Artificial Intelligence HW 4

(All programs have been written in Python 3.5.2)

Answer 1

(a)

Refer to the function KMeans() in the file HW4.py

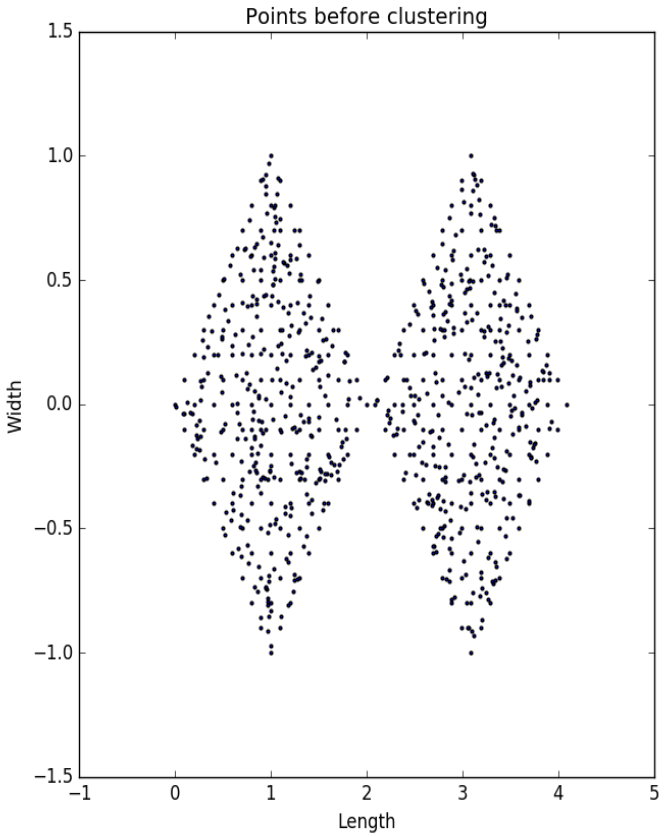
(b)

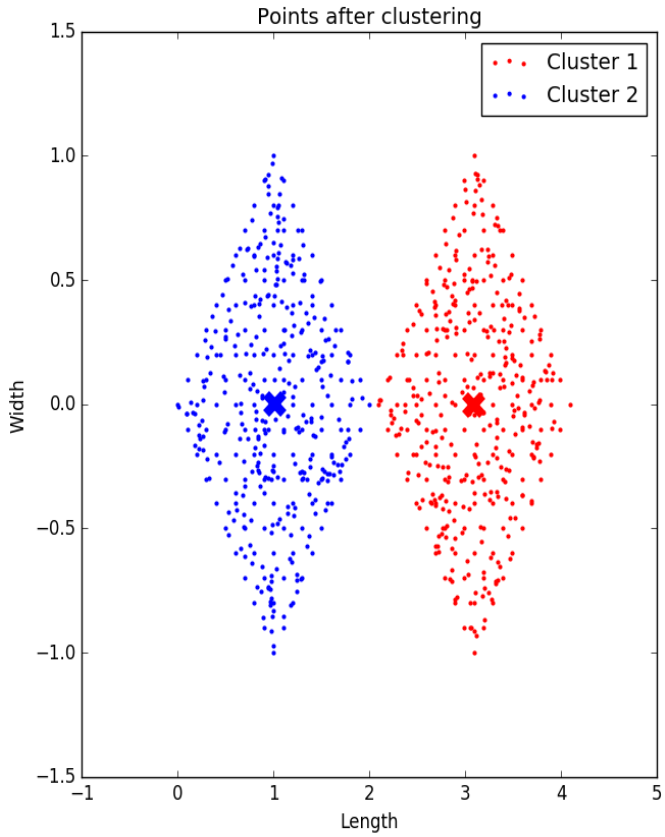
Refer to the file 1b.py. Plotting the data points, it's obvious that the number of clusters must be 2. After the KMeans clustering is performed, each cluster is found to have 400 points. Also, there's a counter variable j in the function that terminates the program after 100 iterations, to prevent a situation where the program takes way too much time. In the event of such an abrupt termination, please re-run the program. On a few runs of the program, it produced just one cluster, although the output was 2 on most runs.

