Programming Assignment 4: Perceptron, Pocket, Logistic Regression and Linear Regression Algorithms

Srihari Akash Chinam
USC ID - 2953497706
Net ID - chinam

Sayan Nanda
USC ID - 2681592859
Net ID - snanda

Implementation

- Language used - Python version 2
- Libraries used - Numpy (arrays and matrices), matplotlib.pyplot, math, SciKit-Learn, pylab

Perceptron Learning Algorithm

The dataset provided for demonstrating the Perceptron Learning Algorithm (PLA) was the first 4 columns of the document “classification.txt”. The dataset was visualized to see the problem clearly. The visualization was done first in 2D by applying PCA on the data and then in 3D space, as shown in Figure 1 and Figure 2 respectively.

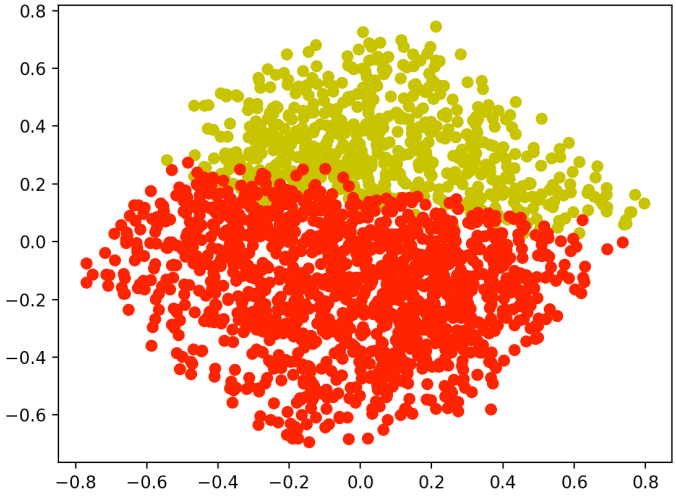

Figure 1 2D Visualization of PLA data

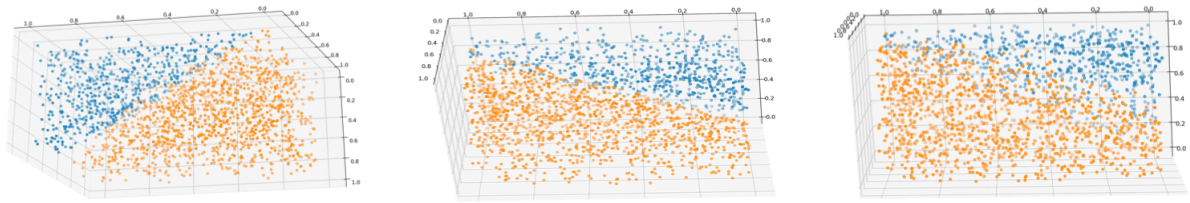

Figure 2 3D Visualization of PLA data

As seen in both the figures, the data is not completely linearly separable. Therefore, the learning algorithm was run till the error was less than 0.5%. All the weights were initialized to zeros with a learning rate of 0.1 and were modified until the error reached less than 0.5%, ending with a final correctness of **1993 of 2000** for the given points. The weights obtained are shown in Table 1.

Table 1 Weights obtained for PLA

	w1	w2	w3	b
Weights	15.72560679	-12.47367845	-9.35763758	-0.03

After this, the same data was passed to the Perceptron implementation of SciKit-Learn and the weights were recorded, as shown in Table 2. This model correctly classified **1986 of the 2000** given points.

Table 2 Weights obtained for SciKit-Learn's PLA

	w1	w2	w3	b
Weights	13.59165618	-10.93149856	-7.87378941	0.0

Pocket Algorithm

The dataset for this algorithm was the first 3 columns as data and the last column as the labels for classification. Upon visualization, it was seen that this problem was not linearly separable at all. The data was converted to 2D using PCA and visualized as shown in Figure 3.

