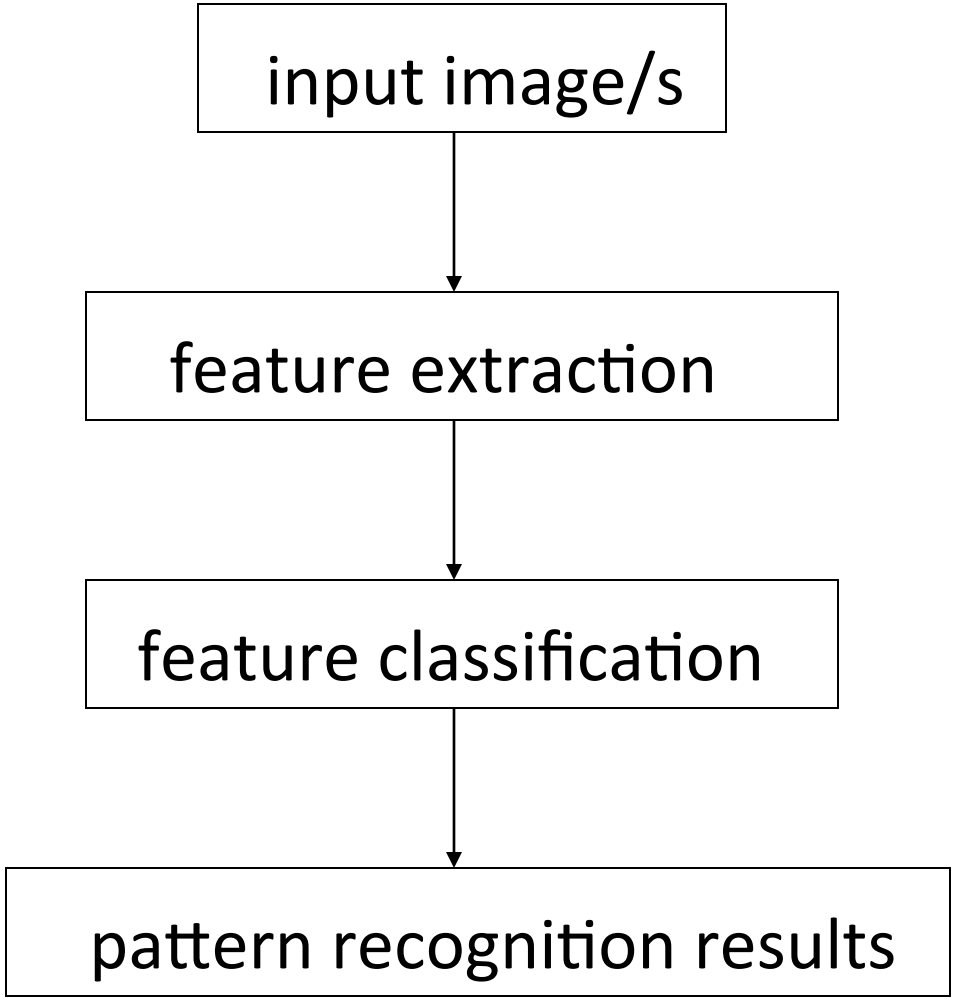
CS4243 Computer Vision & Pattern Recognition

Pattern Recognition

Pattern Recognition Basics

- Features
- Bayes Decision Theory
- Gaussian Normal Density
- Principal Component Analysis
- Support Vector Machines

Pattern Recognition Pipeline



Feature Extraction

- Linear features
  eg. Lines, curves

- 2D features
  eg. Texture

- Feature of features
  eg. usually multidimensional

Feature Extraction

- A good feature should be invariant to changes in
  - scale
  - rotation
  - translation
  - illumination

Feature Classification

- Linear Classifier
  - eg. maximum likelihood
- Non-linear Classifier
  - eg. neural network

Bayes Decision Theory

Suppose we have two classes: c_1 and c_2

with ***a priori*** probability density functions $P(c_1)$ and $P(c_2)$ respectively.

Suppose we have the measurement (observation) x that will give some clue to allow us to guess whether the measurement comes from class c_1 or c_2 .

Depending on whether the class is c_1 or c_2 , the probability of observing x may be different. We describe this using the ***conditional probability density***

$$P(x \mid c_1) \quad \text{and} \quad P(x \mid c_2)$$

Our goal is to find out which class is more likely, given the measurement x. In other words, we want to compare the ***a posteriori probabilities*** $P(c_1 \mid x)$ and $P(c_2 \mid x)$

Bayes rule allows us to express the a posteriori probability in terms of the a priori probability and class conditional density.

Bayes Rule

$$P(c_i \mid x) = \frac{P(x \mid c_i)\, P(c_i)}{P(x)}$$

$$P(x) = \sum_{i=1}^{2} P(x \mid c_i)\, P(c_i)$$

A natural approach:

Decide class c_1 if $P(c_1 \mid x) > P(c_2 \mid x)$

Decide class c_2 if $P(c_2 \mid x) > P(c_1 \mid x)$

If we have a more complicated situation whereby different wrong decisions result in different levels of penalty, the decision process will be slightly more complicated than above.

Let λ_{11} be the penalty for deciding class c_1 if x comes from class c_1

Let λ_{12} be the penalty for deciding class c_1 if x comes from class c_2

Let λ_{21} be the penalty for deciding class c_2 if x comes from class c_1

Let λ_{22} be the penalty for deciding class c_2 if x comes from class c_2

(Usually, λ_{11} and λ_{22} are both set to zero)

Then given a measurement x, the risk of deciding class c_1 is

$$R(c_1 \mid x) = \lambda_{11} P(c_1 \mid x) + \lambda_{12} P(c_2 \mid x)$$

Similarly, the risk of deciding class c_2 is

$$R(c_2 \mid x) = \lambda_{21} P(c_1 \mid x) + \lambda_{22} P(c_2 \mid x)$$

So we will decide class C_1 if

$$R(c_1 \mid x) < R(c_2 \mid x)$$

$$\Rightarrow \quad P(c_1 \mid x)\,[\lambda_{11} - \lambda_{21}] < P(c_2 \mid x)\,[\lambda_{22} - \lambda_{12}]$$

Using Bayes rule,

$$P(c_1 \mid x)\,[\lambda_{11} - \lambda_{21}] < P(c_2 \mid x)\,[\lambda_{22} - \lambda_{12}]$$

$$\Rightarrow \quad \frac{P(x \mid c_1)\,P(c_1)}{P(x)}\,[\lambda_{11} - \lambda_{21}] < \frac{P(x \mid c_2)\,P(c_2)}{P(x)}\,[\lambda_{22} - \lambda_{12}]$$

$$\Rightarrow \quad P(x \mid c_1)\,P(c_1)\,[\lambda_{11} - \lambda_{21}] < P(x \mid c_2)\,P(c_2)\,[\lambda_{22} - \lambda_{12}]$$

How to obtain these in practice ?

Gaussian (Normal) Density

Let X be a d-dimensional feature vector.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_d \end{bmatrix}$$

If each of the elements of the vector X is a Gaussian random variable, the ***multivariate normal density*** is given by

$$P(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(X - \bar{X})^T \Sigma^{-1} (X - \bar{X})\right.$$

where $\overline{X}$ is the mean vector, and Σ is the covariance matrix.

$$\underline{X} = \mathbb{E}\{X\}$$

$$\Sigma = E\{(X - \overline{X})(X - \overline{X})^T\}$$

$|\Sigma|$ is determinant of Σ

The linear combination of Gaussian random vectors is also a Gaussian random vector.

Properties of covariance matrix

- Σ is always symmetric and positive semi-definite
- Σ is usually positive definite. It is positive semi-definite when at least one of the variables have zero variance, or at least 2 of the variables are identical. These are degenerate cases that we will exclude.
- Σ is a symmetric matrix (Hermitian symmetric if x is complex)

$$\left|\Sigma\right| \neq 0 \quad \text{and so} \quad \Sigma^{-1} \text{ exists.}$$

Mahalanobis Distance

$$(X - \bar{X})^T \Sigma^{-1} (X - \bar{X})$$

Maximum Likelihood Estimates

For parameter estimation

Given a set of observations (measurements)

$X = \{x_1, x_2, x_3, \cdots, x_N\}$ corresponding to time

$T = \{t_1, t_2, t_3, \cdots, t_N\}$

we want to estimate the parameter vector θ that satisfies

$$x_i = f(\theta, t_i)$$

The maximum likelihood estimate of θ

is the $\hat{\theta}$ that maximizes $P(X \mid \theta)$

In computing the ML estimates, it is frequently more convenient and easier to work with the logarithm of $P(X \mid \theta)$, i.e.

$$\log P(X \mid \theta)$$

It is ok to do this because logarithm is a monotonically increasing function.

