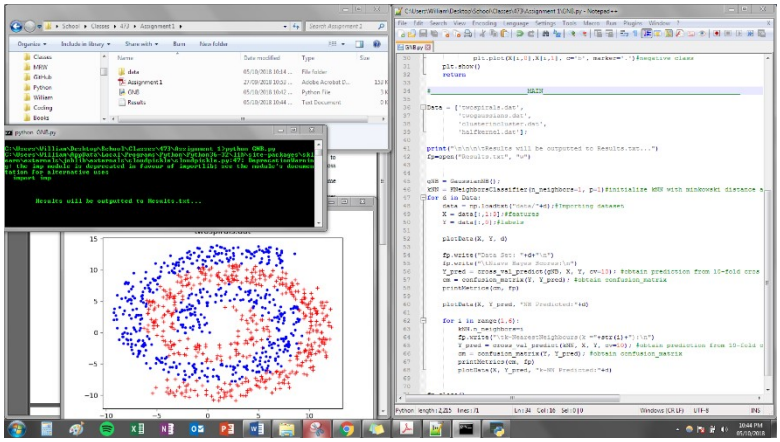
William Briguglio

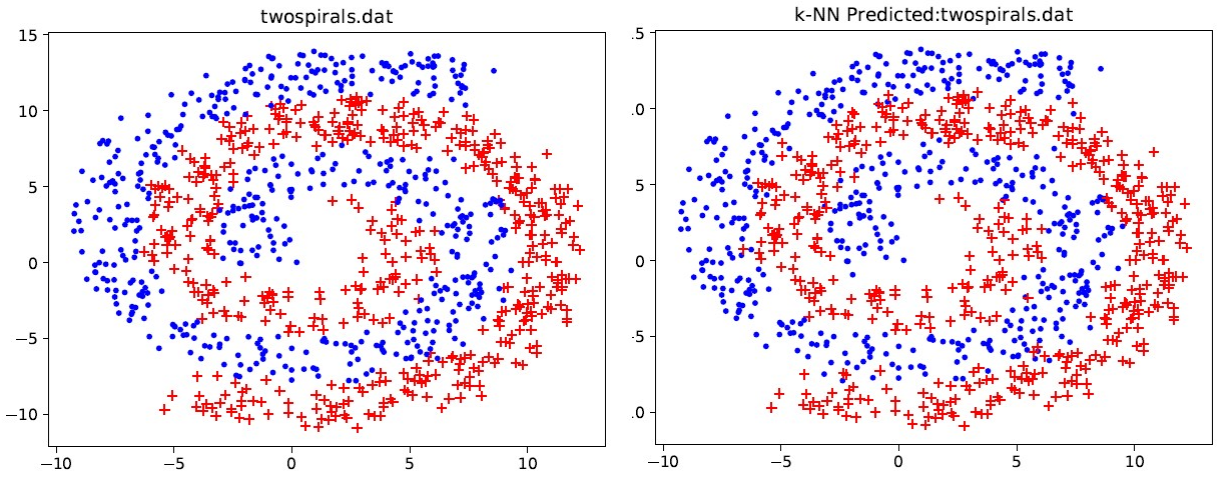
Assignment 1:

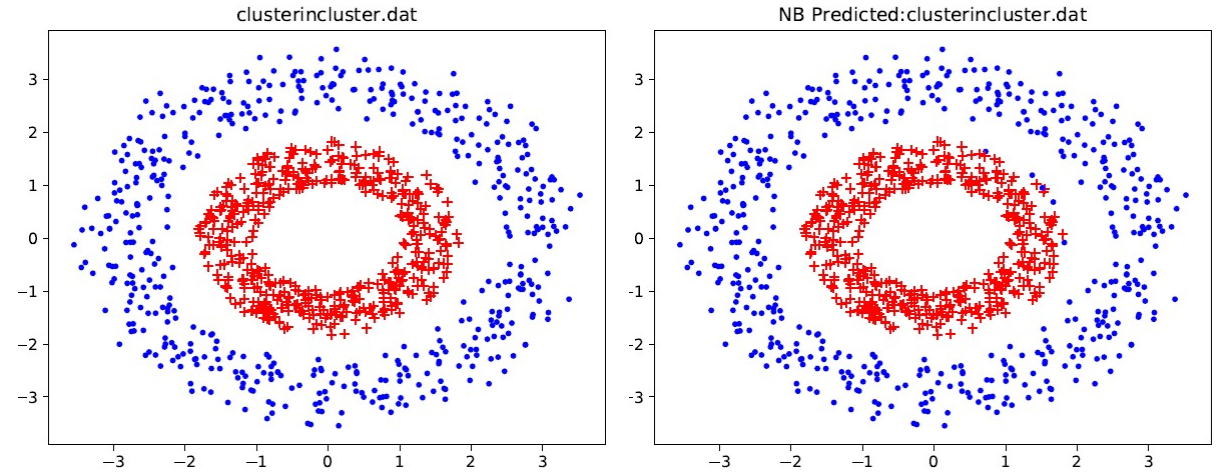
1) Using Scikit's `sklearn` library, I applied both a k-Nearest Neighbour algorithm and a Naïve Bayes algorithm to the four datasets provided. Following are screen shots of the program's execution as well as the plotted data before and after each classifier's predictions on one of the data sets.



