

---

# CS 422 Data Mining

Lecture 7

October 4, 2018

- 
- ❑ The midterm is on October 11, 6:25 PM

---

## □ Feature Selection

---

- ❑ **Missing Values Ratio**

- ❑ Data columns with too many missing values are unlikely to carry much useful information. Thus data columns with number of missing values greater than a given threshold can be removed. The higher the threshold, the more aggressive the reduction.

- ❑ **Low Variance Filter**

- ❑ **High Correlation Filter**

- ❑ **Random Forests / Ensemble Trees**

- ❑ **Principal Component Analysis (PCA)**

- ❑ **Backward Feature Elimination**

- ❑ **Forward Feature Construction**

- 
- ❑ **Missing Values Ratio.**
  - ❑ **Low Variance Filter**
  - ❑ Similarly to the previous technique, data columns with little changes in the data carry little information. Thus all data columns with variance lower than a given threshold are removed. A word of caution: variance is range dependent; therefore normalization is required before applying this technique
  - ❑ **High Correlation Filter**
  - ❑ **Random Forests / Ensemble Trees**
  - ❑ **Principal Component Analysis (PCA)**
  - ❑ **Backward Feature Elimination**
  - ❑ **Forward Feature Construction**

- 
- ❑ **Missing Values Ratio.**
  - ❑ **Low Variance Filter**
  - ❑ **High Correlation Filter**
  - ❑ Data columns with very similar trends are also likely to carry very similar information. In this case, only one of them will suffice to feed the machine learning model. Here we calculate the correlation coefficient between numerical columns and between nominal columns as the [Pearson's Product Moment Coefficient](#) and the [Pearson's chi square value](#) respectively. Pairs of columns with correlation coefficient higher than a threshold are reduced to only one. A word of caution: correlation is scale sensitive; therefore column normalization is required for a meaningful correlation comparison
  - ❑ **Random Forests / Ensemble Trees**
  - ❑ **Principal Component Analysis (PCA)**
  - ❑ **Backward Feature Elimination**
  - ❑ **Forward Feature Construction**

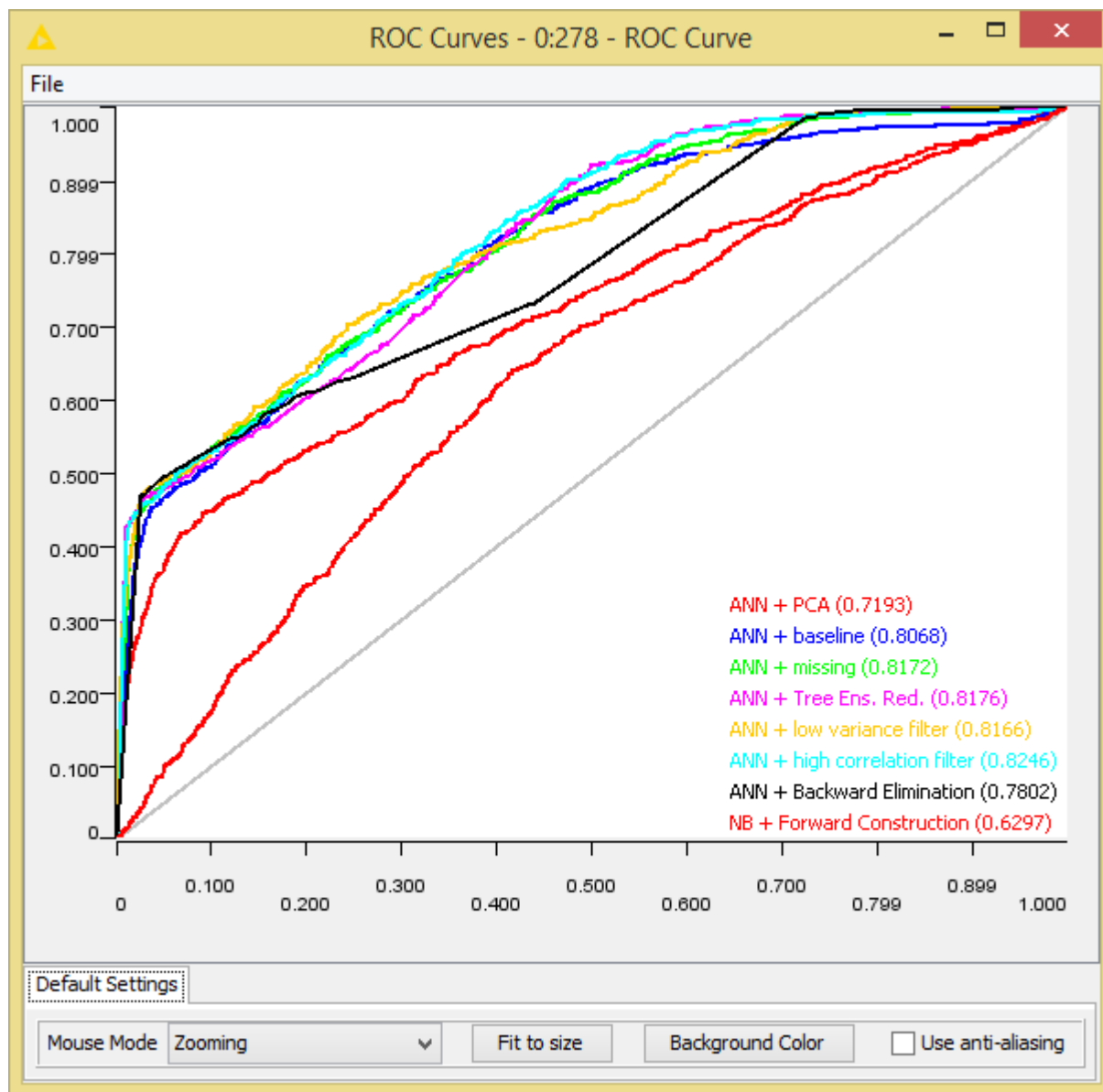
- 
- ❑ **Missing Values Ratio.**
  - ❑ **Low Variance Filter**
  - ❑ **High Correlation Filter**
  - ❑ **Random Forests / Ensemble Trees**
  - ❑ Decision Tree Ensembles, also referred to as random forests, are useful for feature selection in addition to being effective classifiers. One approach to dimensionality reduction is to generate a large and carefully constructed set of trees against a target attribute and then use each attribute's usage statistics to find the most informative subset of features. Specifically, we can generate a large set (2000) of very shallow trees (2 levels), with each tree being trained on a small fraction (3) of the total number of attributes. If an attribute is often selected as best split, it is most likely an informative feature to retain. A score calculated on the attribute usage statistics in the random forest tells us – relative to the other attributes – which are the most predictive attributes.
  - ❑ **Principal Component Analysis (PCA)**
  - ❑ **Backward Feature Elimination**
  - ❑ **Forward Feature Construction**

- 
- ❑ **Missing Values Ratio.**
  - ❑ **Low Variance Filter**
  - ❑ **High Correlation Filter**
  - ❑ **Random Forests / Ensemble Trees**
  - ❑ **Principal Component Analysis (PCA)**
  - ❑ Principal Component Analysis (PCA) is a statistical procedure that orthogonally transforms the original  $n$  coordinates of a data set into a new set of  $n$  coordinates called principal components.
  - ❑ **Backward Feature Elimination**
  - ❑ **Forward Feature Construction**



- 
- ❑ **Missing Values Ratio.**
  - ❑ **Low Variance Filter**
  - ❑ **High Correlation Filter**
  - ❑ **Random Forests / Ensemble Trees**
  - ❑ **Principal Component Analysis (PCA)**
  - ❑ **Backward Feature Elimination**
  - ❑ In this technique, at a given iteration, the selected classification algorithm is trained on  $n$  input features. Then we remove one input feature at a time and train the same model on  $n-1$  input features  $n$  times. The input feature whose removal has produced the smallest increase in the error rate is removed, leaving us with  $n-1$  input features. The classification is then repeated using  $n-2$  features, and so on. Each iteration  $k$  produces a model trained on  $n-k$  features and an error rate  $e(k)$ . Selecting the maximum tolerable error rate, we define the smallest number of features necessary to reach that classification performance with the selected machine learning algorithm
  - ❑ **Forward Feature Construction**

- 
- ❑ **Missing Values Ratio.**
  - ❑ **Low Variance Filter**
  - ❑ **High Correlation Filter**
  - ❑ **Random Forests / Ensemble Trees**
  - ❑ **Principal Component Analysis (PCA)**
  - ❑ **Backward Feature Elimination**
  - ❑ **Forward Feature Construction**
  - ❑ This is the inverse process to the Backward Feature Elimination. We start with 1 feature only, progressively adding 1 feature at a time, i.e. the feature that produces the highest increase in performance. Both algorithms, Backward Feature Elimination and Forward Feature Construction, are quite time and computationally expensive. They are practically only applicable to a data set with an already relatively low number of input columns