OUTPUT:

```
Center of cluster 1 is at : [3.0840872249999993, -0.0020558000000000078]
No. of points in cluster 1 : 400
Center of cluster 2 is at : [1.0043701, 0.0044222525000000004]
No. of points in cluster 2 : 400
```

PLOT:





Answer 2

(a)

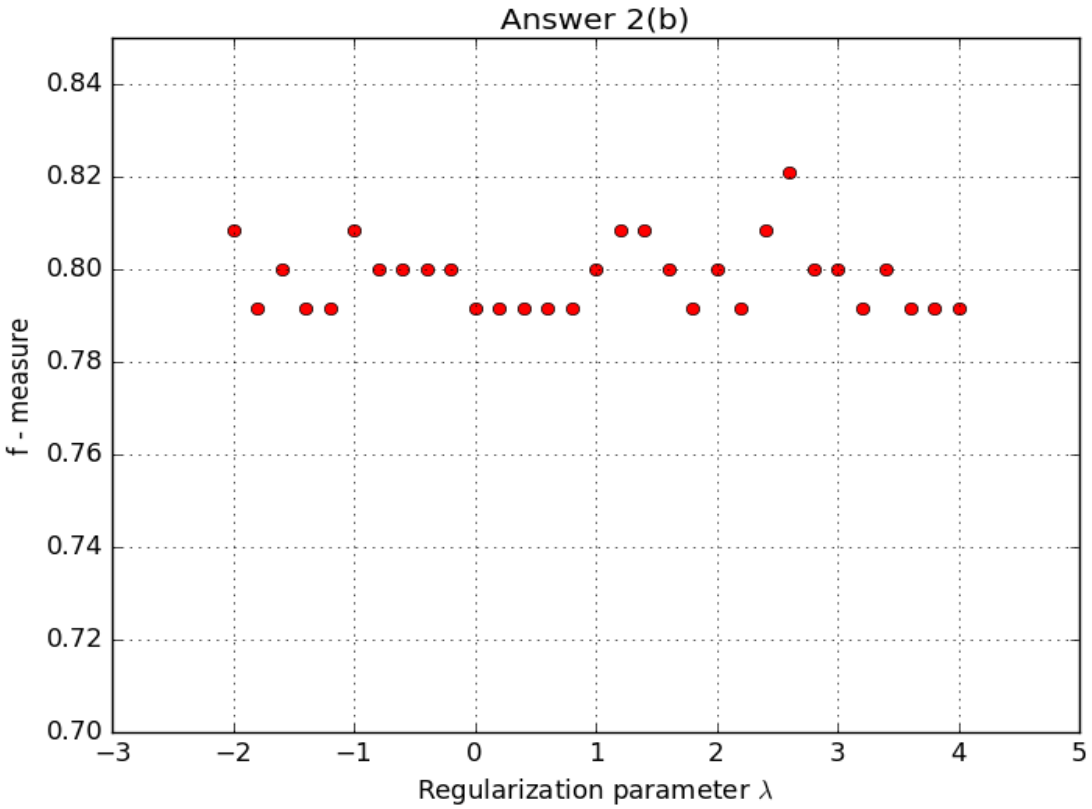
Refer to the function logistic_regression() in the file HW4.py. It invokes a function hyp() which is also included in HW4.py. The function hyp() simply calculates the hypothesis which is the output of the sigmoid function.

(b)

Refer to the file 2b.py. Please note that *Weka* was used to convert the `arff` file into a `csv` file and also to fill in the missing values. The data was further set up for logistic regression using the function process_data() which encodes all non-numeric values as numbers. The learning rate α (`alpha`) has been set as 0.0000001, and the epsilon ε (`ep`) which is used for the convergence test has been set as 0.001. It was very difficult to set values of α and ε such that the gradient descent algorithm converged for all 40 values of the regularization parameter λ (`lamb`). Even for the selected values of α and ε, the algorithm may sometimes not converge, which is why a counter variable count has been used to limit the number of iterations to a maximum of 6000, exceeding which exit() is used to terminate the execution preemptively. The execution time of this program is 55-60 minutes.

