Open lab –Speech/Audio signal Processing using MATLAB

Lab Sheet 6

Data Analysis and Pattern Classification using KNN

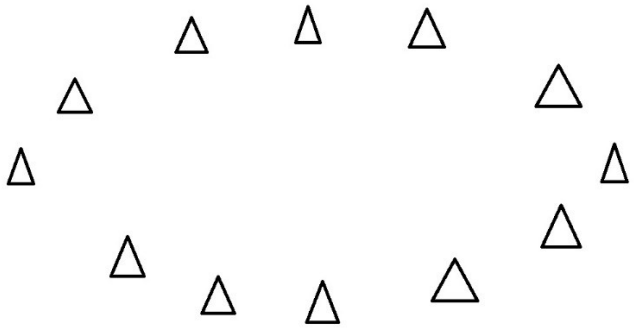
Aim

- To provide a general idea about pattern recognition techniques
- Dimensionality reduction of data
- Familiarizing supervised classification technique- KNN

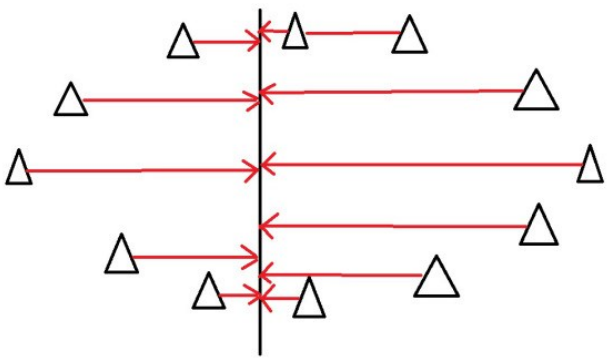
Theory

Principal component analysis (PCA)

Principal component analysis (PCA) is a standard tool in data analysis. With minimal effort PCA provides a roadmap for how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified structures that often underlie it. PCA finds the principal components of data. It is often useful to measure data in terms of its principal components rather than on a normal x-y axis. So what are principal components then? They're the underlying structure in the data. They are the directions where there is the most variance, the directions where the data is most spread out. This is easiest to explain by way of example. Here's some triangles in the shape of an oval

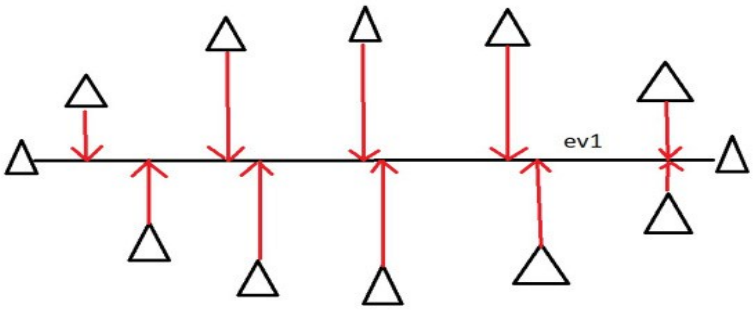


Imagine that the triangles are points of data. To find the direction where there is most variance, find the straight line where the data is most spread out when projected onto it. A vertical straight line with the points projected on to it will look like this:



The data isn't very spread out here, therefore it doesn't have a large variance. It is probably not the principal component.

A horizontal line are with lines projected on will look like this:

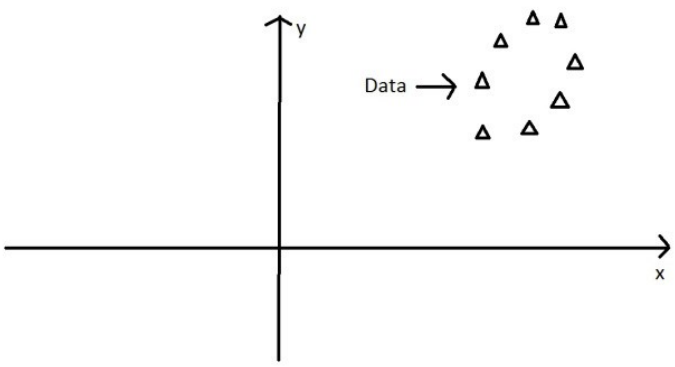


On this line the data is way more spread out, it has a large variance. In fact there isn't a straight line you can draw that has a larger variance than a horizontal one. A horizontal line is therefore the principal component in this example.

