Programmable Embedded Systems

Assignment 2

Submitted by

**Pratyush Jaiswal**

**18EE35014**

Design the android app to classify the motion of cellphones in the x-axis, y-axis, and z-axis direction, and circular motion using Kalman filter and SVM. Submit the codes, results, and a small report in a pdf (Demo will be taken for everyone later on).

# Introduction

Support Vector machines can be defined as systems that use hypothesis space of a linear function in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. Support vector machine was initially popular with the NIPS community and now is an active part of machine learning research around the world.

SVM becomes famous when using pixel maps as input; it gives accuracy comparable to sophisticated neural networks with elaborated features in a handwriting recognition task. It is also being used for many applications, such as handwriting analysis, face analysis and so forth, especially for pattern classification and regression-based applications.

The foundations of Support Vector Machines (SVM) have been developed by Vapnik and gained popularity due to many promising features such as better empirical performance. The formulation uses the Structural Risk Minimization (SRM) principle, which has been shown to be superior, to the traditional Empirical Risk Minimization (ERM) principle, used by conventional neural networks. SRM minimizes an upper bound on the expected risk, whereas ERM minimizes the error on the training data. It is this difference that equips SVM with a greater ability to generalize, which is the goal in statistical learning. SVMs were developed to solve the classification problem, but recently they have been extended to solve regression problems.

# Statistical Learning Theory

The statistical learning theory provides a framework for studying the problem of gaining knowledge, making predictions, making decisions from a set of data. In simple terms, it enables the choosing of the hyper plane space such a way that it closely represents the underlying function in the target space.

In statistical learning theory the problem of supervised learning is formulated as follows. We are given a set of training data {(**x**1,y1)... (**x**l,yl)} in Rn × R sampled according to unknown probability distribution P(**x**,y), and a loss function V(y,f(**x**)) that measures the error, for a given **x**, f(**x**) is "predicted" instead of the actual value y. The problem consists in finding a function f that minimizes the expectation of the error on new data that is, finding a function f that minimizes the expected error: ∫V(y,f(**x**)) P(**x**,y) d**x** dy [6] In statistical modeling we would choose a model from the hypothesis space, which is closest (with respect to some error measure) to the underlying function in the target space. More on statistical learning theory can be found on introduction to statistical learning theory [7].

# Learning and Generalization

Early machine learning algorithms aimed to learn representations of simple functions. Hence, the goal of learning was to output a hypothesis that performed the correct classification of the training data and early learning algorithms were designed to find such an accurate fit to the data [8]. The ability of a hypothesis to correctly classify data not in the training set is known as its generalization. SVM performs better in term of not over generalization when the neural networks might end up over generalizing easily [11]. Another thing to observe is to find where to make the best trade-off in trading complexity with the number of epochs; the illustration brings to light more information about this.

The below illustration is made from the class notes.

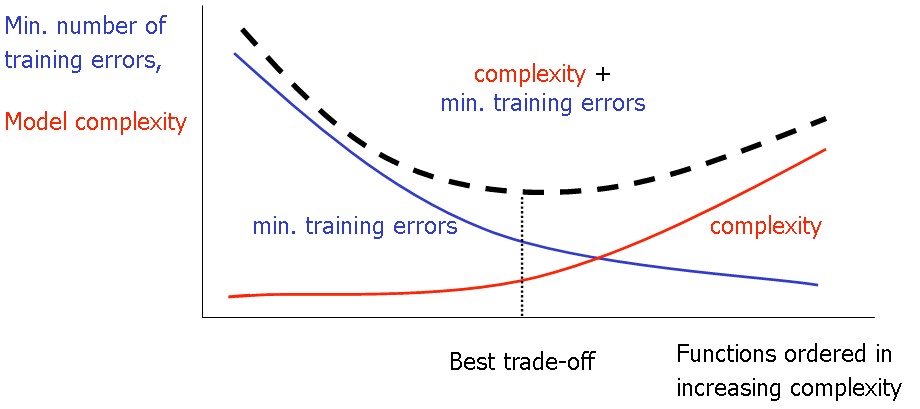


Figure 1: Number of Epochs Vs Complexity. [8][9][11]

**Introduction to SVM: Why SVM?**

Firstly working with neural networks for supervised and unsupervised learning showed good results while used for such learning applications. MLP’s uses feed forward and recurrent networks. Multilayer perceptron (MLP) properties include universal approximation of continuous nonlinear functions and include learning with input-output patterns and also involve advanced network architectures with multiple inputs and outputs [10].