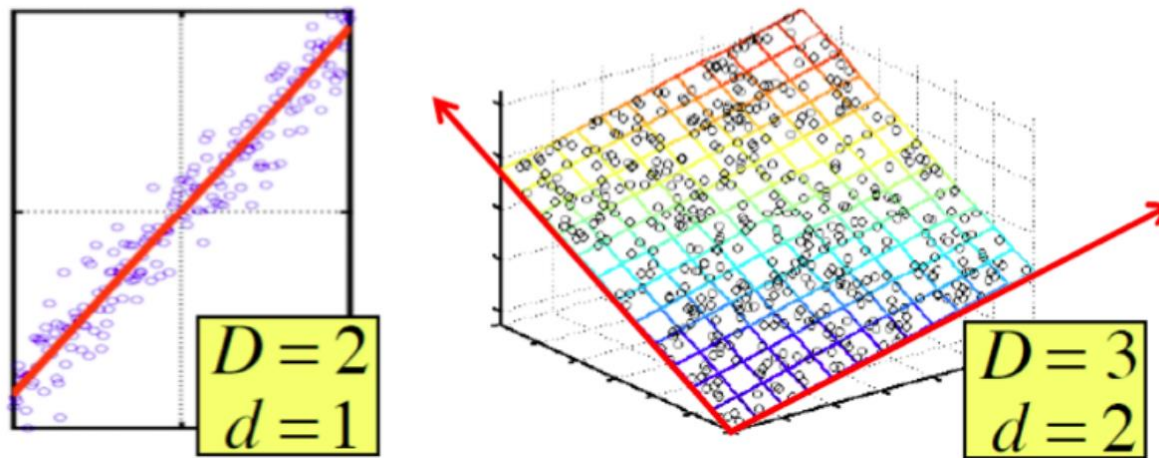


---

## □ Principal Component Analysis (PCA)

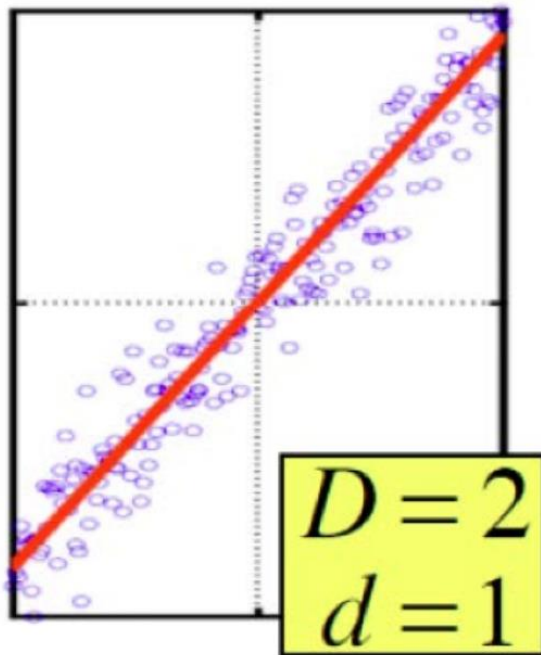
# Subspaces and Projections

---



- **Assumption:** Data lies on or near a low  $d$ -dimensional subspace
- **Axes of this subspace are effective representation of the data**

- 
- Goal of dimensionality reduction is to discover the axis of data!

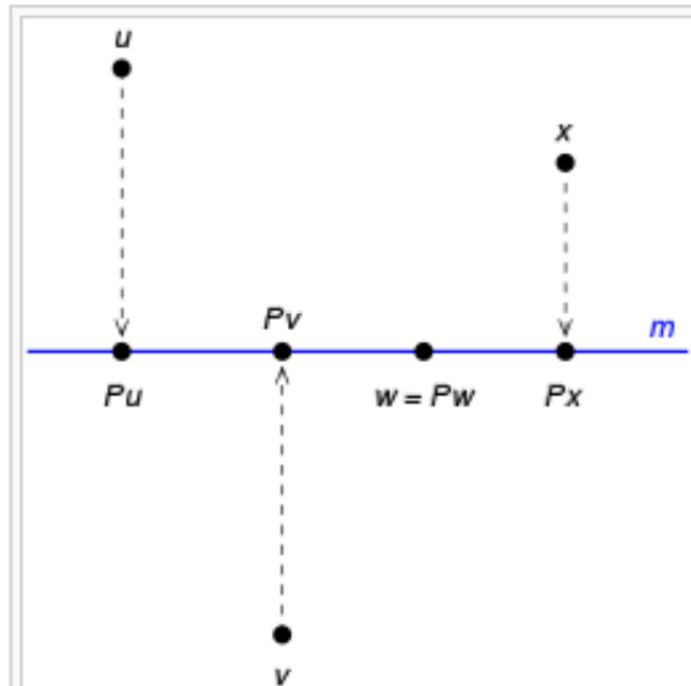


Rather than representing every point with 2 coordinates we represent each point with 1 coordinate (corresponding to the position of the point on the red line).

By doing this we incur a bit of **error** as the points do not exactly lie on the line

---

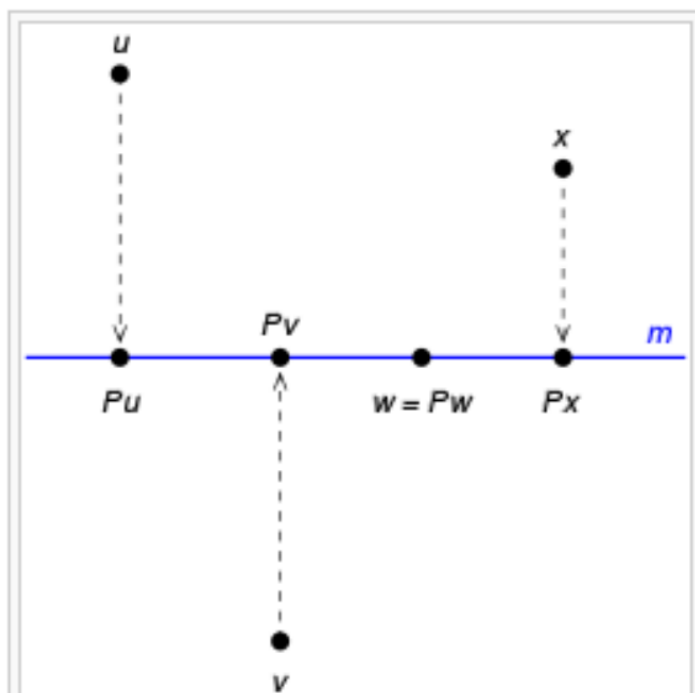
## □ 2 dimensions



The transformation  $P$  is the orthogonal projection onto the line  $m$ .

---

## 2 dimensions



The transformation  $P$  is the orthogonal projection onto the line  $m$ .

## 3 dimensions

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

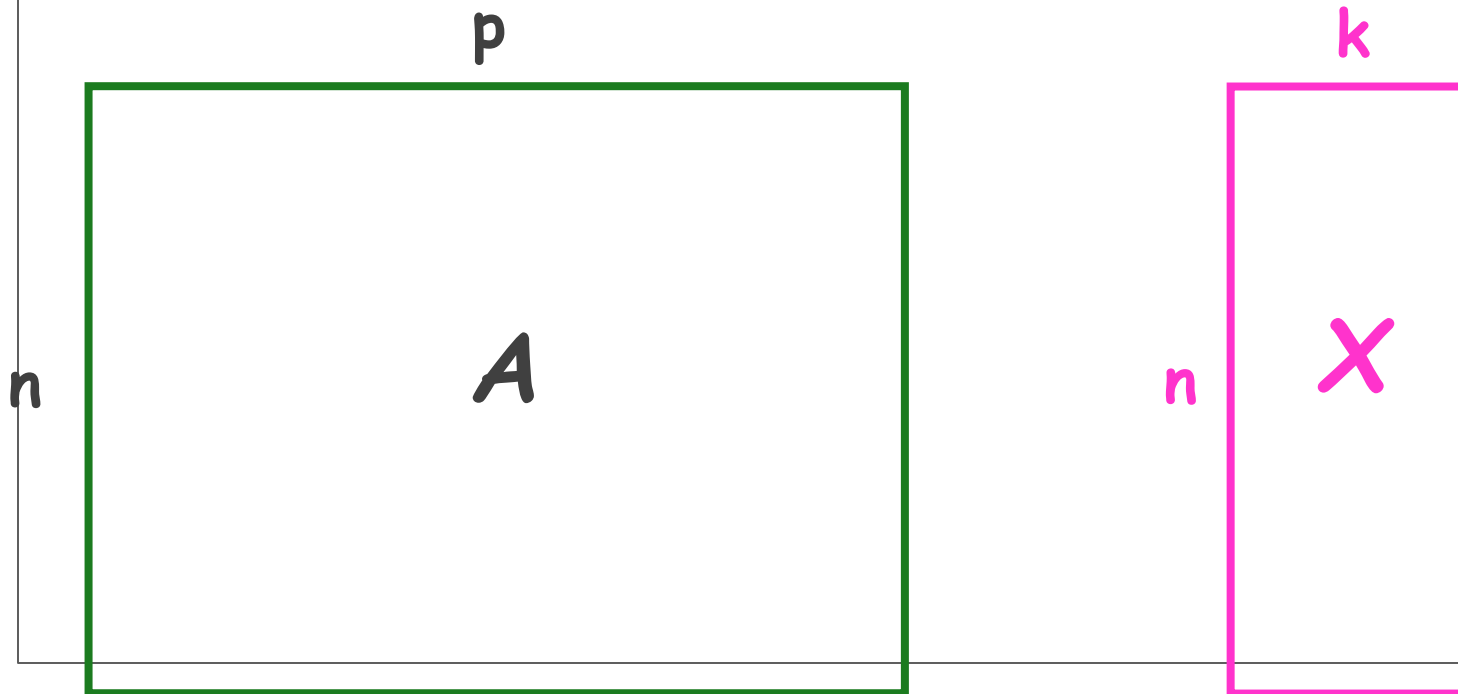
$$P \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$$



