INF 552 HW4

Teammates:
He Chang 5670527576
Ziqiao Gao 2157371827
Fanlin Qin 5317973858

Part 1.

Perceptron Learning Algorithm

Data Structure：

We used 2 np.arrays to store point coordinates and their corresponding result values (1 or -1). When forming the point coordinates, we added X0 with value 1 to each point so it looks like [1, value1, value2, value3]. When generating initial random weights, we make W0 to be 0 as our threshold. The learning rate we picked is 0.5.

Optimization:
We applied 0.5 to the learning rate in case of overstepping.

Challenge:
Implementing the Perceptron Learning Algorithm in python is pretty straightforward and there is not much of a huge challenge.

Result:
Weights:[0. 56.80900531 -45.62959042 -34.0119904]
Passing rate: 100%

Pocket algorithm

Data Structure：

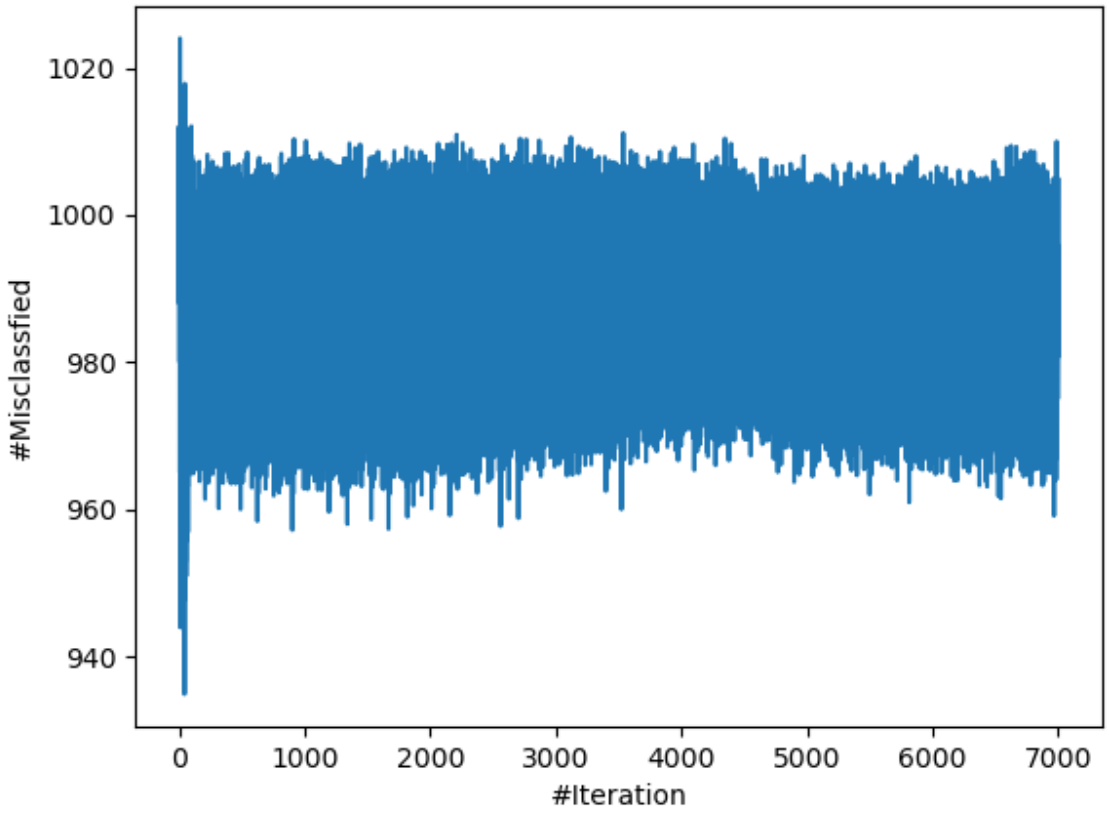
Similar to PLA, we used 2 np.arrays to store point coordinates and their corresponding result values (1 or -1). When forming the point coordinates, we added X0 with value 1 to each point so

it looks like [1, value1, value2, value3]. When generating initial random weights, we make W0 to be 0 as our threshold. The learning rate we picked is 0.5.

