Assignment 1

60-473/574

Dr. Rueda

Joel Rorseth

October 6, 2017

When classifying using the K Nearest Neighbor classifier, a tie resolution scheme will need to be decided upon when a clear winner cannot be established. More specifically, a measure of efficiency must be established to compare the K Nearest Neighbor classifier with various k values. If we take measure the *score* of the classifier, a tie-resolution scheme would be required if the best score of the classifier across all k values was common to more than one value of *k*. This commonly occurs when more than one *k* (used in a K Nearest Neighbor classifier) produces a perfect score of 1.0. Below is the recorded output on the "Two Spirals" dataset, showing the dataset before and classification after.

```
twospirals.csv
-----------------------------------------------
Data before classification:
Training examples:
 [[ 9.6   -2.76 ]
 [-0.913  5.1  ]
 [ 7.96  -1.71 ]
 ...,
 [ 5.96  -5.25 ]
 [ 6.2    9.24 ]
 [-1.87   7.91 ]]
Training labels:
 [ 1.  2.  2.  2.  1.  1.  2.  2.  2.  1.  2.  1.  1.  1.  1.  2.  2.  1.
  2.  1.  1.  1.  2.  1.  2.  1.  1.  2.  1.  2.  2.  2.  1.  1.  1.  2.
  1.  1.  1.  2.  2.  2.  2.  1.  1.  1.  2.  1.  1.  1.  2.  1.  1.  1.
  2.  2.  1.  2.  2.  2.  2.  2.  2.  2.  1.  1.  1.  1.  2.  1.  1.  1.
  1.  2.  2.  2.  1.  1.  2.  1.  2.  1.  2.  2.  1.  1.  2.  2.  2.  2.
  1.  1.  1.  1.  1.  1.  2.  2.  1.  2.  1.  1.  2.  1.  1.  1.  1.  1.
  1.  1.  1.  2.  1.  1.  1.  1.  2.  1.  1.  2.  1.  2.  1.  1.  1.  2.
  1.  1.  2.  2.  1.  1.  2.  1.  1.  2.  2.  1.  2.  2.  1.  1.  2.  1.
  2.  1.  1.  1.  2.  2.  1.  1.  2.  2.  2.  2.  2.  1.  1.  2.  2.  2.
  2.  1.  2.  2.  2.  2.  2.  1.  1.  2.  1.  2.  1.  1.  1.  1.  2.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  2.  2.  1.  2.  2.  2.  2.  2.  2.  1.
  2.  2.  2.  2.  1.  1.  1.  2.  2.  2.  1.  1.  1.  1.  2.  2.  1.  1.
  1.  1.  2.  1.  1.  1.  2.  1.  1.  2.  2.  2.  2.  2.  2.  1.  1.  1.
  2.  1.  1.  1.  1.  1.  2.  2.  1.  1.  1.  1.  1.  2.  2.  2.  2.  1.
  1.  2.  2.  2.  2.  1.  2.  2.  1.  2.  1.  1.  1.  2.  1.  1.  2.  1.
  1.  2.  2.  1.  1.  2.  2.  1.  2.  1.  2.  1.  2.  1.  2.  1.  1.  1.
  2.  1.  1.  1.  1.  1.  1.  2.  2.  2.  2.  1.  1.  1.  2.  1.  2.  2.
  1.  1.  1.  1.  1.  2.  2.  2.  1.  1.  2.  1.  1.  2.  1.  1.  2.  1.
  1.  2.  1.  1.  1.  2.  1.  1.  2.  2.  2.  1.  2.  2.  1.  1.  1.  2.
  2.  2.  2.  1.  1.  2.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  2.  2.
  1.  1.  2.  2.  2.  2.  2.  2.  2.  1.  2.  1.  2.  1.  1.  2.  1.  2.
  2.  2.  2.  1.  1.  2.  2.  1.  1.  2.  1.  2.  1.  1.  1.  2.  1.  1.