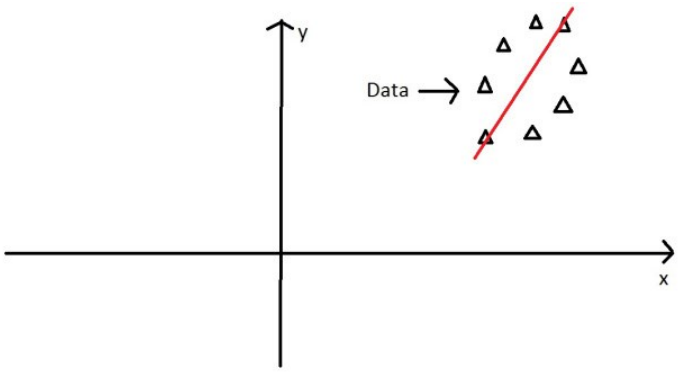
Eigenvectors and Eigenvalues

When we get a set of data points, like the triangles above, we can deconstruct the set into eigenvectors and eigenvalues. Eigenvectors and values exist in pairs: every eigenvector has a corresponding eigenvalue. An eigenvector is a direction, in the example above the eigenvector was the direction of the line (vertical, horizontal, 45 degrees etc.) An eigenvalue is a number, telling you how much variance there is in the data in that direction, in the example above the eigenvalue is a number telling us how spread out the data is on the line. The eigenvector with the highest eigenvalue is therefore the principal component. The amount of eigenvectors/values that exist equals the number of dimensions the data set has. Say I'm measuring age and hours on the internet. There are 2 variables, it's a 2 dimensional data set, and therefore there are 2 eigenvectors/values. If I'm measuring age, hours on internet and hours on mobile phone there's 3 variables, 3-D data set, so 3 eigenvectors/values. The reason for this is that eigenvectors put the data into a new set of dimensions, and these new dimensions have to be equal to the original amount of dimensions. This sounds complicated, but again an example should make it clear.

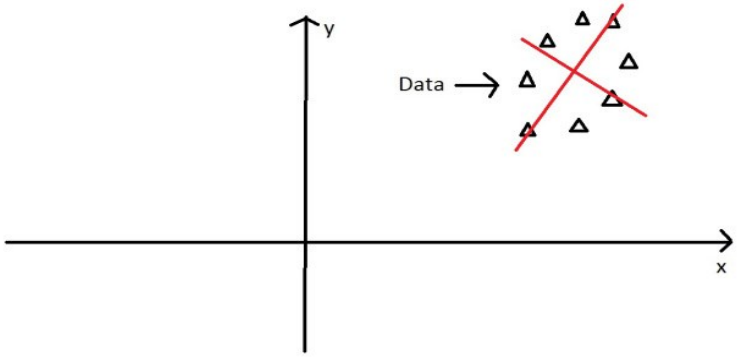
Here's a graph with the oval:



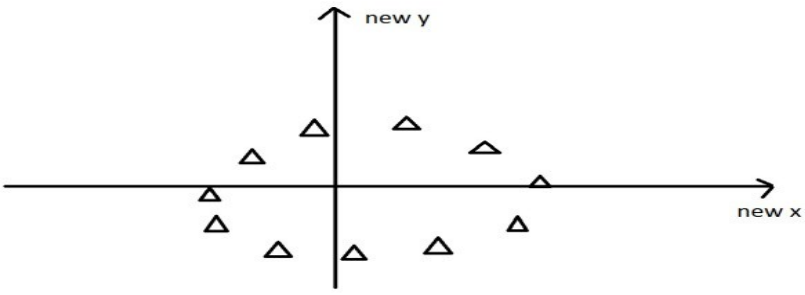
At the moment the oval is on an x-y axis. x could be age and y hours on the internet. These are the two dimensions that my data set is currently being measured in. Now remember that the principal component of the oval was a line splitting it long ways:



It turns out the other eigenvector (remember there are only two of them as it's a 2-D problem) is perpendicular to the principal component. As we said, the eigenvectors have to be able to span the whole x-y area, in order to do this (most effectively), the two directions need to be orthogonal (i.e. 90 degrees) to one another. This why the x and y axis are orthogonal to each other in the first place. It would be really awkward if the y axis was at 45 degrees to the x axis. So the second eigenvector would look like this:



The eigenvectors have given us a much more useful axis to frame the data in. We can now re-frame the data in these new dimensions. It would look like this:

