The University of Birmingham
School of Computer Science
MSc in Advanced Computer Science

Imaging and Visualisation Systems

Visualisation Assignment

John S. Montgomery
msc37jxm@cs.bham.ac.uk

May 2, 2004

Chapter 1

Introduction

1.1 Data Description and Format

The data set chosen for visualisation was of three different varieties of wine. The data consists of a the following thirteen fields:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

All of the fields represent continuous data, which has the advantage of making normalisation much easier.

The data was contained as comma separated values in a plain text file. The first element of each row contains a number (1-3) that gives the samples class. The other fields follow after this element. Parsing was therefore straightforward and mainly involved separating out the class numbers from the rest of the data, as the class information is not, directly, used by the PCA and SOM methods.

Chapter 2

Visualisation

Together with the code for Self-Organising Maps provided for this assignment, the visualisations were made using a simple author crafted Python script (see Appendix A). The "MLab" package from Numeric Python[1] was used for calculating the covariance matrix of the data and then the eigen vectors and values for use in PCA. For actually plotting the various data points Matplotlib[2] was used.

2.1 Basic

To first get an overall feel for the data I simply plotted every pair of fields against each other. With thirteen fields this results in 78 graphs $(n*(n-1)/2)$. This may seem like a lot, but as the graphs were generated by a script is was straightforward. Also by plotting the three different classes in distinct colours it was very easy to scan thumbnail images of graphs in a directory for interesting graphs (Figure 2.1). Most of the graphs showed obvious clusters of each class. Figure 2.2 shows the alcohol content compared with the "flavanoids"[3] content. It shows that two of the classes are fairly easy to separate based purely on alcohol content and those two classes tend to have a similar flavanoids content to each other, but different from the third class. There is a certain amount of overlap, but for a very quick and easy approach this is not bad at all.

Figure 2.3 shows a similar story, but compares the flavanoids with "colour intensity". In this case the third class data points (blue triangles) are completely separate from the other two classes, with the only overlap occurring between those two (red circles and green squares).

By way of comparison Figure 2.3 shows a graph that is not as well separated. In this case two of the classes seriously overlap - to the point that they could

[1]http://www.pfdubois.com/numpy/
[2]http://matplotlib.sourceforge.net/
[3]A type of organic chemical

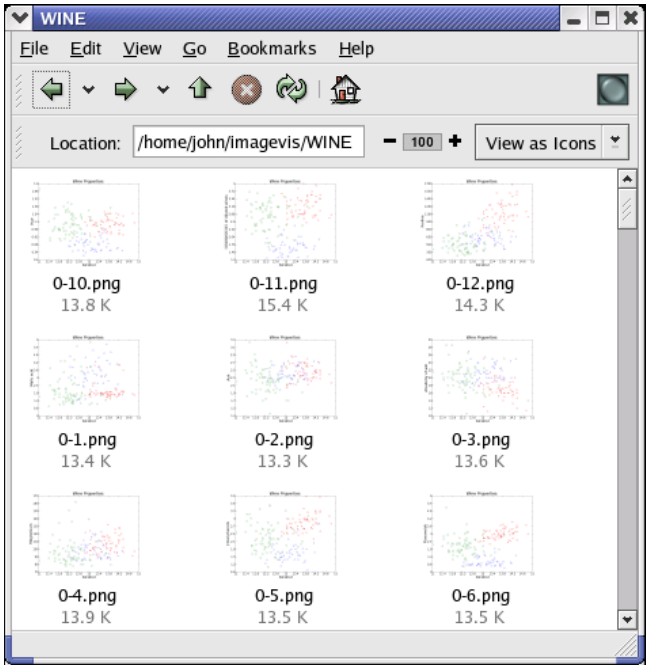

Figure 2.1: Graph Thumbnails.

almost be drawn from the same classes. It would appear that "ash" content, as used in this graph, is not a reliable indicator!

It is obvious from these simple graphs that the data is actually quite two dimensional. No one data field on its own seemed enough to separate all three classes, but two combined seems to do a pretty good job.

