STUDY REPORT FOR HOMEWORK-2

SIVA KRISHNA SIRIGINEEDI
800887277

ALGORITHMS DESCRIPTION

GENERATE DATA

ASSUMPTION

Input matrix X is created with dimensions (N*2) and all elements exist between [-1, 1], numpy.random.uniform is used for creating matrix X but here is a catch

numpy.random.uniform

numpy.random.**uniform**(*low=0.0, high=1.0, size=None*)

Draw samples from a uniform distribution.

Samples are uniformly distributed over the half-open interval `[low, high)` (includes low, but excludes high). In other words, any value within the given interval is equally likely to be drawn by uniform.

Parameters:

low *: float, optional*
Lower boundary of the output interval. All values generated will be greater than or equal to low. The default value is 0.

high *: float*
Upper boundary of the output interval. All values generated will be less than high. The default value is 1.0.

size *: int or tuple of ints, optional*
Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.

Returns:

out *: ndarray*
Drawn samples, with shape *size*.

For covering the 1 inclusively I took upper boundary as 1.00000001

```
>>> import numpy as np
>>> np.matrix(np.random.uniform(1, 1.00000001, size=(10, 1, 2)))
matrix([[ 1.00000001,  1.        ],
        [ 1.        ,  1.        ],
        [ 1.00000001,  1.00000001],
        [ 1.        ,  1.        ],
        [ 1.00000001,  1.        ],
        [ 1.        ,  1.        ],
        [ 1.00000001,  1.00000001],
        [ 1.00000001,  1.        ],
        [ 1.        ,  1.        ],
        [ 1.00000001,  1.        ]])
>>>
```

The above screen shot proves that there is no fraction between 1 and 1.00000001. So for [-1,1] we can use [-1,1.00000001) since numpy.random.uniform generates random number exclusive of maximum bound. According to the documentation it should not include the high bound, but it is including, so I am assuming it is close bracket.

Exception: in the below example it is including the upper boundary also.so I am confused whether the upper bound is inclusive or exclusive.

