


MATLAB Workshop 2


An introduction to Support Vector Machine
implementations in MATLAB


SESSION PLAN	3
OVERVIEW:	3
LEARNING OUTCOMES	3
① MATRICES, VECTORS AND CONSTANTS	4
📄 COPY DATA SETS AND PROGRAMS INTO LOCAL DIRECTORY.....	4
⚡ CLASSES REPRESENTED AS ± 1 FOR SVM'S	4
⚡ RETURN OF THE HYPER-PLANE.....	4
⚡ WHY ARE OPTIMAL SEPARATING HYPER-PLANE SO GOOD?	5
⚡ THE SVM OPTIMISATION PROBLEM (PRIMARY SEPARABLE CASE)	5
① INTUITION OF THE MAIN TERM OF SVM OPTIMISATION	5
① INTUITION OF MAIN TERM AND SIDE CONSTRAINTS	6
⚡ THE QUADPROG FUNCTION.....	6
⚡ ENCODING SVM OPTIMISATION PROBLEMS AS MATRICES AND VECTORS	6
① CONCATENATION OF MATRICES	7
① CREATING DIAGONAL MATRICES	8
① CREATING IDENTITY MATRICES	8
⚡ USING QUADPROG TO SOLVE SVM OPTIMISATION PROBLEMS	8
📄 ADD THE CODE TO FIND THE SVM OPTIMAL SEPARATING HYPERPLANE	9
📄 PLOT THE OPTIMAL SEPARATING HYPERPLANE AND MARGINS	10
📄 PLOTTING NEW TEST DATA POINTS FOR CLASSIFICATION	11
THE FINISHED GRAPH PLOT	13
ADDITIONAL TASKS	14
① WEBSITES	15
① BOOKS	15
MATLAB SOLUTION CODE	16


Session Plan

This work shop is designed to revise the theory and practice of Support Vector Machines (SVM) as well as introduce important functionality in MATLAB. 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!

 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 we will continue our analysis of the tennisweather data set that we analysed in the previous MATLAB workshop. We will implement an SVM on the data and will demonstrate practically how to classify new examples. I will build upon the MATLAB programming techniques introduced in the previous session and introduce more advanced functionality where needed. This session is designed as an informal practical, 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 MATLAB 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 of SVM's
2. Look at how SVM's are applied to real life data
3. Learn more advanced manipulations and creation of matrices in MATLAB such as concatenation, creating diagonal and identity matrices.
4. How to classify new test data using SVM's
5. How to formulate the SVM primal separable problem as a quadratic optimisation problem.
6. How to implement SVM's in MATLAB using the quadprog function

Matrices, vectors and constants

I will try to use consistent notation to distinguish between matrices, vectors and constants throughout this document. I will usually use capital bold notation for matrices (e.g. **A, B, C, ...**), lower case bold for vectors (e.g. **a, b, c, ...**) and italicised for constants (e.g. *a, b, c, ...*). It is very important that you can distinguish between these mathematical objects!

Copy data sets and programs into local directory

As stated in the previous workshop we have created for you a shared directory of resources under the /CS/courses/CS392/ directory

Copy the files that you need for this session into your local directory

```
cp /CS/courses/CS392/ex2SVM/* .
```

You should find the files `firstsvm1.m`, `tennisweather2AttSVMTrain.txt` and `tennisweather2AttSVMTest.txt` in your home directory. The file `firstsvm1.m` is a slightly modified version of the program that you developed in workshop 1. Open this file for editing by typing `emacs firstsvm1.m &`

Classes represented as ± 1 for SVM's

The two dataset files that you have copied is the same data as you worked with in the previous workshop on assessing good 'tennis playing' days. Look at the data, you will notice that I have changed the classes to ± 1 so that it can be analysed by an SVM directly. It will become clearer later why it is useful to encode the classes in this manner. We have also separated the last example from the original data to make a separate *training* and *test* file.

