Matlab Signal & Image Processing

Feature

1 2 3 4 ..

Sample 1
2
3
:

N × M

N → No. of Sample

M → No. of features

auto cylinders engine . . .

* 1
2
3
4
5

* 398 different cars (rows)

* 8 features (columns)

Load data

```
>> mpg_in = xlsread ('MPG.xlsm');
```

```
>> mpg_in (1,:)
```
→ 1st row.

Remove unnecessary

```
>> mpg = mpg_in (:, 2:8);
```
→ 1st row no need.

>> By default: xlsreads – ignores character data

```
[Num, Txt, Raw] = xlsread ('MPG.xlsm');
```

Raw → unprocessed

Txt → with text

Num → only numerical

```
[~, ~, raw] = xlsread ('hello.xlsm');
```

~ ~ → No need

mpg_raw(1,:)
ans =

{'car #'} {'# of cylinders'} → cells

---

* cell: matrix with different data types

* load fisheriris (dataset: flower, petal, sepal data').
* fisheriris → 150×4 cell array.

species(1) → cell array.

{'setosa'}

special {1} → content

'setosa'

Statistical data analysis

* mean, S.D, confidence intervals, hypothesis testing

>> load fisheriris → data(name)
>> V = meas(1:50, 1);
>> Vmean = mean(V) → 5.0000
>> Vmedian = median(V) → 5
>> Vstd = std(V) → 0.3525
>> Vmax = max(V) → 5.0000
>> Vmin = min(V) = 4.3000
>> Vmode = mode(V) → 5 (frequent values)

[vmin, vmind] = min(V)
↳ Index of minimum element.

Sort

```
Vsort = sort(V);
Vsort(1) = 4.3000
Vsort(end) = 5.0000
Vsort(end + 1/2) → 5
```

↳ may not be an integer. round / ceil / floor

Vsort(25) = Vsort(26) = 5

» Vsort(13) = 4.8000

↳ 25 th entry %

Boxplots – Visualize

boxplot(V) → shows mean, range, Total range of data. (outliers)

multiple datasets

```
V2 = mean(5) : 100, 1);
boxplot([V1  V2])
```

→ one data V_1 different from V_2

How?

Common Statistical hypothesis test – Student's t test

* t-test → developed by William Gosset (under pseudonym – Student)
* batches of stout – different results.

↳ most widely used in all scientific test.

matlab

generates p value (prob: incorrect

0.05 → Threshold (Concluding significant difference exists b/w the computations)

$h=1$ if $p < 0.05$ (rejects null hypothesis)

$h=0$ if $p > 0.05$ can't reject " "

```
>> [h, p] = ttest2(v1, v2)

h =
   1                          → reject
p =
   8.9852e-18
```

---

* Swke → For hypothesis testing

Data Visualization

```
>> load fisheriris
>> meas
>> fprintf('Species: %s\n\t Features: %.f %.f %.f %.f',
        species(1), meas(1,:));
```

↳ cell type (error)!

{1}

%.1f → only 1 decimal

```
for q=1:length(meas)
    fprintf('Sample: %3d Species: %10s  Features: %5.1f %5.1f
        %5.1f %5.1f\n', q, species{q},
        meas(q,:));
```

Plot

* Plot(meas(1:50,1)) → over data points - Independent.

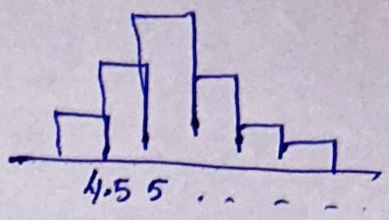
* plot(meas(1:50,1), 'o')

↳ monkeon

(removes connections)