Optimization:
We applied 0.5 to the learning rate in case of overstepping. The range of floating point values for initial weights is set to be between -1 and 1.

Challenge :
The cost to test out some of our ideas by running the program is relatively large. The 7000 run times will averagely take 15 seconds to finish.

Result:
Best weights [0. -2.03241752 1.91138238 -0.0898345]
Accuracy: 0.5325



Logistic Regression

Data Structure：

We used 2 np.arrays to store point coordinates and their corresponding result values (1 or -1).When forming the point coordinates, we added X0 with value 1 to each point so it looks like [1, value1, value2, value3]. When generating initial random weights, we make W0 to be 0 as our threshold.

Optimization:
We separated the process of calculating Ein from the main Logistic Regression loop to make the implementation clearer.
In the initial implementation, when calculating Ein we did it for each point and summed the results up. Later we found it very slow, so we treated the data points as a whole matrix and calculated Ein just for once. Our optimization largely facilitates efficiency.

Challenge：
We spent some time trying to find a way to make the run time faster. The initial cost of testing out some of our ideas by running the program is even larger. The 7000 run times would averagely take 1 minute to finish before. On the other hand, figuring out the math and making sure the matrix calculation works correctly takes most of the time.

Result:
Weights: [-0.03040257 -0.17712695 0.11503189 0.07727183]
Accuracy:53.05%

Linear Regression

Data Structure：

We used 2 np.arrays to store point coordinates and their corresponding result values (1 or -1). When forming the point coordinates, we added X0 with value 1 to each point so it looks like [1, value1, value2, value3]. When generating initial random weights, we make W0 to be 0 as our threshold.

Optimization:
The procedure is already efficient and we did not provide any optimization.

Challenge :
The implementation of Linear Regression algorithm is very simple and straightforward and we did not have any major challenge.

Result:
[0.01523535 1.08546357 3.99068855]

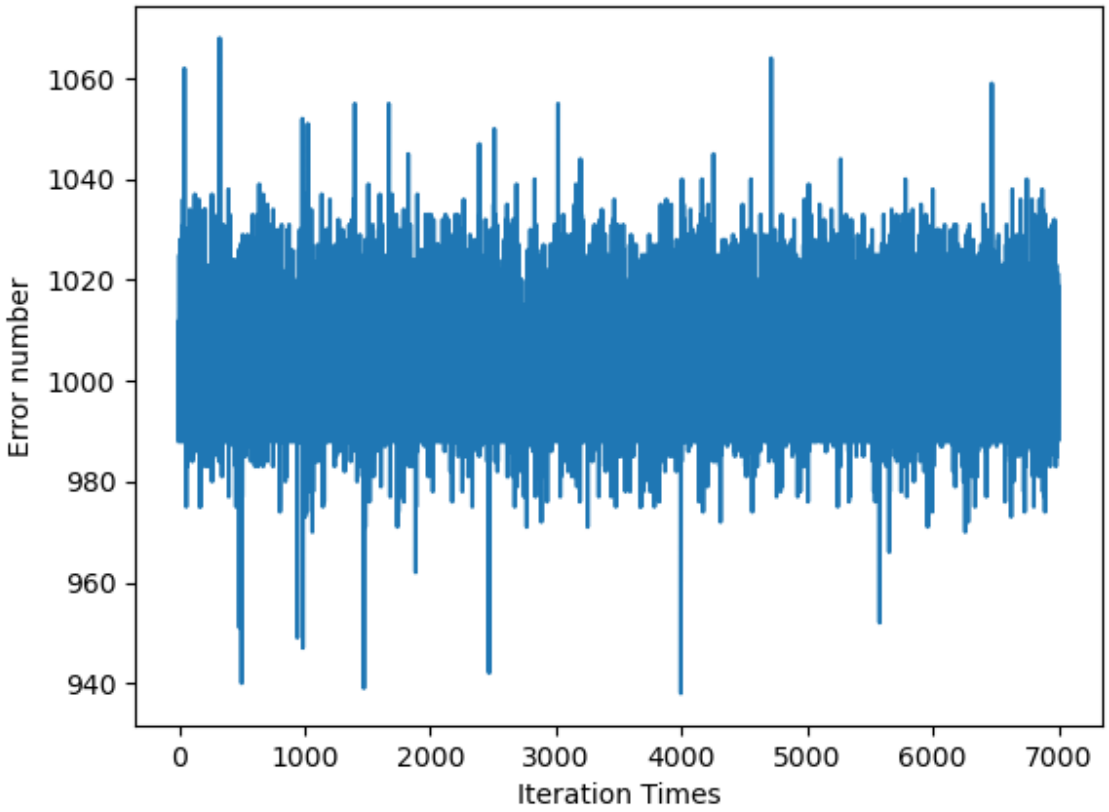
Part 2.