Return of the hyper-plane

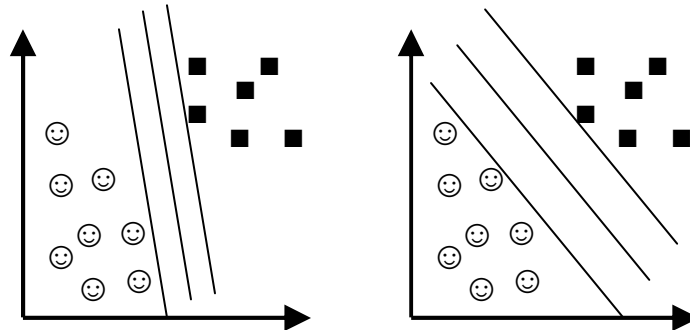
One of the most important concepts to get your head around is that of a hyper-plane in n dimensional space [see Class 2 Slide 8]! We consider our attribute vectors $\mathbf{x}^i = \langle x_1^i, \dots, x_n^i \rangle$ as the n dimensional vectors for the i th example in the data set. We have a training set of l examples (information pairs attribute vector + label) as $\{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^l, y^l)\}$ where the labels $y^i \in \{-1, 1\}$. The equation of a hyperplane H is

$$H = w_0x_0 + w_1x_1 + \dots + w_nx_n + b = (\mathbf{w} \cdot \mathbf{x}) + b = 0$$

using the dot product notation. The hyperplane is defined by the "weights" which are contained in the n dimensional vector w and the constant b .

🔑 Why are optimal separating hyper-plane so good?

If the data is separable then there are infinitely many possible separating hyperplanes that would split the data!



To get an understanding of why finding the largest separating hyper-plane is a good idea consider the above example. The separating hyper-plane on the right drives a larger wedge between the data, than the one on the left. We would hope that this decision rule would give better generalisation of the data than the other. The separating hyperplane (centre of the wedge) has the equation $H = 0$, whereas the margins hyperplanes (the upper and lower planes surrounding the wedge) have equations $H = \pm 1$.

🔑 The SVM optimisation problem (primary separable case)

As specified in the notes [Class 2 Page 12] our SVM in the primal separable case can be formulated as the following optimisation problem:

$$\begin{aligned} &\text{Minimise } \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) \\ &\text{Subject to the constraints} \\ &y^i(\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1 \text{ for } 1 \leq i \leq l \end{aligned}$$

① Intuition of the main term of SVM optimisation

Just why the hell do we want to minimise the term $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w})$? Well without going into too much detail to calculate the distance of the i th example's attribute vector \mathbf{x}^i from the hyperplane H like so:

$$d(\mathbf{x}^i, H) = \frac{\mathbf{w} \cdot \mathbf{x}^i}{\|\mathbf{w}\|}$$

There for if we are maximising the weight vector \mathbf{w} then we are minimising $\|\mathbf{w}\|$ which is in turn maximising the distance of the example \mathbf{x}^i from the hyperplane (this maximising the margin). Note: we are minimising a vector \mathbf{w} therefore we are actually minimising n values!

Intuition of main term and side constraints

In brief the side constraints are making sure the *positive* examples \mathbf{x}^i (which have label $y^i = 1$) lie above the upper margin, and *negative* examples \mathbf{x}^i (which have label $y^i = -1$) lie below the lower margin. For example $y^i = 1$ gives $\mathbf{w} \cdot \mathbf{x}^i + b \geq 1$ which means example \mathbf{x}^i lies above the upper margin hyperplane $H = 1$. Alternatively if we have a negative example $y^i = -1$ then we get $-(\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1 \Rightarrow (\mathbf{w} \cdot \mathbf{x}^i + b) \leq -1$ therefore the example \mathbf{x}^i lies below the hyperplane $H = -1$. Note: We have a side constraint for every example in the data set, therefore we have l side constraints making sure examples lie on the right sides of the hyperplanes! In short we are optimising n variables with l side constraints.