# Data Reduction

---

- summarization of data with many ( $p$ ) variables by a smaller set of ( $k$ ) derived (synthetic, composite) variables.



# Data Reduction

---

- ❑ “Residual” variation is information in  $A$  that is not retained in  $X$
- ❑ balancing act between
  - ❑ clarity of representation, ease of understanding
  - ❑ oversimplification: loss of important or relevant information.

# Principal Component Analysis (PCA)

---

- probably the most widely-used and well-known of the “standard” multivariate methods
- invented by Pearson (1901) and Hotelling (1933)

# Principal Component Analysis (PCA)

---

- ❑ takes a data matrix of  $n$  objects by  $p$  variables, which may be correlated, and summarizes it by uncorrelated axes (principal components or principal axes) that are linear combinations of the original  $p$  variables
- ❑ the first  $k$  components display as much as possible of the variation among objects.

# Geometric Rationale of PCA

- objects are represented as a cloud of  $n$  points in a multidimensional space with an axis for each of the  $p$  variables
- the centroid of the points is defined by the mean of each variable
- the variance of each variable is the average squared deviation of its  $n$  values around the mean of that variable.

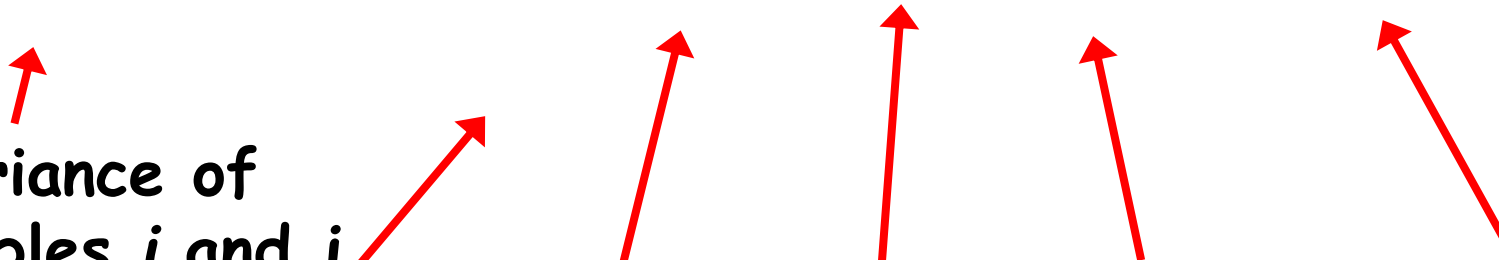
$$V_i = \frac{1}{n-1} \sum_{m=1}^n (X_{im} - \bar{X}_i)^2$$

# Geometric Rationale of PCA

---

- degree to which the variables are linearly correlated is represented by their covariances.

$$C_{ij} = \frac{1}{n-1} \sum_{m=1}^n (x_{im} - \bar{x}_i)(x_{jm} - \bar{x}_j)$$

  
Covariance of variables  $i$  and  $j$   
Sum over all  $n$  objects  
Value of variable  $i$  in object  $m$   
Mean of variable  $i$   
Value of variable  $j$  in object  $m$   
Mean of variable  $j$

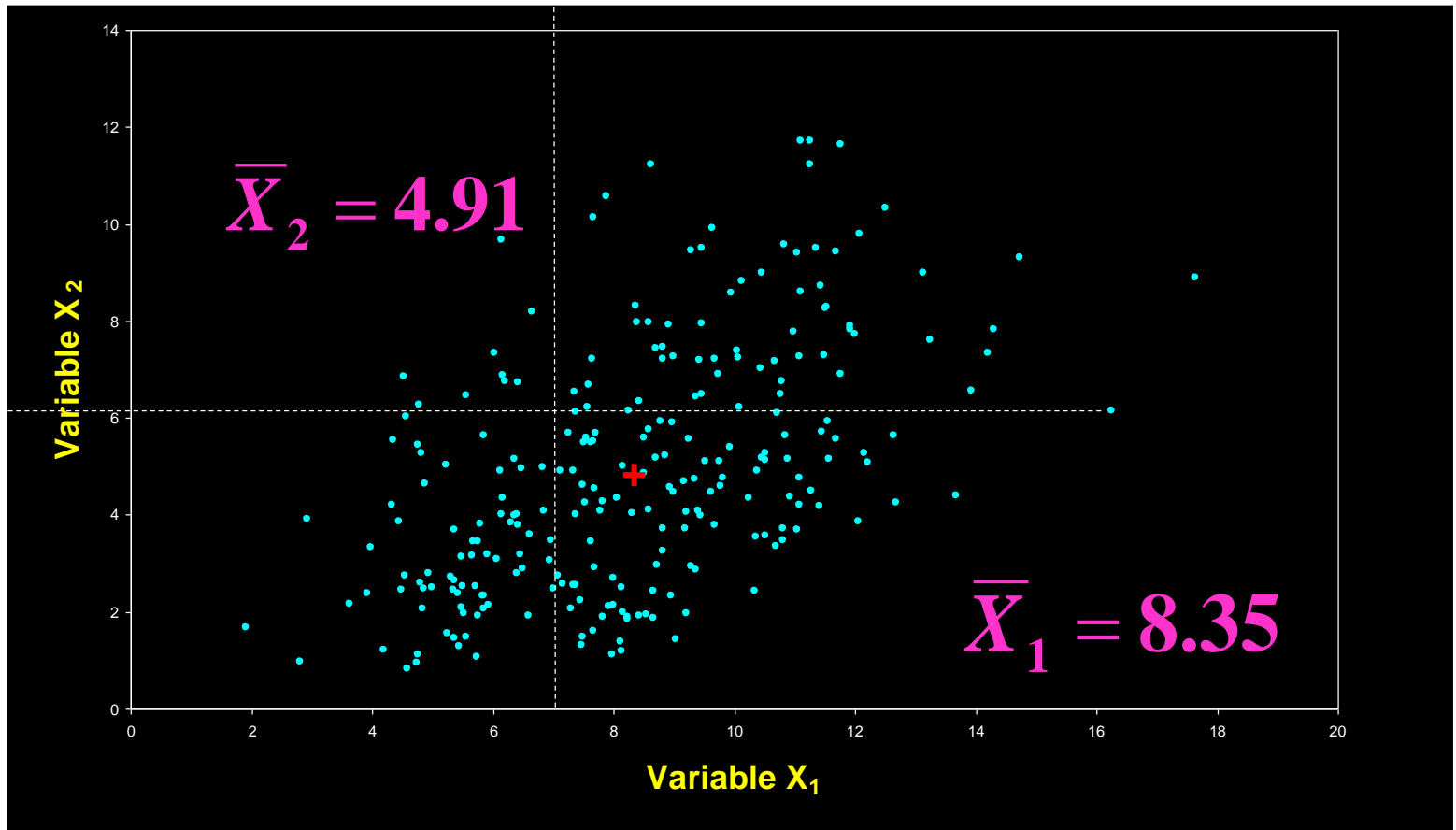
# Geometric Rationale of PCA

---

- ❑ objective of PCA is to rigidly rotate the axes of this  $p$ -dimensional space to new positions (principal axes) that have the following properties:
  - ❑ ordered such that principal axis 1 has the highest variance, axis 2 has the next highest variance, .... , and axis  $p$  has the lowest variance
  - ❑ covariance among each pair of the principal axes is zero (the principal axes are uncorrelated).