  2.  2.  2.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  1.  1.  1.  1.  2.
  2.  1.  2.  1.  2.  1.  1.  1.  1.  1.  1.  1.  1.  1.  2.  2.  2.  2.
  2.  1.  2.  1.  2.  2.  2.  2.  1.  2.  2.  2.  2.  2.  2.  2.  2.  1.
  2.  2.  1.  2.  1.  1.  1.  2.  1.  2.  2.  2.  1.  2.  2.  1.  2.  2.
  1.  2.  1.  2.  2.  1.  1.  2.  2.  2.  1.  1.  1.  1.  2.  2.  1.  1.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.  1.  2.  1.
  2.  1.  2.  1.  2.  2.  2.  2.  2.  2.  1.  2.  1.  2.  2.  1.  2.  2.
  2.  1.  1.  2.  1.  2.  2.  1.  1.  2.  2.  2.  2.  2.  2.  2.  2.  1.
  2.  2.  2.  1.  2.  1.  1.  2.  2.  1.  1.  1.  2.  1.  1.  2.  1.  1.
  1.  2.  1.  2.  1.  1.  1.  1.  2.  1.  2.  2.  2.  2.  1.  2.  1.  1.
  1.  1.  1.  1.  2.  1.  2.  1.  2.  1.  2.  2.  2.  1.  2.  1.  2.  1.
  1.  2.  2.  2.  1.  1.  1.  2.  2.  1.  2.  1.  2.  2.  1.  2.  1.  1.
  2.  2.  2.  1.  1.  1.  2.  2.  1.  1.  1.  1.  2.  2.  1.  2.  2.  1.
  1.  1.  2.  1.  1.  1.  2.  2.  2.  2.  1.  2.  2.  2.  1.  2.  2.  2.
  2.  1.  2.  2.  1.  2.  2.  1.  1.  2.  2.  2.  1.  1.  1.  1.  1.  1.
  2.  1.  1.  2.  1.  1.  1.  2.  1.  2.  1.  2.  2.  1.  1.  1.  1.  1.
  1.  1.  1.  2.  1.  2.  1.  1.  1.  2.  1.  1.]
Test examples:
```

```
Test examples:
 [[ -1.82     5.98  ]  [  6.81    12.7   ]  [ -1.8     13.    ]  [  1.71     9.25  ]  [ -0.698   10.1   ]
 [ -2.93     0.292 ]  [ -6.27     0.73  ]  [ -1.69     0.819 ]  [ -2.86     6.31  ]  [ -5.68     6.88  ]
 [ -1.89     5.49  ]  [ 10.2     -1.88  ]  [  1.36     8.38  ]  [ -0.489   10.5   ]  [ -0.172   -1.92  ]
 [ -7.79     8.01  ]  [ -0.0938   4.8   ]  [  9.93    -3.78  ]  [ 11.4      4.84  ]  [ -2.85    11.2   ]
 [  5.84     0.939 ]  [  2.81     3.63  ]  [  4.28    -1.1   ]  [  5.65     5.8   ]  [ -1.82     3.75  ]
 [  6.11     6.03  ]  [ -4.39     4.2   ]  [  1.84     6.52  ]  [  8.68     6.58  ]  [  1.44     8.39  ]
 [ -8.03     8.04  ]  [  8.03    -1.96  ]  [ -4.97    -4.22  ]  [ -3.65    -9.33  ]  [  7.24     2.69  ]
 [  5.69     3.87  ]  [  8.6     -6.63  ]  [ -3.23     4.94  ]  [  7.53    -7.5   ]  [ -3.47     1.48  ]
 [  5.23     5.62  ]  [ -5.12    11.1   ]  [  4.56     4.94  ]  [ -2.56    -9.64  ]  [ -6.42     5.95  ]
 [  3.32     7.98  ]  [ -0.811   -7.24  ]  [ -2.1      7.53  ]  [  7.39     2.97  ]  [  8.2      6.62  ]
 [ -7.51     5.19  ]  [  0.506   -7.01  ]  [ -7.52    -2.26  ]  [ -0.55     0.224 ]  [  3.63    -1.77  ]
 [  9.61     5.55  ]  [ -2.47    10.2   ]  [  4.65     2.79  ]  [  1.26   -10.3   ]  [  2.79    -3.55  ]
 [  2.17    13.2   ]  [  7.36     6.21  ]  [  8.62     8.46  ]  [  5.06    -6.23  ]  [  3.89    -5.89  ]