Before Classification

After Classification





Using the `confusion_martix()` function from the `sklearn.metrics` library, I was able to obtain the confusion matrix for each algorithm's predictions and from there, easily calculate the PPV, NPV, specificity, sensitivity, and accuracy of each algorithm, outputting the results to "Results.txt". Here are the results for Naïve Bayes and 1-NN classification for each data set:

```
Data Set: twospirals.dat
       Naïve Bayes Scores:
       PPV: 0.64669  NPV: 0.63760  Specificity: 0.51246
       Sensitivity: 0.62600  Accuracy: 0.65244
       k-NearestNeighbours(k =1):
       PPV: 0.94024  NPV: 0.94378  Specificity: 0.49894
       Sensitivity: 0.94400  Accuracy: 0.94012

Data Set: twogaussians.dat
Naïve Bayes Scores:
       PPV: 0.98485  NPV: 0.94554  Specificity: 0.49482
Sensitivity: 0.94660  Accuracy: 0.95545
       k-NearestNeighbours(k =1):
       PPV: 0.96618  NPV: 0.96891  Specificity: 0.48320
       Sensitivity: 0.97087  Accuracy: 0.93705

Data Set: clusterincluster.dat
Naïve Bayes Scores:
       PPV: 1.00000  NPV: 0.99010  Specificity: 0.50251
       Sensitivity: 0.99000  Accuracy: 1.00000
       k-NearestNeighbours(k =1):
       PPV: 0.96525  NPV: 1.00000  Specificity: 0.49084
       Sensitivity: 1.00000  Accuracy: 0.96464

Data Set: halfkernel.dat
       Naïve Bayes Scores:
       PPV: 0.92678  NPV: 0.96050  Specificity: 0.48993
       Sensitivity: 0.96200  Accuracy: 0.92542
       k-NearestNeighbours(k =1):
       PPV: 1.00000  NPV: 1.00000  Specificity: 0.50000
       Sensitivity: 1.00000  Accuracy: 1.00000
```

The `cross_val_predict()` function in the `sklearn.model_selection` library implements 10fold cross validation using the provided algorithm by splitting the data set into 10 equally sized partitions. Then, the algorithm is trained on 9 partitions and tested on the remaining partition. This is repeated 10 times until each partition has been used as the test set once. The predicted labels for each test set is returned in one large matrix. Thus, a predicted label for each sample is obtained which is used to generate the confusion matrix.

To implement the k-NN algorithm with Euclidean distance function, I used the `KNeighborsClassifier()` function from the `sklearn.neighbors` library with the params

`n_neighbours` set to 1, and `p` set to 1. This means that the classifier will use the Minkowski distance with p=1 which is equivalent to the Euclidian distance, as the distance function.

```
gNB = GaussianNB();
kNN = KNeighborsClassifier(n_neighbors=1, p=1)#initialize kNN with minkowski distance and p=2 which is equivalent to the Euclidean distance
for d in Data:
    data = np.loadtxt("data/"+d);#Importing dataset
    X = data[:,1:3];#features
    Y = data[:,0];#labels

    #plotData(X, Y, d)

    fp.write("Data Set: "+d+"\n")
    fp.write("\tNaive Bayes Scores:\n")
    Y_pred = cross_val_predict(gNB, X, Y, cv=10); #obtain prediction from 10-fold cros val
    cm = confusion_matrix(Y, Y_pred); #obtain confusion_matrix
    printMetrics(cm, fp)

    #plotData(X, Y_pred, "NB Predicted:"+d)

    for i in range(1,100):
        kNN.n_neighbors=i
        fp.write("\tk-NearestNeighbours(k ="+str(i)+"):\n")
        Y_pred = cross_val_predict(kNN, X, Y, cv=10); #obtain prediction from 10-fold cros val
        cm = confusion_matrix(Y, Y_pred); #obtain confusion_matrix
        printMetrics(cm, fp)
        #plotData(X, Y_pred, "k-NN Predicted:"+d)
```