If $\{x_1, x_2, x_3, \cdots, x_N\}$

are independent measurements of a random variable x, then

$$P(x \mid \theta) = \prod_{i=1}^{N} P(x_i \mid \theta)$$

$$\log P(x \mid \theta) = \sum_{i=1}^{N} \log P(x_i \mid \theta)$$

Finding $\hat{\theta}$ that maximizes $\log P(x \mid \theta)$ can usually be done by setting

$$\frac{\partial}{\partial \theta} \log P(x \mid \theta) = 0$$

Principal Components Analysis (Karhunen-Loeve Transformation)

- Principal components analysis and its interpretation
- Applications

Principal Component Analysis (PCA)

$$\{X_1, X_2, X_3, \cdots, X_N\}$$

$$\overline{X} = \frac{1}{N} \sum_{i=1}^{N} X_N$$

$$S = \sum_{i=1}^{N} (X - \overline{X})(X - \overline{X})^T$$

$$S\, e_i = \lambda_i\, e_i$$

where the e_i is a principal components with eigenvalue λ_i

One problem with PCA when applied directly to the covariance matrix of the images is that the dimension of covariance matrix is huge! For example, for a 256x256 image, the covariance matrix will be of size 65536x65536 !!! To compute the eigenvector of such a huge matrix is clearly inconceivable.

To solve this problem, Murakami and Kumar proposed a technique in the following paper:

Murakami, H. and Vijaya Kumar, B.V.K., "Efficient Calculation of Primary Images from a Set of Images", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-4, No. 5, Sep 1982.

We shall now look at this method. In the lab, you will practice implementing the algorithm for face recognition.

To begin, we note that our goal is to find the eigenvectors and eigenvalues of the covariance matrix

$$\Sigma = \frac{1}{N} \sum_{i=1}^{N} x_i \, x_i^T$$

We will see later that instead of working directly on $\sum$, which is a huge matrix, we can find the eigenvectors and eigenvalues of an NxN matrix and then use it to deduce the eigenvectors and eigenvalues of $\sum$.

To see how this can be done, we first note that

$$x_i x_i^T e = x_i^T e \, x_i$$

Therefore, $$\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T e = \frac{1}{N}\sum_{i=1}^{N} (x_i^T e) x_i$$

$$= \frac{1}{N}\sum_{i=1}^{N} a_i x_i$$

If *e* is an eigenvector of Σ, then

$$\lambda\, e = \frac{1}{N}\sum_{i=1}^{N} a_i x_i$$

Let $b_i = \frac{a_i}{\lambda\, N}$

then $e = \sum_{i=1}^{N} b_i x_i$

So if e is an eigenvector of $\sum$, e can be expressed as a linear combination of the vectors $\{x_i\}$.

Now,

$$\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T e = \lambda \sum_{i=1}^{N} b_i x_i$$

$$\Rightarrow \quad \frac{1}{N}\sum_{i=1}^{N} x_i x_i^T \left(\sum_{j=1}^{N} b_j x_j\right) = \lambda \sum_{i=1}^{N} b_i x_i$$

$$\Rightarrow \quad \frac{1}{N}\sum_{i=1}^{N} x_i \sum_{j=1}^{N} b_j \, x_i^T x_j = \lambda \sum_{i=1}^{N} b_i x_i$$

$$\Rightarrow \quad \frac{1}{N}\sum_{j=1}^{N} x_i^T x_j \, b_j = \lambda b_i$$

Therefore, the vector $b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$ is the eigenvector of the matrix

with elements $\left\{\frac{1}{N} x_i^T x_j\right\}$

So the eigenvector of the matrix $\frac{1}{N}\sum_{i=1}^{N} x_i\, x_i^T$, i.e. e, is given by

$$e = \sum_{i=1}^{N} b_i x_i$$
$$= \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix} b$$

Note that the eigenvalues of the matrix with elements $\left\{\frac{1}{N} x_i^T x_j\right\}$ is the same as the eigenvalues of the matrix $\frac{1}{N}\sum_{i=1}^{N} x_i\, x_i^T$

In conclusion, the task of finding the eigenvectors of a huge matrix of size M x M (where M is the number of pixels in an image), is reduced to the task of finding the eigenvectors of a NxN matrix.

tentative

Murakami and kumar's work was published in 1982. In 1991, Turk and Pentland gave a shorter proof of the same result in the paper:

Turk, M.A. and Pentland, A.P., "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no.1, pg 71-86, 1991.

We shall now look at the proof:

Supposing e and λ are the eigenvector and the corresponding eigenvalue of the matrix XX^T. Then

$$XX^T e = \lambda e$$

Multiplying both sides by X^T, we have

$$X^T XX^T e = \lambda X^T e$$

Therefore, $X^T e$ is eigenvector of $X^T X$

XX^T and $X^T X$ share the same eigenvalue.

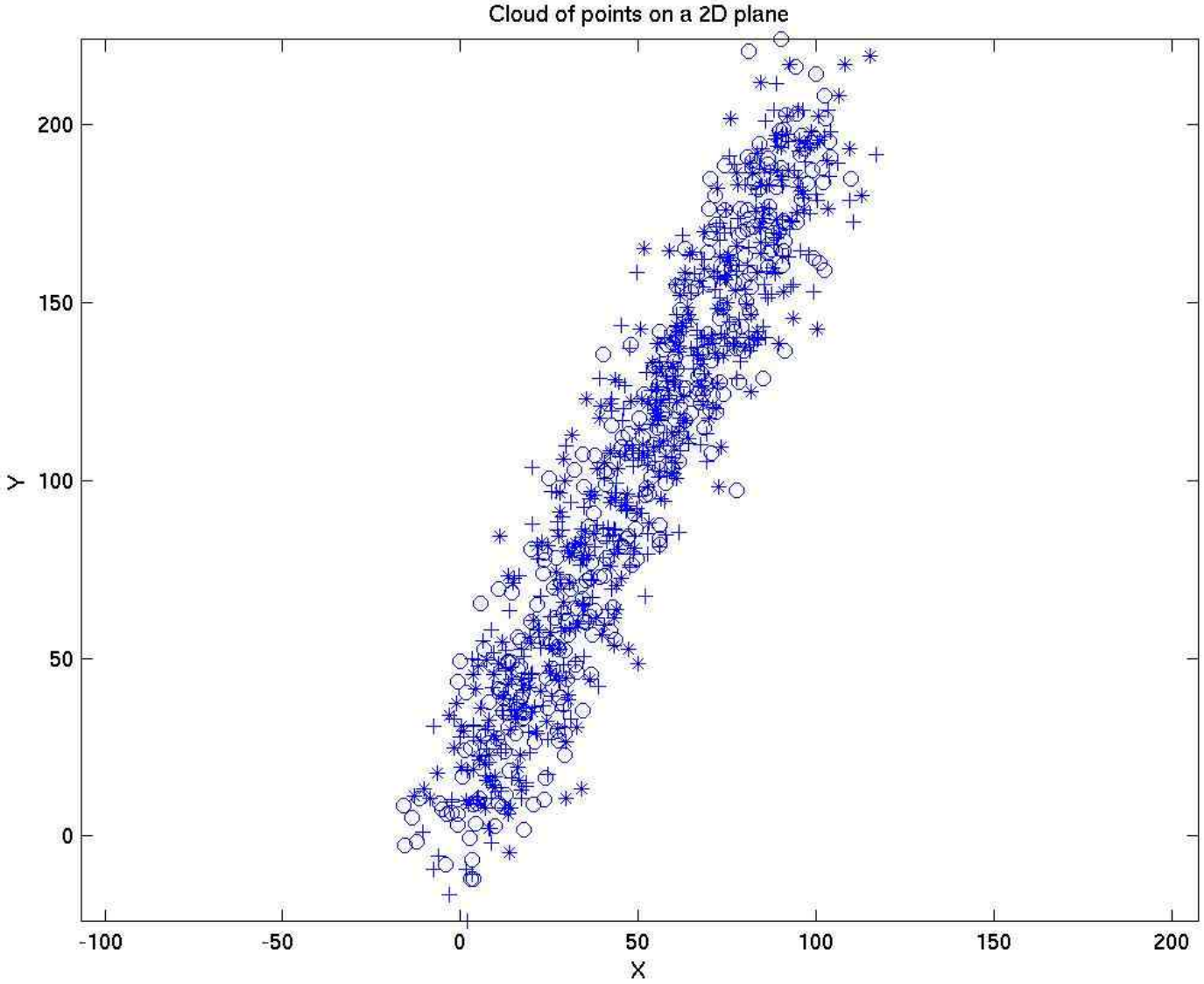
Let b be the eigenvector of $X^T X$, i.e.

$$X^T X b = \lambda b = \lambda X^T e$$

So $b = X^T e$

which means that the components of the eigenvector *b* are the projections of each individual image x_i onto the eigenvector *e*

Suppose we are shown a set of points:



We want to answer any of the following questions:

- What is the trend (direction) of the cloud of points
- Which is the direction of maximum variance (dispersion)
- Which is the direction of minimum variance (dispersion)
- Supposing I am only allowed to use a 1D description for this set of 2D points, i.e. using 1 coordinate instead of 2 coordinates (x,y). How should I represent the points in such a way that the overall error is minimized ?
  --- this is data compression problem

All these questions can be answered using
Principal Components Analysis (PCA)

Principal Component Analysis (PCA)

Suppose we have N feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_N\}$

We take the mean of the feature vectors $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$

Form the covariance matrix $C = \frac{1}{N}\sum_{i=1}^{N} \hat{\mathbf{x}}_i \, \hat{\mathbf{x}}_i^{\,T}$