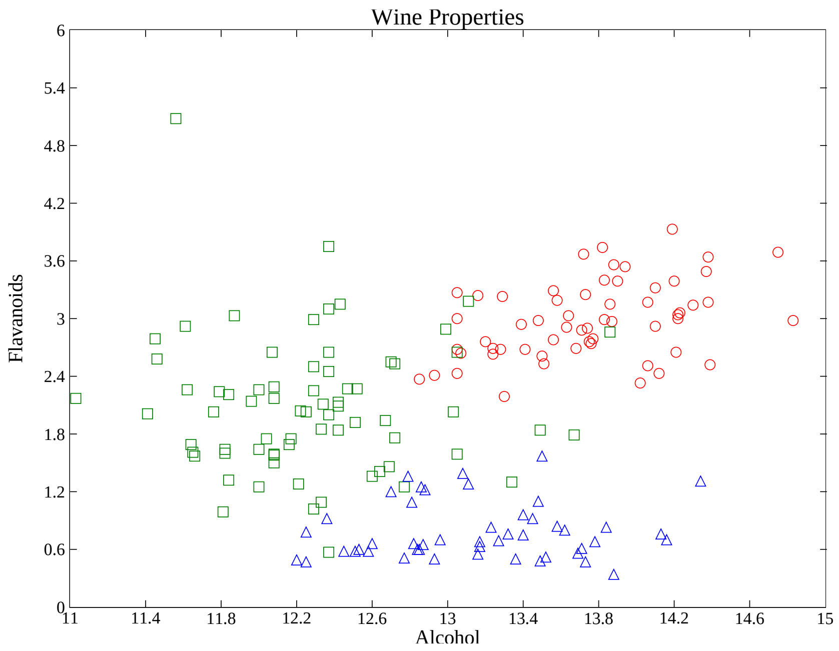

Figure 2.2: Alcohol vs. Flavanoids.

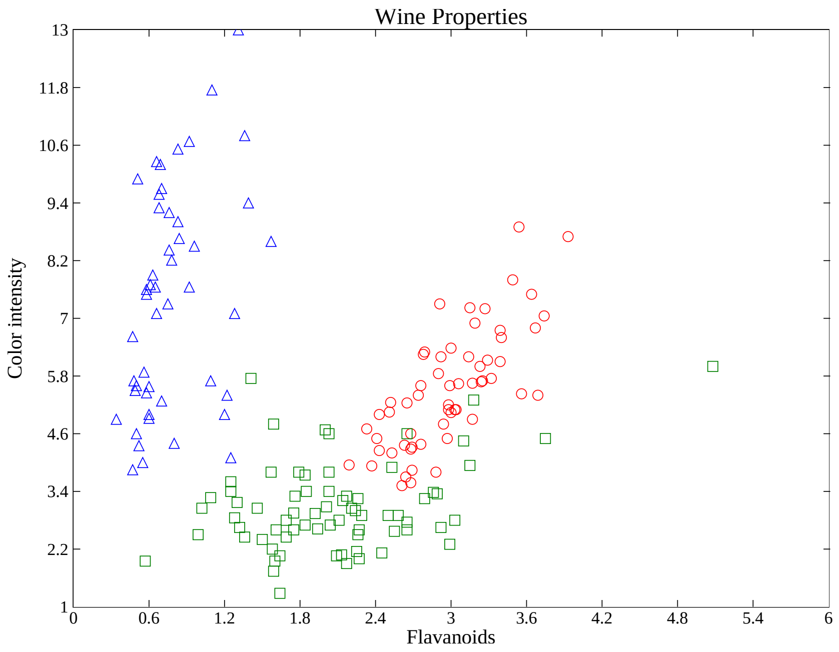

Figure 2.3: Flavanoids vs. Colour Intensity.