To implement the Naïve Bayes algorithm, I used the `gaussianNB()` function from the `sklearn.naive_bayes` library.

```
gNB = GaussianNB();
kNN = KNeighborsClassifier(n_neighbors=1, p=1)#initialize kNN with minkowski distance and p=2 which is equivalent to the Euclidean distance
for d in Data:
    data = np.loadtxt("data/"+d);#Importing dataset
    X = data[:,1:3];#features
    Y = data[:,0];#labels

    #plotData(X, Y, d)

    fp.write("Data Set: "+d+"\n")
    fp.write("\tNaive Bayes Scores:\n")
    Y_pred = cross_val_predict(gNB, X, Y, cv=10); #obtain prediction from 10-fold cros val
    cm = confusion_matrix(Y, Y_pred); #obtain confusion_matrix
    printMetrics(cm, fp)
```

2) When two classes are given, the only time we need to consider a tie-resolution scheme is when k is an even value, since odd values would never have a draw in the number of nearest neighbours belongs to each class. However, the chance of a tie occurring was relatively low and most of the even valued k-NN predictions were just as accurate.

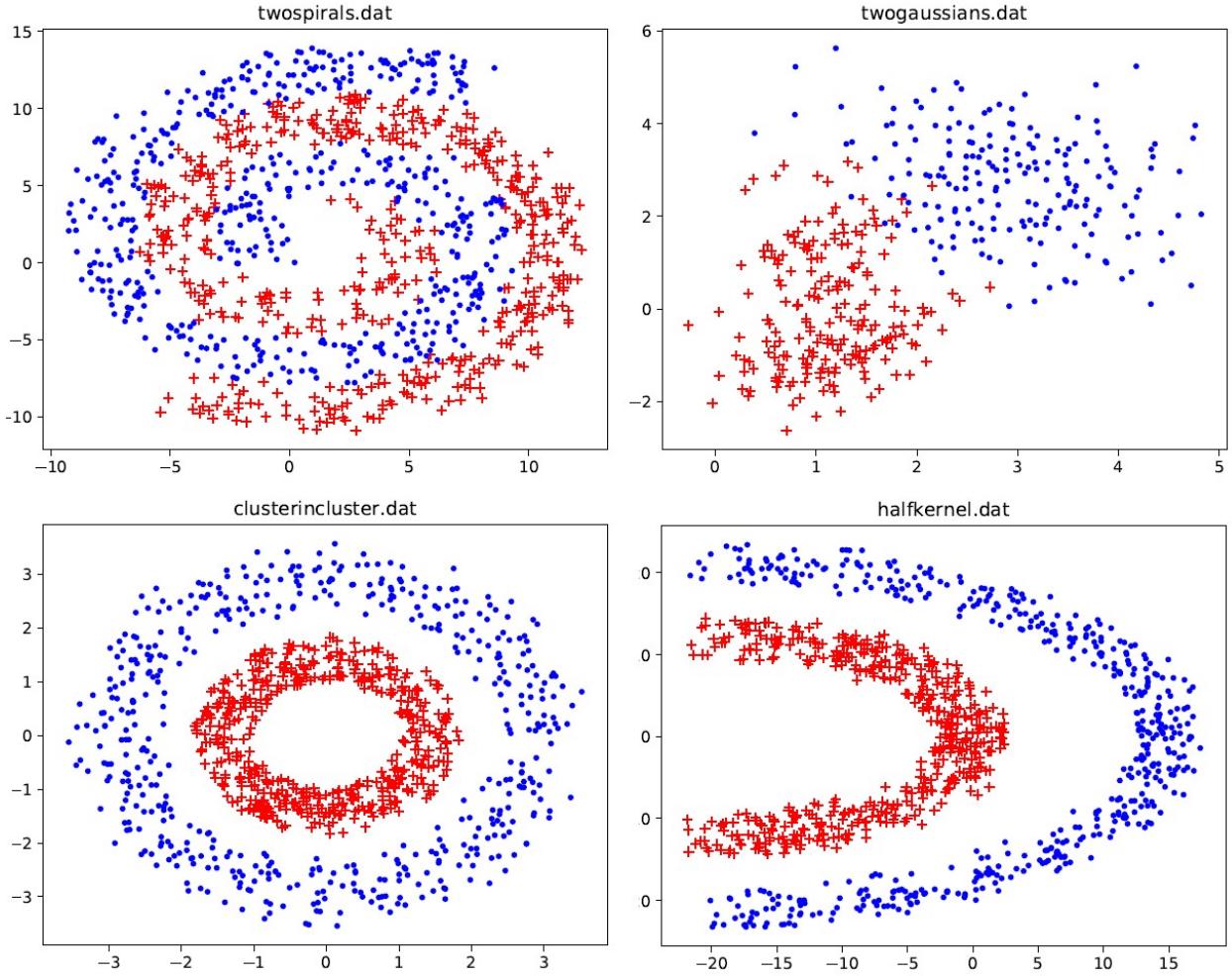
3) For k-NN, the best value for k varies with each dataset and k-NN was not always better or worse then the Naïve Bayes algorithm, because of the this, it only makes sense to say a particular value of k is "best" for a particular dataset. The best value of k is also dependent on your dataset. For example, a data set that is extremely unbalanced would cause even a random classifier to have a very high accuracy. However, because our data sets are all balanced (i.e.$|\omega\omega_1| \cong |\omega\omega_2|$), I will consider accuracy as the primary measure of performance.