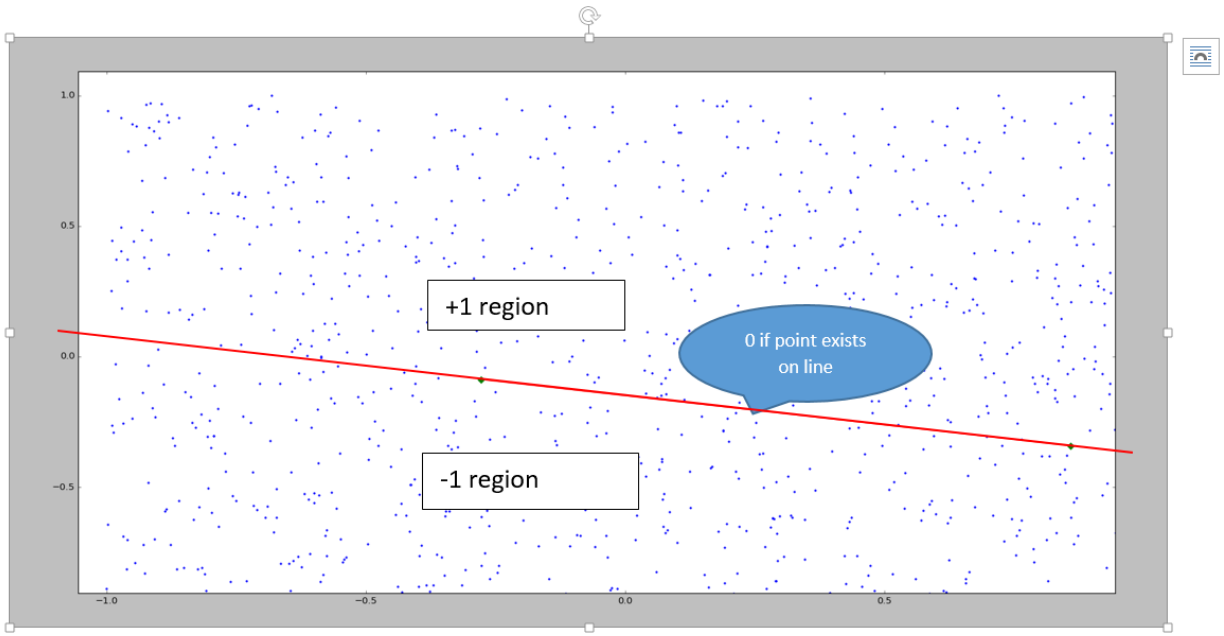
STUDY REPORT FOR HOMEWORK-2

SIVA KRISHNA SIRIGINEEDI
800887277

```
>>> np.matrix(np.random.uniform(0.9999999, 1, size=(100, 1, 2)))
matrix([[ 0.99999997,  0.99999994],
        [ 0.99999999,  0.99999995],
        [ 0.99999994,  0.99999994],
        [ 0.99999993,  1.        ],
        [ 0.99999991,  0.99999994],
        [ 0.99999995,  0.99999997],
        [ 0.99999995,  0.99999996],
        [ 0.9999999 ,  0.99999997],
        [ 0.99999997,  0.9999999 ],
        [ 0.99999993,  0.99999994],
        [ 0.99999994,  0.99999994],
        [ 0.99999993,  0.99999993],
        [ 0.99999991,  0.99999992],
        [ 0.99999992,  0.99999996],
        [ 0.99999998,  0.99999991],
        [ 0.99999999,  0.99999993],
        [ 0.99999992,  0.99999994],
        [ 0.99999998,  0.99999998],
        [ 0.99999999,  0.99999996],
        [ 0.99999995,  0.99999992],
        [ 0.99999991,  0.99999996],
        [ 0.99999994,  0.99999997],
        [ 0.99999999,  0.9999999 ],
        [ 0.99999991,  0.99999997],
        [ 0.99999993,  0.99999992],
        [ 0.99999993,  0.9999999 ],
        [ 0.99999991,  0.99999993],
```

ALGORITHM

- After the generation of input matrix X similarly we are generating 2 points.
- These two points are used for creating a line equation in the version space.
- **y-y1 - m(x-x1) = 0 (line equation passing through two points)**
- **f(x) =sgn(y-y1 - m(x-x1))**
- if we plug the points of matrix X in this equation f(x)
- Now the points in the matrix X are substituted in the linear inequality and signum function is applied which divides the whole Matrix X into y values with +1 or -1. These values are stored in Y matrix in the same order as X matrix with dimensions N*1.



DESIGN DECISIONS

STUDY REPORT FOR HOMEWORK-2

SIVA KRISHNA SIRIGINEEDI
800887277

- Every operation with the data, I am doing it using numpy matrix for faster and easier calculations.
- Linear inequality is used for classifying the input variables into class +1 or -1
- Equation of the Line passing through the two randomly selected points are selected as boundary line between the two classes.
- NUMPY broadcasting is used.

PERCEPTRON LEARNING ALGORITHM

ASSUMPTIONS

- My initial weight vector is zero row matrix of size (1*3).[w0 w1 w2]

ALGORITHM

- Initial weight vector is multiplied with every row of input data for finding out h(x) of every point present in the matrix and summed up.
- Signum function is applied over the summation for predicting the classes of points.
- We will pick out one misclassified point by comparing the predicted class with the label matrix.
- Misclassified point is multiplied with the label and added to the original weight matrix.
- Now again this changed weight is used to find out the h(x).
- This process is done iteratively till all points are classified correctly.
- When this process stops the weight at the last loop is the final weight.
- Final weight and no of iterations are returned.

DESIGN DECISIONS

- For the two dimensional points I am adding an artificial coordinate x0 = 1 to it. For this I am appending column one matrix to the Input matrix which is an artificial coordinate.
- I am using numpy matrix for calculating classes all at a time instead of doing each point at a time.
- For selecting the random misclassified point I am subtracting the label points with the predicted points, I am extracting non zero indexes which represent the misclassified point's location in the main data. From these points I am selecting one random point.

PSEUDO-INVERSE

ASSUMPTIONS

- Points which I have taken as input are two dimensional.

ALGORITHM

- For given input points and their labels we findout the weights such that the square error is minimized for all the points in the given data.

STUDY REPORT FOR HOMEWORK-2

SIVA KRISHNA SIRIGINEEDI
800887277

- Pseudo inverse of the input matrix is calculated and multiplied with label matrix for calculating the weight.
- This weight is feed into the PLA instead of zero initial matrix for finding out the final weight matrix

DESIGN DECISIONS

- For the two dimensional points I am adding an artificial coordinate x0 = 1 to it. For this I am appending column one matrix to the Input matrix which is an artificial coordinate.
- I am using numpy matrix for calculating classes all at a time instead of doing each point at a time.
- For selecting the random misclassified point I am subtracting the label points with the predicted points, I am extracting non zero indexes which represent the misclassified point's location in the main data. From these points I am selecting one random point.

