HW4-Perceptron, Logistic regression and Linear Regression

Che-Pai Kung 5999-9612-95

Part 1: Implementation

Perceptron Learning algorithm /Pocket algorithm

Data structure

In both algorithms, I used numpy array structure to store data and weights for matrix calculations. In addition, an additional column of ones is added into the data and its corresponding weight is seen as the intercept of the linear function derived.

To track the performance of each iteration, a list is created to store the number of violated points after each iteration when designing the Pocket algorithm.

Code-level optimizations

When implementing Perceptron Learning algorithm, it only needs to find one data point that is predicted incorrectly. I compared the label and the predicted label row by row to update the weights once misclassified data was found. Then, the algorithm proceeds to the next iteration. As it is necessary to count the number of misclassified data to select the best result after running the program, instead of calculating the results row by row, I utilized the matrix computation to increase program efficiency with a concise code and attained all predicted labels without using loop.

Moreover, since two algorithms are similar, I combined them into one program. To avoid redundant iterations, the program is stopped once there is no misclassified data. This is achieved by adding a parameter, violated_contraint, that is set to be “True” if violated points are found.

The functions were built independently at first. It was noticed that each function took lots of parameters and not organized. Hence, to increase the readability of the program, the Pocket algorithm program was reorganized into object-oriented style by building all the functions, including print result of each iteration and plot data points, the plane predicted, in perceptron_pocket class.

Challenge

The biggest challenge was implementing the matrix computation. As mentioned above, I used matrix computation function provided by numpy library to get all the predicted labels in one line. At first, it was confusing to decide which matrix to place at the front. Afterwards, I carefully identified dimensions of each matrix and figured out the right answer to it.

Results (Note that the outcomes might differ after each execution, but the accuracy rates are within that range.)

Perceptron Learning algorithm

Weights: [0, 16.1889, -12.9881, -9.69896] (the first number is the intercept)

Accuracy: 100%

The result below shows that the points are well separated by the plane. The X, Y, Z axes represent the first three column of data provides, and the points are colored based on their label.

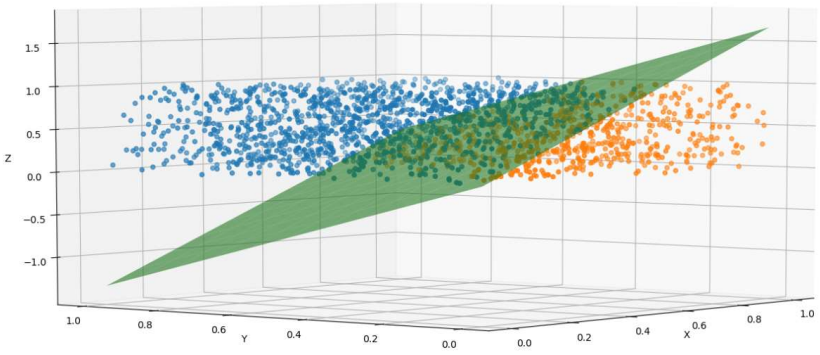

Figure 1

Pocket algorithm

Weights: [0.1, -4.58406004, 3.65972124, 0.83721914] (the first number is the intercept)
Accuracy: 52.9%
The results are presented as below. Figure 2 shows the results after each iteration, and there is no correlation between optimal solution and iterations. Figure 3 is the best solution within 7000 iterations and the points are poor separated by the plane. As the points are scattered evenly, it is difficult to cluster them by only a plane. The X, Y, Z axes represent the first three column of data provides, and the points are colored based on their label.