Perceptron Learning algorithm

Library used: from sklearn.linear_model import Perceptron
Result:
Accuracy = 0.98
Weights = [[0. 24.35173999 -18.56164074 -14.10260498]]
Summary: The PLA function from sklearn is very simple to use. We just need to clean the data that is dirty at first. Then we can see that accuracy is a little bit different from ours. The accuracy isn't 100%. The research shows that the accuracy predicts the percentage of correct results. There are some wrong predictions. But our code passes the wrong predictions. So the accuracy isn't the same.

Pocket algorithm

Library used: from sklearn.linear_model import Perceptron
Result:
Accuracy = 0.531
Weights = [[0. 1.14929708 -0.2587861 -0.95413407]]



Summary: The Pocket PLA uses the same sklearn function as PLA. But it is more complicated than PLA. And it costs about 15 seconds. The running time of it is a little bit long. Though the best weights are a little different from ours, the lowest and the highest of misclassified is very similar.

Logistic Regression

Library used: from sklearn.linear_model import LogisticRegression
Result:
Accuracy = 0.5295
Weights = [[-0.01487118 -0.17357511 0.1116996   0.07491538]]
Summary: The Logistic Regression function from sklearn is very simple to use. And the running time of this function is very fast. It just costs about 5 seconds. It is more and more faster than our code. At first, our weight is very different from this result. Then we found that we did the wrong calculation of Ein. After modification, the weights and accuracy are very similar. From research, we think the little difference of weights is that SKlearn implemented it using regulation.

Linear Regression

Library used: from sklearn.linear_model import LinearRegression
Result:
Weights = [1.08546357 3.99068855]
Summary: The Linear Regression function from sklearn is very simple to use. And the result is the same with us. The main difference is that it has normalization. But the default value of it is false, so we could get the same result with it.

Part 3.

1.Application of linear classification

Linear classification is one of the most commonly used classifiers out there. In the practical applications, linear classifications can be used in text categorization. This algorithm shares similarity by finding the complementary words that roughly separates a class of documents.

2. Application of linear regression

Linear regression is an algorithm that can be used to predict the value of a quantitative variable. Therefore, in practical applications, linear regression can be used in business activities. Linear regression can predict the sale of products in the future based on past buying behaviour. And it can also predict the economic growth of a country or state. In the financial sector, linear regression is very useful.

3. Application of logistic regression

Logistic regression is a machine algorithm for solving binary problems, used to estimate the likelihood of something. In the practical applications, logistic regression can be used in credit scoring. Credit scoring has become a common way in modern banking because of the large number of applications daily and the increased regulatory requirements for banks. Logistic regression focuses on the credit risk context, but not restricted to it.

[1]Tong Zhang, Frank, “Text Categorization Based on Regularized Linear Classification Methods”Information Retrieval Volume 4, 5-31(2001)
[2] Youtube(2018, May 6). Linear Regression Analysis on Youtube [Video file]
[3] Bolton, Christine, “Logistic regression and its application in credit scoring”, Dissertation (MSc)--University of Pretoria, 2010.

Individual contribution:

Ziqiao Gao: Optimize, implement and debug the Perceptron, Pocket, Logistic Regression and Linear Regression algorithm for part 1

He Chang: Optimize, implement and debug the Perceptron, Pocket, Logistic Regression and Linear Regression algorithm for part 1

Fanlin Qin: Research online and use the open source python library to implement Perceptron, Pocket, Logistic Regression and Linear Regression algorithm applications and complete part 2&3.