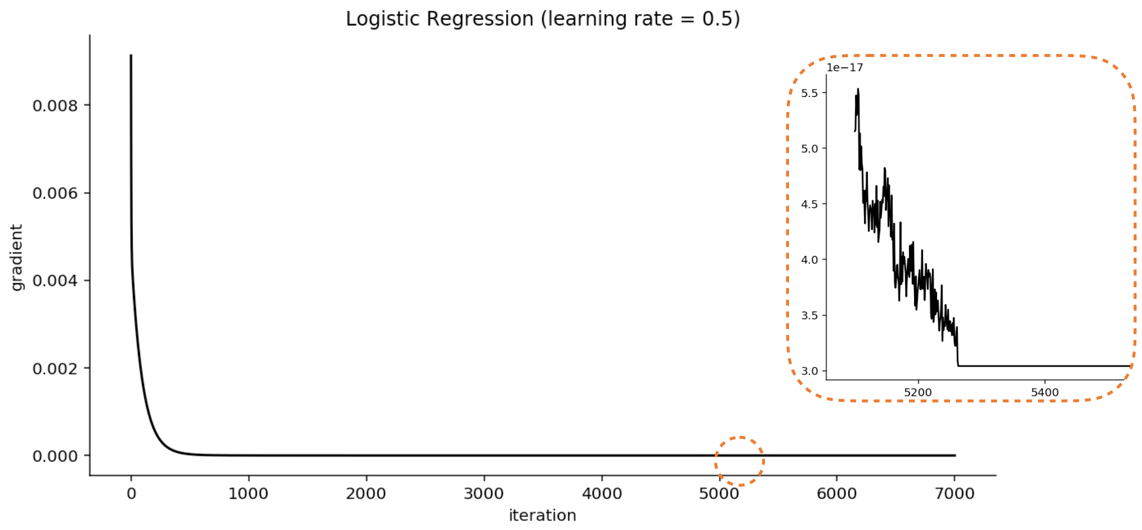
The weight output by my implementation is different from the weight output by sklearn library. Since the weight could be seen as the normal vector of the decision boundary, it is fine to have different length or slightly different direction, as long as it classified data points correctly. Furthermore, my final in-sample accuracy is 100%, which means that my coefficient also works well.

III. Implementation of Logistic Regression Algorithm

Implementation
I update the coefficients using the ideal of gradient descent, that is, setting a learning rate, and update the coefficients (weights) by subtracting the (learning rate * gradient). This learning rate is called fixed learning rate, which indicates the true learning rate of the algorithm changes in tandem with the gradient.

Result
Accuracy: 52.95%
Final weight: [-0.03150075 -0.17769619 0.11445235 0.07670126]*
* coefficients for x0, x1, x2, x3

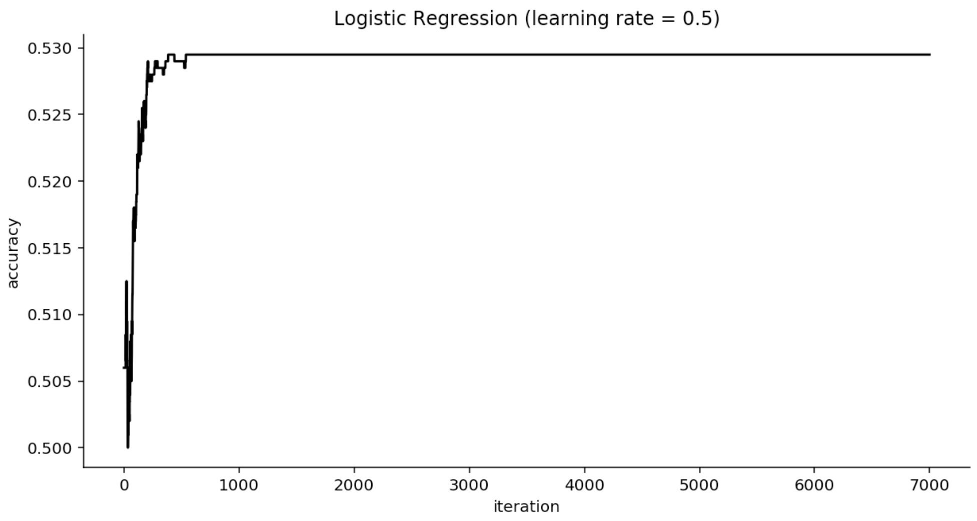
Challenges I Face:
1. Choice of learning rate
   As learning rate greatly influence when the weight converges, I tried many different learning rates in order to get an ideal result. If the learning rate is too small, it's hard for algorithm to converge in 7000 iterations, I finally end up with accuracy 52.95% by setting the learning rate to 0.5. And the gradient converges after around 5300 iterations in this case.



