HOMEWORK-4 INF 552
Submitted By:
Vivek Bhardwaj USC ID# 9237308454
Shushyam Malige Sharanappa USC ID# 8613558815
Raveena Mathur US ID# 9648632816

Ans.1 Code is enclosed in the zip folder. Please read the requirements.txt and the readme file.

Perceptron Algorithm :

Data Structures Used

A class **-** can bundle data and functions together. Models real world objects more closely. perceptron class consists of the following attributes:

- weights : the weights for the given input file
- alpha: the learning rate
- maxIter: the no. of iterations
- errorCounts: the no.of values that were computed incorrectly by the model

Input Data: Is a .txt file of comma separated values.

Function Calls and their sequence:

1. The input is taken using the np.loadtext function
2. create an object of class perceptron
3. Initialise the values of the attributes
4. Store input data as an array

call function train() to compute final weights and final no. of violations. Within function train() following happens:

- Random assignment of weights using np.random.rand function.
- The dot product of the input and the weights are taken.
- If the product is greater than 0 and the corresponding output is less than 0 or if the product is less than 0 and corresponding output is greater than 0 the weight gets updated.
- The code runs 7000 times.

Code Level Optimization

1. maxIter was fixed for this input as 7000.This value was the most optimum one. Any value greater than 7000 gave no change in weights
2. The learning coefficient alpha was fixed to 0.01
3. Built in functions from the libraries is used

.

Requested Output

```
Completed iterations: 500
Completed iterations: 1000
Completed iterations: 1500
Completed iterations: 2000
Completed iterations: 2500
Completed iterations: 3000
Completed iterations: 3500
Completed iterations: 4000
Completed iterations: 4500
Completed iterations: 5000
Completed iterations: 5500
Completed iterations: 6000
Completed iterations: 6500
Completed iterations: 7000
No. of Iterations: 7000
Final Weights [W0, W1, W2,...]: [-0.03516287 -0.17542565  0.11672868  0.07917204
]
Accuracy on the train dataset: 0.5285
                Predicted Correctly: 1057
                Total Samples: 2000
```

Challenges :

1. Fixing maxIter required testing for a wide range of values
2. The learning rate was fixed at 0.01, though other values were tried and tested

---

Ans. 2 **Software Familiarization:** Sklearn offers good implementation. The files in our code with '_SKLearn' suffix give a detailed code of how it is used.
Single layer perceptron is a linear classifier.

Comparison with Our Algorithm

```
Final Weights: [[ 32.50956328 -25.9950333  -19.51079267]]
Accuracy on the train dataset: 1.0
                Predicted Correctly: 2000
                Total Samples: 2000
```

Improvements

1. Sklearn employs a function **`sparsify()`** to convert the matrix in to a sparse matrix. Converts the coef_ member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation. We can consider this in our program to improve the runtime.

Ans. 3 Applications of perceptron Algorithm:

1. Handwritten digit recognition
2. Application of perceptron algorithm in classifying response of an individual to a particular dosage of a drug

---

Pocket Algorithm

Data Structures Used

A class **-** can bundle data and functions together. Models real world objects more closely.

perceptron class consists of the following attributes:

- weights : the weights for the given input file
- alpha: the learning rate
- maxIter: the no. of iterations
- errorCounts: the no.of values that were computed incorrectly by the model
- bestWeights
- bestErrorCount

Run perceptron while keeping extra set of weights in your pocket.Whenever the perceptron has a longest run of consecutive correct classifications, these perceptron weights replace the pocket weights. The pocket weights are the outputs of the program.

Input Data: Is a .txt file of comma separated values.

Function Calls and their sequence:

1. The input is taken using the np.loadtext function
2. create an object of perceptron class and initializes learning rate to 0.01 and maxIter =7000
3. Store input data as an array
4. calls function train() - In the function train() we do the following:
   - The dot product of the input and the weights are taken.
   - If the product is greater than 0 and the corresponding output is less than 0 or if the product is less than 0 and corresponding output is greater than 0 the weight gets updated.
   - We store the best weights so far as pocket weights.
   - Train function computes the attributes described above with respect to the input data

Code Level Optimization :