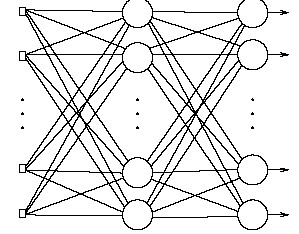
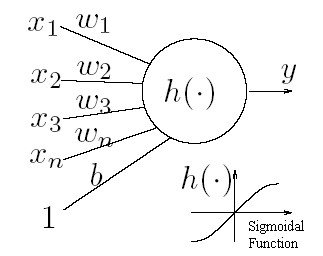


Figure 2: a] Simple Neural Network b]Multilayer Perceptron. [10][11]. These are simple visualizations just to have a overview as how neural network looks like.

There can be some issues noticed. Some of them are having many local minima and also finding how many neurons might be needed for a task is another issue which determines whether optimality of that NN is reached. Another thing to note is that even if the neural network solutions used tends to converge, this may not result in a unique solution [11]. Now let us look at another example where we plot the data and try to classify it and we see that there are many hyper planes which can classify it. But which one is better?

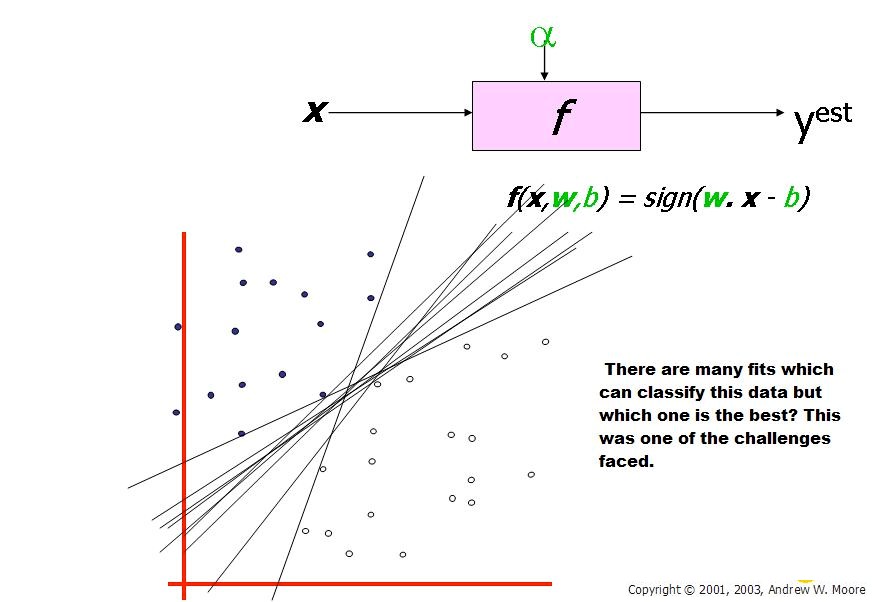


Figure 3: Here we see that there are many hyper planes which can be fit in to classify the data but which one is the best is the right or correct solution. The need for SVM arises. (Taken Andrew W. Moore 2003) [2]. Note the legend is not described as they are sample plotting to make understand the concepts involved.

From above illustration, there are many linear classifiers (hyper planes) that separate the data. However only one of these achieves maximum separation. The reason we need it is because if we use a hyper plane to classify, it might end up closer to one set of datasets compared to others and we do not want this to happen and thus we see that the concept of maximum margin classifier or hyper plane as an apparent solution. The next illustration gives the maximum margin classifier example which provides a solution to the above mentioned problem [8].

