MATLAB Workshop 1

An introduction to machine learning and basics of MATLAB

Session Plan

This work shop is designed to cover the basics of computational learning theory as well as introduce you to the practical implementations of these techniques. The text will be broken down into sections clearly marked using the symbols given below. Read through the exercise and try to answer the questions the best you can amongst yourselves. We have left space at the bottom of each question to write notes/answers.

This is a ***key*** concept that must be understood, if you don't then you must ask!

Comment: Hopefully the exercises are written to emphasise these key concepts

This indicates a ***question*** or exercise that needs to be completed.

This indicates a task that you need to ***do*** as instructed in the exercise.

This is a helpful ***tip*** or piece of useful information.

Overview:

Today I will be giving you an overview of the functionality of the MATLAB programming language that is essential for developing applications in machine learning. I will also mention other freely available software that you might find useful for your project or final year project. This session is designed as an informal practical session, you can work in groups and chat amongst yourselves whilst working through the exercises provided. I encourage you to work through the questions whilst you can so that I may help you with any problems. I have tried to make this session light hearted and highlight the key concepts with a symbol. If you need help at any time, please either refer to your notes, the Help guide or ask me or Tony! I will hand out the solution code to this exercise at the end of this session.

Learning outcomes

1. Review some of the basic concepts in computer learning theory
2. Look at some real life data sets (Places to get some!)
3. Give a quick review of linear algebra concepts needed for using MATLAB
4. Give an introduction to the MATLAB programming language
5. How to structure data in a MATLAB matrix
6. How to read data from a file into MATLAB
7. How to extract columns and concatenate vectors
8. How to plot basic graphs
9. How to perform basic matrix calculations
10. Introduce other freely available packages such as WEKA, NETLAB, and Pattern Classification Toolbox

CS392 Resources Directory

We have set up an area on the computer science department network for electronic resources/files that you might find useful for this course. They are located at
`/CS/courses/CS392`
which has the following subdirectories containing
1. `/CS/courses/CS392/data/`

Several datasets that have been gathered for you. You may want to pick one of these for analysing in your project.
2. `/CS/courses/CS392/MATLAB`

several programs written as MATLAB scripts (.m file extension), as well as downloads of the NETLAB and Pattern Classification toolboxes.
3. `/CS/courses/CS392/java/`

several sample programs written in Java, including the WEKA data mining package.

Comment: We encourage you to have a look at the files in this area to get your self familiar with MATLAB programs and data sets.

4. `/CS/courses/CS392/workshops/`

This area contains all the files ready for you to begin the workshops.
For this lesson we will be working with a simple weather data set (which is used as an example in Mitchell).

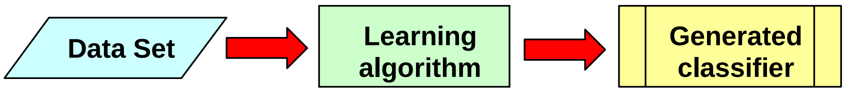
Copy data from shared directory

Copy the workshop files into your user area, and view the dataset using `emacs` like so

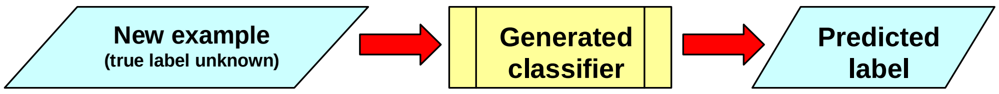
```
cp /CS/courses/CS392/workshops/ex1Basics/* .
emacs tennisweather.arff &
```

How data sets are used for learning

This an example of a typical data set, you should have be vaguely familiar with the set up by now. Try to think of these data sets as databases if you struggle to get the idea. This particular data set (and any other with the *.arff extension) are specially formatted for use with the WEKA data mining package (see Page 19).
The is a simple artificial data set is used in Mitchell (excellent book, see references) which contains a set of measurements taken about past weather details, with the problem to predict using the weather information, whether that day was ideal for "playing tennis".



The data set is introduced to the learning machine algorithm, which then generates a classifier from the data. We can then use this classifier to predict the labels of new examples.



Difference between Pattern Recognition and Regression

There are two main problems that we will consider in this course:

1. **Pattern recognition** ➔ label to predict is *discrete* taking a finite number of possible values. Eg. `Normal`, `Benign` or `Malignant` for cancer diagnostics.
2. **Regression** ➔ label to predict is *continuous* value, eg. speed of a car, rainfall etc.