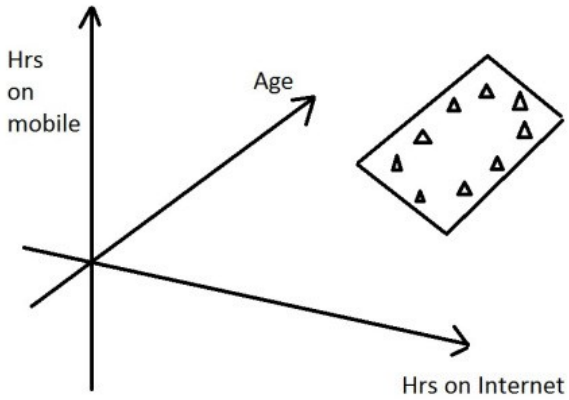


Note that nothing has been done to the data itself. We're just looking at it from a different angle. So getting the eigenvectors gets you from one set of axes to another. These axes are much more intuitive to the shape of the data now. These directions are where there is most variation, and that is where there is more information (think about this the reverse way round. If there was no variation in the data [e.g. everything was equal to 1] there would be no information, it's a very boring statistic – in this scenario the eigenvalue for that dimension

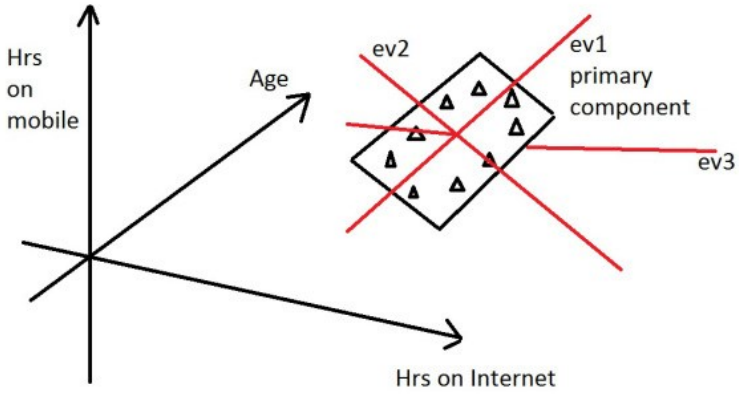
would equal zero, because there is no variation).How does PCA and eigenvectors help in the actual analysis of data? one is dimension reduction.

Dimension Reduction

PCA can be used to reduce the dimensions of a data set. Dimension reduction is analogous to being philosophically reductionist: It reduces the data down into it's basic components, stripping away any unnecessary parts. Let's say you are measuring three things: age, hours on internet and hours on mobile. There are 3 variables so it is a 3D data set. 3 dimensions is an x,y and z graph, It measure width, depth and height (like the dimensions in the real world). Now imagine that the data forms into an oval like the ones above, but that this oval is on a plane. i.e. all the data points lie on a piece of paper within this 3D graph (having width and depth, but no height). Like this:

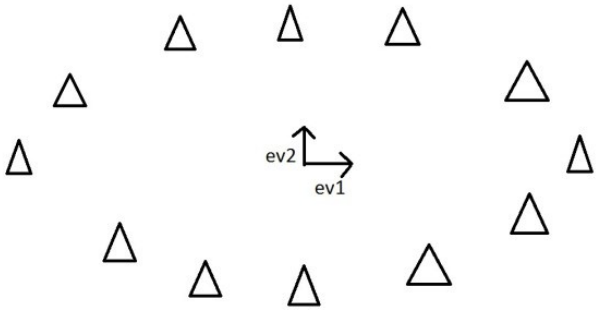


When we find the 3 eigenvectors/values of the data set (remember 3D problem = 3 eigenvectors), 2 of the eigenvectors will have large eigenvalues, and one of the eigenvectors will have an eigenvalue of zero. The first two eigenvectors will show the width and depth of the data, but because there is no height on the data (it is on a piece of paper) the third eigenvalue will be zero. On the picture below ev1 is the first eigen vector (the one with the biggest eigenvalue, the principal component), ev2 is the second eigenvector (which has a non-zero eigenvalue) and ev3 is the third eigenvector, which has an eigenvalue of zero.



We can now rearrange our axes to be along the eigenvectors, rather than age, hours on internet and hours on mobile. However we know that the ev3, the third eigenvector, is pretty useless.

Therefore instead of representing the data in 3 dimensions, we can get rid of the useless direction and only represent it in 2 dimensions, like before:



This is dimension reduction. We have reduced the problem from a 3D to a 2D problem, getting rid of a dimension. Reducing dimensions helps to simplify the data and makes it easier to visualise. Note that we can reduce dimensions even if there isn't a zero eigenvalue. Imagine we did the example again, except instead of the oval being on a 2D plane, it had a tiny amount of height to it. There would still be 3 eigenvectors, however this time all the eigenvalues would not be zero.