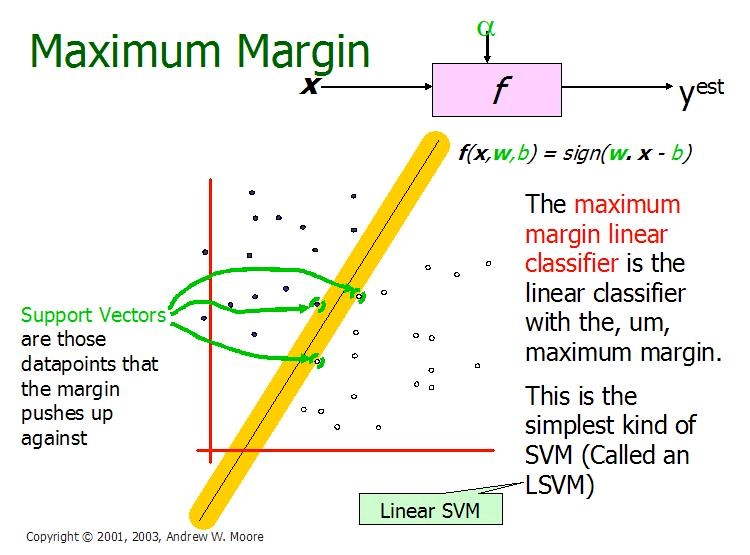
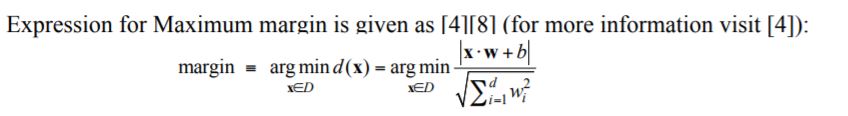


Figure 4: Illustration of Linear SVM. ( Taken from Andrew W. Moore slides 2003) [2]. Note the legend is not described as they are sample plotting to make understand the concepts involved.



The above illustration is the maximum linear classifier with the maximum range. In this context it is an example of a simple linear SVM classifier. Another interesting question is why maximum margin? There are some good explanations which include better empirical performance. Another reason is that even if we’ve made a small error in the location of the boundary this gives us least chance of causing a misclassification. The other advantage would be avoiding local minima and better classification. Now we try to express the SVM mathematically and for this tutorial we try to present a linear SVM. The goals of SVM are separating the data with hyper plane and extend this to non-linear boundaries using kernel trick [8] [11]. For calculating the SVM we see that the goal is to correctly classify all the data. For mathematical calculations we have,

1. If Yi= +1; *wxi* +*b*≥1
2. If Yi= -1; wxi + b ≤ 1
3. For all i; yi (wi + b) ≥ 1

In this equation x is a vector point and w is weight and is also a vector. So to separate the data [a] should always be greater than zero. Among all possible hyper planes, SVM selects the one where the distance of hyper plane is as large as possible. If the training data is good and every test vector is located in radius r from training vector. Now if the chosen hyper plane is located at the farthest possible from the data [12]. This desired hyper plane which maximizes the margin also bisects the lines between closest points on convex hull of the two datasets. Thus we have [a], [b] & [c].

wx+b=1

wx+b=0

wx’+b=

-

1

Figure 5: Representation of Hyper planes. [9]

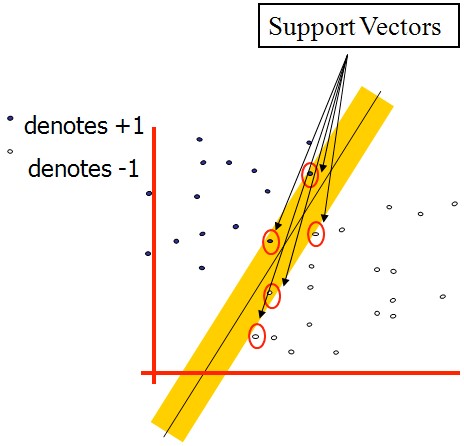
Distance of closest point on hyperplane to origin can be found by maximizing the x as x is on the hyper plane. Similarly for the other side points we have a similar scenario. Thus solving and subtracting the two distances we get the summed distance from the separating hyperplane to nearest points. Maximum Margin = M = 2 / ||w||

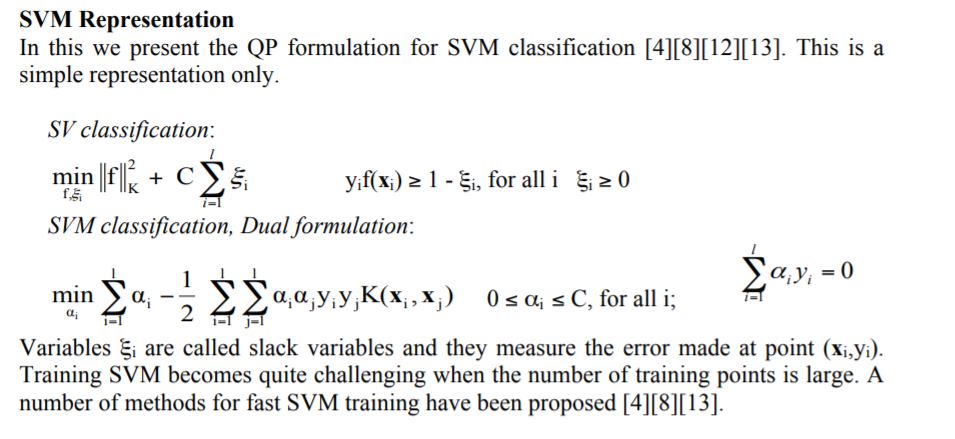
Now maximizing the margin is same as minimum [8]. Now we have a quadratic optimization problem and we need to solve for w and b. To solve this we need to optimize the quadratic function with linear constraints. The solution involves constructing a dual problem and where a Langlier’s multiplier *αi* is associated. We need