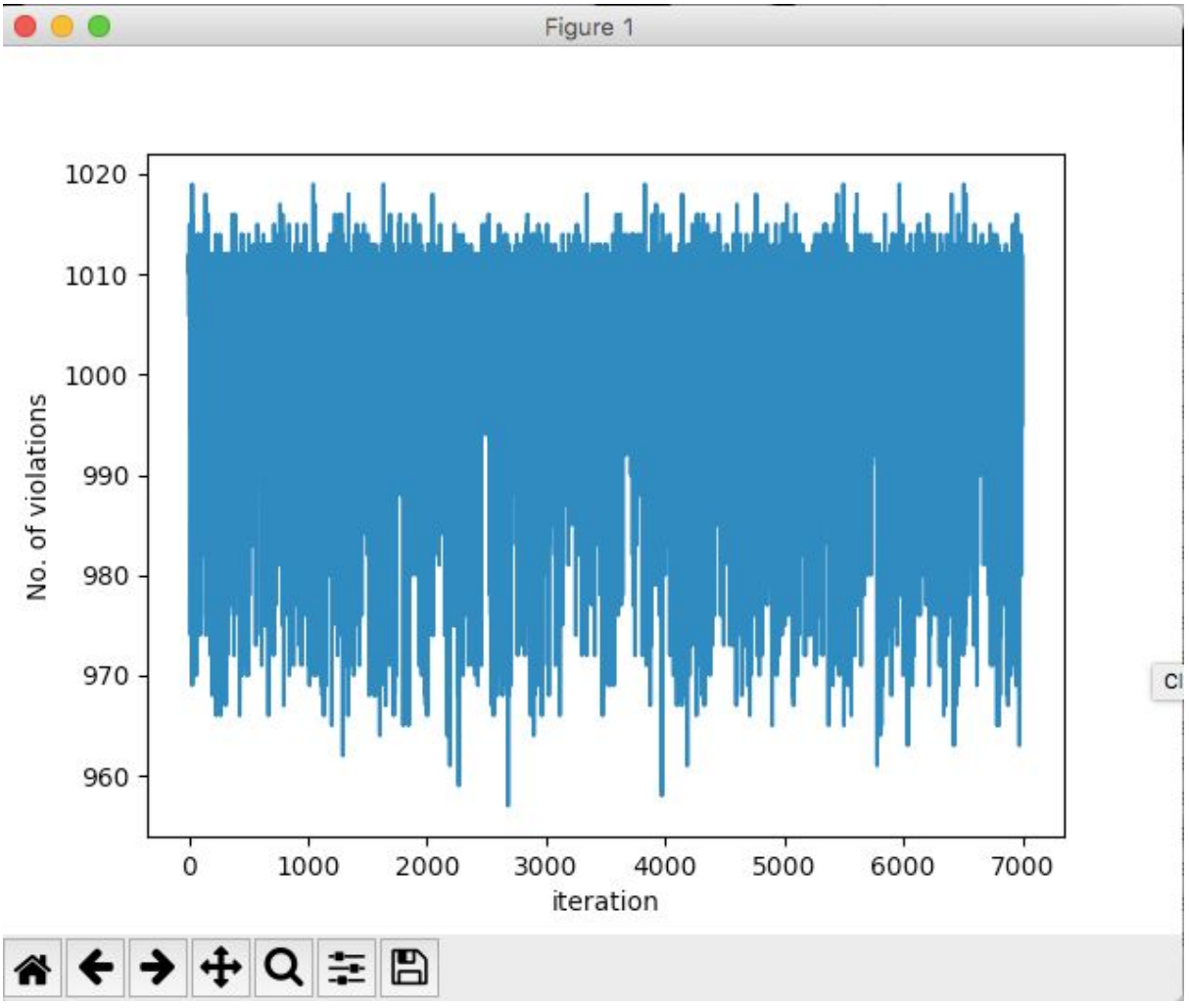
1. Learning rate optimally chosen as 0.01
2. Built in functions from the libraries are used which makes our efficient.

Challenges

1. Fixing the learning rate to an optimal value

Requested Output

```
Completed iterations: 500
Completed iterations: 1000
Completed iterations: 1500
Completed iterations: 2000
Completed iterations: 2500
Completed iterations: 3000
Completed iterations: 3500
Completed iterations: 4000
Completed iterations: 4500
Completed iterations: 5000
Completed iterations: 5500
Completed iterations: 6000
Completed iterations: 6500
Completed iterations: 7000
No. of Iterations: None
Best Weights: [ 0.00859451 -0.00724767  0.0028195  -0.00911422]
Best/Least No. of Violations: 957
Iteration of occurence: 2681
Accuracy on the train dataset: 0.5025
                Predicted Correctly: 1005
                Total Samples: 2000
```



Applications of Pocket Algorithm :

1. Can be applied to credit screening: whether an applicant will be approved a credit card or not based on certain features like income, credit history.