The quadprog function

To run an SVM in MATLAB you will have to use the quadprog function to solve the optimisation problem. For the CS392 course we will use this optimisation tool like a black box, and not give consideration to how it finds solutions (this is *not* the CS349 Computational Optimisation course). You must try and understand the input arguments and the output this function returns [see Class 2a Page 2]. The quadprog function solves generic quadratic programming optimisation problems of the form:

$$\text{Minimise } \frac{1}{2} \mathbf{x}' \mathbf{H} \mathbf{x} + \mathbf{f} \mathbf{x}$$

Subject to the constraints

$$\mathbf{A} \mathbf{x} \leq \mathbf{c}$$

This minimisation problem solves for the vector \mathbf{x} . Do not confuse the notation with that of the SVM (\mathbf{x} above is not the same as the attribute vectors) we could use any letters to denote these matrices and vectors! The quadprog program does not only just solve SVM problems, it just happens that SVM's can be formulated as a quadratic programming problem (which quadprog can solve easily). The first step to solving our problem, is to encode it using the matrices \mathbf{H} , \mathbf{A} and vectors \mathbf{f} , \mathbf{c} as we shall see in the next section! Once we have created the matrices and vectors ($\mathbf{H}, \mathbf{A}, \mathbf{f}, \mathbf{c}$) quadprog function can be used like so:

$$\mathbf{x} = \text{quadprog}(\mathbf{H}, \mathbf{f}, \mathbf{A}, \mathbf{c})$$

which will return the optimal values into vector \mathbf{x} .

Encoding SVM optimisation problems as matrices and vectors

In this section we will explain how to encode the SVM optimisation problem using the matrices and vector formulation that quadprog likes as input. We can encode the main SVM

optimisation term $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) = \frac{1}{2}(w_1 w_1 + w_2 w_2 + \dots + w_n w_n) \rightarrow \min$ like so:

$$\underbrace{\frac{1}{2}(\mathbf{w} \quad b) \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix}}_{\text{Be careful with this notation, we are implicitly concatenating vectors like so}} \rightarrow \min \quad \frac{1}{2} \begin{pmatrix} w_1 & w_2 & \dots & w_n & b \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b \end{pmatrix} \rightarrow \min$$

Similarly we can encode the l side constraints $y^i(\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1$ for $1 \leq i \leq l$ by expanding the notation [see Class 2a, Page 3]

Note: You should always check that the dimensions of your matrices and vectors match up so that the multiplications work as you want. You read the dimensions as the number of rows \times number of columns (as shown with the red brackets and lines on the left).

$$\underbrace{\begin{pmatrix} y_1 & 0 & 0 & 0 \\ 0 & y_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & y_l \end{pmatrix}}_{l \times l} \underbrace{\begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 & 1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 & 1 \\ x_1^3 & x_2^3 & \cdots & x_n^3 & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ x_1^l & x_2^l & \cdots & x_n^l & 1 \end{pmatrix}}_{l \times (n+1)} \underbrace{\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b \end{pmatrix}}_{(n+1) \times 1} \geq \underbrace{\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}}_{l \times 1}$$

$l \times (n+1) \quad \xrightarrow{\quad} \quad l \times 1$

The dimensions of matrices must match in the sense that the number of columns of the first matrix must be the same as the number of rows in the second matrix (e.g. the multiplication of two matrices $(l \times n) \times (n \times m)$ would give a new matrix with dimensions $(l \times m)$).



Use the same technique of checking that dimensions of matrices match on the main optimisation term $\frac{1}{2}(\mathbf{w} \quad b) \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} \rightarrow \min$ mentioned earlier. What are the dimensions of the resulting matrix?

Concatenation of matrices

One very useful (but confusing) technique is to concatenate matrices. Using the *square brackets* “[]” to signify we are constructing a matrix or vector. If we separate elements in the matrix by a *space* “ ” we signify continuing a row; if we separate elements in the matrix using a *semi colon* “;” we signify a new column is about to start.

