Programming Assignment 4

Project Report by Arpit Parwal & Yeon-soo Park

1. Implementation Algorithms

a. Data structures

i. Perceptron Learning algorithm / Pocket algorithm

Pocket algorithm is the modification of Perceptron algorithm so both algorithms share similar data structures. For both algorithms, we have to define the input dataset, a number of iteration, misclassified points, and initial weights. We used a **Numpy array** to store data points and the initial weights. Since the Pocket algorithm keeps the best result in its pocket to get the minimum number of misclassification, it requires additional data structure, list and integer variable, to save misclassified points on every iteration and plot the results.

ii. Logistic Regression

Class Structure to bundle data and functions together.. The class consist of the following variables

1. weights
2. learningRate
3. maxIter

A **NumPy array** to store coordinates and perform inverse functions on them

iii. Linear Regression

A NumPy array to store coordinates and perform inverse functions on them
Input Data: This is a .txt file of comma-separated values.
The data was fetched in an array using **pandas** and **list**

b. Explanation of Modules

i. Perceptron Learning algorithm / Pocket algorithm

- **Main Function**
  This main function reads the dataset from the given CSV file in NumPy array. Then, we get the number of inputs from the dataset and pass this value to the constructor of the Perceptron class. In the perceptron class, we create a construct function def __init__ to initialize the weights vector W including threshold **b and learning rate.** We set the size of W as the number of inputs + 1 (bias **b**). We set the learning rate as 0.0001 which was optimal from our experiments.

- **Train Perceptron Model - Train / Predict Function**
  This model takes input values **data points** as an argument and performs the weighted aggregation of **data points** (dot product between **w.x = w * x**) and call

predict function that returns the value 1 if the aggregation is greater than the threshold **0** else 0. Predict function takes input values **x** as an argument and for every observation in **x** and returns the result between 0 and 1. Next, We update weight based on the result.

*** Weight Update Formula [W1 = W0 + L(O-Y)*X] ***
(w1= weight updated, w0 = current weight, L = learning rate,
O = label, Y = actual output, X = input)
(we set O-Y as -1 or 1 which is error rate because this is binary classification)

Here, we set L as 0.0001. O(label) is given from dataset (-1 or 1) and Y is the result out from the model (0 or 1 from the activation function)

- **Evaluation Function**
  Each data point is evaluated exactly once in every iteration to evaluate the trained model performance.

- **Print Output Function**
  This prints the result of weight and accuracy after the final iteration. In the pocket algorithm, this additionally plots the number of misclassified points
  against the number of iterations.

ii. Logistic Regression

- ***Main Driver Function***
  To create an object of logistic regression and initialize the rate of learning as 0.001 and maxIter as 7000. This function also fetches data and stores it in the required form
- ***Train***
  This function iteratively calls and updates the gradient variable which is computed by calling the ***findGradient***
- ***findGradient***
  This function calculates the gradient according to the sigmoid function.

```
Nr = Yi * Xi
Dr = 1 + exp(Yi * wT dot Xi)
return Nr / Dr
```

The function updates weights according to the current value of weights, learning rate, and the gradient.

- ***Predict***
  This function is used to predict values for the given data. Within the predict function, a call is made to ***findProb***
- ***findProb***
  This function 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.
- **printResult**
  A utility function to print the result in the desired format

iii. **Linear Regression**

- ***Main Driver Function***
  The main driver function gets the data and arranges it into np array matrixes to be used by **runAlgo**
- **runAlgo**
  This function runs the linear algorithm. It finds the inverse using the formula
  `b = (X`X)^-1 X`Y`

c. Challenges

i. **Perceptron Learning algorithm / Pocket algorithm**

- **Configure the learning rate:** To achieve good performance, the optimal weights should be discovered via an empirical optimization procedure by tuning the neural network. We have to discover this via trial and error because we cannot analytically calculate the optimal learning rate.

ii. **Logistic Regression**

- **Learning Rate**
  In order to find an ideal learning rate, there were several tests and run of the algorithms done. We iteratively called the algorithm with different learning rates and compared their results. We started with a broader range with alpha varying from 0-1 then to 0-0.1 and then finally found the best result on 0.05 which is the most ideal learning rate.

iii. **Linear Regression**

- **Finding right function to handle huge dataset**
  In order to run an extensive calculation, we searched for the best available function in NumPy Library

d. Code-level optimization

i. Perceptron Learning algorithm / Pocket algorithm

- To reduce computational load, we use the NumPy array to calculate dot products between weights and input data.
- Define classes for both models and functions to reduce the size of the program and repetitive tasks.

ii. Logistic Regression

- **Numpy Array**
  Using Numpy Array for matrix calculation and as a comprehension over explicit for loops in certain places, making code more compact and execution faster.
- **Class Structure**
  Using of classes as data structure helped in data handling and passing the model directly to predict the functions

iii. Linear Regression

- **Built-in functions**
  Using Numpy Array built-in functions for matrix calculation and inverse as a comprehension over explicit for loops in certain places made the code efficient

e. Output

i. Perceptron Learning algorithm

Experiment 1

```
Iteration 50, misclassified points 4, Evaluation 99.8%

======== Result =========
Iteration 65, misclassified points 0
Evaluation 100.0%
Weights After Final Iteration:  [ 0.0002  0.0774 -0.062  -0.0466]

Process finished with exit code 0
```

