DSCI 552: Machine Learning for Data Science

Programming Assignment 4:
Perceptron Learning algorithm & Pocket algorithm & Logistic Regression & Linear Regression

Team members: Li An, Shengnan Ke

Due Date: 03/10/2022

Contribution:

We followed the same procedure as last week. We reviewed the lecture together, checked some online resources, and had some discussions to help both of us get a better understanding of the models we need to implement. Then each of us worked on the implementation of two algorithms. Our individual contributions are listed below. Then we write the report together and help each other to check on the algorithms that each of us implemented.

Li An: Logistic Regression & Linear Regression

Shengnan Ke: Perceptron Learning algorithm & Pocket algorithm

Part 1: Implementation

■ Perceptron Learning algorithm

Preparation work

- Import necessary libraries
- Load data - edit data to the format we need
  - A dataset of points, with labels.
- Set up variables
  - A number of epochs, n.
  - A learning rate - I set it as small number 0.0001.

Run the algorithm

➢ Pick random weights

➢ If, all constraints are satisfied -> Done

➢ Else, pick any violated constraint and use its geometry to guide update on weight

- Suppose that the i th constraint is violated
  - Case 1:
    - $y_i = +1$
    - $w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots + w_d x_d^{(i)} < 0$
    - Update weight:
      - Weight = weight + learning rate * x
  - Case 2:
    - $y_i = -1$
    - $w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots + w_d x_d^{(i)} > 0$
    - Update weight:
      - Weight = weight - learning rate * x
  - Repeat

➢ PLA converges when all the data points are classified correctly.

Result - weights and accuracy after the final iteration

```
Final Iteration Number: 355 Misclassified Data Points:  0
The accuracy is 100% as default.
Weights: [[-5.39939081e-05  2.69416907e-01 -2.16384769e-01 -1.61246575e-01]]
```

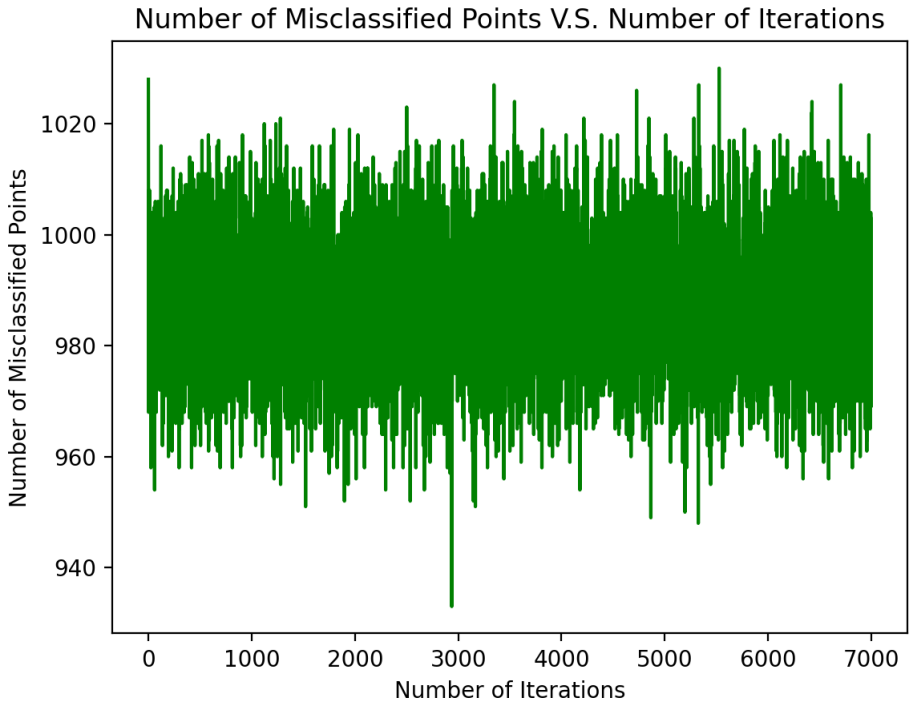
■ Pocket algorithm

- Very similar to Perceptron Learning Algorithm, but PLA can only work with the promise that the data is linearly separable. However, when data is not linearly separable and you want to separate data with minimal error, you can use a pocket algorithm instead.
- Everything is the same except we set up a max iteration for this algorithm. Since this dataset could not perfectly be linearly separated. It runs 7000 iterations and selects the iteration with the least amount of misclassified data points (which means the minimal error).
- Since we set up the max iteration as 7000, this is also the termination rule.

Result

```
[Running] python -u "/Users/kknanxx/.vscode/extensions/DSCI552-A4/Pocket_algorithm.py"
Minimum Misclassifications:  933
Final weights:  [[ 6.68432533e-04  9.40634828e-05  1.52808599e-03 -3.02714727e-04]]
Accuracy:  53.349999999999994 %
```

After multiple run-throughs of the code, we realized the accuracy is usually around 53%.



■ Logistic Regression

Preparation work

- Set up learning rate and maximum iteration
- Load dataset
  - Data structure: numpy array
  - Separate the original dataset to X and y
  - Add x_0 column to the X
- Initialize weights

Run the algorithm

The main part of the implementation includes two parts: compute the optimal weights, and using the weights to compute predicted_y. We use pseudo-code below to illustrate the process of the implementation.

```
set iterations to 0
while iterations smaller than max_iteration:
      updata w by w ← w − η▽_w E_in(w)
```

```
set N to the number of data points
for i=1, 2, ..., N:
    Predicted_y = θ(w^T x^(i))
```

Result

I ran the algorithm several times with different learning rate values (0.1, 0.05, 0.01). And it turned out that the algorithm gained best results when the learning rate was set to 0.1. The weights and accuracy aftering running the algorithm are shown below.

- Weights:
  [-0.03162569, -0.17761756, 0.11452977, 0.07678462]
- Accuracy: 52.949999999999996%

■ Linear Regression

Preparation work

- Load dataset
  - Data structure: numpy array
  - Separate the original dataset to X and y
  - Add x_0 column to the X

Run the algorithm

The main part of the implementation also includes two parts: compute the optimal weights, and using the weights to compute predicted_y.

The optimal weights are computed following this formula:

$$w^* = (DD^T)^{-1}D\overline{y}$$

Predicted y equals to $w^T x^{(i)}$.

Result

Weights:
[0.01523535 1.08546357 3.99068855]

Part 2: Software Familiarization

Sklearn contains almost all of the library functions for machine learning algorithms.

For the perceptron learning algorithm, Linear regression, and Logistic regression, the library function that can be used is listed below:

- from sklearn.linear_model import Perceptron
- from sklearn import linear_model
- from sklearn.linear_model import LogisticRegression

For the pocket algorithm, there is no library model named pocket. Since it's based on the perceptron algorithm, the pocket algorithm can also be implemented merely by extending the Perceptron Learning Algorithm.

Part 3: Applications

■ Perceptron & Pocket

These two algorithms work similarly, they can help us by using datasets trained to learn, recognize patterns, and make predictions in a humanoid fashion as well as solve problems in every business sector.

■ Logistic Regression

Logistic regression can be used to predict classification.
For example, here is a list of people and their income. For people who earn less than 10k, mark as 1; people who earn more than 10k, mark as 0.

■ Linear Regression,

Linear regression is used to quantify the relationship between one or more predictor variables and a response variable.

For example, businesses often use linear regression to understand the relationship between advertising spending and revenue. Medical researchers often use linear regression to understand the relationship between drug dosage and the blood pressure of patients.