For example if we had four vectors $A = [1 \ 2 \ 3]$, $B = [4 \ 5; \ 7 \ 8]$, $C = [6; \ 9]$ we create the following matrices in MATLAB

$$\mathbf{A} = (1 \ 2 \ 3), \quad \mathbf{B} = \begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 6 \\ 9 \end{pmatrix}$$

If we use the concatenation to create a 3×3 matrix using notation $\mathbf{D} = [\mathbf{A}; \ \mathbf{B} \ \mathbf{C}]$

$$\mathbf{D} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

This can be interpreted as make A the first row, then make B and C the second row of the new matrix D. You must be very careful that your dimensions match (use the size function to check).

Creating diagonal matrices

To create a square matrix with specific diagonal elements MATLAB has created the very useful `diag` function. This function takes a row ($d \times 1$) or column ($1 \times d$) vector as an input and places its elements as the diagonal elements of a $d \times d$ matrix.

For example if we defined a vector $A = [1 \ 2 \ 3 \ 4]$ we could create a matrix $B = \text{diag}(A)$ like so:

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Notice that all the off diagonal elements are zero, which is very useful when formulating multiplications in matrix form!

Creating identity matrices

One very useful matrix is the identity matrix which is a square matrix with diagonal elements = 1 and off diagonal elements = 0. We can create identity matrices of any square dimension d using the function `eye(d)`, and have the useful property of not having any effect when multiplied by another matrix.

For example to create a 3×3 identity matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ by typing `eye(3)`.

Using `quadprog` to solve SVM optimisation problems

As mentioned earlier, to use `x=quadprog(H, f, A, c)` to solve our SVM optimisation problem we must construct the matrices and vectors H , f , A and c like so:

1) The vector $\mathbf{x} = \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix}$ returned by `quadprog` is our weight vector that we are seeking that defines the optimal hyperplane.

2) Our main SVM optimisation term $\frac{1}{2}(\mathbf{w} \ b) \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} \rightarrow \min$ can be formulated with

$\mathbf{H} = \begin{pmatrix} I_n & 0 \\ 0 & 0 \end{pmatrix}$ where I_n is a $n \times n$ identity matrix, and $\mathbf{f} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ is a $(n+1)$ column vector.

Sticking the extra 0 in the bottom right corner of the identity matrix makes sure we do not minimise only the weight vector \mathbf{w} and *not* the constant b .

The matrix H can be constructed using the `eye(n+1)` function and f using `zeros(n+1, 1)` which is a $n+1$ column vector of zeros.

3) Finally our constraints are constructed using the matrices **A** and **c** like so:

$$\mathbf{A} = -\text{diag}(\mathbf{Y}) [\mathbf{X} \text{ ones}(l, 1)] \text{ and } \mathbf{c} = -\text{ones}(l, 1)$$

Note: we put an *extra minus sign* in front of the matrix **A** and vector **c** because quadprog solves optimisation problems which are of the form $\mathbf{Ax} \leq \mathbf{c}$ and our SVM constraints

$$\text{diag} \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} \begin{pmatrix} \mathbf{x}^1 & 1 \\ \vdots & \vdots \\ \mathbf{x}^l & 1 \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} \leq (-1) \text{ are of the opposite equality } -\mathbf{Ax} \geq -\mathbf{c}.$$



Add the code to find the SVM optimal separating hyperplane

In this exercise you will add the following code to the program file “firstsvm1.m” that you should have open in your user area using emacs (or any other editor that you prefer!).

Add the code in underneath the comments:

```
%%% Code to find the SVM hyperplane will go here! %%%
```

Run the program by opening up MATLAB and typing firstsvm1 in the *Command Window*.



Create the main SVM optimisation term matrices **H** and **f** using the variable numofAttributes



Create the SVM side constraint matrices **A** and **c** using labels vector **Y**, augmented feature matrix **Z** and numofExamples



Plug all these matrices into quadprog and return the values into a weight vector **w**.

***Plot the optimal separating hyperplane and margins***

In this exercise you will add the following code to the program file “firstsvm1.m” that will plot the separating hyperplanes and respective margins to the data. Make sure you insert your code in between the comments provided to make your code easy to read! Now that quadprog has found the optimal weight vector w you will need to extract the relevant components to plot the straight line. (Refer to previous workshop for detail on graph plotting).