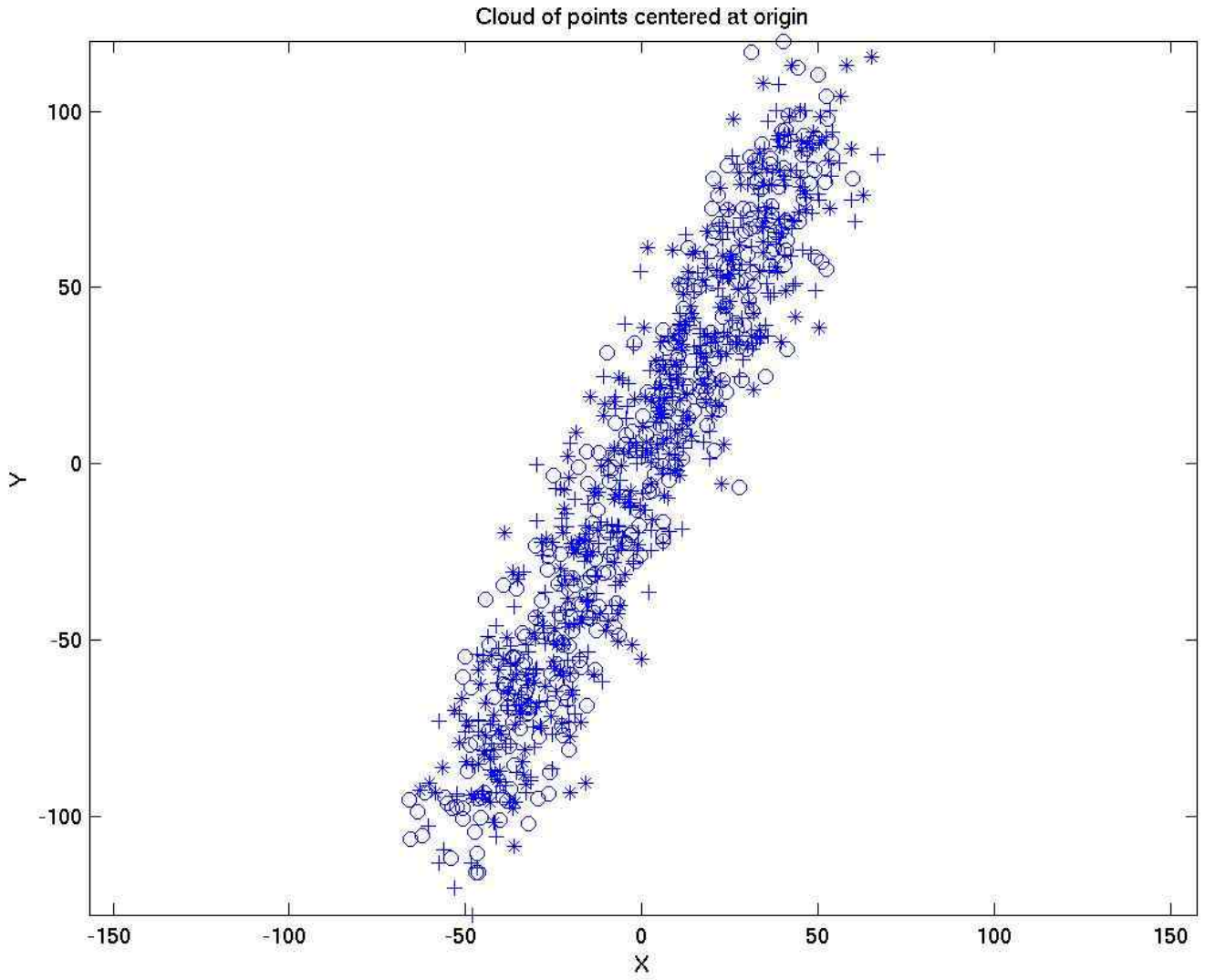
where $\hat{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$

Find the eigenvalues and eigenvectors of covariance matrix

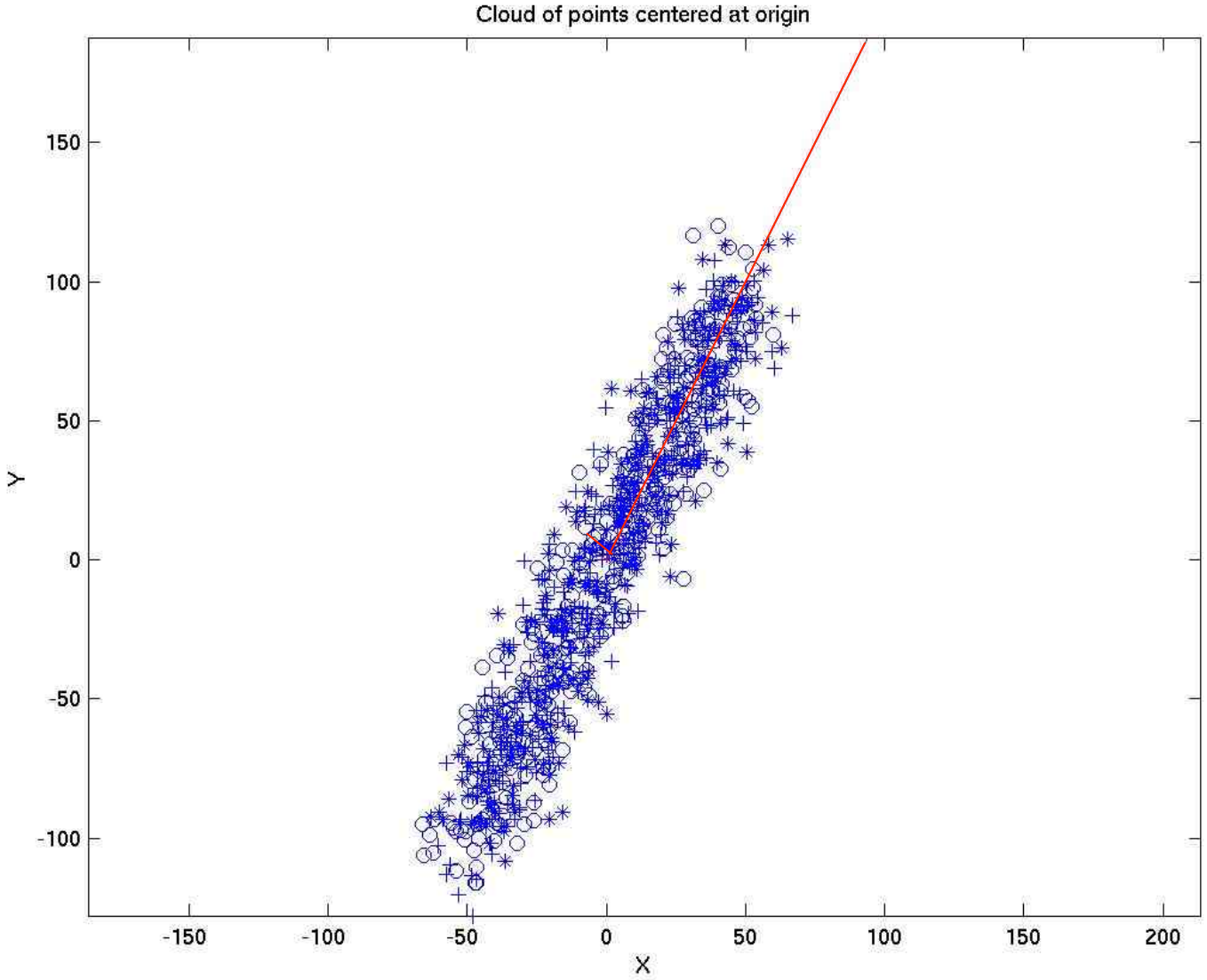
$$C\,\mathbf{e}_i = \lambda_i \, \mathbf{e}_i$$

