LINEAR PERCEPTRON, REGRESSION, AND LOGISTIC REGRESSION

PART 1 IMPLEMENTATION

To allow for debugging, blocks of independent code is commented out per sections within hw4.py. Please comment/uncomment them to test. The running speed for Pocket Algorithm and Logistic regression is VERY slow, as it requires iterating through 7000*2000 rows of data.

LINEAR PERCEPTRON

I have implemented the Linear Perceptron as a class in perceptron.py. You can call digest() for each row and the perceptron will continually update its weights to classify better. After running through the entire dataset, the error of classification for the training set becomes 0.016. After running the training a few more times, the classification error approaches very close to 0.

POCKET ALGORITHM

The perceptron class is extended as the pocket algorithm. However, the digest now takes all rows and values instead of a single data. I tried first running the pocket algorithm with the original perceptron data. Since it's a close modification of the linear perceptron, the errors were about the same. I then changed the column for value to the 4th column. The weight resulted in [0.5, -0.422, 0.150, -0.527], and the error was around .478.

LINEAR REGRESSION

Linear regression is almost entirely math. The formula is implemented in linreg.py, by simply calling getW().

The resulting in weight of [0.015, 1.085, 3.991]

LOGISTIC REGRESSION

Logistic regression feels like the pocket algorithm, but using a different, more complicated weight update function. It is implemented in logitreg.py. After around 1000 iterations, the weights converge to [-0.0315, -0.178, 0.114, 0.077].
Although it is difficult to 'classify' a row of data using logistic regression (as it gives probability instead of classification) I simply split the classification between .5, and tried to obtain an error.
I tried the logistic regression with both column 4 – the simple case. This gave a very low error.
When using the logistic regression on column 5, the error was much worse: .4705.

PART 2 SOFTWARE FAMILIARIZATION

Before I begin, I'd like to note that the while doing some research on the logistic regression (as well as linear perceptron), most lecture notes found use 1 and 0 as the values rather than +1 and -1. This has caused a lot of confusion especially while dealing with the sigmoid function and its gradient. However, during testing, I implemented the weight updating both ways and discovered that the resulting weights are the same no matter what function is used. This is a good sanity check.

LINEAR REGRESSION

For linear regression, I used R to check if the coefficients were correct or not. The following is the output of my R console:

```
df <- read.table("linear-regression.txt", header=TRUE)
mat = data.matrix(df)
fit <- lm(mat[,"Z"]~mat[,"X"]+mat[,"Y"])
fit

Call:
lm(formula = mat[, "Z"] ~ mat[, "X"] + mat[, "Y"])

Coefficients:
(Intercept)   mat[, "X"]   mat[, "Y"]
   0.01524      1.08546      3.99069
```

As you can see, the weights we got from part 1 corresponded exactly to our output from R.

PERCEPTRON AND LOGISTIC REGRESSION

Perceptron and Logistic regression are for the most part being replaced by Neural networks and other more sophisticated algorithms within most libraries. (Especially perceptron)

Within Weka, we can find Logistic and Simple logistic regression, but Perceptron learning has been replaced by Multilayered Perceptron (Neural net) and Voted Perceptron.

We had to do some preprocessing of the data to get them running properly in Weka (it does not expect result value to be numerical) The results are shown below:

USING COLUMN 4 – LINEARLY CLASSIFIABLE DATA

```
=== Run information ===

Scheme:       weka.classifiers.functions.Logistic -C -R 1.0E-8 -M 7000 -num-decimal-places 4
Relation:     classification2
Instances:    2000
Attributes:   4
             X
             Y
             Z
             A
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Logistic Regression with ridge parameter of 1.0E-8
Coefficients...
```

```
                             Class
Variable                        No
===================================
X                        -763.9505
Y                         611.2391
Z                         458.3186
Intercept                  -0.0068


Odds Ratios...
                             Class
Variable                        No
===================================
X                                0
Y            2.8692967922776186E265
Z            1.1097777515546316E199


Time taken to build model: 0.15 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.05 seconds

=== Summary ===

Correctly Classified Instances        1999               99.95   %
Incorrectly Classified Instances         1                0.05   %
Kappa statistic                          0.9989
Mean absolute error                      0.0018
Root mean squared error                  0.0241
Relative absolute error                  0.4161 %
Root relative squared error              5.166  %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC 
Area  Class
                0.999    0.000    1.000      0.999    1.000      0.999    1.000     1.000     No
                1.000    0.001    0.998      1.000    0.999      0.999    1.000     1.000     Yes
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      0.999    1.000     1.000

=== Confusion Matrix ===

   a    b   <-- classified as
1354    1 |    a = No
   0  645 |    b = Yes
```

USING COLUMN 5 – COMPLEX DATA

```
=== Run information ===

Scheme:       weka.classifiers.functions.Logistic -C -R 1.0E-8 -M 7000 -num-decimal-places 4
Relation:     classification4
Instances:    2000
Attributes:   4
             X
             Y
             Z
             B
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Logistic Regression with ridge parameter of 1.0E-8
Coefficients...
              Class
Variable        Yes
```

```
===================
X            -0.1777
Y             0.1145
Z             0.0767
Intercept    -0.0315


Odds Ratios...
              Class
Variable        Yes
===================
X             0.8372
Y             1.1213
Z             1.0797


Time taken to build model: 0.03 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances        1059               52.95   %
Incorrectly Classified Instances       941               47.05   %
Kappa statistic                          0.0562
Mean absolute error                      0.4994
Root mean squared error                  0.4997
Relative absolute error                 99.8935 %
Root relative squared error             99.9467 %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC
Area  Class
                0.406    0.350    0.531      0.406    0.460      0.058    0.521     0.512     Yes
                0.650    0.594    0.529      0.650    0.583      0.058    0.521     0.517     No
Weighted Avg.    0.530    0.473    0.530      0.530    0.522      0.058    0.521     0.515

=== Confusion Matrix ===

  a   b   <-- classified as
401 587 |   a = Yes
354 658 |   b = No
```

Although the weights for column-5 data are very similar from my own results, WEKA's algorithm is much faster even with 7000 iterations. Our weights for the column-4 data looks completely different, however (mine was [-0.299, 11.76, -9.07, -6.92], while Weka's are [-0.007, -764, 611, 458]…a strange set of weights).

I'm not sure how its algorithm that makes it much faster than my algorithm – but it's possible that it uses Stochastic gradient descent, which makes the algorithm O(n) instead of O(n^2).

APPLICATION

LINEAR REGRESSION

Let us first talk about linear regression. This method is not only used for classification, but primarily for statistical analysis of data in the scientific realm. It is extended further for more complicated experiments such as Analysis of Variance.

LOGISTIC REGRESSION AND PERCEPTRON LEARNING

It's true that perceptron and logistic regression are for the most part being phased out by other more sophisticated learning algorithms, during my research, many people say that Neural Network is infamous for being rather slow. In such cases, Logistic Regression is used since it is usually much faster for simpler cases.

One use of Logistic Regression is in Image Segmentation – that is, identifying separate physical objects from an image. The images are pre-processed into 'super pixels' which are bounded on edge detection. It is then trained using human-segmented set.