Create a range of values $X1$ from 60 to 95 using the colon notation



Create variables and extract the weights w_1 , w_2 and b from the vector w



Rearrange $w_1x_1 + w_2x_2 + b = 0$ to get x_2 in terms of x_1 , w_1 and w_2 . This is the equation of the separating hyper-plane (this is the centre of the wedge or sandwich filling). Note: You are trying to get the equation in the $y = mx + c$.



Rearrange $w_1x_1 + w_2x_2 + b = 1$ and $w_1x_1 + w_2x_2 + b = -1$ in the same way. This is the equation of the margin hyper-planes (edges of the wedge)



Using the previous questions create a vector $Y1$ to plot the corresponding vertical values for the separating hyper-plane using the range $X1$ created earlier.

? Plot the separating hyper-plane using a black line.

? Create a further two plots in a similar manner for the margins using vectors YUP and YLOW to define the upper and lower margins. Plot the margins using a magenta dotted line.



Plotting new test data points for classification

Once we have plotted the separating hyperplane we must remember why we were doing this whole messy process in the first place, to *classify new test examples*. In this section we will add new test data points to our graph (your plot should look very close to that on page 13).

? Read in the data for new test examples into a matrix TESTDATA from “tennisweather2AttSVMTest.txt” which is a comma delimited file using the dlmread function.

? Plot the new test using a green pentagram.

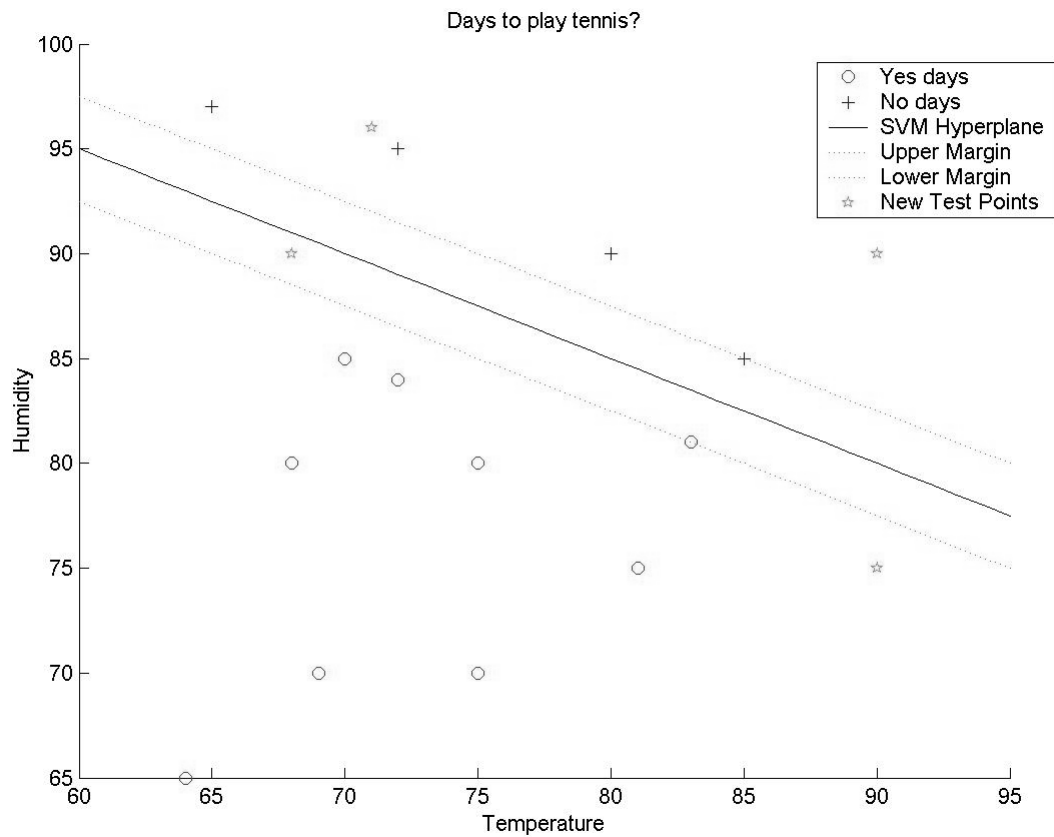
? Edit the legend to label the separating hyperplanes, margins and new test data