## 2D Example of PCA

- variables  $X_1$  and  $X_2$  have positive covariance & each has a similar variance.



$$V_1 = 6.67$$

$$V_2 = 6.24$$

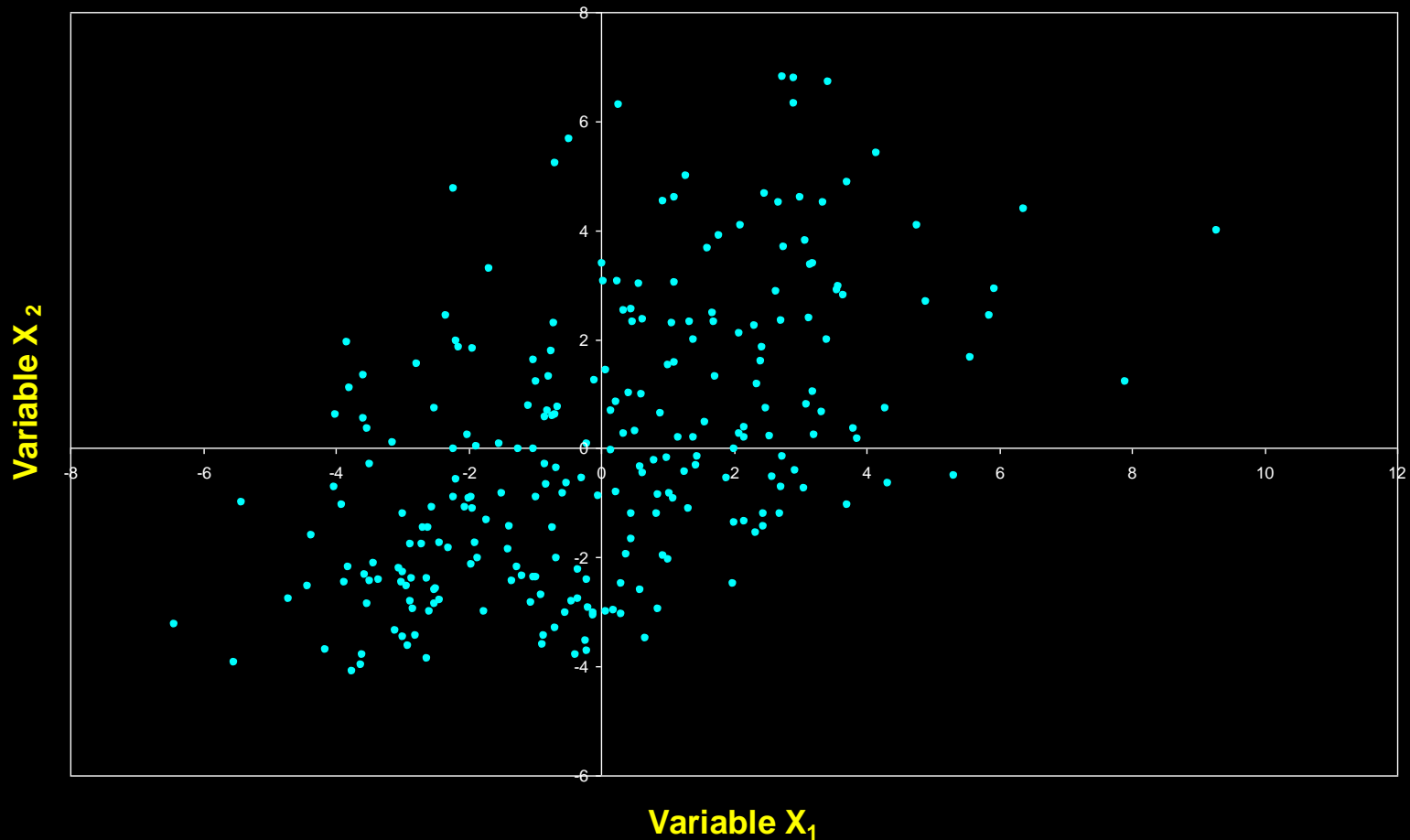
$$C_{1,2} = 3.42$$



# Configuration is Centered

---

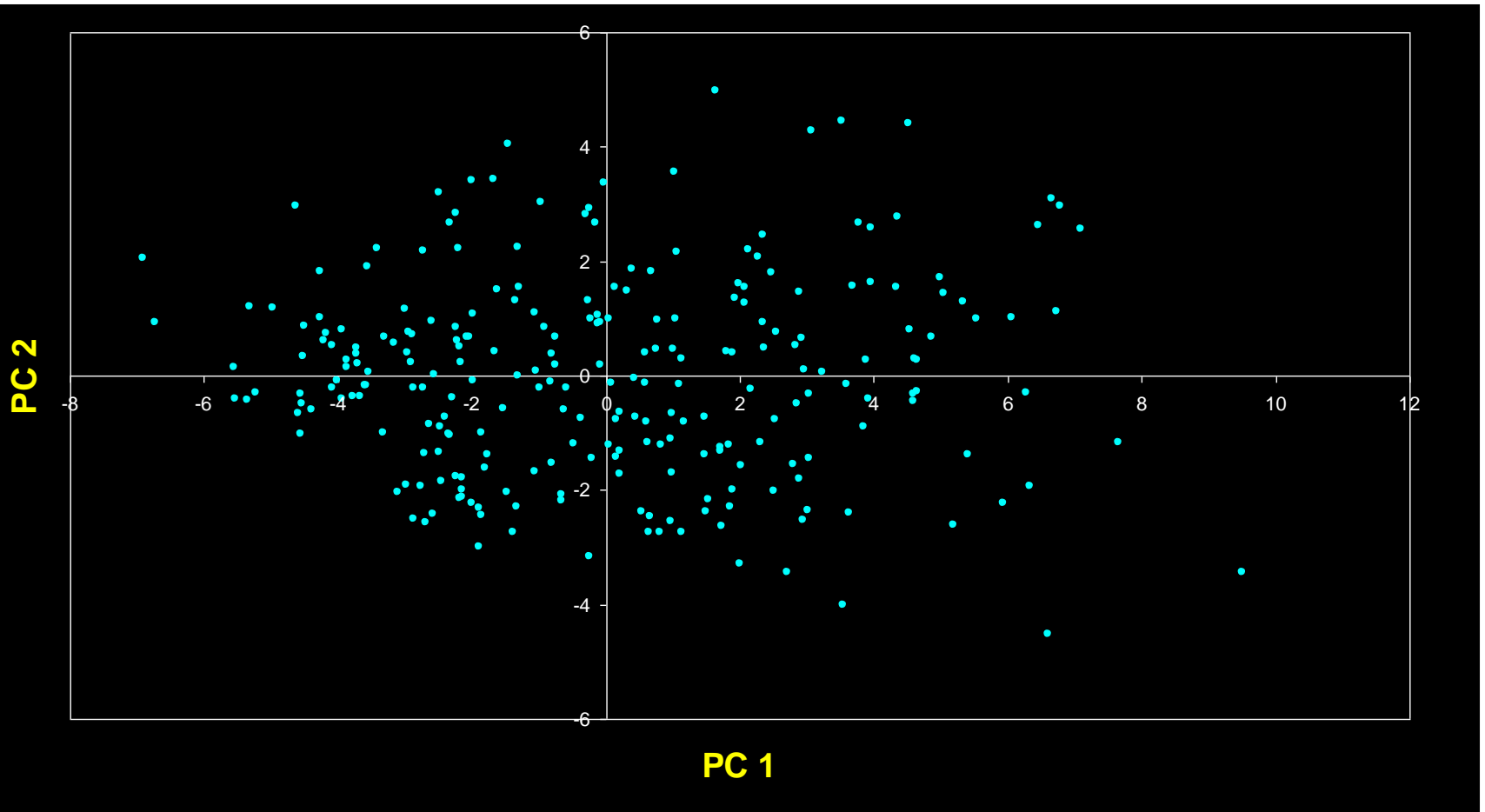
- each variable is adjusted to a mean of zero (by subtracting the mean from each value).



# Principal Components are Computed

---

- ❑ PC 1 has the highest possible variance (9.88)
- ❑ PC 2 has a variance of 3.03
- ❑ PC 1 and PC 2 have zero covariance.



# The Dissimilarity Measure Used in PCA is Euclidean Distance


- ❑ PCA uses Euclidean Distance calculated from the  $p$  variables as the measure of dissimilarity among the  $n$  objects
- ❑ PCA derives the best possible  $k$  dimensional ( $k < p$ ) representation of the Euclidean distances among objects.

---

## Main theoretical result:

The matrix  $G$  consisting of the first  $p$  eigenvectors of the covariance matrix  $S$  solves the following min problem:

$$\min_{G \in \mathbb{R}^{d \times p}} \|X - G(G^T X)\|_F^2 \text{ subject to } G^T G = I_p$$


$$\|X - \bar{X}\|_F^2$$

reconstruction error

PCA projection minimizes the reconstruction error among all linear projections of size  $p$ .