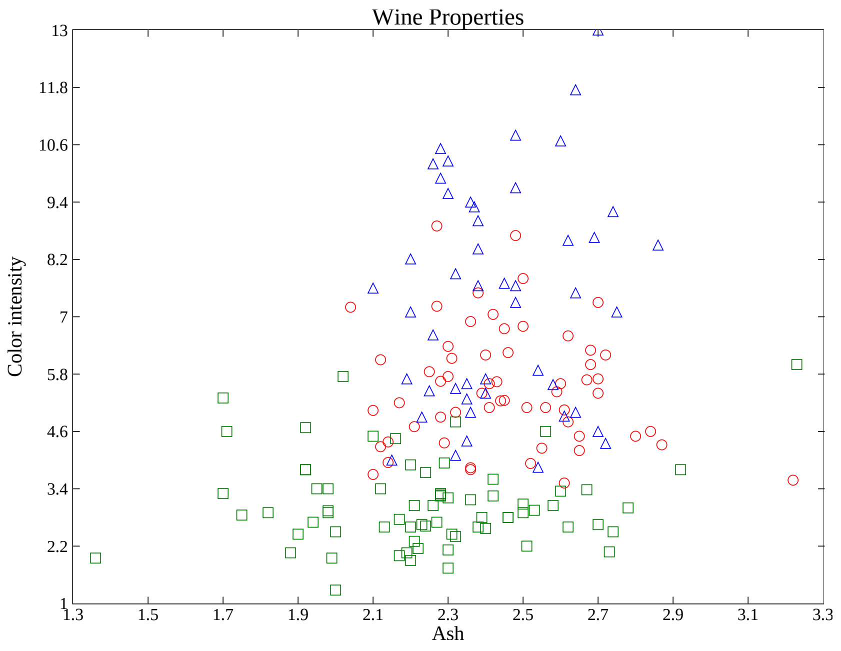

Figure 2.4: Ash vs. Colour Intensity.

2.2 Self Organising Maps

2.2.1 Preprocessing

The raw data (minus the category field) was normalised, so that all values varied in the range 0 to 1. To do this the minimum and maximum values in each field were first found. Then these values are used to rescale the data thus:

$$v_{norm} = (v - v_{min})/(v_{max} - v_{min})$$

Where v is the original value, v_{min} and v_{max} are the minimum and maximum values respectively and v_{norm} is the value after normalising.

2.2.2 Visualisation

The application code provided was used to generate a 10x10 map, as shown in Figure 2.5. The map shows the three classes have been separated very well and in fact there is only one overlapping point. The codebook vectors generated for the final map would probably be very useful for "guessing" which class a new sample of wine belongs to, by using a simple "nearest neighbour" approximation.

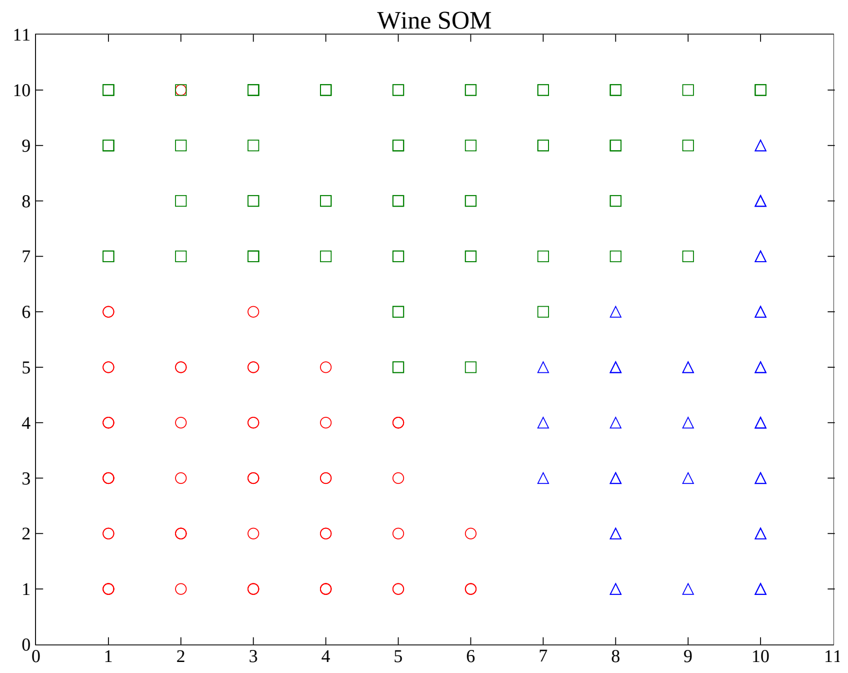

Figure 2.5: SOM.

