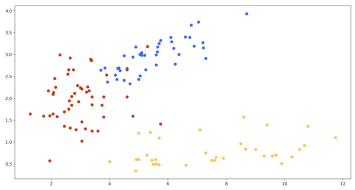
Symbols, Patterns and Signals coursework 2 Report

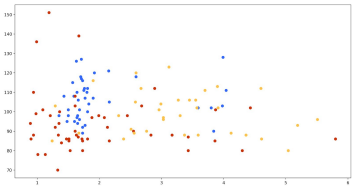
Tyler Ward (tw17530) and Andy Thorn(at17556)

4.1 Feature Selection

The first problem we encountered was that displaying 13 x 13 features makes graphs very small so it's hard to see the distribution to pick a good pair. To help with these, we reduced the size of the markers on the graph which made it a lot clearer which graphs were better suited for the classifier. The features we chose are features 10 and 7. We also considered feature 1 and 7.



With feature 10 (x-axis) and 7 (y-axis) there is not a lot of crossover between different flower types so it should give a more accurate classifier as shown in the figure on the left.



Some of the classifier combinations have a lot of crossover and so it is obvious they wouldn't make a good classifier such as feature 2 with feature 5 shown in the second figure on the left (far from defined classification boundaries).

4.2 K-Nearest Neighbours

Brief explanation of K-Nearest Neighbours:
The K-Nearest Neighbours algorithm is is classification algorithm that assigns a test instance to the majority class among its k nearest neighbours. To classify a test instance, draw the smallest hypersphere around it that contains the k neighbours; then assign majority class among this. Each of the k nearest neighbours contribute equally to the vote for the most likely class of the test instance.

Using the standard classification metrics accuracy to evaluate our classifier on the test set:
k = 1

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 1
Accuracy:  0.9056603773584906
[2, 1, 1, 1, 1, 3, 3, 3, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 3, 1, 1, 2, 1, 2, 3, 2, 1, 3, 2, 1, 2, 1, 2, 3, 2, 1, 3, 3, 3,
 2, 1, 3, 3, 1, 2, 1, 2, 1, 3, 2, 1, 1]
```

k = 2

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 2
Accuracy:  0.8679245283018868
```

k = 3

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 3
Accuracy:  0.8490566037735849
```

k = 4

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 4
Accuracy:  0.8679245283018868
```

k = 5

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 5
Accuracy:  0.7924528301886793
```

k = 6

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 6
Accuracy:  0.8490566037735849
```

k = 7

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 7
Accuracy:  0.8301886792452831
```

Average error with k = 1 to 7 is 0.85175

The k that resulted in the highest accuracy was k = 1; however using a low value of k within your classifier is nonoptimal as the effect of noise on the class prediction is greatly increased. In general, increasing your value of k can reduce this effect of noise however it can also consequently make the classification boundaries less distinct as samples of other classes start to get counted (underfitting - if k rises too high all test points will belong to the same class - the majority class). This can explain why our classifier accuracy started decreasing as the value of k rose too high (started decreasing after k = 4). Looking at our results, a good balance between reducing the effect of noise and not underfitting would be k = 4. This is because it had the highest value of k (good noise reduction) where the accuracy was still also relatively high (classification boundaries were still distinct).

Confusion matrix:
A confusion matrix is another method of analyzing the performance of a classification algorithm. Calculating a confusion matrix can be preferred over traditional classification accuracy as it can allow you to gain a deeper understanding about the performance of a classifier as it gives you a more detailed analysis by identifying potential issues with specific classes.

For the confusion matrix, we decided to calculate it with k = 4 as we concluded that this was the optimal k for this particular dataset (discussed in the paragraph above).

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn --k 4
Accuracy:  0.8679245283018868
Confusion matrix:
 [[1.         0.         0.        ]
 [0.23809524 0.76190476 0.        ]
 [0.         0.14285714 0.85714286]]
[1, 1, 1, 1, 1, 3, 3, 3, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 3, 1, 1, 2, 1, 2, 3, 2, 1, 3, 2, 1, 2, 1, 2, 3, 2, 1, 3, 3, 2,
 2, 1, 3, 3, 1, 2, 1, 2, 1, 3, 2, 1, 1]
```

Row 1 represents class 1; Row 2 represents class 2; Row 3 represents class 3.