In this workshop we will focus on the problem of *pattern recognition.*
For this problem, after generating our classifier, we could get the information about the day (weather forecast, the current temperature, humidity and wind readings) and feed them into our classifier. Our classifier would then predict whether that day would be a great for playing tennis.

Comment: Not a very useful application of a learning machine, you would spend all the effort to design a machine to answer such a trivial question.

? Can you think of any informal decision rules for classifying this data?

(`if` rainy and cold `then` don't play, `if` hot and humid `then` don't play etc.)

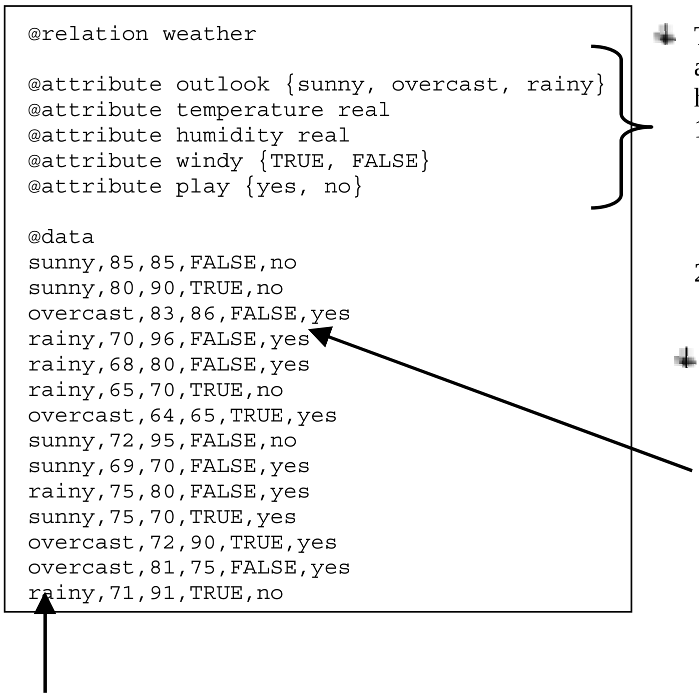
Anatomy of a Data Set

Get yourself familiar with following components of a data set as highlighted in Figure 1 below. The main terms we use are highlighted in bold, but they are sometimes referred by other names.

1) **Attributes**, features, fields
2) **Labels**, classifications, target field
3) **Examples**, instances, records

Comment: Actually the encoding of attributes is a very important step which we will not dwell on.

Figure 1: Identifying the parts of the `tennisweather` dataset



```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

- This describes the formatting of the attributes used in the data. Notice how they fall into two main types:
  1. *Nominal* taking discrete values, eg. Attribute "`windy`" takes values `TRUE`, `FALSE` etc.
  2. *Numeric* taking continuous real values, eg. Attribute "`temp`"
- Each row/line of this file corresponds to the *attribute* values for a particular *example* in the data set. In this data each example represents measurements and findings of the weather of a particular day.

- Each column in the data represent the measurements of a particular *attribute*. In this case the "`outlook`" gives details of the weather forecast for that day.
- Usual convention is to have the classification to be learnt is left as the last attribute
- In this data the target to be learnt is "`play`" which dictates whether one should play tennis given the other attribute values.

? As an expert in Computer Learning (hopefully you will be after taking this course!) a doctor asks you for help. He/She would like your help developing a system for screening patients in need of urgent brain surgery. He/She tells you that they have kept records on each of the patients that they have treated over the years. Describe what would have to be done, ie. How/what would you need to generate a data set, what are the attributes, class labels etc.

Comment:
Ask the doctor what useful patient details could be used to diagnose. For example details like their name, the type of car the y drive would probably be irrelevant. However details like cancer in the family, age, weight, average usage of mobile phone etc migths be useful.

The class label could be whatever is useful to the doctor. They may want to know if the patients is either risk, or not. But they may also want to know if high, medium or low risk.

For working in MATLAB we need to have everything in numeric format, so the first step for us is to convert everything into numbers.

Convert data into numeric format

Convert all the nominal attributes into numbers like so, for "`outlook`" attribute let `sunny = 0, overcast = 1, rainy = 2`. For the "`windy`" attribute let `FALSE = 0` and `TRUE = 1`. For the "`play`" attribute let `no = 0` and `yes = 1`. You may wish to use *replace command* in `emacs` instead of manually changing the data! As you will become aware one tedious and less glamorous part of machine learning is the formatting of data. Finally remove all the attribute information at the top of this data set file, and save the new file as "`tennisweathernumeric.txt`".

Comment: You should be very careful when converting data into numeric values, you must consider what information does that attribute hold!

Open up MATLAB

Now open up MATLAB by typing at the command line:

```
matlab &
```

You will now be presented with the console window. This gives you access to lots of different parts of MATLAB which you can find in the beginners guide that I have provided. However for this session we will mainly be concentrating on the *Command Window* right on right hand side of screen. In here we will be presented with output from our programs, as well as be able to type in program commands directly.

MATLAB (**Mat**rix **Lab**oratory) is an interactive software system for numerical computations and graphics [1]. As the name suggests, MATLAB is especially designed for matrix computations. MATLAB is a weakly typed high level language, which can be very disorientating to you keen C++ programmers. There are no explicit definitions of data types such as int, string, etc. MATLAB has a vast library of routines for all sorts of tasks from file reading, to graph plot creation. You can specify very complex programs in just a few lines of code!

We will now create a new MATLAB program file (aka M-files). Think of this file like a script which we can specify program instructions in.

Create an M-File to enter your program

Click on the **File Menu → New → M-File**
This will bring up a MATLAB editor program for you to start entering your script. We will now go through some of the most important commands that you will need for this course. We will use MATLAB to read in the `tennisweather` data set that you manipulated earlier and plot some nice graphs to *visualise* the problem.
Now that we have converted the data into numeric format we can use a very handy function `dlmread` which can be used to read delimited files of numbers into matrices in MATLAB.

Read the data set into your program using `dlmread`

Type into your open MATLAB script the following code:

```
DATA=dlmread('tennisweathernumeric.txt',',')
```

This will read the data from our "`tennisweathernumeric.txt`" file that we created earlier into a 5×13 matrix called `DATA`. The second argument to the function ',' specifies that the data is comma delimited. Notice how we didn't explicitly specify that the `DATA` variable is a matrix, this can be very confusing! I usually keep the convention that matrices and vector variables are in uppercase, whilst normal numeric variables are in lowercase to save confusion. I advise you strongly to build you MATLAB programs *in small steps* and make sure each part works correctly instead of trying to write the complete program at once.

Comments in MATLAB using the `%` *symbol*

You can add comments to your code by using the % symbol. This can be also useful when debugging code to isolate which of lines of code are giving errors. To run your program, save it in the same directory that you created the "`tennisweathernumeric.txt`" file.

Save and run your program in MATLAB

Click on **File → Save** and specify the filename eg. `firstplot.m`
To can run your program in two ways

1) Click on **Debug menu → run**
2) Or go to the MATLAB Command Window that you started with and type in the name of your file eg. `firstplot` followed by return

If you look at the *Command Window* you will notice that the program has executed and output to screen. As seen on the next page the program should have correctly read in the data set from the file.

```
DATA =

     0    85    85     0     0
     0    80    90     1     0
     1    83    81     0     1
     2    70    85     0     1
     2    68    80     0     1
     2    65    97     1     0
     1    64    65     1     1
     0    72    95     0     0
     0    69    70     0     1
     2    75    80     0     1
     0    75    70     1     1
     1    72    84     1     1
     1    81    75     0     1
     2    71    96     1     0
```

Whenever you define a variable the contents are output to the Command Window unless you explicitly tell the MATLAB program interpreter not to. You can switch off this output by typing the '`;`' semi-colon symbol at the end of the line of code.

For our simple programs this will not be necessary, as we will want to see what we are doing. At the end of this course when you are MATLAB experts, you may find that this output very annoying!

Accessing elements of a matrix, removing columns and rows using the magical colon ":"

Now once you have read the data into a matrix, we can perform lots of useful manipulations. We can extract any element in the matrix `DATA(i,j)` by specifying the appropriate index `i`, `j` of that element. We can also remove columns and rows of a matrix by using the colon notation "`:`", e.g. `DATA(i,:)` would remove the `i`th row and `DATA(:,j)` would remove the `j`th column of the matrix `DATA`.

Indexes in MATLAB

Try to think of a MATRIX as data storage variable much like a two dimensional array of numbers in C++. Be careful, indexes in MATLAB are different to C++, they run from 1 to N and *not* 0 to N like when using arrays in C++.

? Write a line of code that will extract a column in the `DATA` matrix corresponding to the first attribute in the matrix and put it into a vector called `ATTRIBUTE1`.

Comment: ATTRIBUTE1 = DATA(:,1)
EXAMPLE4 = DATA(4:,1)

? Do the same for extracting the fourth example in the `DATA` matrix into a vector `EXAMPLE4`.

Comment: Are these what you expect?

? Do you understand which parts of the matrix are examples and which are attributes?

ⓘ *Sensible variable names*

Use sensible names for variables to make code less confusing! Use consistent notation to distinguish between strings, numbers, and matrices!

Finding the size of a matrix

A particularly useful thing to do in MATLAB is know the size of your matrix that you are dealing with at any one time! You must make sure that these vectors are of the correct size when you manipulate them. This is problem is highlighted by the non-commutativity of matrix multiplication. To find the number of rows in a matrix `X` use the function `size(X,1)`, and to find the number of columns use `size(X,2)`.

Comment: A=[1 2; 3 4;]
B=[2 3; 4 5]
AB = [2+8; 3+10; 6+16 9+20] = [10 13; 22 29]
BA = [2+9; 4+12; 4+15 8+20] = [11 16; 19 28]
Rows times Colums using fingers

? Write a line of code that will read the number of attributes used in the matrix `DATA` (not including the class attribute "`play`") into a variable `numberOfAttributes`, and the number of examples likewise into `numberOfExamples`.

Comment: Make sure students remember number of attributes does not include the play class label so take away 1.
numOfExamples=size(DATA,1)
numOfAttributes=size(DATA,2)-1

Finding help in MATLAB

If you want to find out more about a MATLAB function i.e. a simple description of its usage, what arguments a function takes, or what type of data is returned by the function use MATLAB great help facilities from the menu bar, or via querying a function in the *Command Window* eg. typing `help dlmread` to find out more about the `dlmread` function.

Colon ' : ' for specifying ranges

Now let us return to another magical property of the ' : ' colon in MATLAB – to specify ranges of values. Suppose that you wanted to create a vector `X` of values with values ranging from `1` to `n`. You could specify the vector explicitly ie.