where the $\mathbf{e}_i$ is a principal component with eigenvalue λ_i

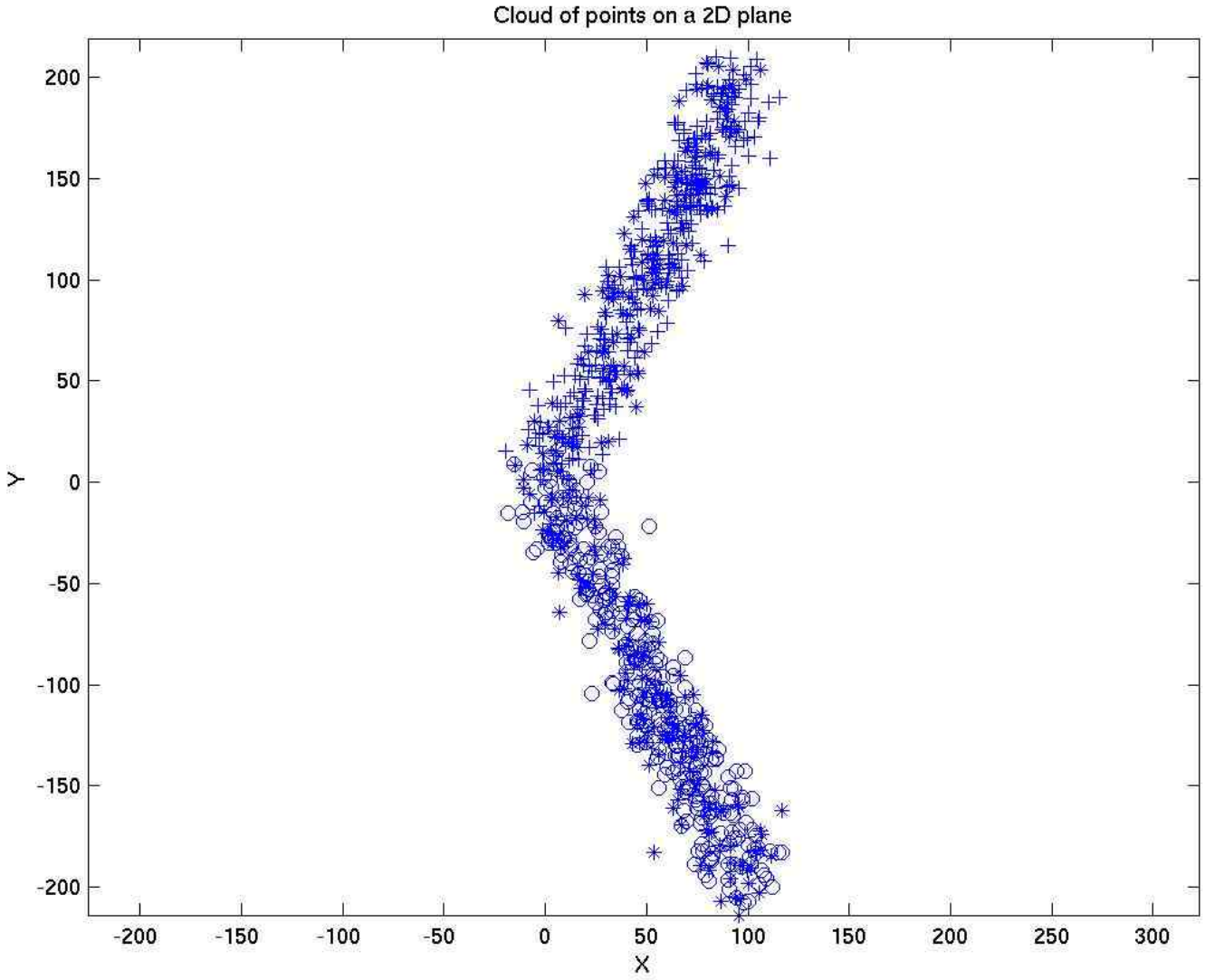
Shift cloud of points so that average position is at origin