Experiment 2

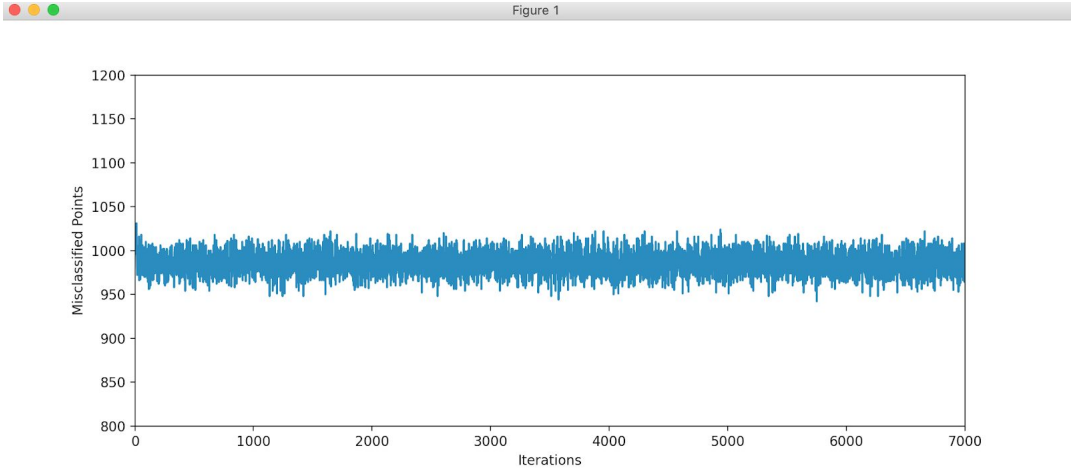
```
Iteration 50, misclassified points 9, Evaluation 99.7%
Iteration 100, misclassified points 4, Evaluation 99.85000000000001%

======== Result =========
Iteration 147, misclassified points 0
Evaluation 100.0%
Weights After Final Iteration:  [-0.     0.145 -0.117 -0.087]
```

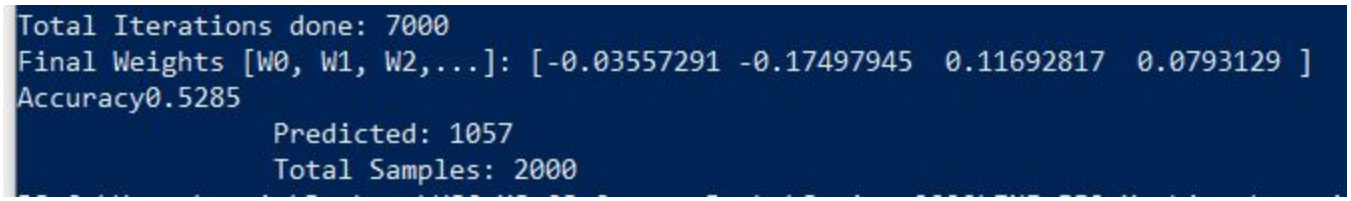
ii. Pocket algorithm

```
Iteration 500, misclassified points 996, Evaluation 49.4%
Iteration 1000, misclassified points 990, Evaluation 50.3%
Iteration 1500, misclassified points 1004, Evaluation 50.6%
Iteration 2000, misclassified points 968, Evaluation 49.4%
Iteration 2500, misclassified points 996, Evaluation 49.4%
Iteration 3000, misclassified points 994, Evaluation 49.3%
Iteration 3500, misclassified points 979, Evaluation 49.4%
Iteration 4000, misclassified points 953, Evaluation 49.4%
Iteration 4500, misclassified points 972, Evaluation 49.8%
Iteration 5000, misclassified points 992, Evaluation 49.4%
Iteration 5500, misclassified points 1000, Evaluation 49.4%
Iteration 6000, misclassified points 982, Evaluation 50.3%
Iteration 6500, misclassified points 980, Evaluation 49.4%
Iteration 7000, misclassified points 986, Evaluation 49.85%

======== Result =========
Iteration 7000, misclassified points 986
Evaluation 49.85%
Minimum Misclassified Points/Best Result:  944
Weight After Final iteration:  [ 0.0000793 -0.0000181  0.0001152 -0.000121 ]
Best Weights of Pocket:  [ 0.0000793 -0.0000124  0.0001266 -0.0000203]
Best Accuracy of Pocket: 52.800000000000004%
```



iii. **Logistic Regression**



iv. **Linear Regression**

2. Software Familiarization

a. Existing Libraries

i. Perceptron

- Scikit: SciKit is the most popular machine learning library for Python, which has so many built in support for Neural Network models including Perceptron. We can customize the parameters; choose the number of layers, learning rate, activation functions and batch size.
- PyPi Perceptron: This Perceptron implements a multilayer *perceptron* network written in Python. This network consists of multiple layers of neurons, the first of which takes the input. The last layer gives the output.