? Add some new data points $x_1 = (68,90)$, $x_1 = (90,75)$ and $x_1 = (90,90)$ to the test file `tennisweather2AttSVMTest` and re-plot the data.

? What are the corresponding labels y_1 , y_2 and y_3 to the new test points you just added?

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.

☞ 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 .

☞ Can you think of a way of classifying data without having to plot the separating hyperplane just using the weight vector w .

☞ Try to generate an SVM classifier using three of the dimensions in the data.

References

Websites

<http://www.math.ohiou.edu/~vnnguyen/papers/Introduction%20to%20Support%20Vector%20Machines.pdf>

This pdf document gives a tutorial on SVM's, there are many others out there!

<http://www.kernel-machines.org/>

This website has lots of data, software, and papers on kernel based learning algorithms such as SVM's.

<http://www.ics.uci.edu/~mllearn/MLRepository.html>

This website contains lots of freely downloadable data sets (including the tennisweather data set you have worked with!) and has a mixture of pattern recognition and regression problems.

<http://www.mlnet.org/>

This website has many resources for people in machine learning including datasets, jobs, links to research groups etc.

Books

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 from this 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 training data from a comma delimited file
TRAINDATA=dlmread('tennisweather2AttSVMTrain.txt','')
% Extract the first attribute, and fourth example
ATTRIBUTE1 = TRAINDATA(:,1)
EXAMPLE4 = TRAINDATA(4,:)
% Find the number of examples and attributes used in the data
numOfExamples = size(TRAINDATA,1)
numOfAttributes = size(TRAINDATA,2)-1
% Extract the attribute matrix X and label vector Y
X = TRAINDATA(:,1:numOfAttributes)
Y = TRAINDATA(:,numOfAttributes+1)
% Split the data into no and yes play days
X_YES_DAYS = X(find(Y==1),:)
X_NO_DAYS = X(find(Y==-1),:)

hold on
% Plot the yes days
plot(X_YES_DAYS(:,1),X_YES_DAYS(:,2),'or')
% Plot the no days
plot(X_NO_DAYS(:,1),X_NO_DAYS(:,2),'+b')

%%% Code to find the SVM hyperplane will go here! %%%
H=eye(numOfAttributes+1)
H(numOfAttributes+1,numOfAttributes+1)=0
f=zeros(numOfAttributes+1,1)

Z = [X ones(numOfExamples,1)]
A=-diag(Y)*Z
c=-1*ones(numOfExamples,1)

w=quadprog(H,f,A,c)

%%% Code to plot the SVM separating hyperplane will go here! %%%

X1=[60:95]
w1=w(1,1);
w2=w(2,1);
b=w(3,1);
Y1=-(w1*X1+b)/w2; %Seperating hyperplane

plot(X1,Y1,'k-')

%%% Code to plot the SVM margins goes here! %%%

YUP=(1-w1*X1-b)/w2; %Margin

plot(X1,YUP,'m:')

YLOW=(-1-w1*X1-b)/w2; %Margin

plot(X1,YLOW,'m:')
```



```
%%% Code to plot the new test point goes here %%%

% Read in the training data from a comma delimited file
TESTDATA=dlmread('tennisweather2AttSVMTest.txt','')
% Plot the new test point(s)
plot(TESTDATA(:,1),TESTDATA(:,2),'gp')

% Add some nice titles to the graph
title('Days to play tennis?')
xlabel('Temperature')
ylabel('Humidity')
legend('Yes days','No days','SVM Hyperplane','Upper Margin','Lower
Margin','New Test Points')

hold off
```