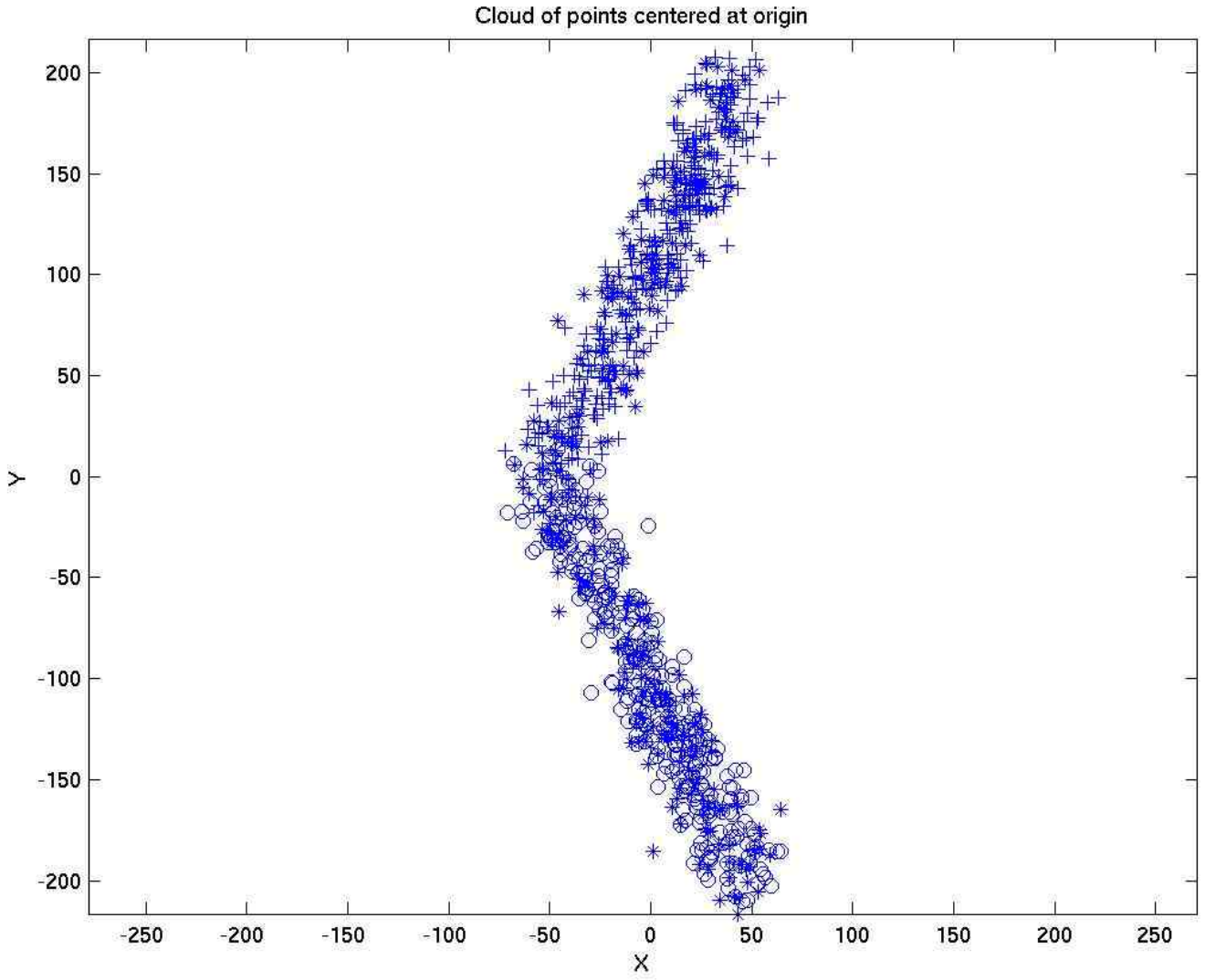
Principal components of cloud of points



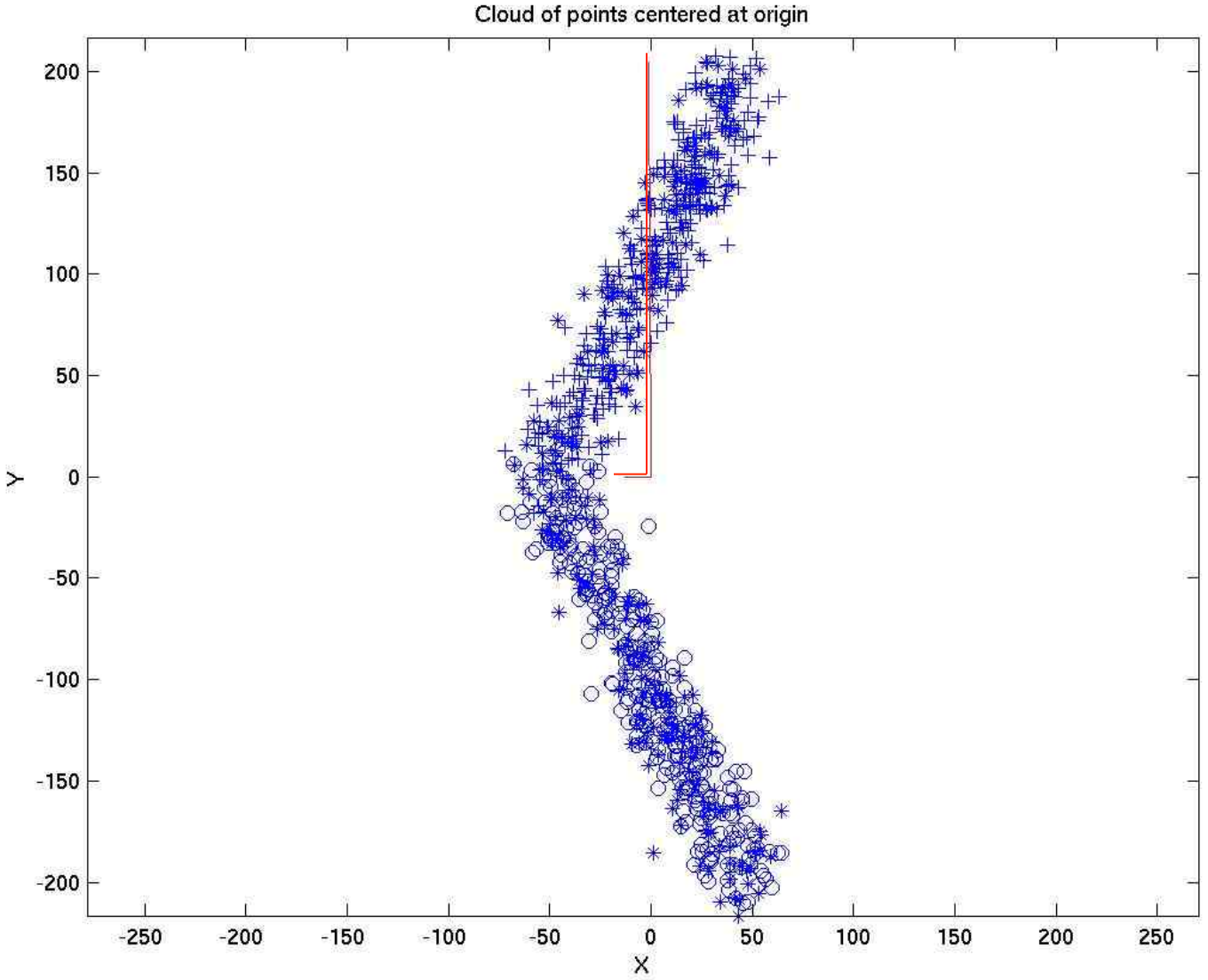
Another Example



Shift cloud of points so that average position is at origin



Principal components of cloud of points



One problem with PCA when applied directly to the covariance matrix of the images is that the dimension of covariance matrix is huge! For example, for a 256x256 image, X will be of dimension 65536x1. So the covariance matrix of X will be of size 65536x65536 !!! To compute the eigenvector of such a huge matrix is clearly inconceivable.

To solve this problem, Murakami and Kumar proposed a technique in the following paper:

Murakami, H. and Vijaya Kumar, B.V.K., "Efficient Calculation of Primary Images from a Set of Images", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-4, No. 5, Sep 1982.

We shall now look at this method.

To begin, we note that our goal is to find the eigenvectors and eigenvalues of the covariance matrix

$$C = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T = \frac{1}{N}\sum_{i=1}^{N}\hat{\mathbf{x}}_i\,\hat{\mathbf{x}}_i^T$$

We will see later that instead of working directly on C, which is a huge matrix, we can find the eigenvectors and eigenvalues of an NxN matrix and then use it to deduce the eigenvectors and eigenvalues of C.

To see how this can be done, we first note that

$$\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T\mathbf{e} = \hat{\mathbf{x}}_i\left(\hat{\mathbf{x}}_i^T\mathbf{e}\right) = \underbrace{(\hat{\mathbf{x}}_i^T\mathbf{e})}_{\text{scalar}}\hat{\mathbf{x}}_i = a_i\,\hat{\mathbf{x}}_i$$

Therefore, if e is an eigenvector of C, then

$$\frac{1}{N}\sum_{i=1}^{N}\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T\mathbf{e} = \lambda\,\mathbf{e}$$

$$\Rightarrow \quad \frac{1}{N}\sum_{i=1}^{N}(\hat{\mathbf{x}}_i^T\mathbf{e})\hat{\mathbf{x}}_i = \lambda\,\mathbf{e}$$

$$\Rightarrow \quad \frac{1}{N}\sum_{i=1}^{N}a_i\hat{\mathbf{x}}_i = \lambda\,\mathbf{e}$$

$$\Rightarrow \quad \sum_{i=1}^{N}\frac{a_i}{\lambda N}\hat{\mathbf{x}}_i = \mathbf{e}$$

Let $b_i = \dfrac{a_i}{\lambda N}$

then $\mathbf{e} = \sum_{i=1}^{N} b_i\hat{\mathbf{x}}_i$

So if $\mathbf{e}$ is an eigenvector of C, $\mathbf{e}$ can be expressed as a linear combination of the vectors $\left\{\hat{\mathbf{x}}_i\right\}$.

Therefore,

$$\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T\right)\mathbf{e}=\lambda\mathbf{e}\ =\lambda\sum_{i=1}^{N}\left(b_i\hat{\mathbf{x}}_i\right)$$

$$\Rightarrow\quad\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T\right)\sum_{j=1}^{N}\left(b_j\hat{\mathbf{x}}_j\right)=\lambda\sum_{i=1}^{N}\left(b_i\hat{\mathbf{x}}_i\right)$$

$$\Rightarrow\quad\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\mathbf{x}}_i\sum_{j=1}^{N}\left(b_j\,\hat{\mathbf{x}}_i^T\hat{\mathbf{x}}_j\right)\right)=\lambda\sum_{i=1}^{N}\left(b_i\hat{\mathbf{x}}_i\right)$$

$$\Rightarrow\quad\sum_{i=1}^{N}\left(\hat{\mathbf{x}}_i\frac{1}{N}\sum_{j=1}^{N}\left(b_j\,\hat{\mathbf{x}}_i^T\hat{\mathbf{x}}_j\right)\right)\ =\ \sum_{i=1}^{N}\left(\hat{\mathbf{x}}_i\lambda b_i\right)$$

Note that $\frac{1}{N}\sum_{j=1}^{N}\left(\hat{\mathbf{x}}_i^T\hat{\mathbf{x}}_j\,b_j\right)=\lambda b_i$ satisfies the above equation.

So one of the solutions for the vector $\mathbf{b}=\left[b_1\ b_2\cdots b_N\right]^T$ is the eigenvector of the matrix with (i,j) elements given by $\left\{\frac{1}{N}\hat{\mathbf{x}}_i^T\hat{\mathbf{x}}_j\right\}$

So the eigenvector of the matrix $\frac{1}{N}\sum_{i=1}^{N}\hat{\mathbf{x}}_i\,\hat{\mathbf{x}}_i^T$, i.e. e, is given by

$$\mathbf{e} = \sum_{i=1}^{N} b_i \hat{\mathbf{x}}_i$$
$$= \begin{bmatrix} \hat{\mathbf{x}}_1 & \hat{\mathbf{x}}_2 & \cdots & \hat{\mathbf{x}}_N \end{bmatrix} \mathbf{b}$$

Note that the eigenvalues of the matrix with elements $\left\{\frac{1}{N}\hat{\mathbf{x}}_i^T\hat{\mathbf{x}}_j\right\}$ is the same as the eigenvalues of the matrix $\frac{1}{N}\sum_{i=1}^{N}\hat{\mathbf{x}}_i\,\hat{\mathbf{x}}_i^T$

In conclusion, the task of finding the eigenvectors of a huge matrix of size M x M (where M is the number of pixels in an image), is reduced to the task of finding the eigenvectors of a NxN matrix.

Summary

Suppose we have N feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_N\}$

We want to find the principal components. Since a direct computation of covariance matrix is not feasible due to huge dimension of covariance matrix, we do the following steps instead:

Step 1: We take the mean of the feature vectors $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$

Step 2: Subtract the mean $\bar{\mathbf{x}}$ from each of the feature vectors $\mathbf{x}_i$

$$\hat{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$$

Step 3: Form the inner product matrix

$$M = \begin{bmatrix} \widehat{\mathbf{x}}_1^T \widehat{\mathbf{x}}_1 & \widehat{\mathbf{x}}_1^T \widehat{\mathbf{x}}_2 & \cdots & \widehat{\mathbf{x}}_1^T \widehat{\mathbf{x}}_N \\ \widehat{\mathbf{x}}_2^T \widehat{\mathbf{x}}_1 & \widehat{\mathbf{x}}_2^T \widehat{\mathbf{x}}_2 & \cdots & \widehat{\mathbf{x}}_2^T \widehat{\mathbf{x}}_N \\ \vdots & \vdots & \vdots & \vdots \\ \widehat{\mathbf{x}}_N^T \widehat{\mathbf{x}}_1 & \widehat{\mathbf{x}}_N^T \widehat{\mathbf{x}}_2 & \cdots & \widehat{\mathbf{x}}_N^T \widehat{\mathbf{x}}_N \end{bmatrix}$$

Step 4: Find the eigenvectors and eigenvalues of M

$$\mathbf{M}\,\mathbf{b} = \alpha\ \mathbf{b}$$

Step 5: If $[\mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3\ \cdots \mathbf{e}_N]$ and $[\lambda_1\ \lambda_2\ \lambda_3\ \cdots \lambda_N]$ are the principal components and the corresponding eigenvalues respectively of $[\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3\ \cdots \mathbf{x}_N]$ then

$$\mathbf{e}_i = [\hat{\mathbf{x}}_1\ \hat{\mathbf{x}}_2\ \hat{\mathbf{x}}_3\ \cdots \hat{\mathbf{x}}_N]\mathbf{b}_i$$

$$\lambda_i = \alpha_i$$

Bayes Rule

Expectant mother:

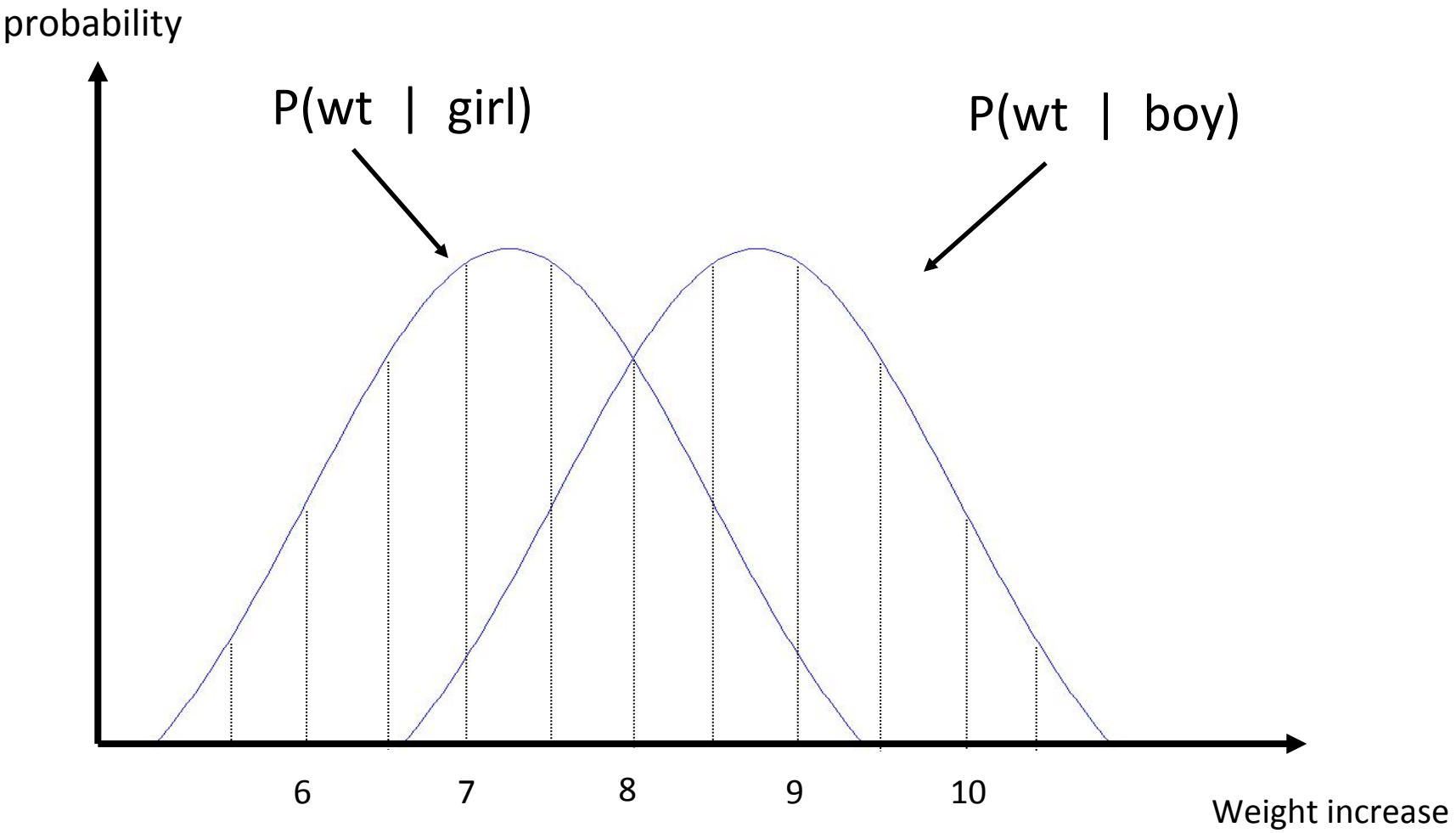
Boy or girl ?

P(boy) = 0.5

P(girl) = 0.5

P(boy or girl) = 1

Conditional Probability Density



Let's suppose the weight increase of all expectant mothers depends on whether the baby is boy or girl.

The **conditional probability** of weight increase for a large sample of expectant mothers is as shown in the graph on the next slide.

Suppose now I tell you the weight increase of the expectant mother. Then you are "better informed" as far as guessing boy or girl is concerned.

If I tell you weight increase is 9, then you have better chance of winning the bet if you guess "boy".
If I tell you weight increase is 7, then you have better chance of winning the bet if you guess "girl".

So the **conditional probaility** tells you the likelihood of event A happening given that event B has happened: P(A | B)

Sometimes, you want to compute P(A | B) but you are not given the conditional probability function P(A | B). Instead, you are given P(B | A). Can you make use of P(B | A) to compute P(A | B) ?

The answer is "Yes", with some additional information.

Bayes Rule allows you to do this.

Bayes Decision Theory

boy → c_1 and girl → c_2

Suppose we have two classes: c_1 and c_2

with ***a priori*** probability density functions $P(c_1)$ and $P(c_2)$ respectively.

weight increase →

Suppose we have the measurement (observation) x that will give some clue to allow us to guess whether the measurement comes from class c_1 or c_2 .

Depending on whether the class is c_1 or c_2 , the probability of observing x may be different. We describe this using the ***conditional probability density***

$$P(x \mid c_1) \quad \text{and} \quad P(x \mid c_2)$$

Our goal is to find out which class is more likely, given the measurement x. In other words, we want to compare the ***a posteriori probabilities*** $P(c_1 \mid x)$ and $P(c_2 \mid x)$

Bayes rule allows us to express the a posteriori probability in terms of the a priori probability and class conditional density.

Bayes Rule

$$P(c_i \mid x) = \frac{P(x \mid c_i)\, P(c_i)}{P(x)}$$

so $$p(b \mid wt) = \frac{p(wt \mid b)\ p(b)}{p(wt)}$$ --- Eqn-1

where

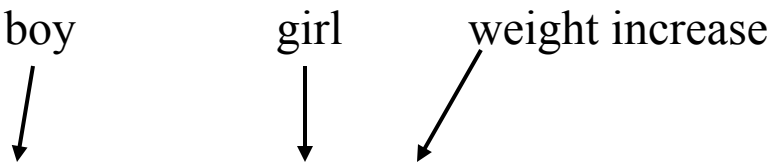
$$p(b) = 0.5$$

$$p(wt) = p(wt \mid girl) * p(girl) + p(wt \mid boy) * p(boy)$$

i.e. $$P(x) = \sum_{i=1}^{2} P(x \mid c_i)\, P(c_i)$$

In this expectant mother problem, we are actually not required to compute the actual probability p(b | wt). All that we need is to have a good way to decide whether it is boy or girl.
Since p(wt) is a constant whether it is boy or girl, all that we are interested in is the numerator of the right hand side of Eqn-1.

A natural approach:



Decide class c_1 if $P(c_1 \mid x) > P(c_2 \mid x)$

Decide class c_2 if $P(c_2 \mid x) > P(c_1 \mid x)$

If we have a more complicated situation whereby different wrong decisions result in different levels of penalty, the decision process will be slightly more complicated than above.

Example: if baby is girl, and you say "boy", you lose $10
if baby is boy, and you say "girl", you lose $20

Then you should attach "cost" in your decision making.

Let λ_{11} be the penalty for deciding class c_1 if x comes from class c_1

Let λ_{12} be the penalty for deciding class c_1 if x comes from class c_2

Let λ_{21} be the penalty for deciding class c_2 if x comes from class c_1

Let λ_{22} be the penalty for deciding class c_2 if x comes from class c_2

(Usually, λ_{11} and λ_{22} are both set to zero)

Then given a measurement x, the risk of deciding class c_1 is

$$R(c_1 \mid x) = \lambda_{11} P(c_1 \mid x) + \lambda_{12} P(c_2 \mid x)$$

Similarly, the risk of deciding class c_2 is

$$R(c_2 \mid x) = \lambda_{21} P(c_1 \mid x) + \lambda_{22} P(c_2 \mid x)$$

So we will decide class C_1 if

$$R(c_1 \mid x) < R(c_2 \mid x)$$

$$\Rightarrow \quad P(c_1 \mid x)\,[\lambda_{11} - \lambda_{21}] < P(c_2 \mid x)\,[\lambda_{22} - \lambda_{12}]$$

Using Bayes rule,

$$P(c_1 \mid x)\,[\lambda_{11} - \lambda_{21}] < P(c_2 \mid x)\,[\lambda_{22} - \lambda_{12}]$$

$$\Rightarrow \quad \frac{P(x \mid c_1)\,P(c_1)}{P(x)}\,[\lambda_{11} - \lambda_{21}] < \frac{P(x \mid c_2)\,P(c_2)}{P(x)}\,[\lambda_{22} - \lambda_{12}]$$

$$\Rightarrow \quad P(x \mid c_1)\,P(c_1)\,[\lambda_{11} - \lambda_{21}] < P(x \mid c_2)\,P(c_2)\,[\lambda_{22} - \lambda_{12}]$$

How to obtain these in practice ?

Gaussian (Normal) Density

Probability density functions (pdf) are often expressed in analytical form. One such pdf that is very commonly encountered is Gaussian Density, also known as Normal Density.