Two Spirals) For this data set, the k-NN algorithm preformed far better then the Naïve Bayes algorithm, in fact the Naïve Bayes algorithm was only 14.2% more accurate then a random classifier. The best value for k in terms of accuracy was k=7 with an accuracy of 95.5%. Two Gaussians) For this data set, the two classifiers preformed close to the same but the kNN algorithm was marginally better. Naïve Bayes had an accuracy of 96.5% and the best value for k, k=23, had an accuracy of 97.75%.

Cluster in Cluster) For this data set, Naïve Bayes outperformed k-NN with a 99.5% accuracy and the best value for k, k=1, having a 98.2% accuracy

Half Kernel) Finally, for this data set, although Naïve Bayes preformed well with a 94.3% accuracy, the k-NN algorithm managed a 100% accuracy for all k values in the range 1 to 99 Another thing to note is the best k values were odd for all of the datasets, this is expected due to the fact that there are two classes and an odd k value makes it impossible for a tie to occur and thus impossible for a sample to be arbitrarily assigned to the wrong class.

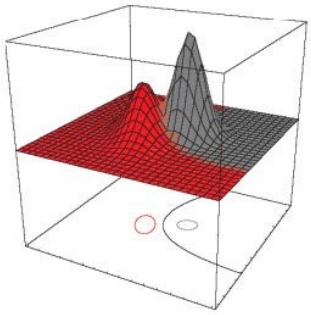
4) Following are the plots of each data set, with a red cross denoting a positive sample(y=1) and a blue circle denoting a negative sample(y=2).



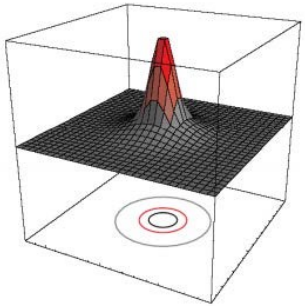
5)

Two Spirals) The Naïve Bayes algorithm preformed poorly for this data set with an accuracy of 64.2%. This is because the data is not gaussian at all, and Naïve Bayes assumes gaussian distribution. Intuitively, the Naïve Bayes algorithm fits gaussian distributions to the data set. If there is no transformation(i.e. making the bell shape taller/shorter or making it wider/narrower in the x_1 or x_2 dimension or rotating it about it's centre) of the gaussian distribution that roughly resembles the dataset, the Naïve Bayes cannot fit the data well, hence the poor performance. The k-NN classifier preforms much better and this is because there are two distinct regions of positive and negative samples meaning that most samples are surrounded by samples of a similar class. In my opinion, the k-NN classifier is better for this data set given its' far superior performance.

Two Gaussians)The Naïve Bayes algorithm preformed much better with this data set as this data set has two regions of samples with one having the positive and the other having the negative class. This "shape" of data can easily be modeled with gaussian distributions and so Naïve Bayes preforms well. One can imagine two bell curve superimposed on these regions where the peak of highest probability is above the centre of each region, and as you approach the other region, the probability that a sample belongs to a certain class drops(similar to the image from Dr. Rueda's slides below). The k-NN classifier also preformed well for the same reason as in the two spirals data set. In my opinion the better classifier for this data set is still the k-NN classifier as it had slightly higher performance measures, although both are very good.



Cluster in Cluster)The Naïve Bayes algorithm preformed well here as well, this is because although one region lies within the other, each region is still somewhat shaped like a gaussian distribution. In this case we can imagine a tall narrow bell curve in the centre superimposed on a wider but shorter bell curve. Meaning that the probability of a sample belonging to the positive class is very high in the centre of the data cluster, but drops quickly as we move outward, dropping to below the probability of the negative class as we move out towards the edges. (similar to the image from Dr. Rueda's slides below). The k-NN algorithm preformed well here too, once again for the same reasons as in the case of the two spirals. In this case the Naïve Bayes had higher accuracy then the k-NN and so in my opinion this classifier is better for this data set.



Half Kernel) Here, the Naïve Bayes algorithm preformed well for the same reasons as with the cluster in cluster data set, only in this case, half of each bell curve is cut off. The k-NN algorithm preformed well here too, for the same reasons as the other three data sets. In this case the k-NN had higher accuracy then the Naïve Bayes and so in my opinion this classifier is better for this data set as well.