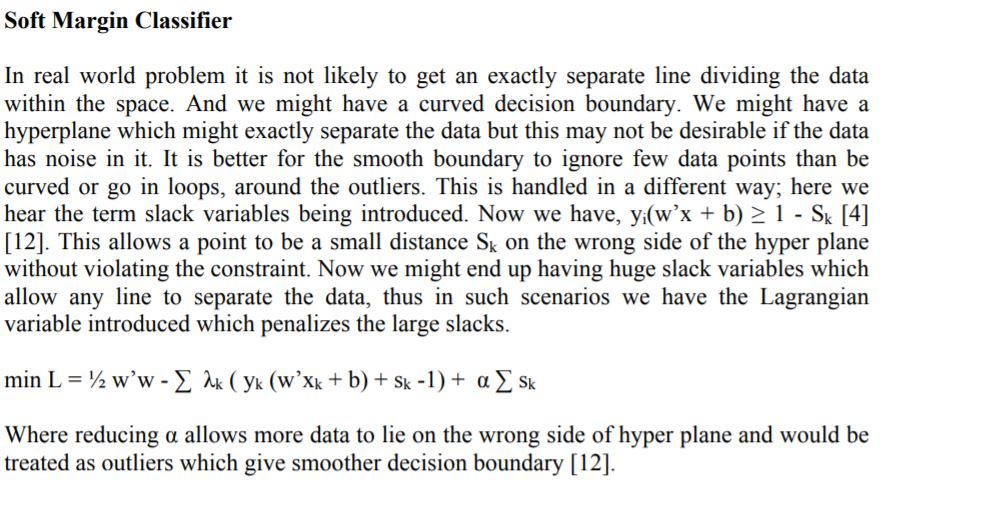
to find w and b such that Φ (w) =½ |w’||w| is minimized; And for all {(xi, yi)}: *yi* (w \* xi + *b*) ≥ 1.