```
X = [1 2 3 4 5 6 …. n]
```

but for large values of `n` this would be dangerous and be detrimental to your fingers.
A much better way is to use the colon to specify the range. Like so

```
X = [1:n]
```

ⓘ *Different incremented ranges*

We can also increment our range using any value by using an extra colon
eg. `X = [1:0.5:n]` to create an vector of numbers

The magical colon for removing several rows/columns

We can use this method to access a range of indexes in a matrix. For example to remove the first n columns of a matrix X and put them into another variable Y we could type
`Y = X(:,1:n)`
This can be a very labour saving device when writing programs. However due to the compactness of the code this can look very confusing, it does take some getting used to. Relating what you are doing to C++, you are implicitly using for loops to copy values and create variables.

A common problem when working with data is to separate the observable features into a matrix `X`, and the labels that are to be learnt in `Y`. Let the number of attributes in the data be *n* and let the number of examples be *l*.

? Without any calculations in MATLAB, what should be the dimensions of attribute matrices `X` and `Y` be in terms of *n* and *l*?

Comment: X is $l \times n$
Y is $l \times 1$

? Relating to our problem, what are the values of *n* and *l*?

Comment: *n=4*
l=14

? Do they match up with what you found in the previous exercise using the `size` function?

Comment: They should do, maybe you didn't take away 1 from the number of attributes, remember you don't use the class attribute

? Finally create the attribute matrix X, and label matrix Y using the colon notation.

Comment:
X=DATA(:,1:numOfAttributes)
Y=DATA(:,numOfAttributes+1)

Now that we have created our data matrices X and Y for our dataset it would be useful to get some kind of visualisation of this data! A common technique is to use scatter plots of examples of each class, of course this is limited to a maximum of 3 dimensions (unless you have special mutated eyes). In this last part of the workshop we will plot the "`yes`" and "`no`" play tennis days on a scatter plot using two variables `temperature` and `humidity` as the axis.

The `find` function for returning vectors of indexes

The function `find(X==n)` searches through the matrix X and returns a matrix of indexes that match the value n. You can construct more complex logical statements, just like you would when writing an `if` statement in C++. If X is a 2D matrix then the indexes returned by `find` will be pairs `[i j]` but if X is a column or row vector (1D) then the find function will return a 1D column or row vector of indexes.

Extract the "`yes`" and "`no`" days from the matrix X

Firstly we must separate the `yes` and `no` days from the attribute matrix X, using MATLAB's wonderful find function. Do this by adding the code:

```
X_YES_DAYS = X(find(Y==1),:)
X_NO_DAYS = X(find(Y==0),:)
```

This will create two matrices `X_YES_DAYS` and `X_NO_DAYS` to store the attribute vectors of yes and no examples. Remember we converted `yes = 1`, and `no = 0` earlier, that is why we use the function `find(Y==1)` which finds the indexes of these respective examples (and returns them into a vector). We then use this vector of indexes to remove a range of rows from the matrix X using the colon notation. Notice how compact this code is! Make sure you understand what it is doing!

Comment: If they are confused explicitly make them create a vector YES_DAY_INDEXES so they can see these indexes and make them match it with the data.

The plot function for creating 2D graphs

This is a very nice feature of MATLAB; far better than the crappy little graphs that Excel produces. Quite simply the command `plot(X,Y,s)` plots a vector of pairs using X for horizontal and Y for vertical coordinates, with s being a string to specify what type of *line style, line colour* and *point type* (for a full list see `help plot`). For example the string `s='r*-.'` creates a ***r**ed* line, with *stars* plotted at each point, with a *dash-dot* line connecting them!

ⓘ *Creating multiple plots using* `hold on` *and* `hold off`

To build up complex graph plots it is often better to use a separate plot command for each element in the graph. This will make the code much easier to read! To do so you must place the `hold on` and `hold off` code above and below your `plot` commands.

? Plot the "`yes`" day vectors with red circles, and "`no`" days with blue plus signs, on a graph using the `temperature` and `humidity` attributes. Do *not* use a line to connect the points! You will need to use the `X_YES_DAYS` and `X_NO_DAYS` matrices that you created earlier, and remove the desired attributes using the colon notation. Write notes below, and run your program! Your finished plot should look like the graph on Page 17 of this worksheet.

Comment:

```
plot(X_YES_DAYS(:,2),
X_YES_DAYS(:,3),
'or')

plot(X_NO_DAYS(:,2),
X_NO_DAYS(:,3),
'b+')
```

Once you have plotted the data we can now think about how to learn from the data. Looking at the data plotted in the two dimensions we can see a rough clustering of the data points, "`no`" tennis days seem to have high temperature and humidity in the *upper-right region* of the plot. Informally we have defined a decision rule, but to define this relationship more precisely we will involve some simple mathematics.

Classification using separating hyper-plane

How can we simply divide up regions of space of data? One simple solution suggested by many early learning algorithms (e.g. Perceptron developed back in the swinging 60's) was to use the concept of a separating hyperplane. Don't get scared of the term hyperplane, all it means is a generalisation of a straight line in many dimensions. For example in our plot (see Page 17) we have successfully split a 2-dimensional space with "`no`" days above the black straight line (separating hyper-plane in 2D) and "`yes`" days below. For 3 dimensions the hyper-plane would be a flat surface, and though its hard to imagine this can be generalised for more dimensions. Using a hyper-plane to separate the data can be very simple method of classifying 2-class data. Relating back to model of learning described earlier (see Page 5) our learning algorithm would find a separating hyper-plane, the generated classifier would be described by that hyper-plane using it to classify data into the two

? Can you think of any potential problems of using a separating hyperplane to classify data?

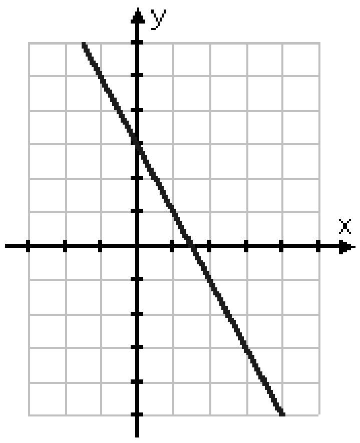
Comment: 1.More than 2 classes, how do we do it?
2.What if not separable as in XOR problem?

Understanding the equation of a straight line

An important aspect of creating separating hyperplanes is to understand how the equation is defined. If we consider the simple 2-dimensional case that you might have experienced in GCSE Maths, we consider two variables $\boldsymbol{x}$ for horizontal and $\boldsymbol{y}$ for vertical. The equation of a straight line is of the form

$$\boldsymbol{y} = m\boldsymbol{x}+c$$

The components of the line can be seen in the plot below for the line $\boldsymbol{y} = -2\boldsymbol{x}+3$,



The gradient (slope of the line) of the graph m can be calculated by considering two points on the line (–1, 5) and (2, –1) like so:

$$m = \frac{y_1 - y_2}{x_1 - x_2} = \frac{(5)-(-1)}{(-1)-(2)} = \frac{6}{-3} = -2$$

To find the c component of the equation we want to find its intercept on the $\boldsymbol{y}$ axis, which is formally the value of $\boldsymbol{y}$ when $\boldsymbol{x} = 0$. In our example the value $c = 3$. Once the constant m has been found we can ***also*** derive c by using a known coordinate ***(x,y)*** on the line and rearranging the equation to let $c = \boldsymbol{y} - m\boldsymbol{x}$.

You should be familiar with this concept and be comfortable with rearranging these straight line formulae, as they will be useful when working with SVM's later in the course.

? Find the equation of the line drawn on the plot shown on Page 17, you may wish to use the points (60,95) and (82,85). Calculate the values of the constants m and c (two decimal places).

Comment:
m=(95-85)/(60-82)= –10/22
≈ –0.45
y= –0.45x+c, ➔
c = 95 – (–0.45×60) = 122.27

? Rearrange the formula in terms of $m_y y + m_x x = c$, and identify the constants m_y and m_x.

Note: this is another way of specifying the equation of a straight line.

Comment: y-mx=c, therfor
y+0.45x=122.27
my =1, mx=0.45

Plotting lines in MATLAB

Graph plotting in MATAB works by:

1. Specifying a sub-domain of values to evaluate the function that you want to plot. **E.g.** we could specify to plot a *sin* curve over the domain of values
   X = $\left[-\pi, \frac{-\pi}{2}, 0, \frac{\pi}{2}, \pi\right]$
2. Calculating the respective function values and plotting these coordinate pairs on a graph and interconnecting each point with a straight line. **E.g.** the corresponding range of values for a *sin* curve Y = $[0,-1,0,1,0]$. Interconnecting these X,Y pairs would give a very crude plot of a *sin* curve!

To specify the range in MATLAB we can use the clever colon notation discussed earlier. For example to plot the *sin* curve we can use the following code:

```
X=[-pi:pi/2:pi]
Y=sin(X)
plot(X,Y,'rx-')
```

It can be confusing to think of passing a vector `X` into a function like `sin`. Try to think of it as evaluating the function at each value in `X`. To make a less grainy plot of the sin curve we can use a smaller increment size (for example on page 16 we show plots of sin curves using increments `0.01` rather than `pi/2`).

? Using the colon notation specify a vector `HX` for the range of values for the hyper-plane to be plot between 60 and 85.

Comment: HX=[60:85]

? Using the values of *m* and *c* that you computed on the previous page try, create a vector `HY` of corresponding values specified by the separating hyper-plane defined by *m* and *c* for the values in `HX`.

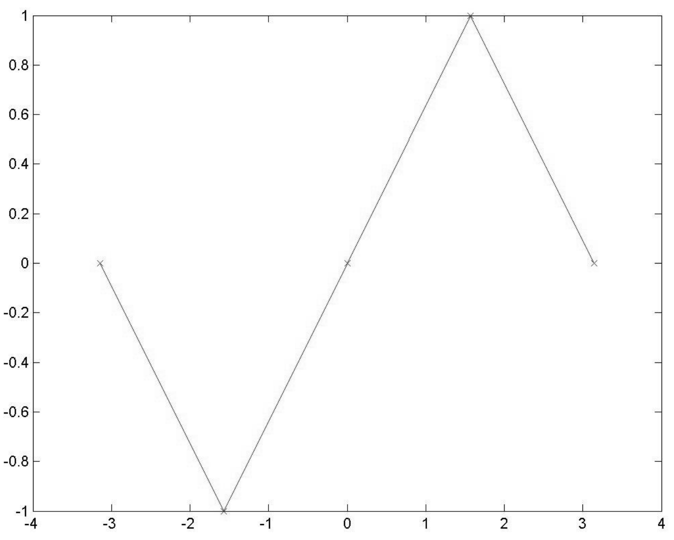
Comment: `HY=(-0.45*HX)+122.27`

? Plot the separating hyperplane with a black line using vectors `HX` and `HY`.

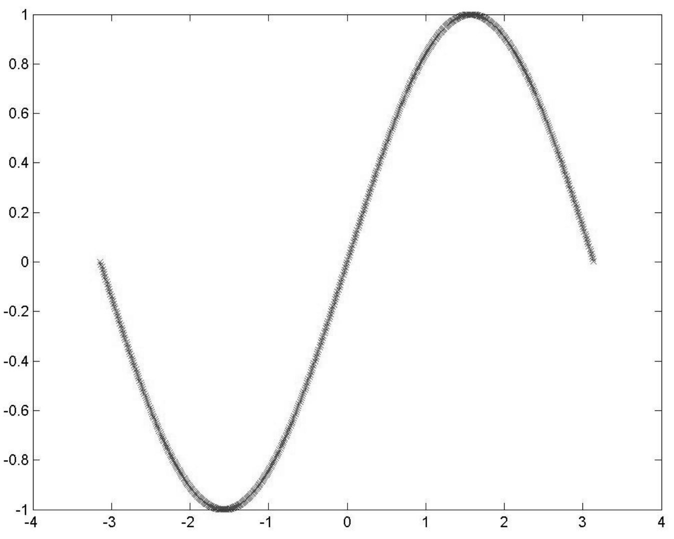
Comment: plot(HX,HY,'k-')

Plots of the sin curve using different increment sizes

Using the increment size `pi/2` we get a very crude plot of 5 points on the sin curve:

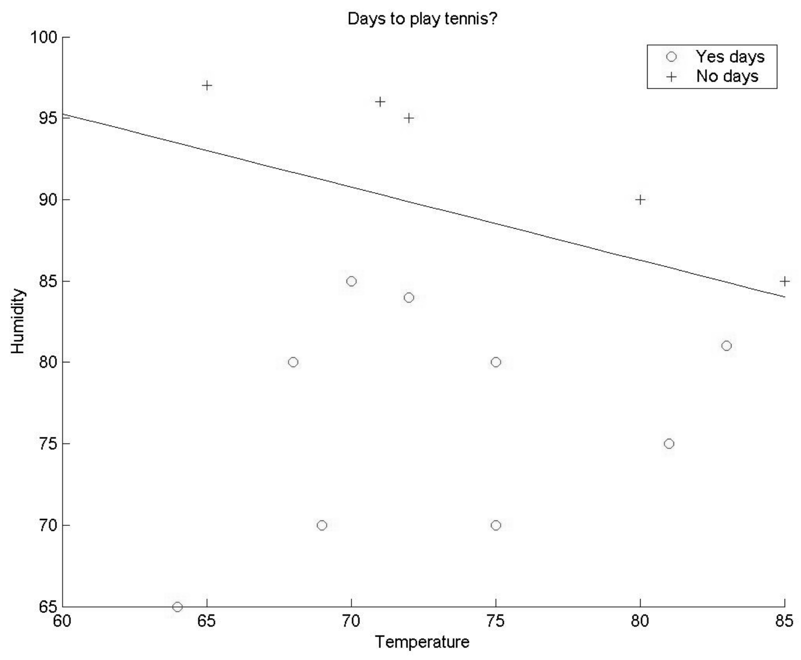


Using a smaller increment size `0.01` we get a much smoother plot of 629 points on the sin curve:



The finished graph plot

Your finished plot should look like this! If it doesn't then something has gone wrong!



Additional Tasks

I cannot recommend how important it is to just mess around with functions in MATLAB. Try writing programs to tackle fun little problems. Practise is the best way to learn any programming language. Here are a list of possible task that you do to improve your background knowledge to this subject. Use the MATLAB's `help` to find out about other functions, concentrate on the arguments and return values of a function.

🏆 Try to plot the data set using three variables in a 3D plot using MATLAB. (**Hint:** use the `plot3` function).

🏆 Read chapter 1 of Tom Mitchell's book (see references), don't worry if the notation is different just make sure you are familiar with the main components of data sets and the concept of learning.

🏆 How would you define a hyperplane in 3 dimensions? Rearrange the formula $a_1x_1 + a_2x_2 + a_3x_3 = b$ in terms of the $y = mx + c$ format discussed earlier treating x_3 as the vertical axis *y*.

Comment: x_3 = (b-a_1x_1-a_2x_2)/a_3

Publicly available tools for machine learning

In this section I will discuss some publicly available tools and programs for machine learning research. We recommend that you try to write your project using your own MATLAB code, but if you are interested in machine learning then you may want to look at these packages for ideas on how to write good code and also to open your eyes to the vast number of learning algorithms that are out there other than SVM. If you are researching a final year project in machine learning then you may find these resources very useful. (I wish I knew about them in my third year!).