# Generalization to p-dimensions

---

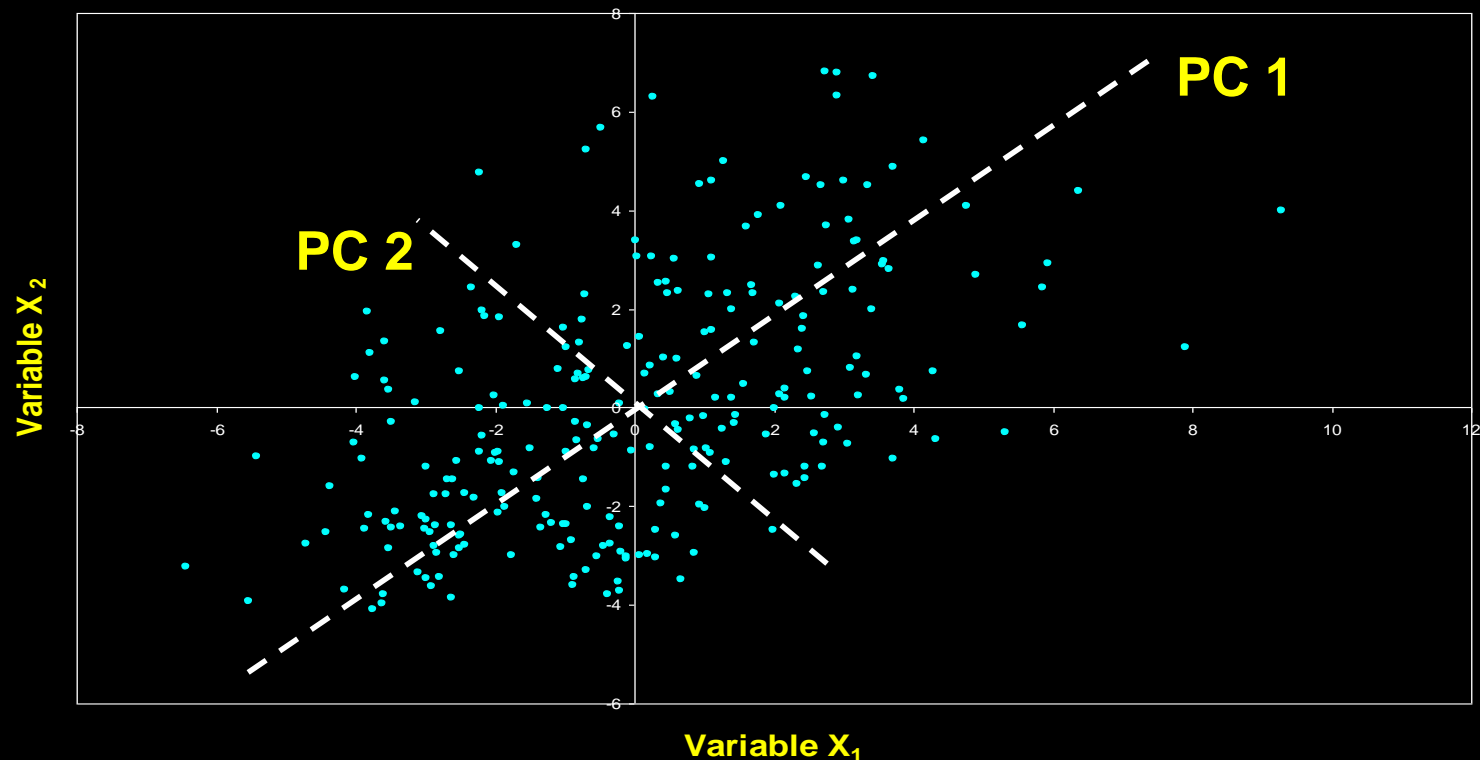
- ❑ In practice nobody uses PCA with only 2 variables
- ❑ The algebra for finding principal axes readily generalizes to  $p$  variables
- ❑ PC 1 is the direction of maximum variance in the  $p$ -dimensional cloud of points
- ❑ PC 2 is in the direction of the next highest variance, subject to the constraint that it has zero covariance with PC 1.

# Generalization to p-dimensions

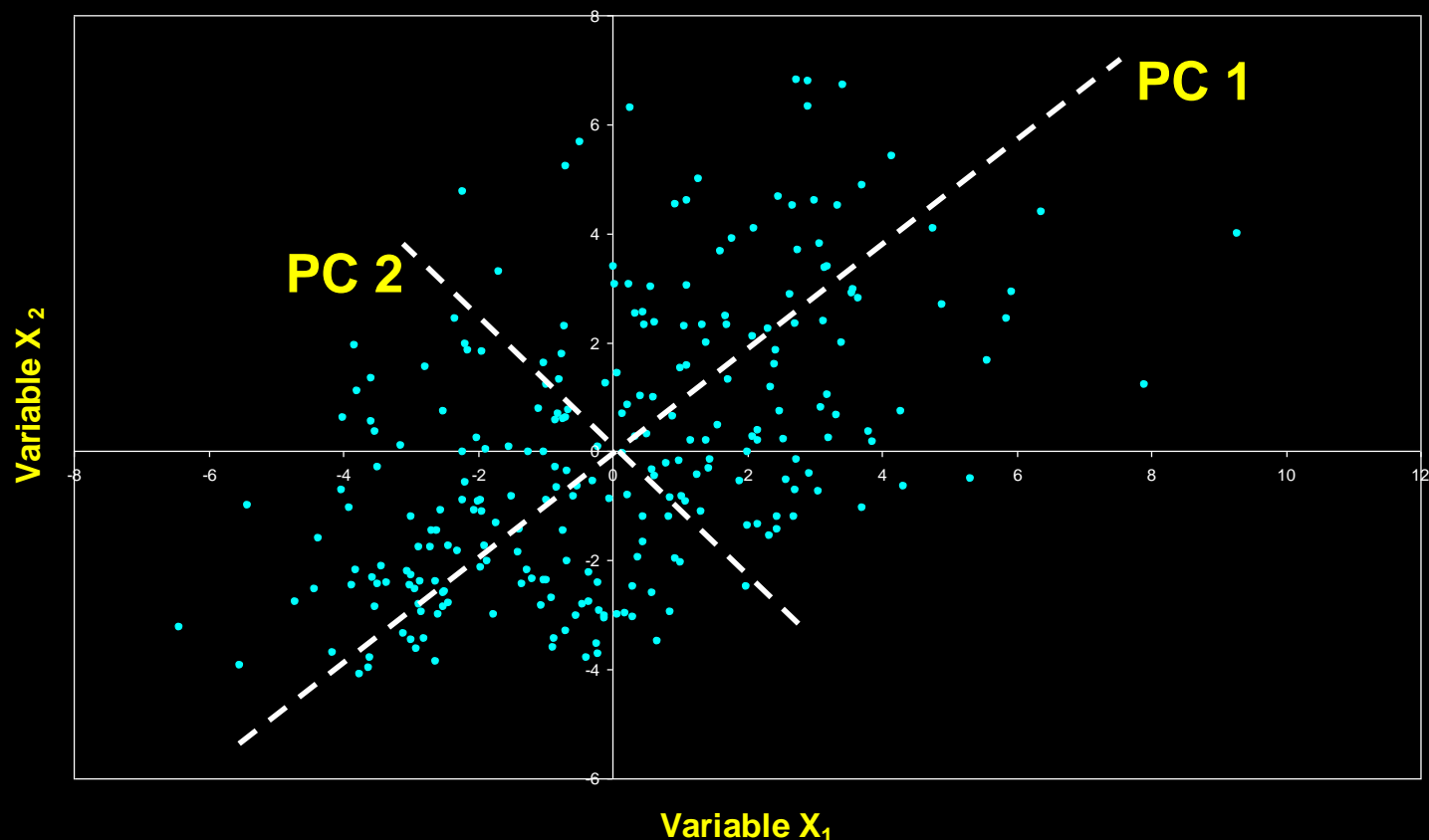
---

- PC 3 is in the direction of the next highest variance, subject to the constraint that it has zero covariance with both PC 1 and PC 2
- and so on... up to PC p

- 
- each principal axis is a linear combination of the original two variables
  - $PC_j = a_{i1}Y_1 + a_{i2}Y_2 + \dots a_{in}Y_n$
  - $a_{ij}$ 's are the coefficients for factor  $i$ , multiplied by the measured value for variable  $j$



- 
- ❑ PC axes are a rigid rotation of the original variables
  - ❑ PC 1 is simultaneously the direction of maximum variance and a least-squares “line of best fit” (squared distances of points away from PC 1 are minimized).