 [ -5.95    -0.565 ]  [  4.       6.06  ]  [ -9.25     2.08  ]  [  6.83    -6.14  ]  [  8.79     4.11  ]
 [  4.3     12.6   ]  [ 10.4     -5.46  ]  [  0.928   13.9   ]  [  3.74     6.5   ]  [ -0.853   -2.06  ]
 [  5.84     6.84  ]  [ -4.01    -7.49  ]  [  8.15     7.27  ]  [ -0.885   12.2   ]  [ -4.87     4.3   ]
 [  1.99     8.51  ]  [ -5.71     6.63  ]  [  5.29    11.3   ]  [  2.12     8.24  ]  [  3.06    -3.39  ]
 [  5.86    -5.68  ]  [  5.75    -3.82  ]  [  0.198    0.0244]  [ -3.93    -8.8   ]  [ -6.35    -1.91  ]
 [ -8.04     0.758 ]  [  3.87     8.7   ]  [  0.601   12.2   ]  [  9.99    -2.75  ]  [ -1.27     3.3   ]
 [ -0.147   -4.57  ]  [  9.55     5.55  ]  [  5.36    11.8   ]  [ -7.37     5.86  ]  [ -1.39     7.6   ]
 [  6.06    11.6   ]  [  5.03    13.8   ]  [ -4.48    -0.569 ]  [  5.13    -4.49  ]  [ 10.2      2.48  ]
 [  4.52    10.2   ]  [  6.11    -7.38  ]  [  3.58    -5.45  ]  [ -3.13    10.9   ]  [  9.02     3.59  ]
 [ -2.16     9.23  ]  [ -7.83    -1.8   ]  [ -0.049   12.2   ]  [ -0.844    4.79  ]  [ -4.65     7.39  ]
 [ -9.04     2.73  ]  [ -2.77    -3.54  ]  [  2.03     8.91  ]  [  5.36     4.03  ]  [  6.12    -0.601 ]
 [  2.19    -9.3   ]  [  4.81     5.14  ]  [  4.43     1.87  ]  [ -3.61     0.961 ]  [  0.403   10.4   ]]
 [ -4.05     5.34  ]  [ 11.6      2.61  ]  [  7.33    -7.96  ]  [ -5.84     3.15  ]
 [ -2.59     8.33  ]  [  2.57     9.74  ]  [  0.579  -10.7   ]  [  9.8      0.0154]
 [ -2.41     1.3   ]  [ 10.2      1.1   ]  [ -3.13     3.85  ]  [  0.326   -4.56  ]
 [ -3.07     3.46  ]  [  1.04    -9.61  ]  [ 10.8     -2.89  ]  [ -5.19     3.98  ]
 [ -6.86    -1.33  ]  [  6.77    10.8   ]  [  1.36    -9.07  ]  [ -6.79     2.79  ]
 [  3.77    -7.27  ]  [ -0.987   13.6   ]  [ -7.06     0.184 ]  [ -2.24   -10.2   ]
 [ -5.81     4.85  ]  [  4.81     4.69  ]  [  2.19    -4.    ]  [ -2.39    -7.9   ]
 [  7.03     2.19  ]  [ -2.19    -6.73  ]  [  6.49     5.28  ]  [  7.2     11.6   ]
 [  9.29     5.31  ]  [  5.17     8.67  ]  [  3.81    13.2   ]  [ -3.07    -5.37  ]
 [  9.66     2.74  ]  [ 12.       1.69  ]  [ -0.791    0.818 ]  [ -7.88     5.05  ]
 [  4.81    -1.41  ]  [  8.5      5.87  ]  [  2.2     -8.25  ]  [  6.11     4.87  ]
 [  5.73    -3.03  ]  [  2.62     5.35  ]  [ -6.54     5.04  ]  [  8.83     4.1   ]
 [  4.9     -5.26  ]  [ -5.82     3.77  ]  [ 10.2      1.82  ]  [  7.33    -6.26  ]
 [  2.46    -6.67  ]  [ -0.0517   4.68  ]  [  6.93    -1.15  ]  [ -1.51    -6.65  ]
 [  2.5     10.8   ]  [  4.91    -8.52  ]  [  3.81     1.94  ]  [  2.67    10.4   ]
 [ -2.48     5.28  ]  [ -3.28    10.3   ]  [ -2.37    -7.43  ]  [  7.74    -0.682 ]