Info: how the data is distributed - regardless (order)

histogram




```
histogram(meas(1:50,1));
xlabel('value');
ylabel('occurance');
min(meas(:)) ⟶ 0.1000
max(meas(:)) ⟶ 7.9000
```

```
subplot(2,2,1);
axis([0,8,0,40]);
        ⌣   ⌣
      xaxis yaxis
```

```
subplot(2,2,1); axis([0,8,0,40]);
subplot(2,2,2); axis "
subplot(2,2,3);  "
subplot(2,2,4);  "
for f=1:3      % species
figure(f);
if f==1
data range= [1:50];
elseif f==2
datarange= [51:100];
else
 datarange= [101:150];
end
for s=1:4  % features
subplot(2,2,s)
histogram(meas(data_range,s),[0:0.25:8])
                              ↳ range
axis([-.25,8.25,0,35]);
title(['species', species{data_range(1)}, '; Feature ', numstr(s)])
xlabel('value')
ylabel('occurance')
```

end

end.

Important – Compare

* Similar Scale
* Similar bin sizes

] If not mentioned: MATLAB guesses

---

* Here: distribution
* Relationship b/w different features: may not be

Independent like this. – Correlated.

Scatter plotting – relationship.

* 2-D plot – Symbol for each instance
* If correlated – upward diagonal slope

(value of one – somewhat predictive of the value of other)

* Negative – correlation → 'Larger value happens – smaller value of other feature'

'downward diagonal slope'

No diagonal slope – No correlation.

```
plot(meas(:,2), meas(:,1), 'o')
xlabel('Feature 1');
ylabel('Feature 2');
```

} Not correlated!

Separate species – Call array

```
u = unique(Species)
```

3 x 1

{ 'setosa'
'versicolor'
'virginica' }

[u, v, s] = unique(species)

↓ elements

└→ indices $\begin{pmatrix}1\\2\\3\end{pmatrix}$

plot(meas(s==1,2), meas(s==1,1), 'ro');

└→ circle.

└→ red

plot(meas(s==2,2), meas(s==2,1), 'gx');

Legend

legend('Setosa', 'Versi', 'Virginica')

↓ 1st plotted ↓ 2nd plotted

[r, p] = corr(meas(s==1,1), meas(s==1,2))

↓ row (equal 1)

└→ Probability variable

If good linear mapping - obtainable

+ve correlated (r=1)

No! (r ≈ 0)

$r > 0.7$ ⟶ strong correlation

$0.7 \geq r \geq 0.3$ → moderate

< 0.3 ⟶ weak

} Correlation

→ scatter(x,y) (Scatter plot)

Dimensionality reduction - techniques

* 6 Scatter plots - 4 features
* 5 feature → 10 plots
* 6 feature → 15 plots

'Some have 1000s & 10000s eg plots'

Analyze efficiently – dimensionality reduction

* Techniques – (given: right circumstances) – allow us to transform info from a large set of features into a new – more compact set of features – easy to visualize & understand.

Concept: Different plots change (tend to) when together there is correlated with eachother.

Linear algebra: dependent : one vector is linear combination of other.

$$\begin{bmatrix} 2 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{matrix} y_1 \\ y_2 \\ y_3 \end{matrix}$$

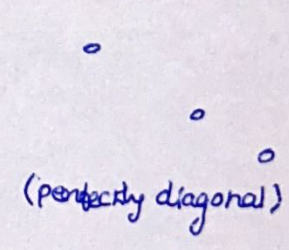
$$y_1 = y_2 + 2y_3$$

Importance:

If we assume: features are normally distributed, it turns out that if different features are correlated, we can represent the feature space of n features in a reduced linear subspace of m less than n dimensions.

"Though we have n dimension, they vary only in n dimensions"

```
>> D = [-5, 2.5; 2, -1; 10, -5; -12 6]
>> plot (D(:,1), D(:,2), 'o')
```



Feature 1 = −2 (feature 2)

(perfectly diagonal)

"Though −2d" – Not

They represent a line in space!

1-d vector space!