Example Code

```
load fisheriris
x = meas(:,1:2);
gscatter(x(:,1),x(:,2),species,[], [], [], 'on', 'x', 'y')
% legend('Location','best')
title('normal Iris data'), grid on
figure


[pc,score,latent,tsquare] = princomp(meas);
gscatter(score(:,1), score(:,2), species, [], [], [], 'on', 'PC1', 'PC2')
title('Projected Iris data'), grid on
```

Pattern Classification

Introduction to K-Nearest Neighbour Algorithm

The purpose of the k Nearest Neighbours (kNN) algorithm is to use a database in which the data points are separated into several separate classes to predict the classification of a new sample point. Suppose each sample in our data set has n attributes which we combine to form an n-dimensional vector:

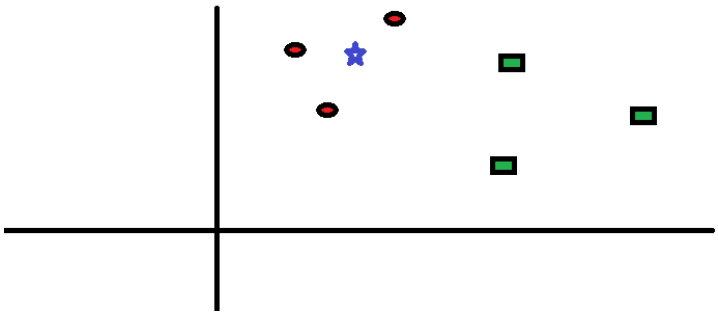
$$x = (x1, x2, \ldots, xn).$$

These n attributes are considered to be the independent variables. Each sample also has another attribute, denoted by y (the dependent variable), whose value depends on the other n attributes$x$. We assume that y is a variable, and there is a scalar function, f, which assigns a class, $y = f(x)$ to every such vectors. We do not know anything about f except that we assume that it is smooth in some sense.

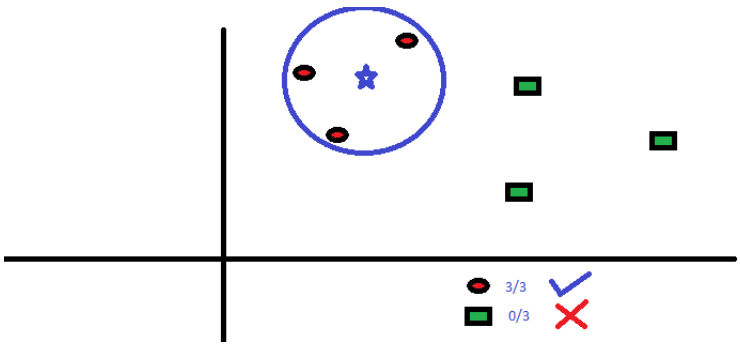
We suppose that a set of T such vectors are given together with their corresponding classes: $x(i), y(i)\ for\ i = 1, 2, \ldots, T$. This set is referred to as the training set. The problem we want to solve is the following. Supposed we are given a new sample where $x = u$. We want to find the class that this sample belongs. If we knew the function f , we would simply compute $v = f(u)$ to know how to classify this new sample, but of course we do not know anything about f except that it is sufficiently smooth.

The idea in k-Nearest Neighbor methods is to identify k samples in the training set whose independent variables x are similar to u, and to use these k samples to classify this new sample into a class, v. If all we are prepared to assume is that f is a smooth function, a reasonable idea is to look for samples in our training data that are near it (in terms of the independent variables) and then to compute v from the values of y for these samples. When we talk about neighbors we are implying that there is a distance measure that we can compute between samples based on the independent variables. For the moment we will concern ourselves to the most popular measure of distance: Euclidean distance.

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :



You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The "K" is KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:



The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next we will understand what are the factors to be considered to conclude the best K.

Example Code

```
clear
load fisheriris
figure
x = meas(:,3:4);
%  choose query point
newpoint = [5 1.45];

[n,d]=knnsearch(x,newpoint,'k',10);

gscatter(x(:,1),x(:,2),species)

line(newpoint(1),newpoint(2),'marker','x','color','k',...
   'markersize',10,'linewidth',2)
line(x(n,1),x(n,2),'color',[.5 .5 .5],'marker','o',...
   'linestyle','none','markersize',10)
set(legend,'location','best')
tabulate(species(n,:))
```