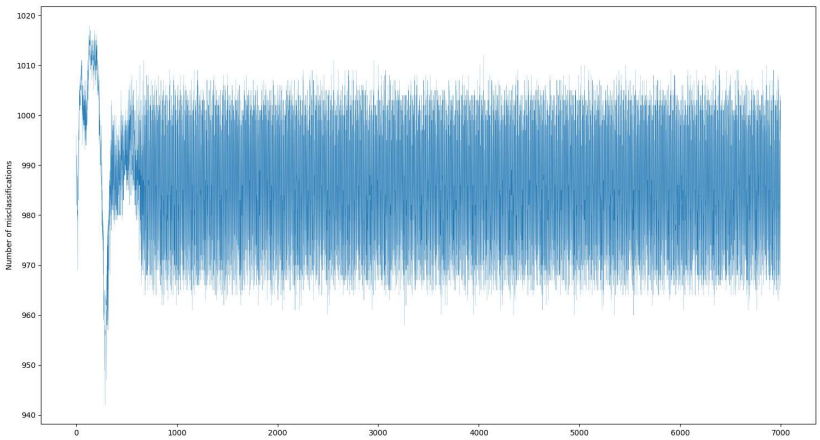

Figure 2

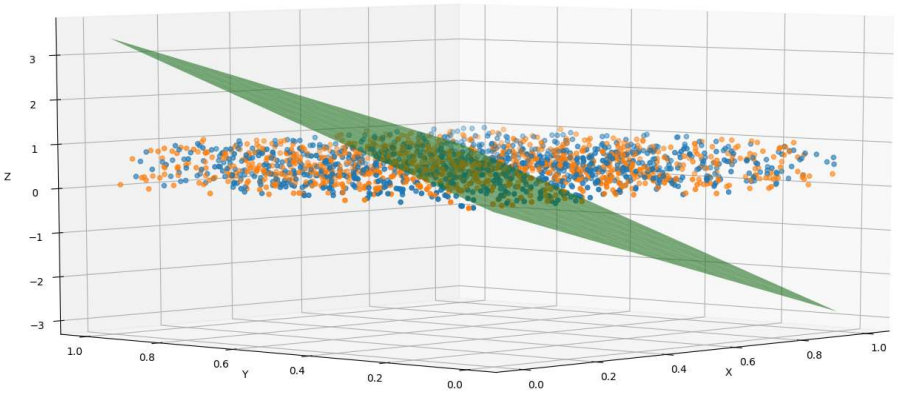

Figure 3

Logistic Regression

Data structure

The data structure used for logistic regression is the same as the pocket algorithm. I used array to store data and weights; list to save the loss of each iteration.

Code-level optimizations

The same optimizations as above were also implemented on logistic regression. Matrix computations are used to make the codes neater and all the functions were built into logistic class after writing each function separately.

Challenge

Other than the matrix computation mentioned above, I was also struggled at deriving the gradient of the loss function. Although Professor Satish has provided us with the formula, I thought it is better to derive it myself. After understanding how the formula is derived, the next challenge was how to turn it into matrix computation without looping by each data. As described above, I wrote down the dimension of each matrix as well as the desired results, and then figured out how to manipulate the computations between them.

Results (Note that the outcomes might differ after each execution, but the accuracy rates are within that range.)

Weights: [-0.03728945, -0.17402848, 0.11802634, 0.08055266] (the first number is the intercept)

Accuracy: 52.85%

The graphs below show how loss and accuracy changed after each iteration. It is indicated that generally there is negative correlation between loss and accuracy but not always. I think that is due to the loss being calculated based on the probability while accuracy is either true or false. This leads to the inconsistent outcomes.