Basis

$V = [2 \quad -1]$

∴ >> $-2.5 * V \longrightarrow D(1,:)$

>> $V \longrightarrow D(2,:)$

>> $5V \longrightarrow D(3,:)$

>> $-6V \longrightarrow D(4,:)$

---

* we can represent this with a single feature – instead of 2 (Coordinate in 1d linear vector space)

$a = [-2.5 ; 1 ; 5, -6] \longrightarrow$ Subspace Coordinated

ax basis vector(v) = $\begin{bmatrix} \text{our data} \end{bmatrix}$ → example of dimensionality reduction.

Scalar multiple of V'

(∴ Since they are Correlated !)

Assume: normally distributed (Take new guesses !)

>> an = [-10 : .25 : 10]

hold on

>> cmap = jet(length(an)); → creates a matrix of RGB ranging from blue (1st row) Red (last row)

Use for loop to plot each scaling !

>> for i = 1 : length(an)

plot(an(i) * V(1), an(i) * V(2), '*', 'color', cmap(i,:))

end

(Different color)

>> D = [-2, -8, -7 ; -1 -3 -3 ; 4 0 6 ; 1 5 4 ; -2 4 -5];

>>

-2	-8	-7
-1	-3	-3
4	0	6
1	5	4
-2	-4	-5

$V_1 = [1 \;\; 1 \;\; 2]$;

$V_2 = [1 \;\; -1 \;\; 1]$;

$-5V_1 + 3V_2$ → over fitted vector.

>> $2V_1 + V_2$

-1 -3 -3

>> $2V_1 + 2V_2$

4 0 6

>> $-3V_1 + V_2$

-2 -4 -5

-2 -8 -7

∴ 3d-features → entirely lines in 2-d subspace?

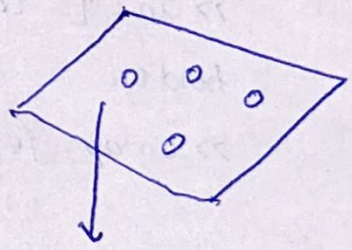
"2-d plane"

```
plot3(D(:,1), D(:,2), D(:,3), 'ro')
plane = [V1+V2; V1-V2; -V1-V2; -V1+V2; V1+V2] * 5;
hold on;
Plot3(plane(:,1), plane(:,2), plane(:,3), 'g')
```

Algorithmically find bases

* Large vector: trial & error → not possible!

"Best vector bases"



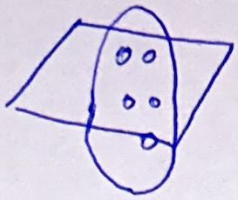
Principle component analysis! → Dimensionality reduction.

» Fitting an n dimensional hyper ellipsoid to n dimensional data set.

equivalent to an ellipse in 2-D

* 3-d version of an ellipse → ellipsoid!
* higher dimensions — hyper ellipsoids (beyond 3)
* PCA - assumes - our sample are from the space modeled as n-d hyper ellipsoid.

* This process – well defined by multi dimensional Gaussian distribution



→ 2-d ellipse to 3d data (features)

```
>> d = [5.5 -4; -1.5 2; -9 2; 5 0];
>> plot(d(:,1), d(:,2), 'bo');   ('LineWidth', 2)
>> axis([-15 15 -15 15])
>> axis equal
```

→ This ensures that distance along the x & y axis is rep. with same no. of pixel on our screen.

```
[eigvects, ~, eigvals] = PCA(D)
```

principle components of the data – mathematically called as – Eigen vectors!
of the covariance matrix of data D.

The magnitude / variants of the data along each of those principle components → Eigen values.

* with 2-d data, fitting ellipse to the data

* 1st eigen vector → eigvects(:,1) → 0.9486, -0.3166 → Eigen values in 1st column has the largest eigen values.

* eigvals → 50.9500, 3.2167

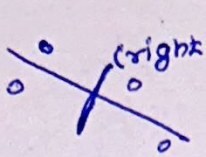
Eigen values gives the variance of the data along the direction defined by the eigen vector.

```
>> plot(eigenvects(1,1)*[2; -2]*Sqrt(eigvals(1)),
        eigenvects(2,1)*[2; -2]*Sqrt(eigvals(1)));
```

2 standard deviation from mean.