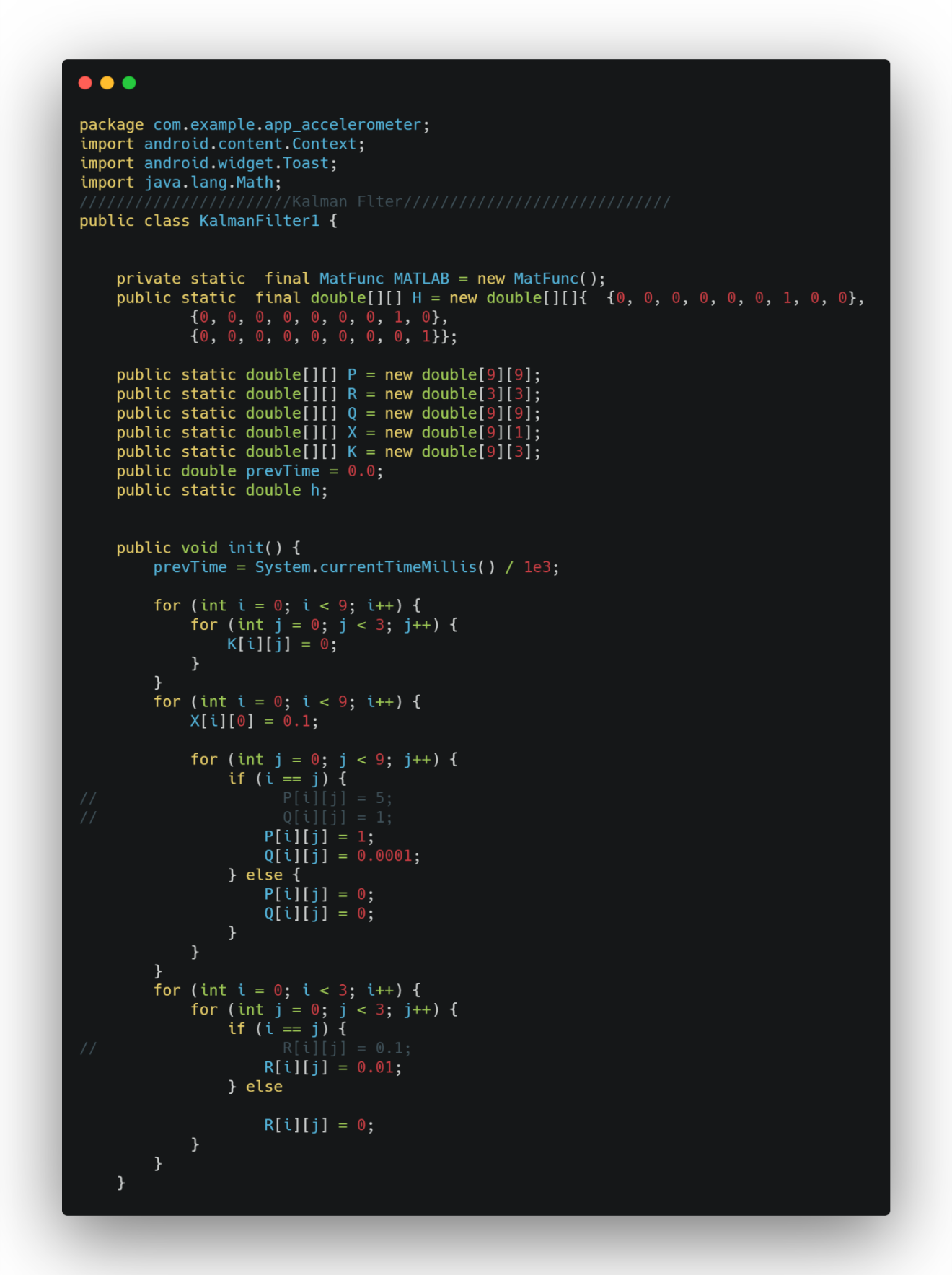
Now solving: we get that w =Σ*αi \** xi; *b*= *yk*- w \*xk for any xk such that *αk*≠0

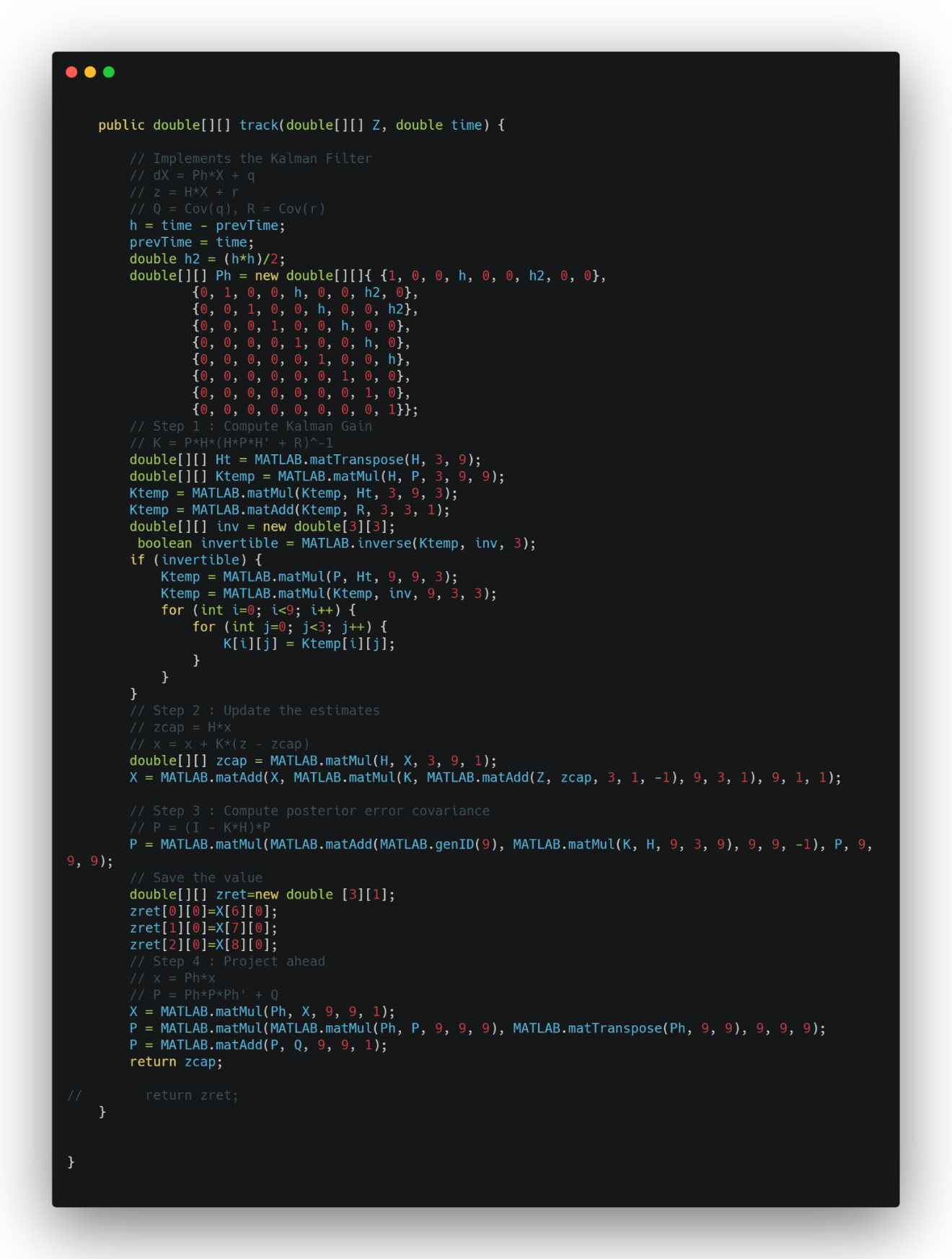
Now the classifying function will have the following form: *f*(x) = Σ*αi yi* xi \* x + *b*