 [ -3.38     3.24  ]  [ 11.3     -1.36  ]  [  8.96    -1.66  ]  [ -1.18    -2.64  ]
 [  2.39    -9.14  ]  [ 10.8      0.154 ]  [ -8.71    -1.09  ]  [ 10.1      1.81  ]
 [ -1.15     6.07  ]  [  8.46     1.37  ]  [  2.8     10.5   ]  [ -5.39     6.43  ]
 [ 10.5     -1.6   ]  [ -2.47     0.877 ]  [  4.73    10.6   ]  [  5.98     3.2   ]
 [  7.1      7.77  ]  [ 11.2     -1.62  ]  [  5.67    -4.65  ]  [ -6.55     8.17  ]
 [  8.12    -3.7   ]  [ -1.61     2.94  ]  [ -5.93     0.481 ]  [  1.03     7.66  ]
 [  6.03    -2.02  ]  [ 10.1      0.0597]  [ -0.0173  -7.73  ]  [ -6.39     6.55  ]
 [  7.15    -6.37  ]  [ -4.27    11.2   ]  [ -3.41    11.    ]  [  8.79    -4.17  ]
 [ -0.546   -6.14  ]  [  6.04    13.6   ]  [ -8.91     0.715 ]  [  1.33     5.88  ]
 [ -1.47    -5.32  ]  [ -4.88     8.77  ]  [  1.67    13.4   ]  [ -8.3     -1.73  ]
 [ -7.55    -0.155 ]  [  7.58     3.19  ]  [  8.96    -2.47  ]  [ 10.7      1.79  ]
 [ 11.4      3.88  ]  [ -0.473   -3.39  ]  [  5.31    -5.5   ]  [ -0.102    1.5   ]
 [  1.3      7.12  ]  [  5.15    11.1   ]  [  6.35     1.96  ]  [ -2.21    -0.0332]
 [ -2.98     8.12  ]  [ -7.75     6.62  ]  [  4.44     8.92  ]  [ -4.38    -1.29  ]
 [ -4.32     6.86  ]  [ -6.44     5.45  ]  [  2.21     5.13  ]  [  4.64    12.9   ]
 [  6.81    12.7   ]  [ -1.8     13.    ]  [ -2.82    -8.72  ]  [ -0.698   10.1   ]
                     [ -1.69     0.819 ]  [  1.71     9.25  ]  [ -5.68     6.88  ]
```

```
Test labels:
 [ 2.  2.  2.  2.  1.  2.  2.  2.  2.  1.  2.  1.  2.  2.  2.  2.  1.  2.
  2.  1.  2.  1.  1.  2.  1.  1.  1.  2.  2.  2.  1.  1.  2.  1.  1.  1.
  2.  2.  2.  1.  1.  1.  1.  2.  1.  1.  2.  2.  1.  2.  2.  2.  1.  2.
  1.  1.  2.  1.  1.  2.  1.  1.  2.  1.  2.  2.  2.  2.  1.  2.  1.  1.
  2.  2.  1.  1.  2.  1.  2.  1.  2.  1.  1.  1.  1.  2.  2.  2.  2.  1.
  1.  1.  2.  1.  2.  1.  2.  1.  1.  2.  2.  1.  2.  1.  2.  2.  2.  2.
  1.  2.  2.  2.  2.  2.  1.  1.  1.  2.  2.  1.  2.  1.  2.  1.  1.  2.
  2.  1.  2.  2.  2.  2.  1.  2.  2.  1.  1.  1.  1.  1.  1.  1.  2.  1.
  2.  2.  2.  1.  2.  1.  2.  1.  2.  1.  2.  1.  1.  2.  1.  2.  2.  2.
  2.  2.  2.  2.  1.  2.  1.  1.  1.  1.  1.  2.  1.  1.  1.  1.  2.  2.
  1.  2.  1.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  1.  1.  1.  1.  1.
  2.  1.  1.  2.  2.  2.  2.  2.  1.  2.  1.  2.  1.  1.  2.  2.  2.  1.
  2.  1.  2.  2.  1.  2.  2.  1.  2.  1.  2.  1.  2.  2.  1.  2.  1.  2.
  1.  1.  1.  2.  1.  1.  1.  1.  2.  2.  1.  1.  1.  1.  1.  1.]

After classification:
KNN with k = 1  classifier score:        0.924
Gaussian Naive Bayes classifier score:   0.628
```