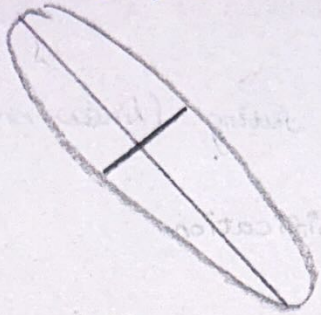
S.D = $\sqrt{\text{Variance}}$

= captured by eigen values.



(right angle to 1st one) – shorter than 1st one.
(∵ data has less variance in that direction)

>> plot (eigvecks(1,2)* [2;-2]* sqrt(eigvals(2)), eigvecks(1,2) *[2;-2] * sqrt(eigvals(2)), 'r')



principal axes.

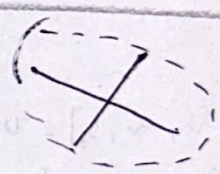
>> theta = [0:0.05:2*pi];

>> ellipse_x = 2 * sqrt(eigvals(1)) * eigvecks(1,1) * cos(theta) +
2 * sqrt(eigvals(2)) * eigvecks(1,2) * sin(theta)

"

>> ellipse_y = eigvals(1)) * ... (2,1) * ...
(2,2) ...

plot (ellipse_x, ellipse_y)



pca → works to n dim data

* Eigen values & vectors are ordered → most important principal axis (most variance) first.
* Shortest variance (least important) → at last.

dim reduction

↙ Lossy ↘ Lossless (removal – when no variance in data)

(depending upon – data)

* Here: 1d line not possible (not lossless)
* Lossy: preserve as much as possible

pca

» [eigvecks, D_pca, eigvals] = pca(D);

» D_ pca =

4×8 matrix

» fitlm(x,y)

↳ map using fitting (linear model)

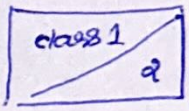
Data classification

eg: From data → healthy

→ Unhealthy – need maintainance

Translate, Interpret, Classify → audio commands

* classification: partition – Anything falls – in certain partition – belongs there.



» [~, ~, s] = unique (species);

» figure (1)

» plot (meas(s==1, 1), meas(s==1, 2), 'rx');

» holden

» plot (meas(s==2, 1), meas(s==2, 2), 'go');

» plot (meas(s==3, 2), meas(s==3, 2), 'go');

» xlabel ('feature 1');

» ylabel ('feature 2');

» legend ('setosa', 'versi color', 'virginica');

» edit

```
function class = my_iris_classifier (feat)
```

Note: mask points of species 3 → values > 3 (yaxis)

```
function class = my_iris_classifier(feat)
    class = zeros(size(feat,1),1);
    if feat(1) > 6
            class(i) = 3;
    else if feat(2) > 3
            class(i) = 1;
    else
            class(i) = 2;
    end
end
```

for i = 1:size(feat,1) — Row matrix.

* we can - Sampling datapoints uniformly across the features space (using meshgrid)

* [x, y] = meshgrid(4:.01:8, 2:0.01:4.5);
* cls = my_iris_classifier([x(:), y(:)]);

'use - contour' → Decision boundary

↓

generates 2-d matrix as i/p to create 2-d contour

```
>> cls = my_iris_classifier([x(:), y(:)]);
>> cls = reshape(cls, size(x));
>> Contour(x, y, cls == 1, [1, 1], 'r--');
```

↳ at the interface where grid points = 1.