Figure 6: Representation of Support Vectors (Copyright © 2003, Andrew W. Moore)[2]





**Kalman Filter Code**

****

**MainActivity Code**





**Matlab Code to generate plots**

% Matlab Code to generate plots:

load testmat\_pratyush;

% Initial Guess

% state

x=randn(9,1);

% Covariance

P=eye(9);

% Process Noise covariance Q

Q=0.0007\*eye(9);

% Measurement Noise covariance R

R=0.07\*eye(3);

X=[]; Z=[]; Gain=[]; Err=[];

N=length(t);

% Construct H matrix

H=[zeros(3,6) eye(3)];

for n=1:N-1

h=t(n+1)-t(n);

h2=h^2/2;

% Construct Phi matrix

phi=[eye(3) h\*eye(3) h2\*eye(3)

         zeros(3) eye(3)  h\*eye(3)

         zeros(3) zeros(3)    eye(3)];

%  Compute the Kalman Gain K

K=P\*H'\*inv(H\*P\*H'+R);

% Update the states

z=a(n,:)';

err=(z-H\*x);

x=x+K\*err;

% Update the P

P=(eye(9)-K\*H)\*P;

%Save the values

X=[X;x(:)'];

Z=[Z;z(:)'];

Gain=[Gain;K(:)'];

Err=[Err;err(:)'];

% Project Ahead

x=phi\*x;

P=phi\*P\*phi'+Q;

end

ae=[X(:,7:9);a(end,:)];

subplot(311)

plot([a(:,1) ae(:,1)])

grid on

subplot(312)

plot([a(:,2) ae(:,2)])

grid on

subplot(313)

plot([a(:,3) ae(:,3)])

grid on

shg

**Matlab Code for training and plotting from generated data**

clc;

clear all;

axyzr = readmatrix('Filtered\_Data\_run.txt');

axyzr2 = readmatrix('Filtered\_Data\_run2.txt');

axyzw = readmatrix('Filtered\_Data\_walk.txt');

axyzw2 = readmatrix('Filtered\_Data\_walk2.txt');

axyzs = readmatrix('Filtered\_Data\_still.txt');

axyzd = readmatrix('Filtered\_Data\_down1.txt');

axyzd2 = readmatrix('Filtered\_Data\_down2.txt');

axyzd3 = readmatrix('Filtered\_Data\_down3.txt');

axyzu = readmatrix('Filtered\_Data\_up1.txt');

axyzu2 = readmatrix('Filtered\_Data\_up2.txt');

axyzu3 = readmatrix('Filtered\_Data\_up3.txt');

ar = [];

aw = [];

as = [];

ad = [];

au = [];

for i=1:size(axyzr, 1)

ar = [ar; norm(axyzr(i,:), 2)];

aw = [aw; norm(axyzw(i,:), 2)];