In these screenshots, we can see the results of the *train_rest_split* utility, which, as I explain later, randomly splits the original dataset into test and train partitions. The training examples, seen abbreviated in the first screenshot, total to 75% (by default) of the samples from the dataset. The classification seen above is the mean accuracy of the fit after this test data is compared against the classifier trained with aforementioned training data.

To run the K Nearest Neighbor and Naïve Bayes classifiers on the datasets, I chose to implement a solution in Python's Scikit tool. The dataset must first be separated into a list of feature samples (examples) and labels. In Question 1, the *train_test_split* utility from the model_selection Scikit class was used. This randomly selects 25% of the sample data for a test set, 75% for a training set, then returns the samples in lists. Once obtained, the KNeighborsClassifier and GaussianNB classifiers from the *neighbors* and *naive_bayes* classes were used to fit the training data. The output, determined by sklearn's *score* function, determines the mean accuracy in comparison with the test data.

In Question 2, 10 fold cross validation is instead used to perform estimates on the fit for both classifiers. As opposed to directly dividing the dataset into test and train partitions of fixed size, 10 fold (or k fold) cross validation considers every possible dataset partition of size k (in percentage). Each partition will take a turn being the test set, with the remainder being the training set. For reaching partitioning, the efficiency of the classifier is calculated, with the overall accuracy becoming the average efficiency across every distinct partitioning. This technique allows the *entire dataset* to be used in training and testing.

```
------------------------------
twogaussians.csv
------------------------------
K Nearest Neighbor Classifier with k = 1
Accuracy:        0.9675
Sensitivity:     0.969072164948
Specificity:     0.966019417476
PPV:             0.964102564103
NPV:             0.970731707317

Naive Bayes Gaussian Classifier
Accuracy:        0.9725
Sensitivity:     0.984536082474
Specificity:     0.961165048544
PPV:             0.959798994975
NPV:             0.985074626866


------------------------------
halfkernel.csv
------------------------------
K Nearest Neighbor Classifier with k = 1
Accuracy:        1.0
Sensitivity:     1.0
Specificity:     1.0
PPV:             1.0
NPV:             1.0

Naive Bayes Gaussian Classifier
Accuracy:        0.943
Sensitivity:     0.924
Specificity:     0.962
PPV:             0.960498960499
NPV:             0.926782273603
```

```
------------------------------
twospirals.csv
------------------------------
K Nearest Neighbor Classifier with k = 1
Accuracy:        0.94
Sensitivity:     0.94
Specificity:     0.94
PPV:             0.94
NPV:             0.94

Naive Bayes Gaussian Classifier
Accuracy:        0.637
Sensitivity:     0.656
Specificity:     0.618
PPV:             0.631984585742
NPV:             0.642411642412


------------------------------
clusterincluster.csv
------------------------------
K Nearest Neighbor Classifier with k = 1
Accuracy:        1.0
Sensitivity:     1.0
Specificity:     1.0
PPV:             1.0
NPV:             1.0

Naive Bayes Gaussian Classifier
Accuracy:        1.0
Sensitivity:     1.0
Specificity:     1.0
PPV:             1.0
NPV:             1.0
```

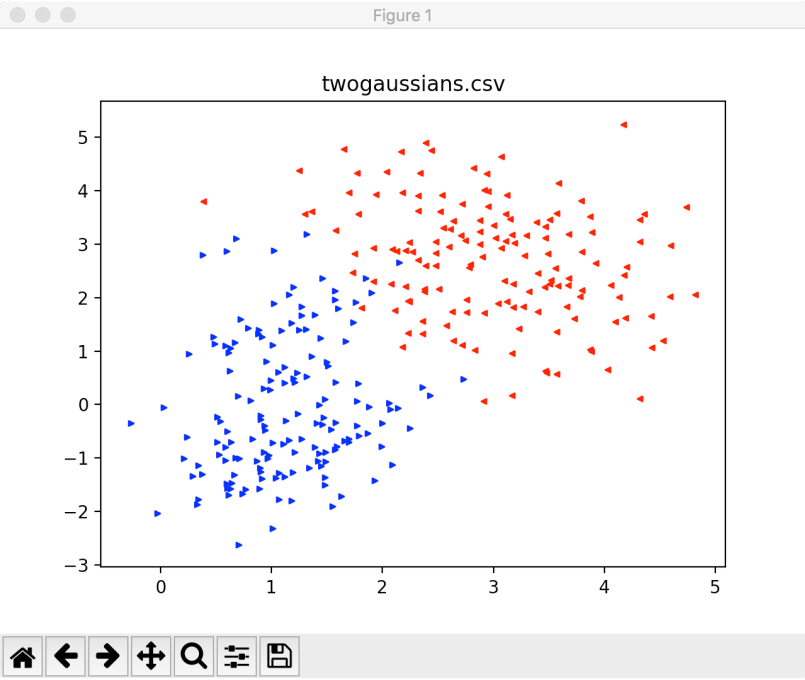
Using the 10-fold cross validation evaluation technique, the optimal values of *k* for the K Nearest Neighbor classifier are 19 for the "Two Gaussians" dataset, 1 for "Half Kernel", 9 for "Two Spirals" and 1 for "Cluster in Cluster". To determine the 'best' value of *k*, this conclusion has been reached using *accuracy* as the comparison metric. The accuracy is determined using statistics retrieved from the 10-fold cross validation, obtained using Scikit's confusion matrix implementation. The accuracy is defined as the proportion of successful predictions made, in this example, by the *K Nearest Neighbor* classifier.