As you can see from the diagram, our classification algorithm excels in classifying test instances that belong to class 1. With samples that belong in class 1, the algorithm was able to predict that it belonged to class 1 with a 100% record. This can be seen by the ‘1’ in the top left corner of the matrix. With the classification of samples of class 2 the classifier was less successful. As you can see from the matrix, the classifier sometimes classified them as class 1 with a error rate of 23.8%. Finally, with the classification of samples that belong to class 3, the performance of the classifier was a lot better however not perfect. The classifier sometimes misidentified samples as being class 2 instead of class 3; and it made this error with a percentage of 14.3%. The reason for these errors may be caused by a large amount of noise within the dataset, or because the classification boundaries of class 2 and 3 not distinct enough.

To summarise, the classifier performs very well among all 3 classes in general, however it was most successful in classifying test instances that belonged to class 1 (100% accuracy), followed by classifying samples of class 3 (85.7% accuracy), and finally classifying samples of class 2 (76.2% accuracy).

4.3 Alternative Classifier
The classifier we opted to use is the Naive Bayes classifier.

As shown in the figure below, the accuracy of the Naive Bayes classifier is less than that of the k-nn classifier average accuracy for k = 1-7. The figure also shows the counts of each class. As you can see, the counts are quite varied. In a Naive Bayes classifier, if the training set isn't representative of the overall population, it won't be able to correctly classify other data and so could possibly be a cause of the inaccuracy show.

```
Count 1: 41
Count 2: 50
Count 3: 34
Accuracy:  0.8301886792452831
```

A lot of the time with Naive Bayes classifier inaccuracy, the problems are due to the data used. To get a better accuracy for this classifier we could consider using different features.

4.4 Use Three Features
For the third feature, we decided to pick feature 1 (on top of the current 2 features we use - 7 & 10) as plotting feature 1 against feature 7 gave us very good separation among the data points. In fact, plotting feature 1 against 7 was our second best alternative to the current two features we plot currently.

Using the standard metrics accuracy to evaluate our classifier on the test set:
k = 1

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 1
Accuracy:  0.9245283018867925
```

k = 2

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 2
Accuracy:  0.9056603773584906
```

k = 3

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 3
Accuracy:  0.9245283018867925
```

k = 4

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 4
Accuracy:  0.8867924528301887
```

k = 5

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 5
Accuracy:  0.8867924528301887
```

k = 6

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 6
Accuracy:  0.8679245283018868
```

k = 7

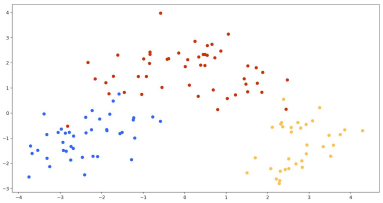
```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_3d --k 7
Accuracy:  0.8490566037735849
```

Average error with k = 1 to 7 is 0.89218

As you can see from the accuracy results, increasing the number of features used from 2 to 3 had a positive impact on the accuracy of the knn classifier. The percentage accuracy rose a total of 4% from 85.2% to 89.2%. This is as expected as increasing the number of dimensions increases the volume of space of which the sample data is plotted; thus allowing the classifier to become more accurate at differentiating the different classes.

4.5 Principal Component Analysis (PCA)

Scatter plot of the PCA-reduced training set:



Running the KNN Classifier on the PCA-transformed data:

```
C:\Users\athorn\Desktop\spscoursework\lab_sheets_public\Coursework_2>py wine_classifier.py knn_pca --k 4
Accuracy:  0.9433962264150944
[2, 1, 1, 1, 2, 3, 3, 3, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 3, 1, 1, 2, 1, 2, 3, 1, 1, 3, 2, 2, 2, 1, 2, 3, 2, 1, 3, 3, 3,
 2, 1, 3, 3, 1, 2, 2, 2, 1, 3, 2, 1, 1]
```

Average error with k = 1 to 7 is 0.95386

As you can see from the results, the performance of the KNN classifier using the PCA-transformed data produced a significantly higher accuracy rate when compared to manually selecting the features. This is most likely because the PCA-transformed data has much more distinct separations between the class boundaries, and also the amount of scatteredness within the clusters is noticeably less. This results in a higher probability of which the surrounding points around a sample are of the same class, therefore allowing the KNN algorithm to more likely correctly classify the sample.