2.3 Principle Component Analysis

2.3.1 Preprocessing

The data was normalised, as in Section 2.2.1, but the data was also centred, so that it could be used for PCA. The centring merely involved subtracting the means of each field from every element of that field.

2.3.2 Visualisation

After normalising and centring the data the next step involved computing the covariance matrix of the data and the calculating the eigen vectors and corresponding values of that matrix. The ordered eigen values are shown in 2.6. The first two eigen values are clearly, overall, the most significant. Therefore the eigen vectors used for plotting the data for the visualisation in Figure 2.7 were (rounded to 2 d.p.):

```
[ -0.55 -0.23 -0.16 0.08 -0.19 -0.07 -0.00 -0.01 -0.03 -0.52  0.24  0.22 -0.44 ]
[ -0.13  0.25 -0.00 0.18 -0.09 -0.40 -0.41  0.33 -0.25  0.09 -0.25 -0.47 -0.29 ]
```

This shows that most of the fields are at least partially useful for separating out the data. The only field which seems to be under represented slightly is field 3 (Ash), which has a low value in the first eigen vector (-0.16) and an even lower value in the second (when rounded it was 0.0). The largest fields in each vector are for alcohol (-0.55) and "OD280/OD315 of diluted wines" (-0.47) meaning that these might be quite good for classification, even without the other fields.

As it stands Figure 2.7 shows data that has been very well separated. Two of the classes are *very* well separated (red squares and blue triangles) and the only overlap that occurs is with the third class. This would seem to suggest that the data is quite two dimensional.

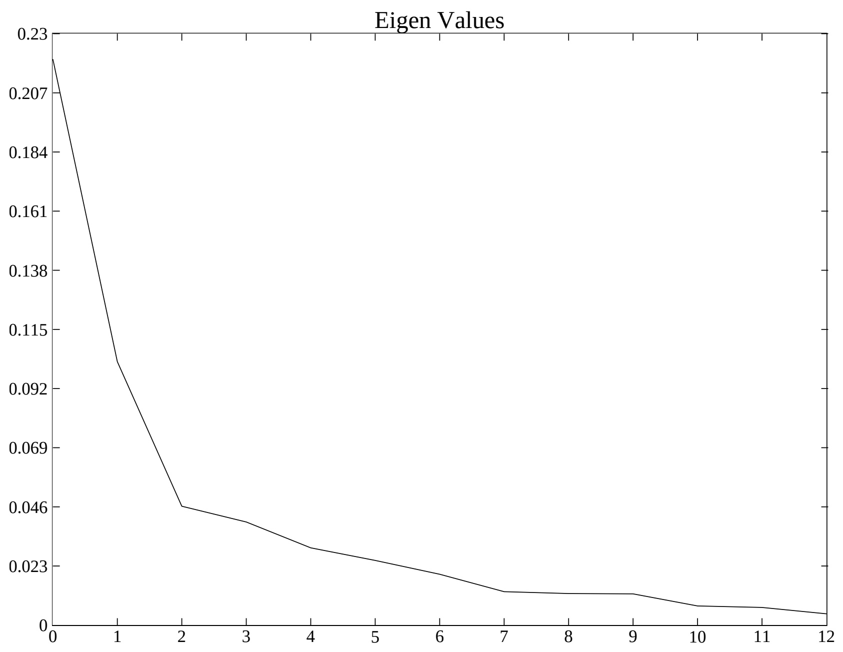

Figure 2.6: Eigen Values.