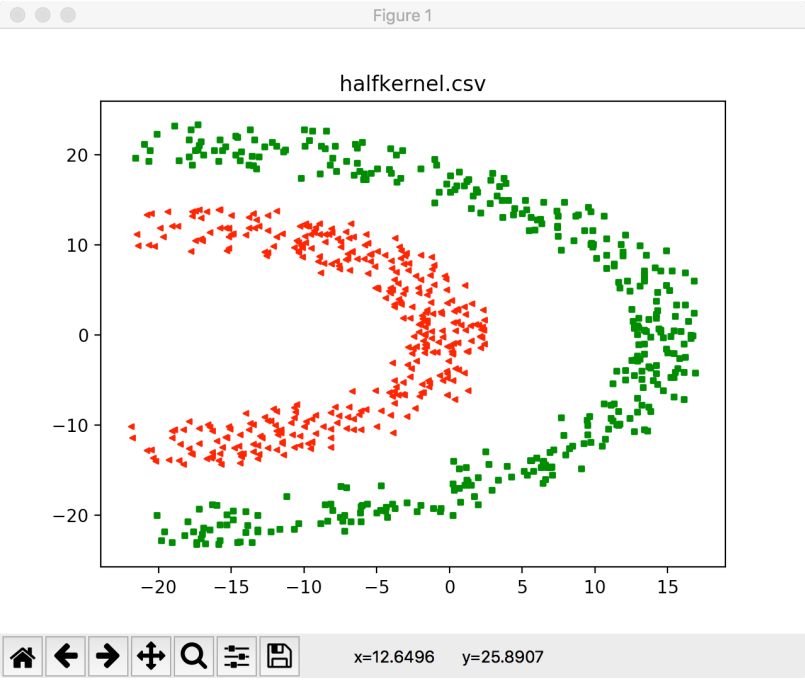
```
------------------------------
twogaussians.csv
------------------------------
Best k for KNN classifier on twogaussians.csv is 19 with 0.975
The accuracy of KNN with k=1 is 0.9675
The accuracy of Naive Bayes is 0.9725

------------------------------
halfkernel.csv
------------------------------
Best k for KNN classifier on halfkernel.csv is 1 with 1.0
The accuracy of KNN with k=1 is 1.0
The accuracy of Naive Bayes is 0.943

------------------------------
twospirals.csv
------------------------------
Best k for KNN classifier on twospirals.csv is 9 with 0.952
The accuracy of KNN with k=1 is 0.94
The accuracy of Naive Bayes is 0.637

------------------------------
clusterincluster.csv
------------------------------
Best k for KNN classifier on clusterincluster.csv is 1 with 1.0
The accuracy of KNN with k=1 is 1.0
The accuracy of Naive Bayes is 1.0
```

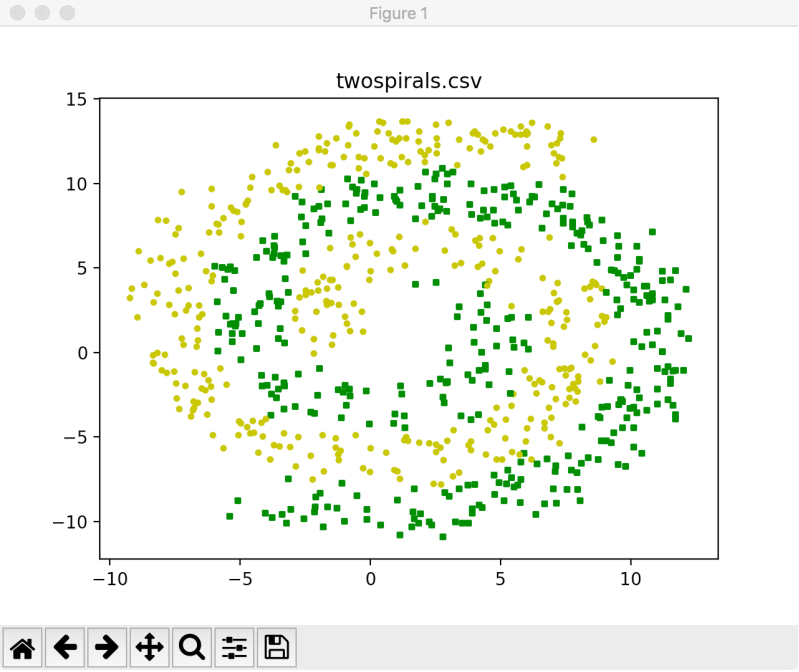
The accuracy of the K Nearest Neighbor classifier with the most optimal k consistently beats the accuracy of both the 1 Nearest Neighbor and Naïve Bayes classifiers, as seen in the screenshot below. When testing for the best k value, it becomes apparent that the accuracy of the classifier decreases as k increases (seen in the screenshot below). The best k values across the datasets are all relatively small integers. This is due to the fact that smaller numbers yield better results, and because the tie-breaking scheme employed in my program awards the best accuracy to the *smallest k across all tied values*.