ⓘ *NETLAB*

The main website can be found at:

http://www.ncrg.aston.ac.uk/netlab/

"The Netlab toolbox is designed to provide the central tools necessary for the simulation of theoretically well founded neural network algorithms and related models for use in teaching, research and applications development."

The toolbox is written in MATLAB and has a very extensive range of algorithms, however these are mainly Neural Network and Bayesian algorithms which are not the main focus of this course. It does not have an implementation of SVM. This toolbox has the advantage that it has an accompanying book. It has been designed for teaching purposes and has some nice GUI's.

ⓘ *WEKA*

This is a very extensive package of data mining tools written in Java and can be downloaded from

http://www.cs.waikato.ac.nz/~ml/weka/

The code is distributed under GNU public licence agreement and is very well documented. Many researchers around the world in both industry and academia have contributed to this project since 1999. The list of algorithms that this package offers is by far the most extensive, including the all important SVM algorithm. This package has been very carefully designed with careful objects to allow rapid implementation of learning algorithms. The data format (ARFF seen earlier) is very natural and allows the researcher to ignore low level problems of reading and writing data to files and concentrate on how to process the data. Another great benefit of the WEKA package is the excellent book that accompanies it (see references), which carefully explains how to implement your own algorithms into the WEKA system! The programs can be run either from the command line or a very tasty GUI system.

ⓘ *Torch*

This is a set of tools specialising in SVM implementations written in C++ and can be downloaded from the "SVMTorch" web site:

http://old-www.idiap.ch/learning/SVMTorch.html

This system is part of the new "Torch" package which is a general object orientated approach to machine learning much like WEKA. The Torch machine learning library is relatively new, and has a growing number of learning algorithms. The code is open source and the creators actively encourage any feedback and contributions. The code can be downloaded from:

http://www.torch.ch/

I would strongly recommend looking at each of these packages if you are struggling to understand how to implement learning algorithms. Your preference will probably be swayed by which programming language you find the easiest to develop in. If you find any other useful packages then please tell me and I will add them to this list.

References

ⓘ *Websites*

http://www.math.mtu.edu/~msgocken/intro/intro.html
This is a great website for an overview of what MATLAB is all about.