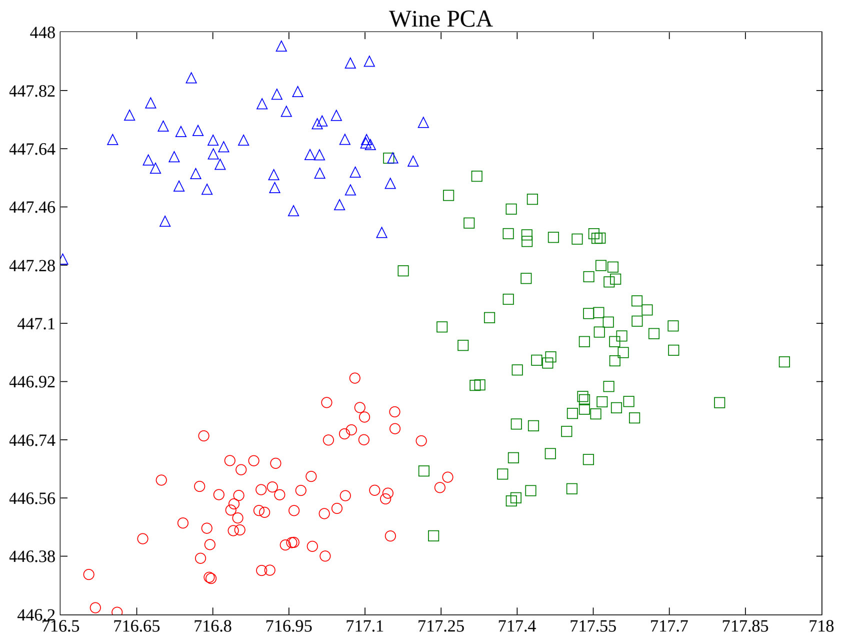

Figure 2.7: PCA.

Chapter 3

Conclusion

The wine data set proved to be very two dimensional. Even just performing basic plots of some of the data fields yielded moderate separation of the three classes. So in some respects applying self organising maps and principle component analysis was overkill. However both of these techniques did increase the separation previously found, so would probably help greatly with accurately classifying new wine samples.

The self organising map in particular worked very well, achieving near "perfect" separation of all three classes. However from a computational point of view it did take a considerably longer time to perform, whereas PCA took a fraction of the time (at least on this data set). Coupled with the apparent low dimensionality of the data, this may make PCA the best method overall, in terms of "bang for buck".