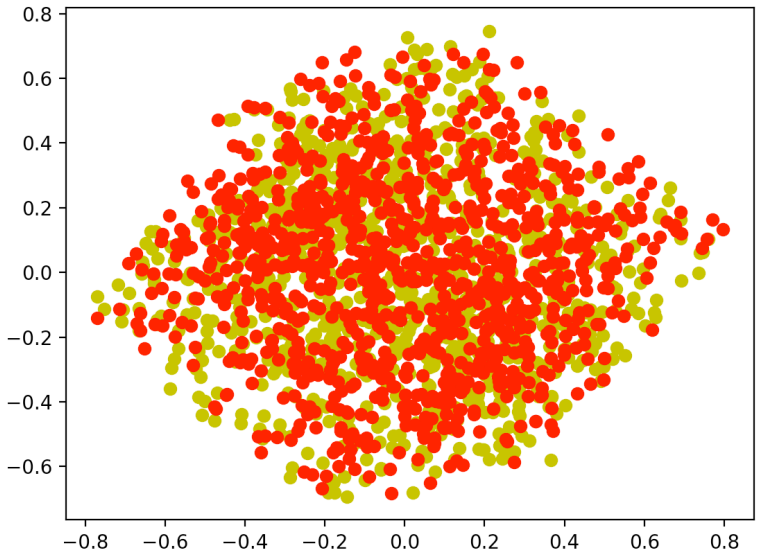

Figure 3 2D Visualization of data for Pocket Algorithm

The Pocket algorithm was run for 7000 iterations on this data with a learning rate of 0.1. The best result obtained was with **1013 correct classifications**. The misclassifications per iteration are shown as a graph in Figure 4. The weights obtained for this case are shown in Table 3.