'Seperates others'

```
>> Contour(x, y, cls == 2, [1, 1], 'g--');
>> Contour(x, y, cls = 3, [1, 1], 'b--');
```

'many' - didn't fall under ours'

```
cls = my_iris_classifier(meas);
sum(cls == s)/length(s) * 100;
```

ans=

74% accuracy!

```
>> [v, D, PCa] = PCa (meas);
>> Plot (D_PCa (S==1,1), D_PCa (S==1,2), 'rx')
>> hold on
>> plot (D_PCa (S==2,1), D - PCa (S==2,2), 'go');
>> plot (D_PCa (S==3,1), D_PCa (S==3,2), 'b*');
>> xlabel ('PCA 1');
>> ylabel ('PCA 2');
>> axis equal
```

```
function class = my_iris_PCA_classifier (feat)
    class = zeros (size (feat,1), 1);
    b = -1.5; m = 1;
    for i = 1: size (feat,1)
    if  feat (i,1) < -1.5
            class(i) = 1;
    elseif feat (i,1) * m + b > feat (i,2)
            class (i) = 3;
    else
            class (i) = 2;
    end
end
end
```

```
>> [x, y] = meshgrid (-4 : 0.01 : 4, -2 : .01 : 2);
>> cls = my_iris_PCA_classifier ([x(:), y(:)]);
>> cls = reshape (cls, size (x));
>> Contour (x, y, cls == 1, [1, 1], 'r--');
>>
>>                                    'g--');
```

Pretty close - diagonal almost classifying!

```
>> cls = my_fcns_PCA_classifier(D_pca(:,1:2));
>> sum(cls==S)/length(S)*100

ans.=
    96
```

Automate! — Supervised ML
— Unsupervised ML.

*

* Supervised methods: Trained with dataset includes examplars with both feature values & known classification.

* The method then aims to partition the feature space in a way that seperates the known classes

Unsupervised

*Trained with examplars with feature values, but without known classification. This method aim to find partitions that result in distinct clusters of data points in the features space. If the data tend to cluster from different classes, the resulting partition - can be used as a classifier.

---

* popular unsupervised method: K means clustering

↓

partitions by finding centroid of K clusters of data points,

K → i/p parameter (how many cluster to find)

```
>> plot(D_pca(S==1,1), D_pca(S==1,2), 'rx')
hold on
>> plot(D_pca(S==2,1), D_pca(S==2,2), 'go')
>> plot(D_pca(S==3,1), D_pca(S==3,3), 'b*')
axis equal
```

K-means → Stochastic (rely on randomly gen. numbers)

```
>> rng(0);
>> Kmeans(D_pca(:,1:2),3)        =[cls,C]
                  ↳ K
>> plot(C(:,1), C(:,2), 'ko', 'LineWidth', 3)
```

* find the distance from each point in our meshgrid - designed with X & Y

Euclidian distance: $\sqrt{\text{Sum of squared differences across each dimension}}$

```
>> dists1 = sqrt((C(1,1)-x).^2 + (C(1,2)-y).^2);
>> dists2 = sqrt((C(2,1)-x).^2 + (C(2,2)-y).^2);
>> dists3 = sqrt((C(3,1)-x).^2 + (C(3,2)-y).^2);
>> cls3 = ((dists3 < dists1) & (dists3 < dists2));
>> cls2 = ~cls3 & (dists2 < dists1);
>> cls1 = ~cls2 & ~cls3;
>> contour(x,y,cls1,[1,1],'k--')
>> contour(x,y,cls2,[1,1],'k--')
>> contour(x,y,cls3,[1,1],'k--')
>> sum(cls==s)/length(s)*100
ans =
    31.3333
```

"Not Separating"