as = [as; norm(axyzs(i,:), 2)];

end

for i=1:size(axyzr2, 1)

ar = [ar; norm(axyzr2(i,:), 2)];

aw = [aw; norm(axyzw2(i,:), 2)];

end

for i=1:size(axyzd, 1)

ad = [ad; norm(axyzd(i,:), 2)];

au = [au; norm(axyzu(i,:), 2)];

end

for i=1:size(axyzd2, 1)

ad = [ad; norm(axyzd2(i,:), 2)];

au = [au; norm(axyzu2(i,:), 2)];

end

for i=1:size(axyzd3, 1)

ad = [ad; norm(axyzd3(i,:), 2)];

au = [au; norm(axyzu3(i,:), 2)];

end

% ar = ar - mean(ar);

% aw = aw - mean(aw);

% as = as - mean(as);

% ad = ad - mean(ad);

% au = au - mean(au);

XIr = 0:40/100:40;

size(XIr)

[Fr, XIr] = ksdensity(ar, XIr);

[Fw, XIw] = ksdensity(aw, XIr);

[Fs, XIs] = ksdensity(as, XIr);

[Fd, XId] = ksdensity(ad, XIr);

[Fu, XIu] = ksdensity(au, XIr);

Fr = Fr.\*(1/sum(Fr));

Fw = Fw.\*(1/sum(Fw));

Fs = Fs.\*(1/sum(Fs));

Fd = Fd.\*(1/sum(Fd));

Fu = Fu.\*(1/sum(Fu));

for i=1:size(XIr, 2);

Fr(i) = Fr(i)+eps;

Fw(i) = Fw(i)+eps;

Fs(i) = Fs(i)+eps;

Fd(i) = Fd(i)+eps;

Fu(i) = Fu(i)+eps;

if Fr(i) == 0 | Fw(i) == 0 | Fs(i) == 0 | Fd(i) == 0 | Fu(i) == 0

disp("Warning");

end

end

markersize = 10;

linewidth = 1;

fontsize = 18;

figure(1)

hold on

plot(XIr, Fr, 'LineWidth', linewidth ,'MarkerSize',markersize)

plot(XIw, Fw, 'LineWidth', linewidth ,'MarkerSize',markersize)

plot(XIs, Fs, 'LineWidth', linewidth ,'MarkerSize',markersize)

plot(XId, Fd, 'LineWidth', linewidth ,'MarkerSize',markersize)

plot(XIu, Fu, 'LineWidth', linewidth ,'MarkerSize',markersize)

legend('Running', 'Walking', 'Still', 'Going down', 'Going up', 'FontSize', fontsize);

xlabel('Acceleration m/s^2')

ylabel('Probability')

title('Standard Probability Distributions')

hold off

fileID = fopen('ExtractedPDF\Run.txt', 'w');

for i=1:length(Fr)

fp = Fr(i);

fprintf(fileID, "%e, ", fp);

end

fclose(fileID);

fileID = fopen('ExtractedPDF\Walk.txt', 'w');

for i=1:length(Fw)

fp = Fw(i);

fprintf(fileID, "%e, ", fp);

end

fclose(fileID);

fileID = fopen('ExtractedPDF\Still.txt', 'w');

for i=1:length(Fs)

fp = Fs(i);

fprintf(fileID, "%e, ", fp);

end

fclose(fileID);

fileID = fopen('ExtractedPDF\Up.txt', 'w');

for i=1:length(Fu)

fp = Fu(i);

fprintf(fileID, "%e, ", fp);

end

fclose(fileID);

fileID = fopen('ExtractedPDF\Down.txt', 'w');

for i=1:length(Fd)

fp = Fd(i);

fprintf(fileID, "%e, ", fp);

end

fclose(fileID);

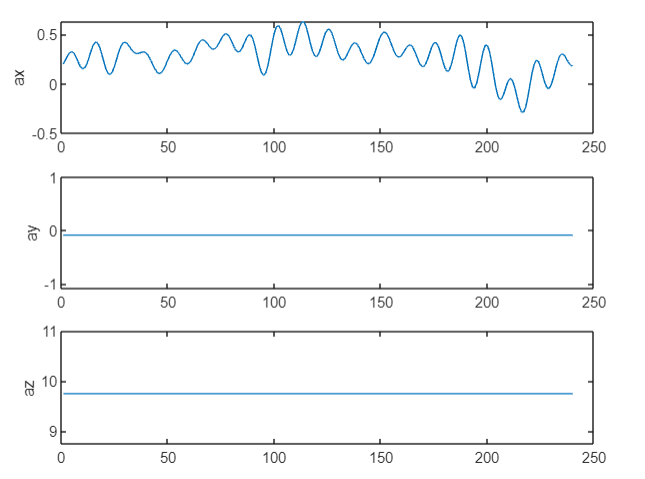
Taking fourier transform of accelerometer data along each axis