http://www.mathworks.com/search/
This is the website of the official creators of MATLAB.
http://www.purplemath.com/modules/slope.htm
This site has some helpful tips on basics of line plotting.

ⓘ *Books*

Netlab: Algorithms for Pattern Recognition, I. Nabney, Springer, Cost approx £24.50

Machine Learning, T. Mitchell, McGraw Hill. Cost approx £28.99
Sadly this book doesn't cover SVM's, but it has a very nice overview of the field of machine learning.

Data Mining Practical Machine Learning Tools and Techniques with Java Implementations, I. Witten, E. Frank, Morgan Kaufman. Cost approx £27.00
This is the book that accompanies the WEKA data mining package. This book has a little bit about SVM theory but really is useful for understanding the practicalities of the algorithms.

Pattern Classification, R. Duda, P. Hart, D. Stork, John Wiley & Sons. Cost approx £75.00
Very exspensive, and doesn't cover SVM's in depth. However this has a very in depth problem of pattern recognition from a statistics perspective.

An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, N. Cristianini, J. Shawe-Taylor, Cambridge University Press, Cost approx £32.00
As the name suggests this is very focused on SVM's but its not very easy to read! Does have some pseudo code for implementing an SVM without the need for an quadratic optimisation package.

Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond (Adaptive Computation and Machine Learning Series), B. Scholkopf, A. J. Smola, The MIT Press, Cost approx £39.00
A bit more rigorous in its explanations of the mathematics than the above book but quite complicated in places.

MATLAB Solution code

Below is what your code should look like. If you have not finished the exercise, do not cheat by looking at the solution, there is no point, you will not learn fro mthis workshop exercise. Cheaters never prosper! Try to work through the rest of the exercises and only use the solution if you really need to.

```
% Read in the data from a comma delimited file
DATA=dlmread('tennisweathernumeric.txt',',')
% Exctract the first attribute, and fourth example
ATTRIBUTE1 = DATA(:,1)
EXAMPLE4 = DATA(4,:)
% Find the number of examples and attributes used in the data
numOfExamples = size(DATA,1)
numOfAttributes = size(DATA,2)-1
% Extract the attribute matrix X and label vector Y
X = DATA(:,1:numOfAttributes)
Y = DATA(:,numOfAttributes+1)
% Split the data into no and yes play days
X_YES_DAYS = X(find(Y==1),:)
X_NO_DAYS = X(find(Y==0),:)

hold on
% Plot the yes days
plot(X_YES_DAYS(:,2),X_YES_DAYS(:,3),'or')
% Plot the no days
plot(X_NO_DAYS(:,2),X_NO_DAYS(:,3),'+b')

%Plot the separating hyperplane
HX = [60:85]
HY = (-0.45*HX)+122.27
plot(HX,HY,'k-')

% Add some nice titles to the graph
title('Days to play tennis?')
xlabel('Temperature')
ylabel('Humidity')
legend('Yes days','No days')

hold off
```