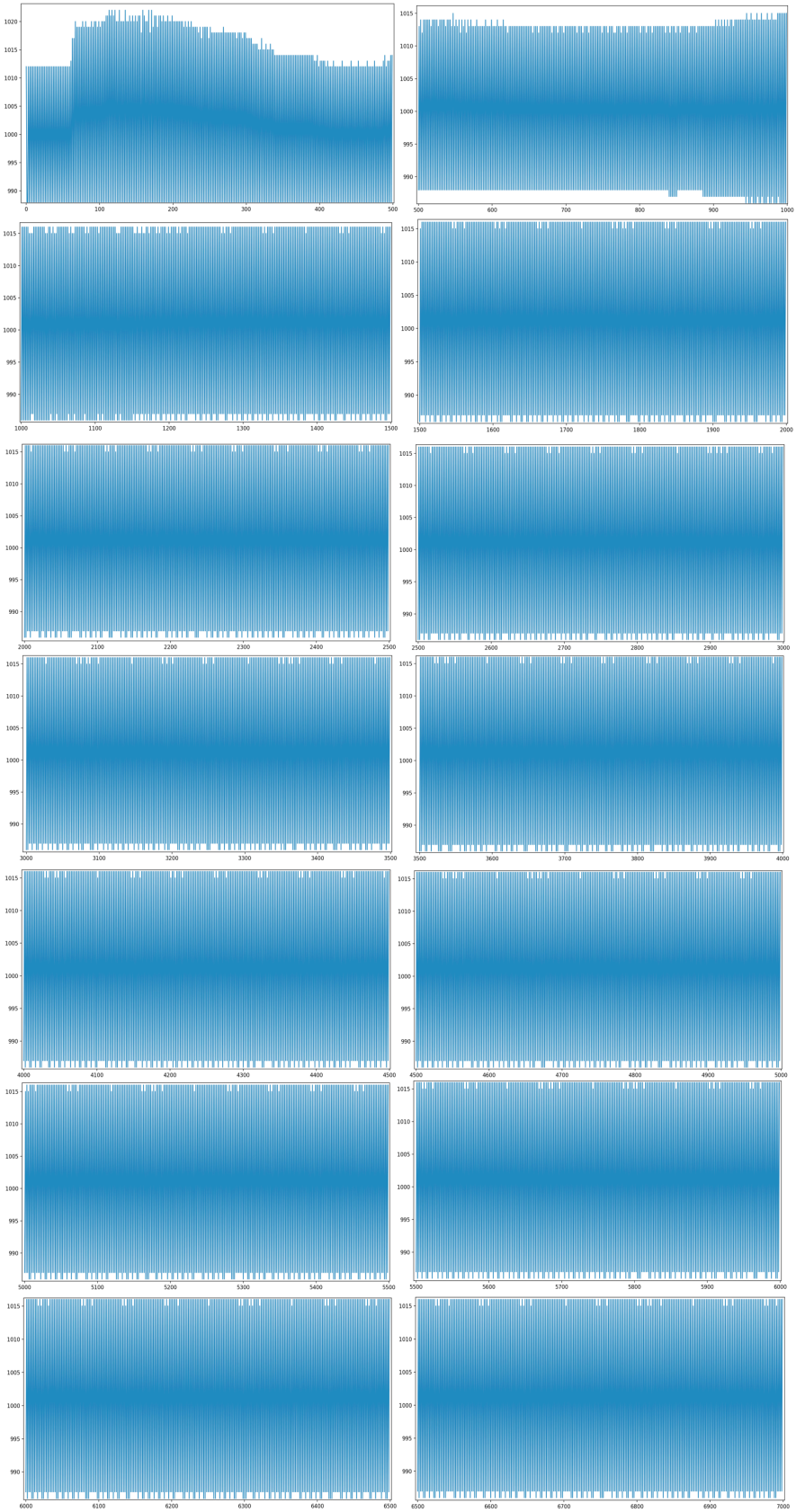

Figure 4 Misclassifications per iteration broken down in 14 parts for 7000 iterations

Table 3 Best case weights obtained for Pocket Algorithm

	w1	w2	w3	b
Weights	-12.8331663	4.63806244	3.59011913	10.33

Logistic Regression

The data used for Pocket algorithm was used again for Logistic Regression (shown in Figure 3). The logistic regression algorithm implemented used the sigmoid function as the non-linear activation function. This ensured that all outputs are in between 0 and 1, giving the probability of a point belonging to a class. The number of points correctly classified after 7000 iterations are **1059 of 2000**. The weights for this case are shown in Table 4.

Table 4 Weights obtained for Logistic Regression

	w1	w2	w3	b
Weights	-0.17768283	0.11446576	0.07671557	-0.0315221931581

The same data was passed to the in-built Logistic Regression Model of SciKit-Learn. The weights obtained are very similar to the weights obtained by this assignment's implementation, as observed from comparing Table 4 and Table 5. With the weights obtained by SciKit-Learn, **1058 of 2000** samples were correctly classified.

Table 5 Weights obtained for Logistic Regression of SciKit-Learn

	w1	w2	w3	b
Weights	-0.17393989	0.11141144	0.07456303	-0.03071168

Linear Regression

The data for linear regression was taken from the file 'linear-regression.txt' with the first two columns being the independent data points and the last column being the dependent value. When passed to the linear regression algorithm and trained for 7000 iterations, the following weights were obtained, shown in Table 6. The error (cost) for this implementation was **1.37667655054e-14**.

Table 6 Weights obtained for Linear Regression

	w1	w2	b
Weights	1.08546357	3.99068855	0.0152353482889

When the same data was passed to SciKit-Learn's implementation of Linear Regression, the error obtained was **1.60862434484e-11**. The weights for this implementation are shown in Table 7.

Table 7 Weights obtained for SciKit-Learn's Linear Regression

	w1	w2	b
Weights	1.08546357	3.99068855	0.0152353482889

Even though the weights seem exactly the same, the errors are different for both of them. This is because the last few decimals of both implementations. The differences between the weights are shown in Table 8.

Table 8 Difference in weights for both implementations

	w1	w2	b
Difference in weights	-1.33226763e-14	-2.70894418e-14	1.48197426553e-14

Software Familiarization

The python library, SciKit learn offers implementations for the perceptron learning algorithm, linear regression and logistic regression.