Derive the magnitudes of fourier coefficients using log operation

Threshold the Fourier Space data using magnitude as criteria

Taking the Inverse fourier transform

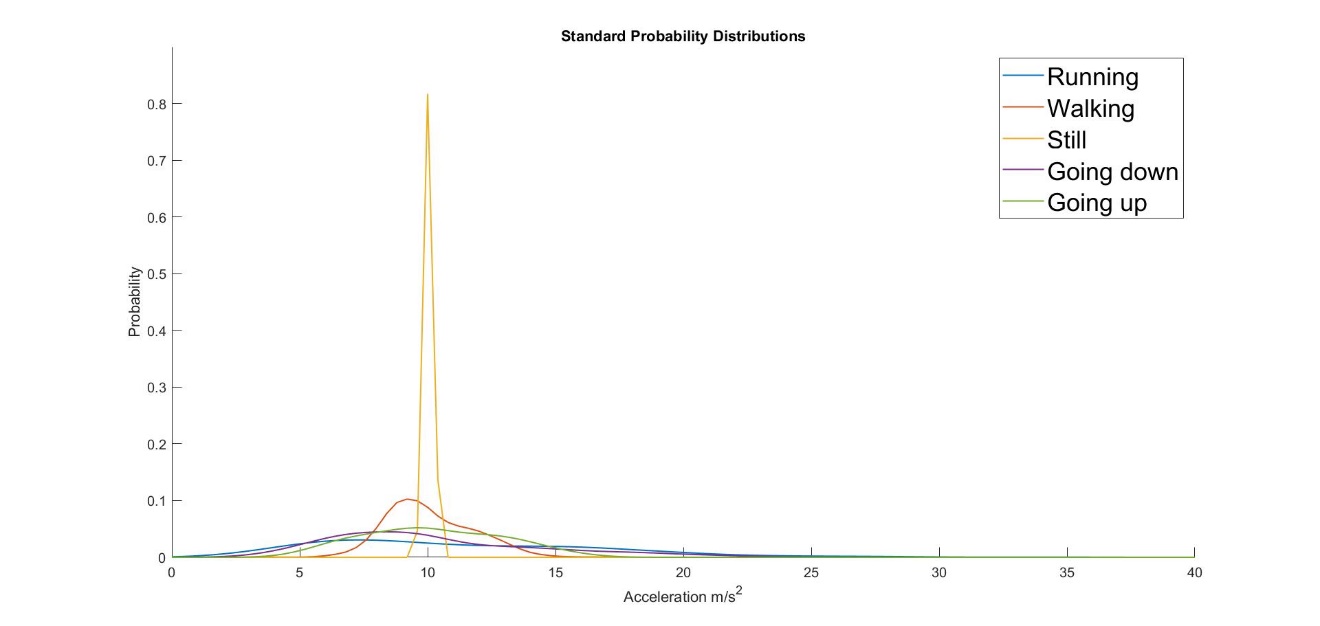
For x-axis motion :



Next the SVM is designed and solved using quadprog

|  |
| --- |
| %% Design SVM  H = zeros(n,n);  for i=1:n     for j=i:n         H(i,j) = y(i)\*y(j)\*x(:,i)'\*x(:,j);         H(j,i) = H(i,j);     end  end  f = -ones(n,1);  Aeq=y;  beq=0;  lb=zeros(n,1);  ub=C\*ones(n,1);  Alg{1}='trust-region-reflective';  Alg{2}='interior-point-convex';  options=optimset('Algorithm',Alg{2},...     'Display','off',...     'MaxIter',20);  alpha=quadprog(H,f,[],[],Aeq,beq,lb,ub,[],options)';  AlmostZero=(abs(alpha)<max(abs(alpha))/1e5);  alpha(AlmostZero)=0;  S=find(alpha>0 & alpha<C);  w=0;  for i=S     w=w+alpha(i)\*y(i)\*x(:,i);  end  b=mean(y(S)-w'\*x(:,S)); |

**Results**

****