The output stated below is an array of 40 f-measures for the 40 regularization parameters λ from -2.0 to 4.0 in steps of 0.2. The plot indicates the f-measures are not dependent on λ, though it could be inferred that for λ >= 1.0, the f-measures are mostly >= 0.8, while for λ < 1.0, the f-measures are mostly <= 0.8.

PLOT:



OUTPUT:

```
array([ 0.80851064,  0.79166667,  0.8       ,  0.79166667,  0.79166667,
        0.80851064,  0.8       ,  0.8       ,  0.8       ,  0.8       ,
        0.79166667,  0.79166667,  0.79166667,  0.79166667,  0.79166667,
        0.8       ,  0.80851064,  0.80851064,  0.8       ,  0.79166667,
        0.8       ,  0.79166667,  0.80851064,  0.82105263,  0.8       ,
        0.8       ,  0.79166667,  0.8       ,  0.79166667,  0.79166667,
        0.79166667])
```

(c)

Refer to the file 2c.py. The features are scaled using the standardization protocol:

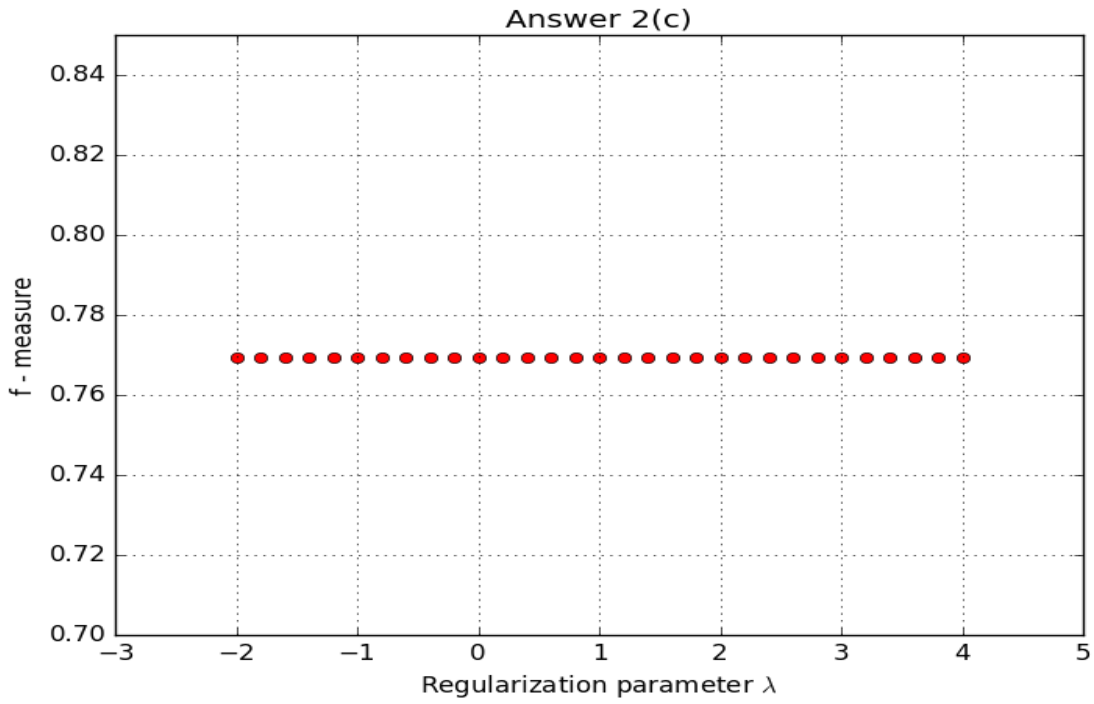
$$x_i \leftarrow \frac{x_i - \mu}{\sigma}, \text{ where } \mu \text{ is the mean and } \sigma \text{ is standard deviation}$$

Due to this operation, the execution is greatly speeded up, and the f-measures for all regularization parameters λ are calculated in less than 5 seconds. The values of the constants α and ε were kept the same as in part ***(b)***. An interesting observation is that due to the standardization, the f-measures seem to be independent of the regularization parameter λ. All the 40 f-measures are exactly the same; however, they are lower than the lowest f-measure in part ***(b)***.