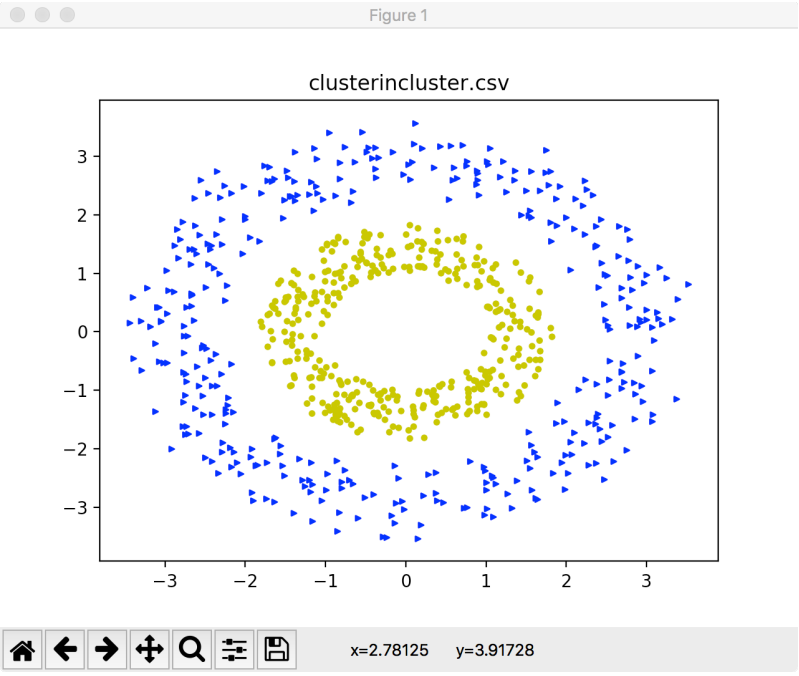
```
------------------------------
twospirals.csv
------------------------------
0.940  0.934  0.942  0.945  0.949  0.947  0.950  0.950  0.952  0.945  0.946  0.944  0.943  0.943  0.942  0.947
0.948  0.943  0.940  0.940  0.940  0.939  0.939  0.936  0.939  0.936  0.943  0.940  0.942  0.937  0.939  0.937
0.930  0.933  0.928  0.932  0.924  0.927  0.929  0.927  0.925  0.926  0.924  0.923  0.923  0.924  0.920  0.922
0.920  0.917  0.914  0.913  0.907  0.909  0.904  0.905  0.908  0.906  0.900  0.899  0.896  0.894  0.890  0.884
0.884  0.873  0.875  0.869  0.867  0.866  0.858  0.855  0.854  0.850  0.850  0.841  0.842  0.836  0.835  0.827
0.824  0.815  0.813  0.809  0.802  0.798  0.799  0.785  0.789  0.782  0.783  0.771  0.772  0.766  0.760  0.746
0.749  0.739  0.738  0.730  0.731  0.720  0.720  0.712  0.707  0.700  0.692  0.680  0.681  0.672  0.671  0.670
0.666  0.660  0.658  0.653  0.648  0.642  0.640  0.633  0.635  0.628  0.628  0.617  0.620  0.614  0.614  0.605
0.603  0.594  0.596  0.592  0.589  0.583  0.584  0.582  0.580  0.571  0.577  0.573  0.570  0.569  0.571  0.570
0.568  0.566  0.565  0.564  0.560  0.557  0.558  0.552  0.554  0.549  0.553  0.552  0.555  0.554  0.553  0.553
0.550  0.550  0.546  0.548  0.546  0.546  0.541  0.542  0.536  0.541  0.539  0.534  0.531  0.532  0.531  0.531
0.533  0.531  0.532  0.533  0.531  0.533  0.534  0.535  0.534  0.536  0.535  0.532  0.534  0.534  0.536  0.533
0.534  0.532  0.532  0.530  0.534  0.533  0.537  0.533
Best k for KNN classifier on twospirals.csv is 9 with 0.952
The accuracy of KNN with k=1 is 0.94
The accuracy of Naive Bayes is 0.637

------------------------------
clusterincluster.csv
------------------------------
1.000  1.000  1.000  1.000  1.000  0.999  1.000  0.998  0.999  0.999  1.000  1.000  1.000  1.000  1.000  1.000
1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
1.000  1.000  1.000  0.999  0.999  0.999  0.999  0.999  0.999  0.999  1.000  0.999  0.999  0.998  0.999  0.999
0.999  0.999  0.999  0.998  0.999  0.998  0.998  0.997  0.998  0.997  0.999  0.997  0.998  0.997  0.997  0.996
0.996  0.996  0.996  0.996  0.997  0.997  0.997  0.995  0.995  0.995  0.995  0.994  0.995  0.993  0.995  0.993
0.995  0.991  0.995  0.994  0.994  0.992  0.994  0.990  0.991  0.988  0.989  0.988  0.989  0.988  0.989  0.987
0.989  0.986  0.988  0.986  0.989  0.986  0.987  0.986  0.986  0.984  0.985  0.981  0.984  0.979  0.982  0.979
0.980  0.979  0.980  0.977  0.977  0.977  0.977  0.971  0.973  0.968  0.969  0.966  0.968  0.965  0.966  0.962
0.964  0.958  0.961  0.952  0.954  0.948  0.952  0.942  0.944  0.939  0.942  0.934  0.936  0.931  0.934  0.931
0.931  0.925  0.929  0.920  0.921  0.916  0.916  0.910  0.912  0.905  0.909  0.900  0.903  0.894  0.898  0.885
0.895  0.880  0.889  0.871  0.878  0.863  0.871  0.859  0.865  0.849  0.856  0.838  0.846  0.837  0.841  0.825
0.829  0.812  0.816  0.803  0.812  0.797  0.806  0.799
Best k for KNN classifier on clusterincluster.csv is 1 with 1.0
The accuracy of KNN with k=1 is 1.0
The accuracy of Naive Bayes is 1.0
```