# Generalization to p-dimensions

---

- if we take the first  $k$  principal components, they define the  $k$ -dimensional “hyperplane of best fit” to the point cloud
- of the total variance of all  $p$  variables:
  - PCs 1 to  $k$  represent the maximum possible proportion of that variance that can be displayed in  $k$  dimensions
  - i.e. the squared Euclidean distances among points calculated from their coordinates on PCs 1 to  $k$  are the best possible representation of their squared Euclidean distances in the full  $p$  dimensions.

# Covariance vs Correlation

---

- ❑ using covariances among variables only makes sense if they are measured in the same units
- ❑ even then, variables with high variances will dominate the principal components
- ❑ these problems are generally avoided by standardizing each variable to unit variance and zero mean.

$$X'_{im} = \frac{(X_{im} - \bar{X}_i)}{SD_i}$$

Mean variable  $i$

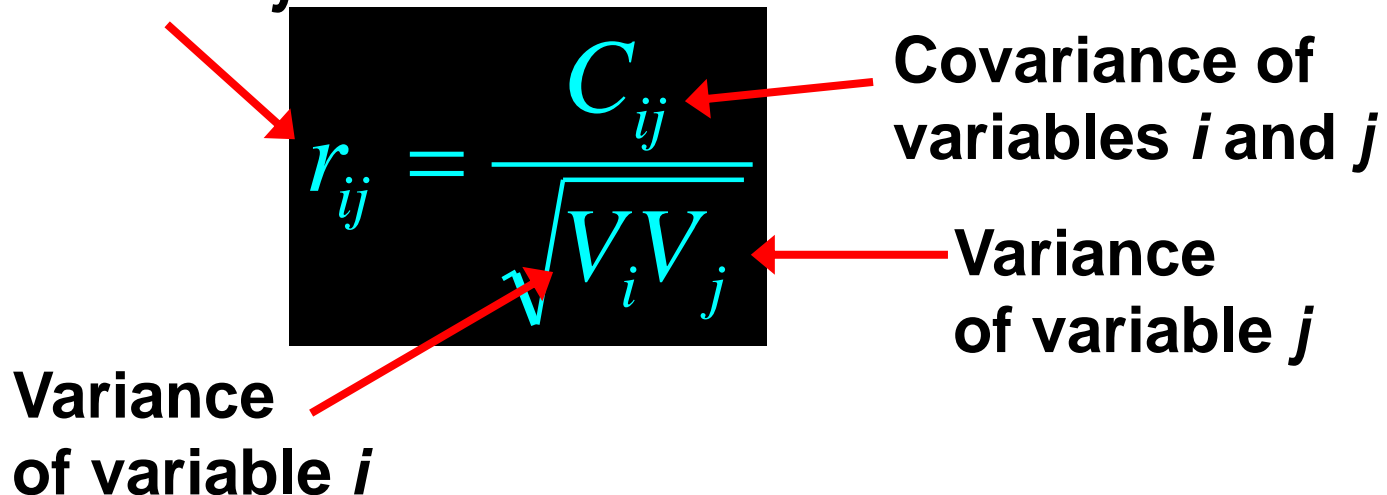
Standard deviation of variable  $i$

# Covariance vs Correlation

---

- ❑ covariances between the standardized variables are correlations
- ❑ after standardization, each variable has a variance of 1.000
- ❑ correlations can be also calculated from the variances and covariances:

**Correlation between variables  $i$  and  $j$**



The diagram shows the formula for the correlation coefficient  $r_{ij}$  between variables  $i$  and  $j$ . The formula is 
$$r_{ij} = \frac{C_{ij}}{\sqrt{V_i V_j}}$$
 where  $C_{ij}$  is the covariance and  $V_i$  and  $V_j$  are the variances. Red arrows point from text labels to the corresponding parts of the formula: one from 'Correlation between variables  $i$  and  $j$ ' to  $r_{ij}$ , one from 'Covariance of variables  $i$  and  $j$ ' to  $C_{ij}$ , one from 'Variance of variable  $i$ ' to  $V_i$ , and one from 'Variance of variable  $j$ ' to  $V_j$ .

$$r_{ij} = \frac{C_{ij}}{\sqrt{V_i V_j}}$$

**Covariance of variables  $i$  and  $j$**

**Variance of variable  $j$**

**Variance of variable  $i$**

# The Algebra of PCA

---

- ❑ first step is to calculate the cross-products matrix of variances and covariances (or correlations) among every pair of the  $p$  variables
- ❑ square, symmetric matrix
- ❑ diagonals are the variances, off-diagonals are the covariances.

	$X_1$	$X_2$
$X_1$	6.6707	3.4170
$X_2$	3.4170	6.2384

**Variance-covariance Matrix**

	$X_1$	$X_2$
$X_1$	1.0000	0.5297
$X_2$	0.5297	1.0000

**Correlation Matrix**

# The Algebra of PCA

---

- in matrix notation, this is computed as

$$S = X'X$$

- where  $X$  is the  $n \times p$  data matrix, with each variable centered (also standardized by SD if using correlations).

	$X_1$	$X_2$
$X_1$	6.6707	3.4170
$X_2$	3.4170	6.2384

**Variance-covariance Matrix**

	$X_1$	$X_2$
$X_1$	1.0000	0.5297
$X_2$	0.5297	1.0000

**Correlation Matrix**

# Manipulating Matrices

---

- transposing: could change the columns to rows or the rows to columns

- $$X = \begin{bmatrix} 10 & 0 & 4 \\ 7 & 1 & 2 \end{bmatrix}$$

$$X' = \begin{bmatrix} 10 & 7 \\ 0 & 1 \\ 4 & 2 \end{bmatrix}$$

- multiplying matrices

- must have the same number of columns in the premultiplicand matrix as the number of rows in the postmultiplicand matrix

# The Algebra of PCA

---

- ❑ sum of the diagonals of the variance-covariance matrix is called the trace
- ❑ it represents the total variance in the data
- ❑ it is the mean squared Euclidean distance between each object and the centroid in p-dimensional space.

	$X_1$	$X_2$
$X_1$	6.6707	3.4170
$X_2$	3.4170	6.2384

**Trace = 12.9091**

	$X_1$	$X_2$
$X_1$	1.0000	0.5297
$X_2$	0.5297	1.0000

**Trace = 2.0000**

# The Algebra of PCA

---

- ❑ finding the principal axes involves eigenanalysis of the cross-products matrix (S)
- ❑ the eigenvalues (latent roots) of S are solutions ( $\lambda$ ) to the characteristic equation

$$|\mathbf{S} - \lambda\mathbf{I}| = 0$$



# The Algebra of PCA

---

- ❑ the eigenvalues,  $\lambda_1, \lambda_2, \dots, \lambda_p$  are the variances of the coordinates on each principal component axis
- ❑ the sum of all  $p$  eigenvalues equals the trace of  $S$  (the sum of the variances of the original variables).

	$x_1$	$x_2$
$x_1$	6.6707	3.4170
$x_2$	3.4170	6.2384

**Trace = 12.9091**

$$\lambda_1 = 9.8783$$

$$\lambda_2 = 3.0308$$


$$\text{Note: } \lambda_1 + \lambda_2 = 12.9091$$

# The Algebra of PCA

---

- each eigenvector consists of p values which represent the “contribution” of each variable to the principal component axis
- eigenvectors are uncorrelated (orthogonal)
  - their cross-products are zero.

## Eigenvectors



	$u_1$	$u_2$
$x_1$	0.7291	-0.6844
$x_2$	0.6844	0.7291

$$0.7291 * (-0.6844) + 0.6844 * 0.7291 = 0$$

# The Algebra of PCA

---

- coordinates of each object  $i$  on the  $k$ th principal axis, known as the scores on PC  $k$ , are computed as

$$z_{ki} = u_{1k}x_{1i} + u_{2k}x_{2i} + \cdots + u_{pk}x_{pi}$$

- where  $Z$  is the  $n \times k$  matrix of PC scores,  $X$  is the  $n \times p$  centered data matrix and  $U$  is the  $p \times k$  matrix of eigenvectors.