Linear Regression

Data Structures Used
A class **-** can bundle data and functions together. Models real world objects more closely.
Linear Regression class consists of the following attributes:
weights

Input Data: Is a .txt file of comma separated values.

Function Calls and their sequence:

1. A Linear regression object is created
2. Store input data as an array
3. A function linearregression() is called that computes the weights

The mathematical formula used to compute linear regression is as follows:

Least Squares Solution

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

X: matrix of values of input data

Code Level Optimization :

1. Built in functions from the numpy library for computing inverse of matrices.

Challenges

1. Finding the right function in the numpy library that offers a good implementation of matrix operations
2. Finding the right function for computing inverse.

Requested Output

```
guest-wireless-nup-1622-10-121-100-199:code raveenamathur$ ./RunLinearRegression
.sh
Final Weights [W0, W1, W2,...]: [[ 0.01523535  1.08546357  3.99068855]]
```

Software Familiarization

Sklearn offers good implementation. The files in our code with '_SKLearn' suffix give a detailed code of how it is used.

Comparison with Our Algorithm

The output of sklearn is as under

```
Raveenas-Air:code raveenamathur$ ./RunLinearRegression_SKLearn.sh
Final Weights: [ 1.08546357  3.99068855]
```

Improvements

1. **fit_intercept** if set to false the input is not normalised. If set to true the input is normalised. Normalising is a linear transformation of data to better the accuracy of the algorithm sometimes. We can add a configurable code to do normalisation on a need basis depending on the dataset.

.

Applications of Linear Regression

1. Predicting house prices with increasing size of houses
2. Predicting crop yields on rainfall

Logistic Regression

Data Structures Used
A class - can bundle data and functions together. Models real world objects more closely. LogisticRegression class consists of the following attributes:

- weights
- alpha
- maxIter

Input Data: Is a .txt file of comma separated values.

Function Calls and their sequence:

1. create an object of logistic regression and initialize the rate of learning as 0.001 and maxIter as 7000
2. store input data as array
3. call function train(). this function calls gradient which computes the gradient descent according to the sigmoid function. The function updates weights according to the current value of weights, learning rate and the gradient.
4. call function predict() to predict values for the given data.
   Within the predict function a call is made to findProb that computes the dot product of weights with x (the features of the data) and multiplies that to the actual output of classification on those features (y) . Then it computes the sigmoid on this computed value.

Code Level Optimization :

1. Built in functions from the numpy library is used.
2. The runtime was reduced from an initial 4 minutes to around 2 minutes by using inbuilt functions for matrix operations.

Challenges

1. Fixing the rate of learning. The final value of weights change with the rate of learning. Therefore fixing this value required lot of trials.

Requested Output

```
Completed iterations: 500
Completed iterations: 1000
Completed iterations: 1500
Completed iterations: 2000
Completed iterations: 2500
Completed iterations: 3000
Completed iterations: 3500
Completed iterations: 4000
Completed iterations: 4500
Completed iterations: 5000
Completed iterations: 5500
Completed iterations: 6000
Completed iterations: 6500
Completed iterations: 7000
No. of Iterations: 7000
Final Weights [W0, W1, W2,...]: [-0.03516287 -0.17542565  0.11672868  0.07917204
]
Accuracy on the train dataset: 0.5285
                Predicted Correctly: 1057
                Total Samples: 2000
```

Software Familiarization

Sklearn offers good implementation. The files in our code with '_SKLearn' suffix give a detailed code of how it is used.

Comparison with Our Algorithm

The output of sklearn is as under

```
[guest-wireless-nup-1622-10-121-100-199:code raveenamathur$ ./RunLogisticRegressi
on_SKLearn.sh
Final Weights: [[-0.17393989  0.11141144  0.07456303]]
Accuracy on the train dataset: 0.529
                Predicted Correctly: 1058
                Total Samples: 2000
guest-wireless-nup-1622-10-121-100-199:code raveenamathur$
```

Improvements

1. Sklearn employs a function **`sparsify()`** to convert the matrix in to a sparse matrix. Converts the coef_ member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual

numpy.ndarray representation. We can consider this in our program to improve the runtime.

.

Applications of Logistic Regression

1. Crime Data Mining : Predicting the crime rate of a state based on drug usage, number of gangs, human trafficking, and Killings.
2. Assessing the risk of cardiovascular diseases based on current habits and health