Let X be a d-dimensional feature vector.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_d \end{bmatrix}$$

If each of the elements of the vector X is a Gaussian random variable, the ***multivariate normal density*** is given by

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{C}|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}})^T \mathbf{C}^{-1}(\mathbf{x}-\bar{\mathbf{x}})\right)$$

where $\bar{\mathbf{x}}$ is the mean vector, and $\mathbf{C}$ is the covariance matrix.

$$\underline{\mathbf{x}} = \mathrm{E}\{\mathbf{x}\}$$

$$\mathbf{C} = E\{(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T\}$$

$$\mathbf{C} = \frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix} (\mathbf{x}(i) - \bar{\mathbf{x}})^2 & (\mathbf{x}(i) - \bar{\mathbf{x}})(\mathbf{y}(i) - \bar{\mathbf{y}}) \\ (\mathbf{y}(i) - \bar{\mathbf{y}})(\mathbf{x}(i) - \bar{\mathbf{x}}) & (\mathbf{y}(i) - \bar{\mathbf{y}})^2 \end{bmatrix}$$

$|\mathbf{C}|$ is determinant of $\mathbf{C}$

The linear combination of Gaussian random vectors is also a Gaussian random vector.

Properties of covariance matrix

- $\mathbf{C}$ is always symmetric and positive semi-definite
- $\mathbf{C}$ is usually positive definite. It is positive semi-definite when at least one of the variables have zero variance, or at least 2 of the variables are identical. These are degenerate cases that we will exclude.
- $\mathbf{C}$ is a symmetric matrix (Hermitian symmetric if x is complex)

$$|\mathbf{C}| \neq 0$$ and so $\mathbf{C}^{-1}$ exists.

Maximum Likelihood Estimates

For parameter estimation

Given a set of observations (measurements)

$\mathbf{x} = \{x_1, x_2, x_3, \cdots, x_N\}$ corresponding to time

$\mathbf{t} = \{t_1, t_2, t_3, \cdots, t_N\}$

we want to estimate the parameter vector θ that satisfies

$$x_i = f(\theta, \mathbf{t}_i)$$

The maximum likelihood estimate of θ

is the $\hat{\theta}$ that maximizes $P(\mathbf{x} \mid \theta)$

In computing the ML estimates, it is frequently more convenient and easier to work with the logarithm of $P(\mathbf{x} \mid \theta)$, i.e.

$$\log P(\mathbf{x} \mid \theta)$$

It is ok to do this because logarithm is a monotonically increasing function.

If $\{x_1, x_2, x_3, \cdots, x_N\}$

are independent measurements of a random variable x, then

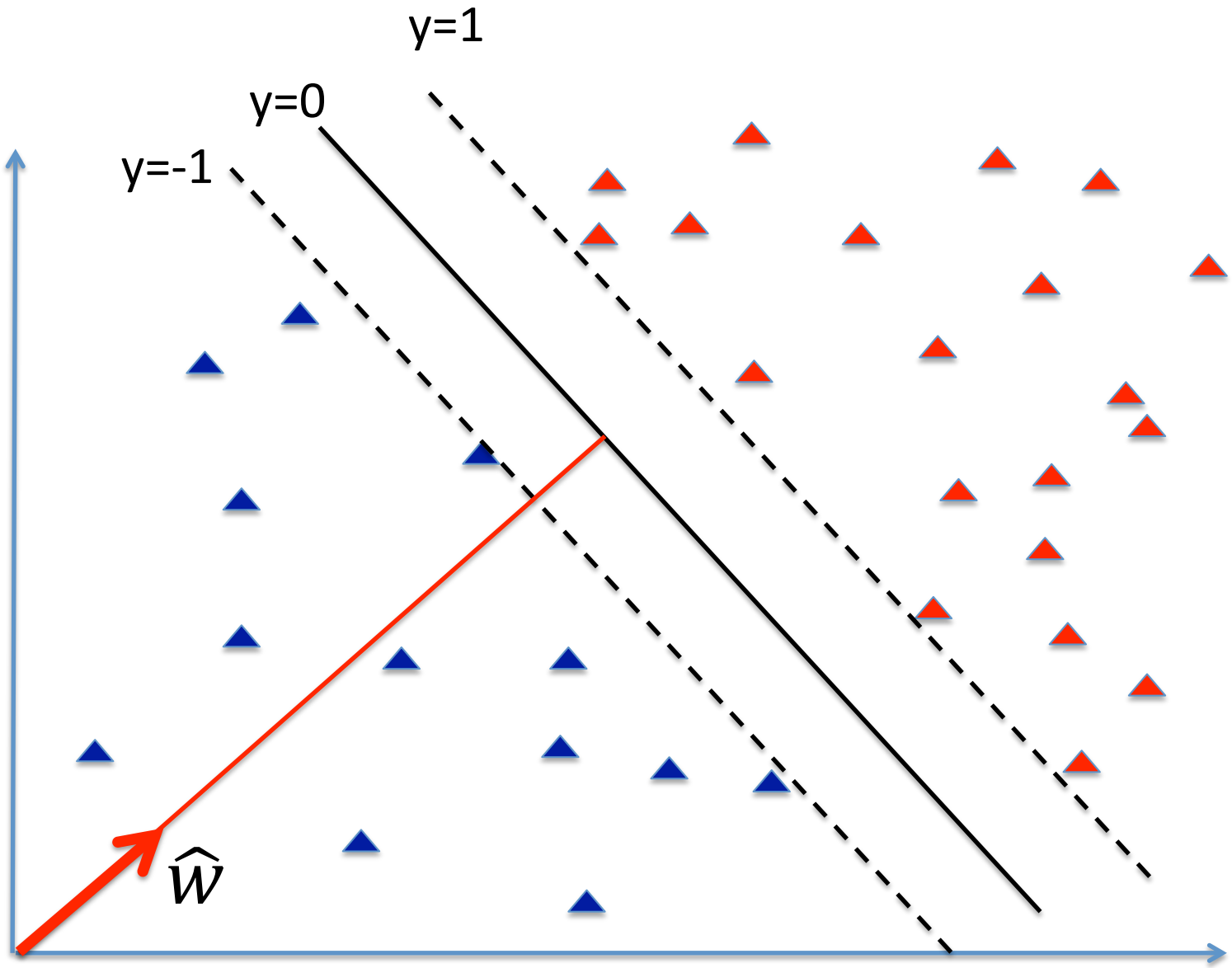
$$P(\mathbf{x} \mid \theta) = \prod_{i=1}^{N} P(x_i \mid \theta)$$

$$\log P(\mathbf{x} \mid \theta) = \sum_{i=1}^{N} \log P(x_i \mid \theta)$$

Finding $\hat{\theta}$ that maximizes $\log P(\mathbf{x} \mid \theta)$ can usually be done by setting

$$\frac{\partial}{\partial \theta} \log P(\mathbf{x} \mid \theta) = 0$$

Support Vector Machines


y=1
y=0
y=-1
$\hat{w}$

$$x^T \hat{w} - |w| = d$$

$$x^T w - |w|^2 = d|w|$$

Let B = -|w|^2 and treat B as 'independent' of w

$$x^T w + B = d|w|$$

note that the equation can be scaled s.t.

$$d|w| = 1 \implies d = \frac{1}{|w|}$$

so in order to maximize the margin d, we need to minimize |w|, or equivalently, minimize $w^T w$ subject to condition

$$y_i\left(x_i^T w + B\right) \geq 1$$

$$\mathrm{L} = \frac{1}{2} w^T w - \sum_i \alpha_i \left(y_i\left(x_i^T w + B\right) - 1\right)$$

$$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i y_i x_i = 0$$

$$w = \sum_i \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial B} = -\sum_i \alpha_i y_i = 0$$

$$\sum_i \alpha_i y_i = 0$$

$$L_D = \frac{1}{2}\sum_i\sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_i \alpha_i y_i (x_i^T \sum_j \alpha_j y_j x_j + B) + \sum_i \alpha_i$$

$$L_D = \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i^T x_j - \sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i$$

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$H_{ij} = y_i y_j x_i^T x_j$$

$$L_D = \sum_i \alpha_i - \frac{1}{2} \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_N \end{bmatrix} H \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}$$

maximize L_D subject to the following conditions

$$\alpha_i \geq 0 \quad \forall i$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

from the α that minimizes L_D , we get w.

The data points that satisfy $\alpha_i > 0$

are the support vectors. Denoting these data points as support vector x_s , we have