Appendix A

Python Data Processing Script

```
#!/usr/bin/python

import string
import os
from matplotlib.matlab import plot, title, savefig, figure, close, set, gca, xlabel, ylabel
from MLab import cov, eig, array, innerproduct

names = [ 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium',
      'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
      'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
      'Proline' ]

makescharts = True
runsom = True

maxvalues = []
minvalues = []
sumvalues = []
categories = []

def extractvalues( line ):
    values = string.split( string.strip( line ), "," )
    categories.append( int( values[ 0 ] ) )
    values = values[1:]
    values = map( float, values )
    for i in range( 0, len( values ) ):
        v = values[ i ]
        if i < len( maxvalues ):
            maxvalues[ i ] = max( maxvalues[ i ], v )
            minvalues[ i ] = min( minvalues[ i ], v )
            sumvalues[ i ] = sumvalues[ i ] + v
        else:
            maxvalues.append( v )
            minvalues.append( v )
            sumvalues.append( v )
    return values

def normalisedata( values ):
    norm = []
    for i in range( 0, len( values ) ):
        v = values[ i ];
        maxv = maxvalues[ i ]
        minv = minvalues[ i ]
        norm.append( (v-minv)/(maxv-minv) )
    return norm

def totext( values ):
    return string.join( map( str, values ), " " ) + "\n"

origfile = file( "wine.data" )

lines = origfile.readlines()
origfile.close()

values = map( extractvalues, lines )
# extract un-normalised values into the three categories
values1 = map( extractvalues, filter( lambda x: string.index( x, "1" ) == 0, lines ) )
values2 = map( extractvalues, filter( lambda x: string.index( x, "2" ) == 0, lines ) )
values3 = map( extractvalues, filter( lambda x: string.index( x, "3" ) == 0, lines ) )

# plot charts of the raw data
if makescharts:
    for i in range( 0, 13 ):
        x1 = map( lambda x: x[ i ], values1 )
        x2 = map( lambda x: x[ i ], values2 )
        x3 = map( lambda x: x[ i ], values3 )
        for j in range( i+1, 13 ):
            y1 = map( lambda x: x[ j ], values1 )
            y2 = map( lambda x: x[ j ], values2 )
            y3 = map( lambda x: x[ j ], values3 )
            figure()
            plot( x1, y1, 'ro' )
            plot( x2, y2, 'gs' )
            plot( x3, y3, 'b^' )
            title( 'Wine Properties' )
            xlabel( names[ i ] )
            ylabel( names[ j ] )
            savefig( str( i ) + "-" + str( j ) )
            close()

#normalise the data and write it to file for SOM to use
```

```
normalisedlines = map( totext, map( normalisedata, values ) )
normalisedfile = open( "wine.data.norm", "w" )
normalisedfile.writelines( normalisedlines )
normalisedfile.close()

if runsom:
    os.spawnlp( os.P_WAIT, './som', 'som', '178', '13', 'wine.data.norm', 'wine.som.config' )

coordsfile = open( "som.winners" )
coordlines = coordsfile.readlines()
coordsfile.close();

x1 = []
y1 = []
x2 = []
y2 = []
x3 = []
y3 = []

for i in range( 0, len( coordlines ) ):
    xy = string.split( coordlines[ i ] )
    x = int( xy[ 0 ] )
    y = int( xy[ 1 ] )
    c = categories[ i ]
    if c == 1:
        x1.append( x )
        y1.append( y )
    elif c == 2:
        x2.append( x )
        y2.append( y )
    elif c == 3:
        x3.append( x )
        y3.append( y )
    else:
        print "unknown category: " + c

# use matplotlib functions to make the graph
figure()

plot( x1, y1, 'ro' )
plot( x2, y2, 'gs' )
plot( x3, y3, 'b^' )
set( gca(), 'xticks', range( 0, 12 ) )
set( gca(), 'yticks', range( 0, 12 ) )
title( "Wine SOM" )
savefig( "som" )
close()

# PCA
averages = array( map( lambda x: x/len( values ), sumvalues ) )
```

```
normvalues = map( normalisedata, values )
cenvalues = array( map( lambda x: array( x ) - averages, normvalues ) )
covmatrix  = cov( cenvalues )
eigvalues,eigvectors = eig( covmatrix )

eigvlist = list( eigvalues )
eigvlist.sort()
eigvlist.reverse()

figure()
plot( range( 0, len( eigvlist ) ), eigvlist, 'k-' )
set( gca(), 'xticks', range( 0, len( eigvlist ) ) )
title( 'Eigen Values' )
savefig( 'eigenvalues' )
close()

# find vectors with largest eigen value
eigval1 = 0
eigval2 = 0
eigvec1 = []
eigvec2 = []

for i in range( len( eigvalues ) ):
    eigval, eigvec = eigvalues[ i ], eigvectors[ i ]
    if eigval > eigval2:
        eigval2, eigvec2 = eigval1, eigvec1
        eigval1, eigvec1 = eigval, eigvec


print 'eigvalue1=' + str( eigval1 )
print eigvec1
print 'eigvalue2=' + str( eigval2 )
print eigvec2

x1 = []
y1 = []
x2 = []
y2 = []
x3 = []
y3 = []

for i in range( 0, len( coordlines ) ):
    v = cenvalues[ i ]
    c = categories[ i ]
    x = innerproduct( eigvec1, v )
    y = innerproduct( eigvec2, v )
    if c == 1:
        x1.append( x )
        y1.append( y )
    elif c == 2:
        x2.append( x )
        y2.append( y )
    elif c == 3:
        x3.append( x )
        y3.append( y )
    else:
        print "unknown category: " + c

figure()

plot( x1, y1, 'ro' )
plot( x2, y2, 'gs' )
plot( x3, y3, 'b^' )
title( "Wine PCA" )
savefig( "pca" )
close()
```