# The Algebra of PCA

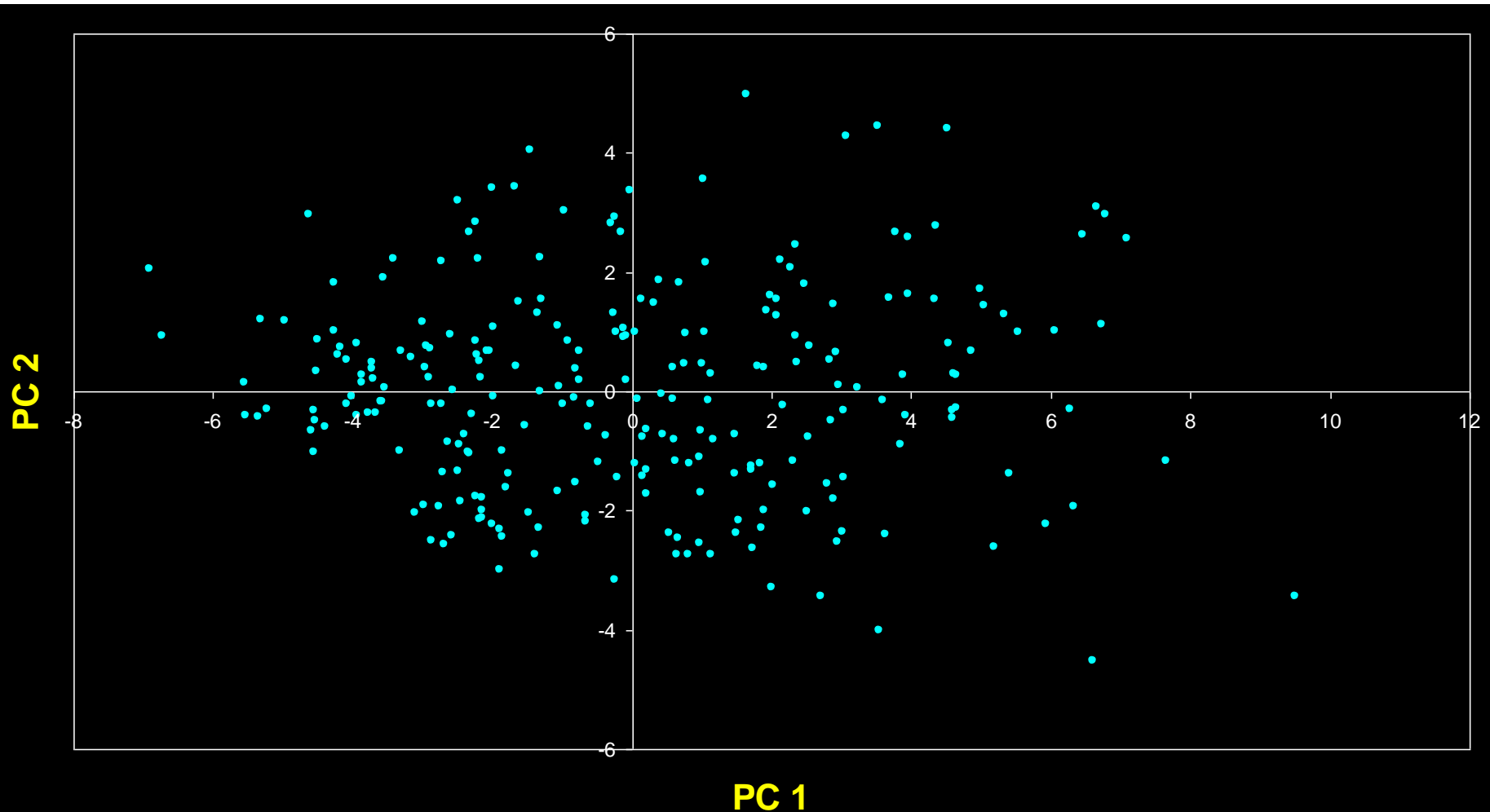
---

- ❑ variance of the scores on each PC axis is equal to the corresponding eigenvalue for that axis
- ❑ the eigenvalue represents the variance displayed (“explained” or “extracted”) by the  $k$ th axis
- ❑ the sum of the first  $k$  eigenvalues is the variance explained by the  $k$ -dimensional ordination.

$\lambda_1 = 9.8783$      $\lambda_2 = 3.0308$     Trace = 12.9091

PC 1 displays ("explains")

$9.8783/12.9091 = 76.5\%$  of the total variance



# The Algebra of PCA

---

- The cross-products matrix computed among the  $p$  principal axes has a simple form:
  - all off-diagonal values are zero (the principal axes are uncorrelated)
  - the diagonal values are the eigenvalues.

	$PC_1$	$PC_2$
$PC_1$	9.8783	0.0000
$PC_2$	0.0000	3.0308

**Variance-covariance Matrix  
of the PC axes**

## A more challenging example

---

- ❑ data from research on habitat definition in the endangered Baw Baw frog
- ❑ 16 environmental and structural variables measured at each of 124 sites
- ❑ correlation matrix used because variables have different units



*Philoria frosti*

# Eigenvalues

---

Axis	Eigenvalue	% of Variance	Cumulative % of Variance
1	5.855	36.60	36.60
2	3.420	21.38	57.97
3	1.122	7.01	64.98
4	1.116	6.97	71.95
5	0.982	6.14	78.09
6	0.725	4.53	82.62
7	0.563	3.52	86.14
8	0.529	3.31	89.45
9	0.476	2.98	92.42
10	0.375	2.35	94.77



# Interpreting Eigenvectors

- correlations between variables and the principal axes are known as loadings
- each element of the eigenvectors represents the contribution of a given variable to a component

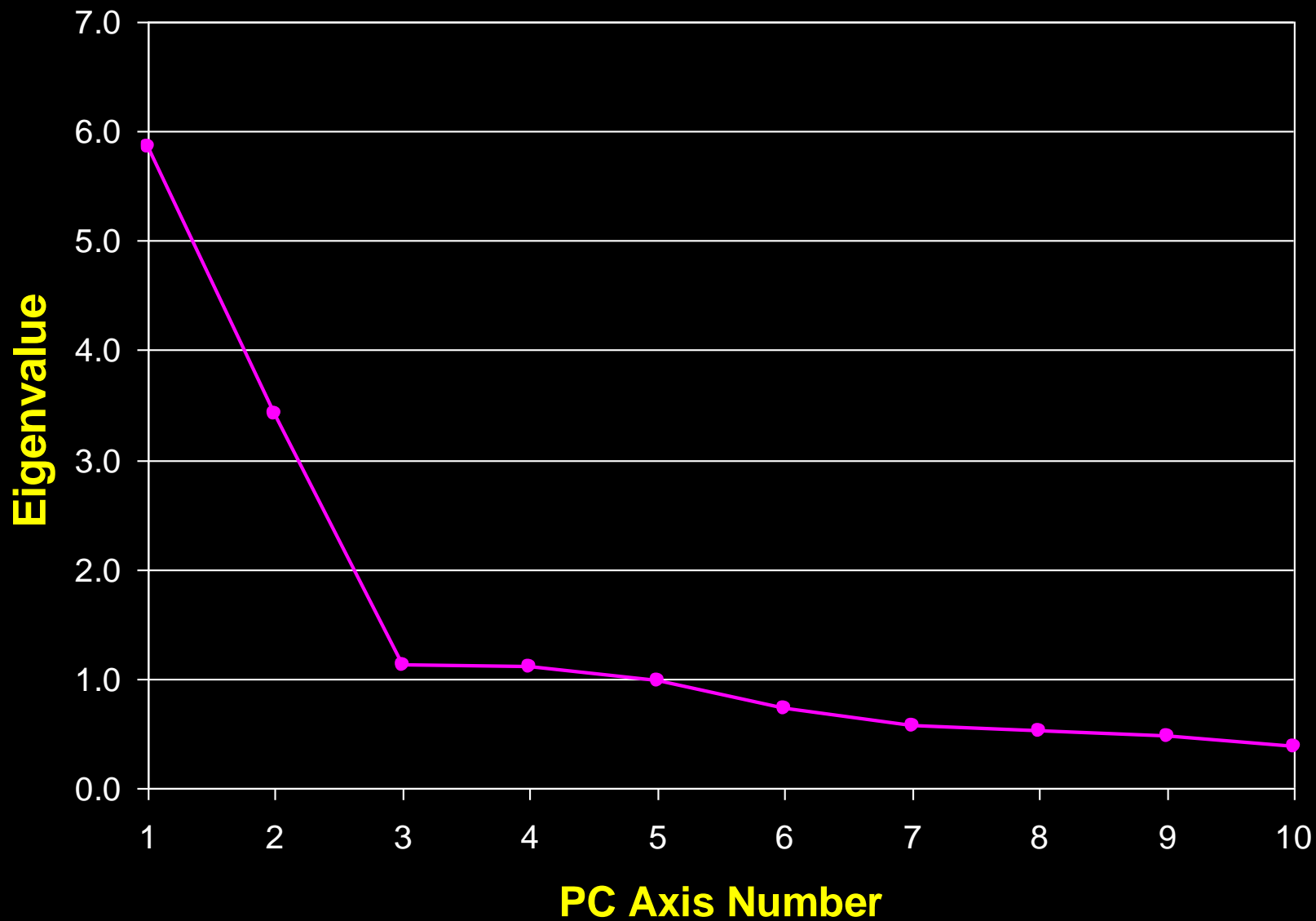
	1	2	3
Altitude	0.3842	0.0659	-0.1177
pH	-0.1159	0.1696	-0.5578
Cond	-0.2729	-0.1200	0.3636
TempSurf	0.0538	-0.2800	0.2621
Relief	-0.0765	0.3855	-0.1462
maxERht	0.0248	0.4879	0.2426
avERht	0.0599	0.4568	0.2497
%ER	0.0789	0.4223	0.2278
%VEG	0.3305	-0.2087	-0.0276
%LIT	-0.3053	0.1226	0.1145
%LOG	-0.3144	0.0402	-0.1067
%W	-0.0886	-0.0654	-0.1171
H1Moss	0.1364	-0.1262	0.4761
DistSWH	-0.3787	0.0101	0.0042
DistSW	-0.3494	-0.1283	0.1166
DistMF	0.3899	0.0586	-0.0175