$$y_s(x_s^T w + B) = 1$$

$$y_s\left(\sum_{m \in S} \alpha_m y_m x_s^T x_m + B\right) = 1$$

s = set of support vectors x_s

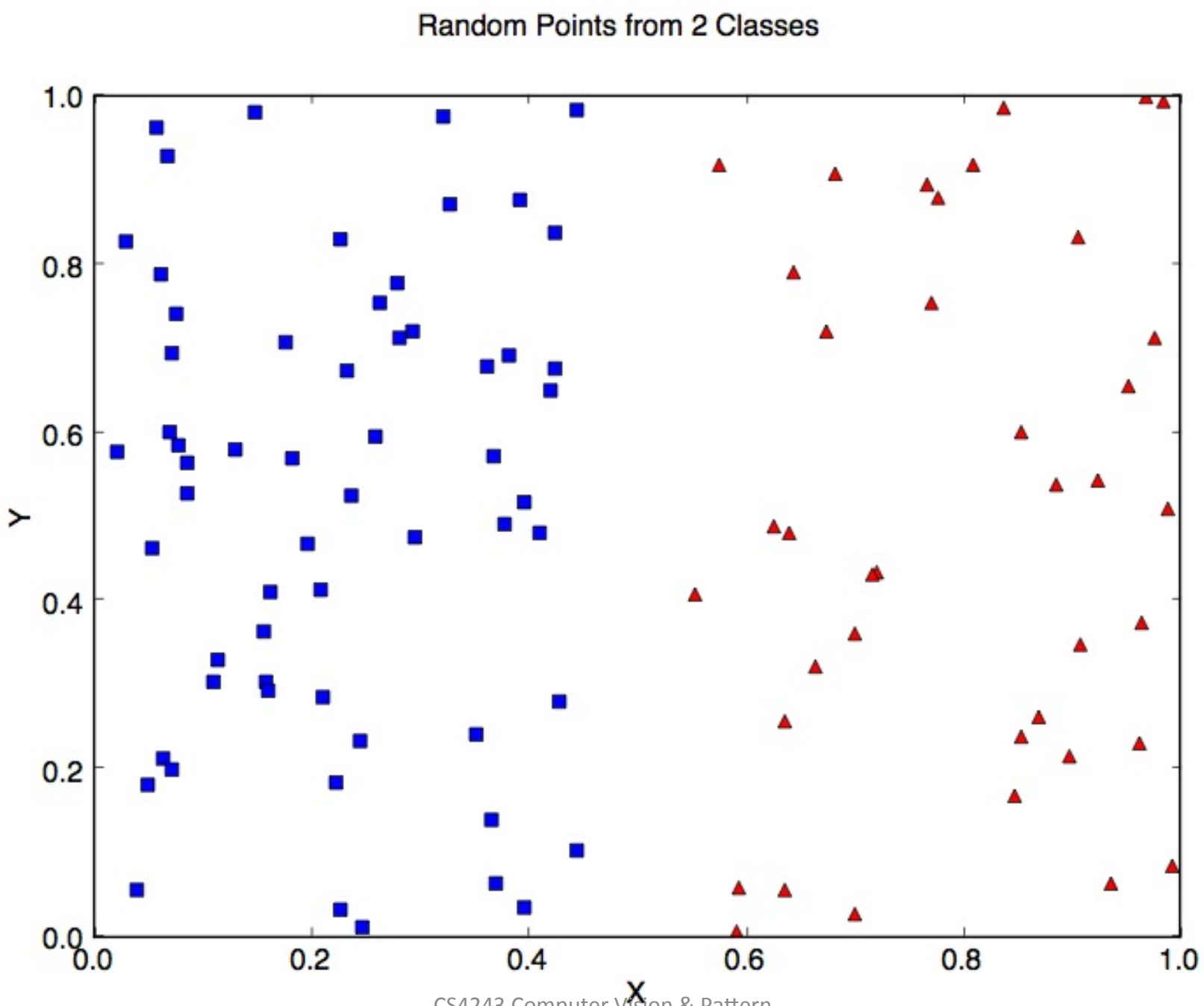
noting that $y_s^2 = 1$, we have

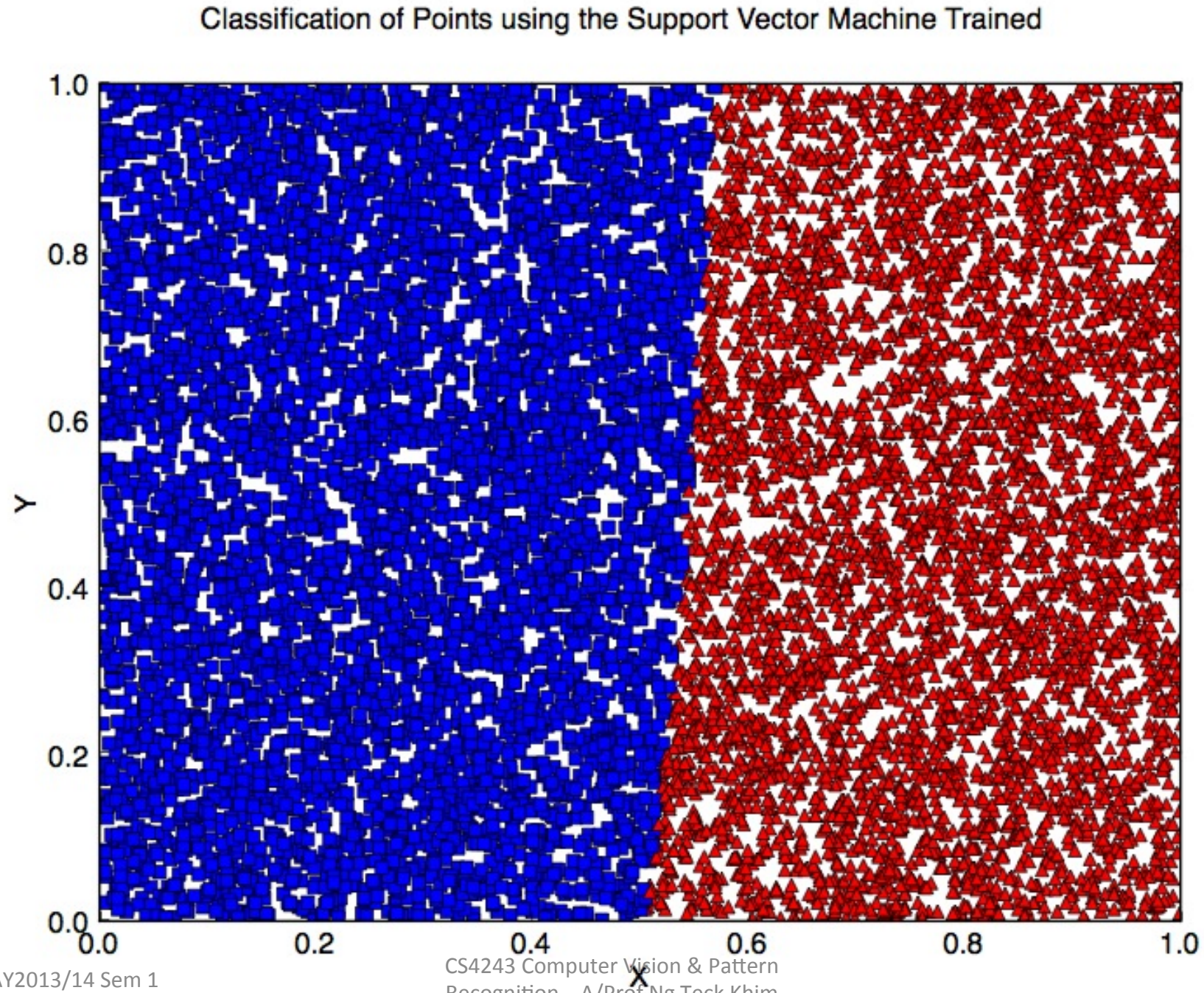
$$\sum_{m \in S} \alpha_m y_m \, x_s^T x_m + B = y_s$$

$$B = y_s - \sum_{m \in S} \alpha_m y_m \, x_s^T x_m$$

making full use of all support vectors, we have

$$B = \frac{1}{K_s} \sum_{s \in S} \left(y_s - \sum_{m \in S} \alpha_m y_m \, x_s^T x_m \right)$$


Random Points from 2 Classes
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X


Classification of Points using the Support Vector Machine Trained
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X

```
import random
import numpy as np
import cv2

import matplotlib.pyplot as plt

npts   = 100

points = np.array(np.random.random((npts, 2)), dtype=np.float32)
labels = np.array(np.random.randint(0, 2, npts), dtype=np.float32)

for p in range(0,npts):
      if points[p][0] <= 0.5:
            labels[p] = 0;
            if points[p][0]>=0.45:
                  points[p][0] = points[p][0] - 0.1
      else:
            labels[p] = 1;
            if points[p][0]<=0.55:
                  points[p][0] = points[p][0] + 0.1
```

```
# Train the SVM:
model = cv2.SVM(points, labels)

# Store it by using OpenCV functions:
model.save("model.xml")

# Now create a new SVM & load the model:
model2 = cv2.SVM()
model2.load("model.xml")

# Predict with model2:
for p in range(0,npts):
        y_val = model2.predict(points[p])
        print y_val, labels[p]

nSupportVectors = model.get_support_vector_count()

nVariables = model.get_var_count()
```

```
f3 = plt.figure()

#for p in labels:
for p in range(0,npts):
    if int(labels[p]) == 1:
        plt.plot(points[p][0], points[p][1], 'r^')
    else:
        plt.plot(points[p][0], points[p][1], 'bs')
plt.ylabel('Y')
plt.xlabel('X')
plt.suptitle('Random Points from 2 Classes')

plt.show()

f3.savefig('RandomPts.jpg')
```

```
f4 = plt.figure()

# generate a big set of test data points to test the support vector
machine
nTestPoints = 10000
testPoints = np.array(np.random.random((nTestPoints, 2)),
dtype=np.float32)

for p in range(0, nTestPoints):
    y_val = model2.predict(testPoints[p])
    if int(y_val) == 1:
        plt.plot(testPoints[p][0], testPoints[p][1], 'r^')
    else:
        plt.plot(testPoints[p][0], testPoints[p][1], 'bs')

plt.ylabel('Y')
```

```
plt.xlabel('X')

plt.suptitle('Classification of Points using the Support Vector Machine Trained')

plt.show()

f4.savefig('SVMClassificationResults.jpg')
```