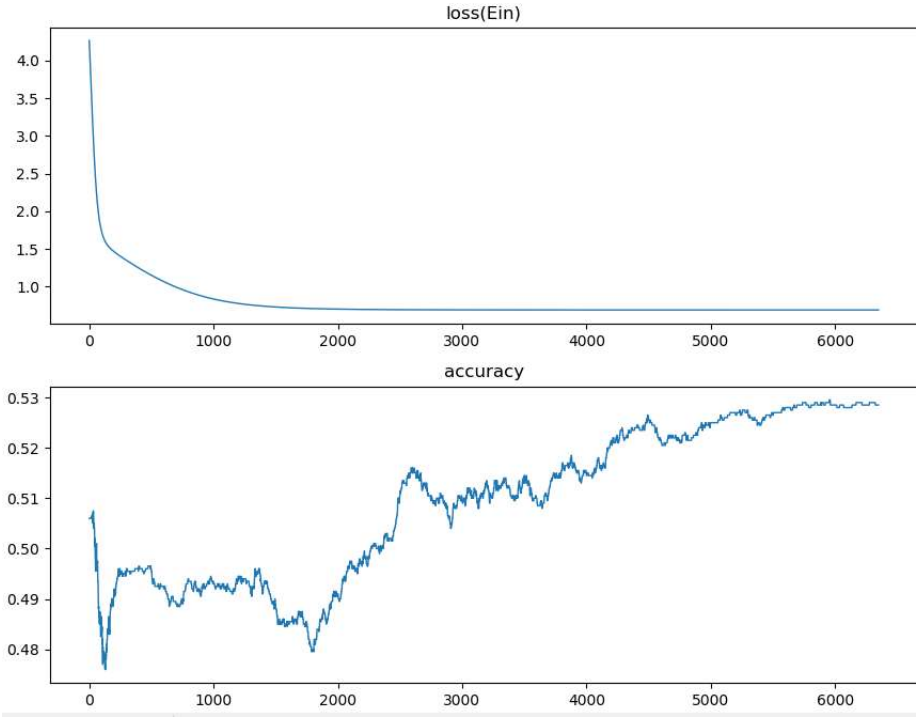

Figure 4

Linear Regression

Data structure

The data structure used for linear regression is the same as pocket algorithm. I used array to store data and weights; list to save the loss of each iteration.

Code-level optimizations

The same optimizations as above were also implemented on linear regression.

Challenge

After implementing the above algorithms, it was easier to do linear regression as there is a numerical solution for it, which meant there is no need for doing gradient descent. Nonetheless, I tried to implement gradient descent for it as well to check if they produce the similar outcomes. At first, the result was strange, as the loss did not decrease after each iteration but keep increasing. Hence, the result was far from the close form solution. I checked the codes and searched for information to make sure that my gradient function is correct. After a few days, I finally found that I forgot to add "self" before the matrix, leading to a wrong computation of the gradient. The derived answer is close to the numerical answer after revising the code.

Results

Weights (numerical solution): [0.01523535, 1.08546357, 3.99068855] (the first number is the intercept)

Weights (generated by gradient descent method): [0.02084511 1.08031071 3.98516824]

Figure 5 illustrates that the plane predicts the data well. The X, Y axes represent the independent X and Y variables and Z represents the dependent Z variable.