RESULTS

PLA to converge with and without using Linear Regression to provide initial weights. For N = {10, 50, 100, 200, 500, 1000} compute the number of iterations of PLA with and without weight initialization using linear regression. For the uninitialized case, start the algorithm with the weight vector **w** being all zeros. Run each trial 100 times and compute the average for different randomly generated data sets.

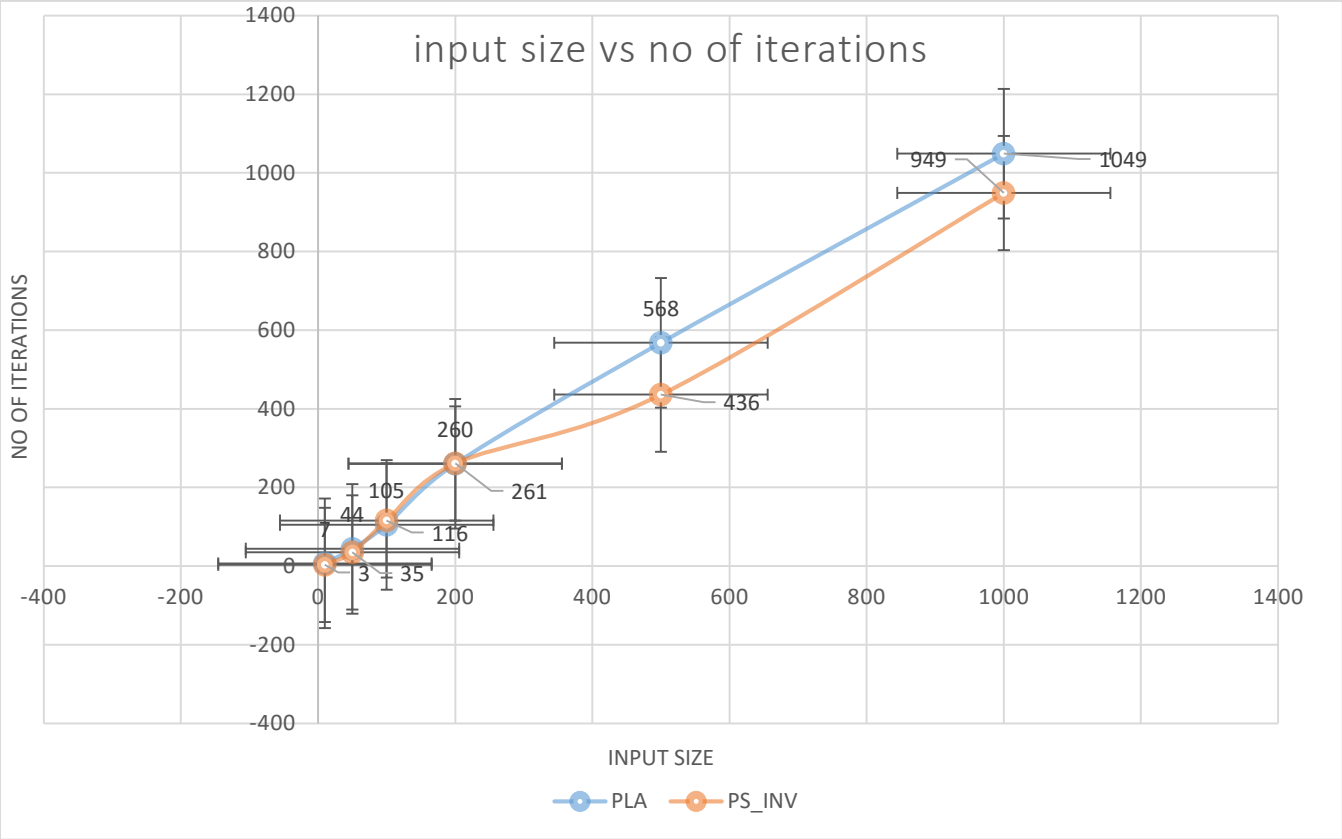
No of Iterations

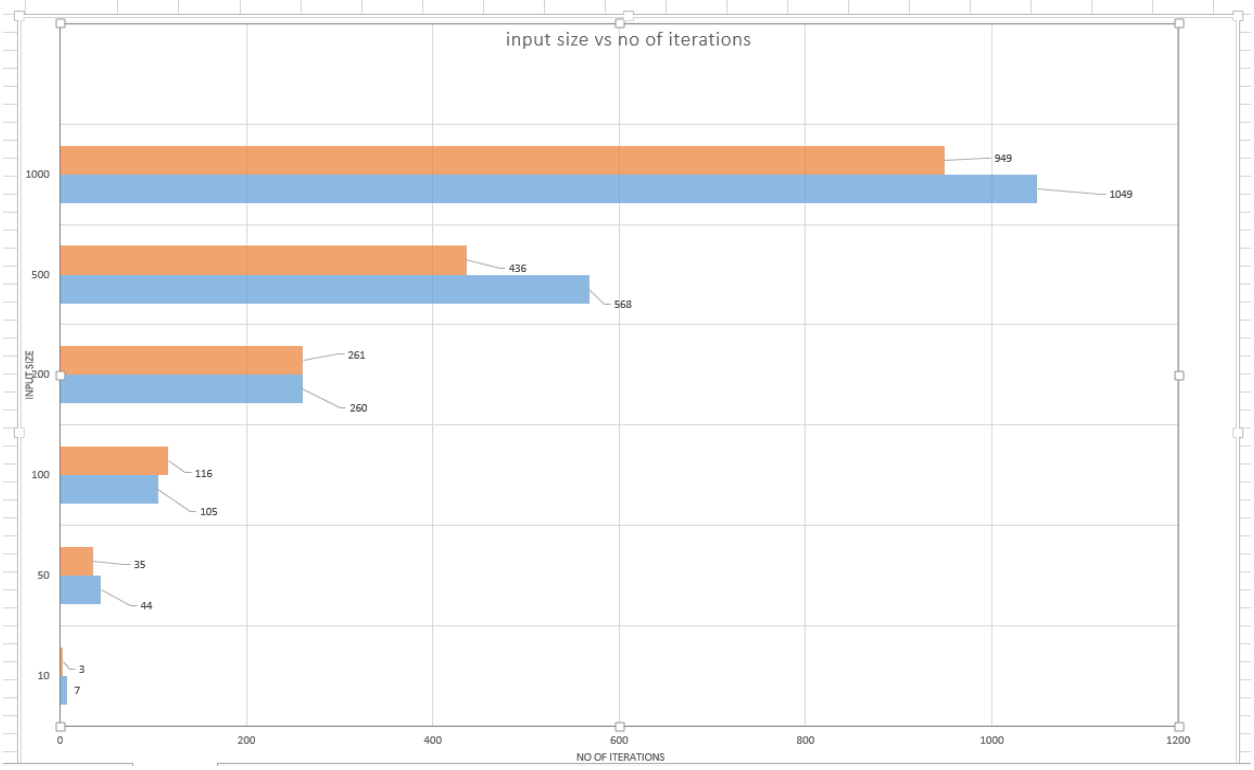
N(input size)	PLA	PS_INV
10	7	3
50	44	35
100	105	116
200	260	261
500	568	436
1000	1049	949

Lower no of iterations is better.

STUDY REPORT FOR HOMEWORK-2

SIVA KRISHNA SIRIGINEEDI
800887277





STUDY REPORT FOR HOMEWORK-2

SIVA KRISHNA SIRIGINEEDI
800887277

INFERENCES FROM GRAPH

- As input size increases no of iterations are increased irrespective of algorithm.
- Sometimes PLA has more no of iterations compared to PLA with LR weights.
- There is almost linear relation between input and no of iterations.
- No of iterations difference between the two algorithms is not much different.

We can conclude that there is no clear winner.

DESIGN DECISION FOR WHOLE HOMEWORK

- For loops are not used anywhere in the whole assignment just for repeating assignment with different input sizes.
- Misclassified points are picked randomly.
- Average no of iterations are rounded off to nearest integers

OBSERVATIONS

- Every time you run the experiment, no of iterations as well as weights change, this is because of change in the hypothesis function every time and randomly generated points every time we run the experiment. The points in the order in which they are fed into PLA also vary no of iterations as well as weights.
- Linear Regression weights could not help in decreasing the no of iterations for PLA, this is because the points are generated randomly and they are uniformly distributed, so finding initial weights by linear regression won't help much.
- Initially I just took the first misclassified point due to which no of iterations where high in all my experiments. When I changed the experiment for taking random misclassified point the no of iterations have been decreased and it is a welcome move. **(improvement changes)**
- For decreasing the time taken for computation I have removed all the **FOR** loops which has decreased the time drastically. **(improvement changes)**
- I created a validation function which checks whether the given weights are satisfying the hypothesis or not. This one I have commented.
- Using numpy matrices has improved the speed of experiment significantly .(10x times faster) **(improvement changes)**