[cls, s] → Sorts which new one belongs to which old one

~~>> sum((cls + (-~~

```
>> sum((4-cls)==s)/length(s)*100);
    88.667
```

K-means supervised

```
>> C = [mean(D_pca(s==1,:)); mean(D_pca(s==2,:)); mean(
                                D_pca(s==3,:))]
>> plot(D_pca(s==1,1), D_pca(s==1,2), 'rx')
    :
>> axis equal
```

```
>> plot(C(:,1), C(:,2), 'mo', 'LineWidth', 3)
>> dists1 = sqrt((C(1,1)-x).^2 + (C(1,2)-y).^2);
    :
```

```
>> dists2 = sqrt((C(2,1) - X).^2 + (C(2,2) - Y).^2);
>> dists3 = sqrt(C(3,1) - X).^2 + (C(3,2) - Y).^2);
>> cls3 = ((dists3 < dists1) & (dists3 < dists2));
>> cls2 = ~cls3 & (dists2 < dists1);
>> cls1 = ~cls2 & ~cls3;
>> contour(x, y, cls1, [1,1], 'k--')
>> contour(x, y, cls2, [1,1], 'k--')
>> contour(x, y, cls3, [1,1], 'k--')
>> dists1 = sqrt((C(1,1) - D_PCA(:,1)).^2 + (C(1,2) -
                 D_PCA(:,2)).^2);
>> dists2 = sqrt((C(2,1) - D_PCA(:,1).^2 + (C(2,2) - D_PCA(:,2))
                 .^2);
>> dists3 = sqrt((C(3,1) - D_PCA(:,1)).^2 + (C(3,2) - D_PCA(:,2)).
                 ^2);

>> cls3 = "
>> cls2 = "
>> cls1 = "
>> cls = 3*cls3 + 2*cls2 + cls1;
>> sum(cls == 5)/length(  ) * 100
```

92.667. → using means of the actual class instead of K means
↑ accuracy.

Fit an n dimensional Gaussian to a dataset

↓

we can use metric on distance to a Gaussian-distribution called 'Mahalanobis distance'

```
function md1 = my_fitPCA(D, class)
    class_labels = unique(class);
    numclasses = length(class_labels);
    for i = 1:numclasses
        [eigvects, ~, eigvals, ~, ~, mu] = PCA(D(class == class_labels(i), :));
        md1.class(i).eigvects = eigvects';
        md1.class(i).eigvals = eigvals;
        md1.class(i).mu = mu;
    end
end
```

```
>> md1 = my_fitPCA(D_PCA(:, 1:2), S);
3
        class_labels = unique(class);

>> md1
md1 =
    struct with fields:
        class: [1×3 struct]
```

```
Centroid_x = md1.class(1).mu(1);
Centroid_y = md1.class(1).mu(2);
major_axis_x = md1.class(1).eigvects(1,1)*2*sqrt(md1.class(1).
                                                 eigvals(1));
major_axis_y = md1.class(1).eigvects(1,2)*2*sqrt(md1.class(1).
                                                 eigvals(1));
minor_axis_x = md1.class(1).eigvects(2,1)*2*sqrt(md1.class(1).
                                                 eigvals(2));
minor_axis_y = md1.class(1).eigvects(2,2)*2*sqrt(md1.class(1).
                                                 eigvals(2));
```

```
>> plot(D_PCA(S==1,1), D_PCA(S==1,2), 'rx');
hold on
>> plot(D_PCA(S==2,1), D_PCA(S==2,2), 'go');
>> plot(D_PCA(S==3,1), D_PCA(S==3,2), 'b*');
>> axis equal;
```

```
theta = 0:pi/32:2*pi;
for i=1:3
    major axis-x = md1.class(i).eigvects(1,1)*2*sqrt(md).class(i).eigvals(1);
    majoraxis-y = md1.class(i).eigvects(1,2)*2*sqrt(md1.class(i).eigvals(1));
    minoraxis-x = md1.class(i).eigvects(2,1)*2*  "    "    eigvals(2));
    minmaxis-y =  "     (i).  "   "  (2,2)*2*  "    "    eigvals(2));

    centroid-x = md1.class(i).mu(1);
    Centroid-y = md1.class(i).mu(2);
    plot(centroid-x+cos(theta)*majoraxisx + sin(theta)*minoraxis-x,
         "     -y+ "          "     y + sin(theta)*min-y
end
```

```
function md = MahalanobisDistance(PCAmd1, data)
    b = PCAmd1.eigvects*(data - PCAmd1.mu)';
    Std_Pcon_mode = abs(b)./sqrt(PCAmd1.eigvals);
    md = sqrt(sum(Std_Pcon_mode.^2));
end
```

```
>> data = [1, 1.5];
>> plot(data(1), data(2), 'kx', 'linewidth', 3)
>> mdScore = MahalanobisDistance(md1.class(1), data);
>> 2        b=PCAmd1.eigvects*(data-PCAmd1.mu)';
>> for i=1:length(md1.class)
mdScore(i) = MahalanobisDistance(md1.class(i), data);
end
>> mdScore =
    17.9915    5.5112    7.4231
```

```
function md1 = my_fitSVM(D, class, KernelScale)
    class_labels = unique(class);
    numclasses = length(class_labels);
    for i=1:numclasses
        S = class == class_labels(i);
        md1.class(i).SVM = fitSVM(D, S, 'KernelFunction', 'rbf',
```

```
'KernelScale', KernelScale);
    end
end
```

```
function Class = my_PredictSvm(mdl, data)
    [r,c] = size(data);
    for g=1:length(mdl.Class)
        [~, mdlScores] = predict(mdl.Class(g).Svm, data);
        Score(:,g) = mdlScores(:,1);
    end
    [v,Class] = min(Score');
    Class = Class';
end
```

```
>> plot(D_Pca(S==1,1), D_Pca(S==1,2), 'rx');
>> hold on
>> plot(D_Pca(S==2,1), D_Pca(S==2,2), 'go');
>> plot(D_Pca(S==3,1), D_Pca(S==3,2), 'b*');
>> axis equal
>> trainCls = S(1:2:end)
>> mdl = my_fitSvm(traindata, trainCls, 0.75);
>> pdct = my_PredictSvm(mdl, traindata);
>> sum(pdct==trainCls)/length(trainCls) * 100
ans =
    96
>> testCls = S(2:2:end);
>> pdct = my_PredictSvm(mdl, testdata);
>> sum(pdct==testCls)/length(testCls) * 100
ans =
    97.3333
>> [X,Y] = meshgrid(-4:.1:4, -2:.1:2);
>> cls = my_PredictSvm(mdl, [X(:), Y(:)]);
>> cls = reshape(cls, size(X));
```

```
>>contour(X, Y, cls, [1.5, 1.5], 'r')
>>Contour(X, Y, cls, [2.5, 2.5], 'k')
```