OUTPUT:

0.76923077 for all λ

PLOT:



Answer 3

Refer to the file q3.py. All the 3 clustering techniques viz. `K-means clustering`, `Agglomerative clustering ('ward' linkage)`, and `Affinity Propagation clustering` have been implemented by importing KMeans, AgglomerativeClustering, and AffinityPropagation respectively from sklearn.cluster. While the first two clustering methods allow the number of clusters to be specified explicitly, the third method doesn't, and so the number of clusters for `Affinity Propagation` was brought to 10 by tweaking the preference parameter of AffinityPropagation. Furthermore, AffinityPropagation takes more time than the other two, and its *Fowlkes and Mallows index* is the lowest. AgglomerativeClustering produces the highest *Fowlkes and Mallows index*.

OUTPUT:

```
K-means clustering confusion matrix:
[[  0   1 177   0   0   0   0   0   0   0]
 [ 24   0   0   0   1  99  55   1   2   0]
 [147   0   1   4  13   8   2   0   0   2]
 [  0   0   0   7 154   7   0   2   0  13]
 [  0 162   0  10   0   2   7   0   0   0]
 [  0   2   0   0   1   0   0 136   1  42]
 [  0   0   1   0   0   2   1   0 177   0]
 [  0   0   0 175   0   2   2   0   0   0]
 [  3   0   0   5   2 101   6   4   2  51]
 [  0   0   0   9   6   1  20   5   0 139]]
Cluster labels according to majority population: [2 4 0 7 3 8 1 5 6 9]
----------------------------------------------------------------------
Agglomerative clustering confusion matrix:
[[  0   0   0   0   0   0   0 178   0   0]
 [  0   0  27   0   0   0   0   0  59  96]
 [  0   0 166   1  10   0   0   0   0   0]
 [  0 169   0   1  13   0   0   0   0   0]
 [  0   0   0   3   0 178   0   0   0   0]
 [179   2   0   0   0   0   1   0   0   0]
 [  0   0   0   0   1   0 180   0   0   0]
 [  0   0   0 179   0   0   0   0   0   0]
 [  0   1   4   1 165   0   0   0   1   2]
 [  2 145   0  11   2   0   0   0  20   0]]
Cluster labels according to majority population: [5 3 2 7 8 4 6 0 1 1]
----------------------------------------------------------------------
Affinity Propagation clustering confusion matrix:
[[  0   0   0   1 176   0   0   0   1   0]
 [  2  52   0   0   0   0 105  22   0   1]
 [  0   0  10   0   2   9   8 140   0   8]
 [  5   0 139   0   1   6   0   2   0  30]
 [  2   9   0   0   0   6  10   0 154   0]
 [104   1   4   7   3   2   1   0   1  59]
 [  1   0   0 172   1   0   7   0   0   0]
 [  0   0   0   0   0 170   9   0   0   0]
 [ 22   6  29   0   7   5  88   3   1  13]
 [  4  17   7   0   9   8   0   0   0 135]]
Cluster labels according to majority population: [5 1 3 6 0 7 1 2 4 9]
----------------------------------------------------------------------
K-Means clustering Fowlkes and Mallows index              : 0.699501787685
Agglomerative clustering Fowlkes and Mallows index        : 0.816751686074
Affinity Propagation clustering Fowlkes and Mallows index : 0.631065522676
```

Answer 4

Refer to the file q4.py. Using the same dataset as was made use of in **Answer 2**, the program makes use of process_data() from HW4.py. Also, the dataset used here is the same `csv` file created by *Weka* with the missing values filled in.

The `RandomForestClassifier` technique always has the best performance, with its f-measure very close to 1.0 every time this program is run. Using a `Support Vector Machine with linear kernel` also has a comparable performance, with its f-measure almost as good as that of `RandomForestClassifier's`. When compared to f-measures of these two, the performance of `Support Vector Machine with RBF kernel` lags behind.

OUTPUT:

```
F-measure for SVM with linear kernel      = 0.979591836735
F-measure for SVM with rbf kernel         = 0.769230769231
F-measure for RandomForestClassifier      = 0.989898989899
```