Below are screenshots of each dataset plotted in two dimensional space using the *pyploy* utility.









Each classifier clearly has different behaviour, with certain advantages over one another. The K Nearest Neighbor classifier is not known to handle large amounts of data, nor is it particularly fast. Naïve Bayes, in comparison, is faster with a smaller memory footprint. Due to it's decision tree structure, kNN

will use up memory during decision making and will have to remember all instances of its data in order to classify new samples. These disadvantages are due to kNN being an instance-based or non-generalizing learning algorithm.

To determine which classifier would be best suited to each dataset would require analysis of the accuracy of our 10-fold cross validation in comparison to the Naïve Bayes accuracy. As seen in the output for Question 4, the K Nearest Neighbor classifier appears to outperform Naïve Bayes in terms of accuracy for k=1 and the best case k. However, we may also notice that many other measures of efficiency can be used to compare the classifiers.

Using the measures shown in the Question 2 output, it is clear that Naïve Bayes consistently outperforms Accuracy, Sensitivity and NPV in the "Two Gaussians" dataset, and ties kNN when evaluating the "Cluster in Cluster" dataset. These are two cases where Naïve Bayes outperforms, and is possibly due to the large sample size and distribution.

In contrast, the "Half Kernel" and "Two Spirals" datasets are far from meeting the various measures of efficiency of the kNN classifier. The Two Spirals dataset is much better suited to kNN due to its distribution, which as you can see in the plot, is interspersed much more than any other plot.

For these reasons, Naïve Bayes should be used to classify Two Gaussians to allow for more types of efficiency, and could arguably outperform on the Cluster in Cluster dataset due to its superior speed. The remaining two datasets are clearly better classified in all measures of efficiency by the kNN classifier. The output of Question 2 is itself a thorough analysis of the performance of both classifiers, and should be referred to for more specific details.