I also notice that by looking closer to the gradient, it's not exactly equals to zero. However, after doing some research online, I realize that it's common that one can't reach an exactly zero gradient, so in most of the cases, the stopping condition would be either the gradient is close enough to zero (a threshold had been set) or maximum iteration is reached.

2. Stopping condition
   After realizing that the stopping condition could be either the gradient is close enough to zero or the maximum iteration is reached, I start to think about which one works better for our data. Although stopping condition had been set in this assignment, that is, when iteration hits 7000. I would like to know if our algorithm really needs 7000 iterations.

   Gradient plot attached above show that the gradient converges around 5300 iterations, the accuracy, which is visualized as below, however, converges to 52.95% after 600 iterations.



   In conclusion, in our case, it's unnecessary to run 7000 iterations. Either setting a threshold indicating that the gradient is close enough to zero, or setting the maximum iteration to 1000 will do.

<u>Software Familiarization: sklearn.linear_model.**LogisticRegression**</u>
It has a special parameter for user to Indicate which algorithm to use in the optimization problem, some are faster for large dataset. The default algorithm is "Large-scale Bound-constrained Optimization", I have tried algorithm "saga" and "sag", both are related to gradient descent and my answer is close to both of the results. And the accuracy of the library output is also 52.95%.

IV. Implementation of Linear Regression Algorithm

<u>Result</u>
Final weight: [0.01523535 1.08546357 3.99068855]
* coefficients for x0, x1, x2

<u>Software Familiarization: sklearn.linear_model.**LinearRegression**</u>
It also allows user to decide if the intercept should be estimated, just like the perceptron library. The training result (weight) of sklearn is identical to my implementation.

V. Code Level Optimization

1. I write this code in an object-oriented manner. This code contains four classes, three are related to the algorithms and another one "Evaluation" is for calculating accuracy and misclassified points. It is necessary to write it as an individual class since all algorithms used in this assignment except linear regression need to calculate accuracy.

2. <u>Before:</u>
   I use zeros as my initial weight of PLA and Pocket algorithm.

After:
I use the weight output by linear regression as initial weight of my PLA and pocket algorithm. Both result in improvement. Reduction of number of iteration for perceptron algorithm and increase in accuracy for pocket algorithm is observed.

Performance comparison between using zeros and using linear regression output as initial weight are attached as below:

Perceptron Algorithm		
Initial Weight	[0 0 0 0]	[-0.08021714 1.86204263 -1.36400004 -1.03326183]
Accuracy	100%	100%
Number of Iteration	1237	**538**
Weight	[0. 93.99358666 -75.3947041 -56.31315876]	[-0.08021714 69.62813721 -55.77353248 -41.6615246]

Pocket Algorithm		
Initial Weight	[0 0 0 0]	[-0.01572516 -0.08876243 0.05716364 0.03830543]
Accuracy	52.75%	**53.30%**
Number of Iteration	7000	7000
Weight	[0. -3.5325214 3.57266164 -0.33081481]	[-0.01572516 -3.83583945 3.2871186 0.50330905]

VI. Applications

One interesting application of linear classification is music information retrieval, namely melody classification. Along with other information retrieval technique, linear classification is able to classify musical pieces represented by trees.

Logistic regression could be used in biomedical research for binary class prediction where the outcome can be for instance alive/dead, or therapeutic success/failure. Logistic regression model can predict the emergency room stratification of a novel breast tumor to select the appropriate treatment for breast cancer by analyzing both clinical and genomic data.

Linear regression is very powerful and easy to understand, it can be used to generate insights on consumer behavior, understanding business and factors influencing profitability. Linear regressions can be used in business to evaluate trends and make estimates or forecasts. For example, if a company's sales have increased steadily every month for the past few years, by conducting a linear analysis on the sales data with monthly sales, the company could forecast sales in future months.

Reference:
https://www-sciencedirect-com.libproxy2.usc.edu/science/article/pii/S092523121630532X
https://go-gale-com.libproxy1.usc.edu/ps/i.do?id=GALE%7CA557737528&v=2.1&u=usocal_main&it=r&p=AONE&sw=w