Data estimation & prediction

* Estimate some other feature – using one feature (As direct observation costly)

'Continuous estimation'

Signals

* pure tone – single frequency (Hz → $\frac{1}{\text{seconds}}$)
* A – 440Hz
* Sampling: (Analog to digital representation)
* n-dimensional signal.
* Time to the data (signal) → Order matters!

```
>>rads = 0.4 * pi/100 : 4 * pi;
>>plot (rads/pi, cos(rads), 'o')
>>xlabel ('radians');
>>ylabel ('cosine function');
```

$F_S = \frac{1}{T_S} = 2Hz$ (As $T_S = 0.5s$)

$Hz = \frac{\text{No. of sample}}{\text{second}}$

```
Fs = 44100; f1 = 440; f2 = 660;
T = 5; t = [0: 1/Fs : T];
y1 = cos(2*pi*f1*t);
y2 = cos(2*pi*f2*t);
y = y1 + y2;
figure(1);
plot(t, y)
axis([0, 5/200, -2, 2])
sounds(Y, Fs)
```

» soundsc (y1, Fs) » soundsc (y2, fs) » soundsc (y, Fs), » sound sc (y2, Fs)

» sound sc (y1, FS * 660/440)

» yfd = audioread ('funky drums.mp3');

» [yst, fsst] = audioread ('Main street - wav');

Convolution

Fir1 → Finite impulse response filters (diminishes to 0 over time)

↳ creates Fir filters.

Fir1(W, 'high') → Cuts frequency!

W: window size

'high': Freq prop

5000 / (Fs/2)

Image