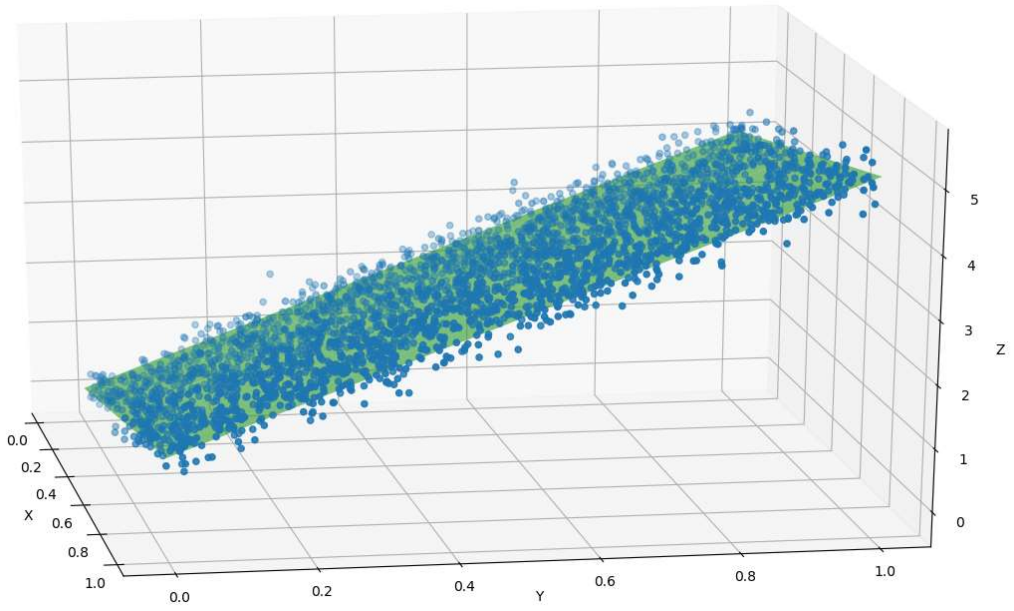

Figure 5

Part 2: Software Familiarization

Linear Classification

The library I used for Linear regression is sklearn.linear_model. SGDClassifier. When parameters are set as the following: "loss="perceptron", eta0=1, learning_rate="constant", penalty=None", its result for predicting column 4 is [0, 16.96657342, -12.7546922, -10.10797002] (the first number is the intercept) and the accuracy rate is 99.1%; the result for predicting column 5 is [-1, -0.69780427, 0.06876568, 0.16214574] and the accuracy rate is 50.6%. Note that the outcomes might differ after each execution, but the accuracy rates are within that range. It was confusing to me that why the accuracy rate is not 100% for the first dataset, and I realized it was due to the parameter "tol" being set as 0.001. If I set it as None, then the result is 100%.

The package also provides other loss functions and penalty choices. It might be helpful to utilize other loss functions to deal with different conditions of data and penalty functions to avoid overfitting if there is a test data.

Logistic Regression

The library I used for Logistic regression is sklearn.linear_model.LogisticRegression. Its weights are [-0.03071168, -0.17393989, 0.11141144, 0.07456303] (the first number is the intercept) and accuracy rate is 52.9%, which is similar to my result. Note that the outcomes might differ after each execution, but the accuracy rates are within that range. Nonetheless, I noticed that the package uses only two iterations to get the answer while my implementation uses 6121 iterations. After reading the document of the package, I found out that it uses several solvers ('liblinear' by default) to optimize the program. The program might converge faster if it was using more advanced gradient descent algorithm. In addition, parameter "penalty" is also introduced in the library. After adding penalty to the loss function, I believe it is likely to avoid overfitting in the training dataset and have higher accuracy in the testing dataset.

Linear Regression

The library I used for Linear regression is sklearn.linear_model.LinearRegression. Its weights are [0.015235348288889838, 1.08546357, 3.99068855] (the first number is the intercept), which is the same as my output. According to the information I searched, the library computes the closed form solution (using ordinary least squares) instead of using gradient descent, which means the algorithm does not do many optimizations. However, I noticed that there are other linear models with penalty, such as Lasso and Ridge. They utilize gradient descent to find the optimal solutions and avoids overfitting by setting parameter "alpha".

Part 3: Applications

Linear Classification

A linear SVM (Supported Vector Machine) is a type of linear classification. SVM applies kernel trick on data, separating data linearly in higher dimensional space. Therefore, it is a powerful classifier and is used in many areas. For instance, using SVM to classify the images has proved to gain higher accuracy than traditional techniques. Also, SVM can distinguish parts of an image as a face or non-face.

Reference:https://data-flair.training/blogs/applications-of-svm/

Logistic Regression

Logistic regression is a form of supervised learning, specifically, learning to classify the data based on the variables. Customer loyalty is imperative for a company to keep operating in the markets. It is observed that service quality, price, brand and other factors affect customers' behavior. If they are satisfied with what they get, then it is likely that they are loyal customer, otherwise, they might switch to other brands. Logistic Regression can be used to predict whether the customers are loyal to the product, and the company may utilize the results to keep potential customers or provide rewards to those who are likely to become loyal customers, increasing their willingness to shop.

Linear Regression

Linear regression shows the relationship between independent variables and dependent variables, and therefore it is widely used in many fields. For instance, economists use it to predict the economic growth based on domestic expenditures, employment rate, interest rate and more variables. Additionally, linear regression can be used to predict the revenue of a company according to the previous performance.