In the library implementation for perceptron learning, there are many parameters which allow more flexibility for the execution. There is a tolerance parameter, which may be set. This is the stopping criterion. There is also an option to shuffle the training data and run the algorithm multiple times. At the end of which the best result is chosen. The weights may be randomly assigned at the start. This is in contrast to the approach in our algorithm wherein we start with the weights 0. There is also the option to assign weights to the classes. This is applicable in other problems where some classes are said to be more significant.

Possible changes that we could make to our algorithm, include assigning random weights at the start of the algorithm. This may be helpful in the pocket problem, we may run the program multiple times and return the case where the number of incorrectly classified points is minimum.

The library implementation for linear regression allows the user to calculate the intercept for the model. This is applicable when the data is not centered. For the given dataset, the data is already centered hence implementing this is not useful. Another parameter is the possibility of normalizing the data. This is used when the data is already centered however like the previous parameter, this will not be useful because the dataset given to us is already normalized.

The SciKit learn implementation of logistic regression include more functionality. It allows parameters for penalty and formulation. The tolerance can be specified as well as the inverse regularization strength. Like the implementation of linear regression, there are option to determine the intercept of the model. There are options to add weights to the classes. Depending on the problem at hand, a variety of solvers can be chosen for optimization. This depends on the dataset size, whether it's a multiclass problem or not and the penalty.

Possible changes that can be made to our algorithm are, changing the inverse regularization strength. We can run multiple iterations to find the best result. The tolerance value can also be experimented with.

Applications

Linear regressions find a lot of use in businesses. They are useful for predicting sales, pricing, performance, risk. It is useful for identifying trends and making forecasts. It finds many applications for identifying the trend line. This identifies whether a data set has increased or decreased over a period of time. This gives it applications in statistics, finance and economics. For many years, linear regressions have been used in astronomy [1].

Logistic regressions have been used in regional hazard management. Multivariate statistical analysis in the form of logistic regression was used to produce a landslide susceptibility map in the Kakuda-Yahiko Mountains of Central Japan. These were combined with bivariate statistical analyses to explain the model obtained at the end of the study [2]. Optimally bounded score functions were studied to estimate regression parameters in generalized linear models. To conclude the study the bounded-influence estimator was compared with maximum likelihood [3].

Linear classifiers have found much of their use in advanced machine learning algorithms. Neural networks and support vector machines use perceptron learning and linear classifiers as a building block. It has recently been used in genetic programming as well [4]. One interesting application of linear classifiers is its use in analysis of pork storage time. Due to a large number of factors (temperature, gas composition, pH value, etc.), such an analysis proves very interesting. To analyze the spoilage of the meat, Fourier transforms were used near infrared spectroscopy [5].

Individual Contributions

Sayan Nanda

- Implementation of models
    - Perceptron Learning Algorithm
    - Pocket Learning Algorithm

- Documentation
    - Software Familiarization
    - Applications

Srihari Akash Chinam

- Implementation of models
    - Perceptron Learning Algorithm
    - Pocket Learning Algorithm
    - Logistic Regression
    - Linear Regression

- Documentation
    - Implementation of PLA, Pocket algorithm, Logistic and Linear Regression

References

[1] Linear Regression for Astronomical Data with Measurement Errors and Intrinsic Scatter, https://arxiv.org/abs/astro-ph/9605002

[2] The application of GIS-based logistic regression for landslide susceptibility mapping in the Kakuda-Yahiko Mountains, Central Japan, http://www.sciencedirect.com/science/article/pii/S0169555X04001631

[3] Optimally hounded score functions for generalized linear models with applications to logistic regression, https://academic.oup.com/biomet/article/73/2/413/338969/Optimally-hounded-score-functions-for-generalized

[4] Application of Genetic Programming to Induction of Linear Classification Trees, https://link.springer.com/chapter/10.1007/978-3-540-46239-2_18

[5] Application of linear/non-linear classification algorithms in discrimination of pork storage time using Fourier transform near infrared (FT-NIR) spectroscopy, http://www.sciencedirect.com/science/article/pii/S0023643811001630