ii. Pocket Algorithm

- Since pocket algorithms are based on perceptron learning algorithm, there is no specific library that supports pocket algorithm. we can easily implement the Pocket algorithm by keeping the best result during the entire process.

iii. Logistic Regression

- **Sklearn**
  It's a machine learning library that includes various machine learning algorithms. It uses the following inbuilt library to implement logistic regression algorithm

sklearn.linear_model ⇒ LogisticRegression

iv. **Linear Regression**

- **Scikit-Learn (for Linear Regression)**
  It's a machine learning library. It includes various machine learning algorithms which use the following inbuilt library
  sklearn.linear_model ⇒ LinearRegression

b. Comparison

i. Perceptron Learning algorithm

- Perceptron Learning Algorithm gives different results for each execution because various parameters such as initial weights, learning rate, or normalized inputs can affect the results represented as the Weight Update Formula. Since our scope is limited to an efficient implementation of the Perceptron Algorithm, advanced customization is not supported. On the other hand, libraries listed here cannot update model weights using customer formulas, because when we set parameters, they define the formula before running the model.

ii. Pocket Algorithm

- We were not able to find any popular implementation of the Pocket Algorithm, but we were able to manage to find a few implementations from some papers. (Gallant, S. I. (1990). Perceptron-based learning algorithms. IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191.)

iii. Logistic Regression

- The output had no significant difference and it gave a similar result as to ours. However, there were some minor changes noted on the weights. This can be due to a different learning rate.

```
Final Weights: [[-0.17393989  0.11141144  0.07456303]]
Accuracy on the train dataset: 0.529
                Predicted Correctly: 1058
                Total Samples: 2000
```

iv. Linear Regression

The output had no significant difference and it gave a similar result as to ours. However, the runtime was way better and the result was much faster.

c. Improvements

i. **Perceptron Learning algorithm/Pocket Algorithm**

- We can improve the performance by finding a faster method for the computation of dot products. When we process a big dataset or run a complicated computation on a massive matrix with Numpy, we can accelerate the training of the model by using existing libraries that support running the model on GPU.

ii. **Logistic Regression**

- Sklearn employs a function sparsify() to convert the matrix into 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.

iii. **Linear Regression**

- Normalizing is a linear transformation of data to better the accuracy of the algorithm sometimes. We can add a configurable code to do normalization on a need basis depending on the dataset.

3. Applications

a. **Predict user behavior (Perceptron Algorithm)**
Perceptron is an algorithm for binary classification that uses a linear prediction. One of the applications of Perceptron we can predict whether a web user will click on an ad for a refrigerator. Given the features from the users who bought a refrigerator recently, usually don't click ads, and searched reviews of refrigerators, we can predict that if users will click the ads or not.

b. **Credit Screening (Perceptron/Pocket Algorithm)**
Another application of Perceptron is credit screening to determine if an applicant will be approved or not on data in the future. Pocket Learning Algorithm can work on this data by finding the best weight vector to predict the screening.

c. **Image Segmentation(Logistic Regression)**
This method is based on feeding artificial features to a framework of logistic regression classifiers with $\ell 1$ norm regularization and Markov Random Field prior, which has been studied, e.g., in hyperspectral image segmentation. This method generates a large set of artificial features and passes them to a $\ell 1$ regularized logistic regression classifier. Finally, a spatial prior imposes additional homogeneity. [Reference]: http://ieeexplore.ieee.org/document/6334177

d. **Analysis of traffic rules (Linear regression)**
Linear regression is used when the response is a continuous variable(CV). This study mainly discusses the overtaking and lane-changing problem in highways, the differences between traveling on the left and on the right lane. In order to promote traffic smoothly effectively, an overtaking model is constructed. The left lane changing rule considers its influence on the human heart, and then multiple linear regression models. In the country with the cars driving on the left, this model is especially suitable for the near S-shaped highway. By studying the joint influence of the curve

radius of curvature, steering angle and the road friction coefficient on the driving speed of the curve and the obtained speed is within the speed limit
http://www.jocpr.com/articles/application-of-multiple-linear-regression-model-inthe-performance-analysis-of-traffic-rules.pdf

4. Individual Contribution

Student Name	Programming Part	Documentation
Arpit Parwal **aparwal@usc.edu**	**Logistic Regression** Algorithm- Read data file, Run **Linear regression** algorithm, calculate distances, modifying distance function on each iteration, plotting data on a 2D graph	Data structure, Modules, Challenges, improvements, Existing libraries, applications for Logistic and linear Regression
Yeon-soo Park **yeonsoop@usc.edu**	Perceptron/Pocket algorithm - the main function, calculating dot products, Read csv file, Plotting the results on a graph	Data structure, Modules, Comparison, improvements, Code-level optimization