# How many axes are needed?

---

- ❑ does the  $(k+1)$ th principal axis represent more variance than would be expected by chance?
- ❑ several tests and rules have been proposed
- ❑ a common “rule of thumb” when PCA is based on correlations is that axes with eigenvalues  $> 1$  are worth interpreting

## Baw Baw Frog - PCA of 16 Habitat Variables



# What are the assumptions of PCA?

---

- ❑ assumes relationships among variables are LINEAR
  - ❑ cloud of points in  $p$ -dimensional space has linear dimensions that can be effectively summarized by the principal axes
- ❑ if the structure in the data is NONLINEAR (the cloud of points twists and curves its way through  $p$ -dimensional space), the principal axes will not be an efficient and informative summary of the data.

---

PCA is “an orthogonal linear transformation that transfers the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (*first principal component*), the second greatest variance lies on the second coordinate (*second principal component*), and so on.”

# Background for PCA

---

- Suppose attributes are  $A_1$  and  $A_2$ , and we have  $n$  training examples.  $x$ 's denote values of  $A_1$  and  $y$ 's denote values of  $A_2$  over the training examples.
- Variance of an attribute:

$$\text{var}(A_1) = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{(n-1)}$$

---

□ Covariance of two attributes:

$$\text{cov}(A_1, A_2) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

- If covariance is positive, both dimensions increase together. If negative, as one increases, the other decreases. Zero: independent of each other.

---

## □ Covariance matrix

□ Suppose we have  $n$  attributes,  $A_1, \dots, A_n$ .

□ Covariance matrix:

$$C^{n \times n} = (c_{i,j}), \text{ where } c_{i,j} = \text{cov}(A_i, A_j)$$



	<i>Hours(H)</i>	<i>Mark(M)</i>
Data	9	39
	15	56
	25	93
	14	61
	10	50
	18	75
	0	32
	16	85
	5	42
	19	70
	16	66
	20	80
Totals	167	749
Averages	13.92	62.42

Covariance:

<i>H</i>	<i>M</i>	$(H_i - \bar{H})$	$(M_i - \bar{M})$	$(H_i - \bar{H})(M_i - \bar{M})$
9	39	-4.92	-23.42	115.23
15	56	1.08	-6.42	-6.93
25	93	11.08	30.58	338.83
14	61	0.08	-1.42	-0.11
10	50	-3.92	-12.42	48.69
18	75	4.08	12.58	51.33
0	32	-13.92	-30.42	423.45
16	85	2.08	22.58	46.97
5	42	-8.92	-20.42	182.15
19	70	5.08	7.58	38.51
16	66	2.08	3.58	7.45
20	80	6.08	17.58	106.89
Total				1149.89
Average				104.54

Table 2.2: 2-dimensional data set and covariance calculation

---


$$\begin{pmatrix} \text{cov}(H, H) & \text{cov}(H, M) \\ \text{cov}(M, H) & \text{cov}(M, M) \end{pmatrix}$$

$$= \begin{pmatrix} \text{var}(H) & 104.5 \\ 104.5 & \text{var}(M) \end{pmatrix}$$

$$= \begin{pmatrix} 47.7 & 104.5 \\ 104.5 & 370 \end{pmatrix}$$

**Covariance matrix**

---

## □ Eigenvectors:

□ Let **M** be an  $n \times n$  matrix.

□ **v** is an *eigenvector* of **M** if  $\mathbf{M} \times \mathbf{v} = \lambda \mathbf{v}$

□  $\lambda$  is called the *eigenvalue* associated with **v**

□ For any eigenvector **v** of **M** and scalar  $a$ ,

$$\mathbf{M} \times a\mathbf{v} = \lambda a\mathbf{v}$$

□ Thus you can always choose eigenvectors of length 1:

$$\sqrt{v_1^2 + \dots + v_n^2} = 1$$

□ If **M** has any eigenvectors, it has  $n$  of them, and they are orthogonal to one another.

□ Thus eigenvectors can be used as a new basis for a  $n$ -dimensional vector space.

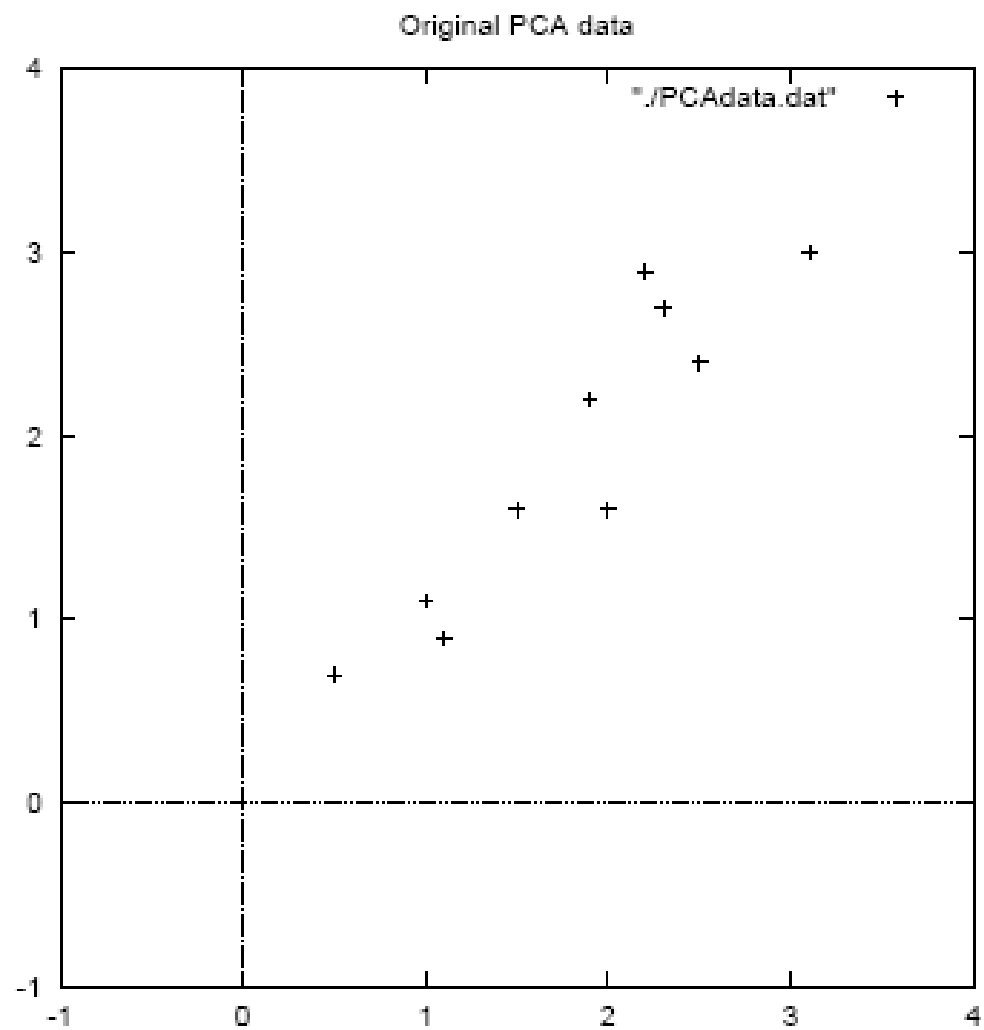
# PCA

---

1. Given original data set  $S = \{\mathbf{x}^1, \dots, \mathbf{x}^k\}$ , produce new set by subtracting the mean of attribute  $A_i$  from each

	$x$	$y$
	2.5	2.4
	0.5	0.7
	2.2	2.9
	1.9	2.2
Data =	3.1	3.0
	2.3	2.7
	2	1.6
	1	1.1
	1.5	1.6
	1.1	0.9
Mean: 1.81		1.91

	$x$	$y$
	.69	.49
	-1.31	-1.21
	.39	.99
	.09	.29
DataAdjust =	1.29	1.09
	.49	.79
	.19	-.31
	-.81	-.81
	-.31	-.31
	-.71	-1.01
Mean: 0		0



2. Calculate the covariance matrix:

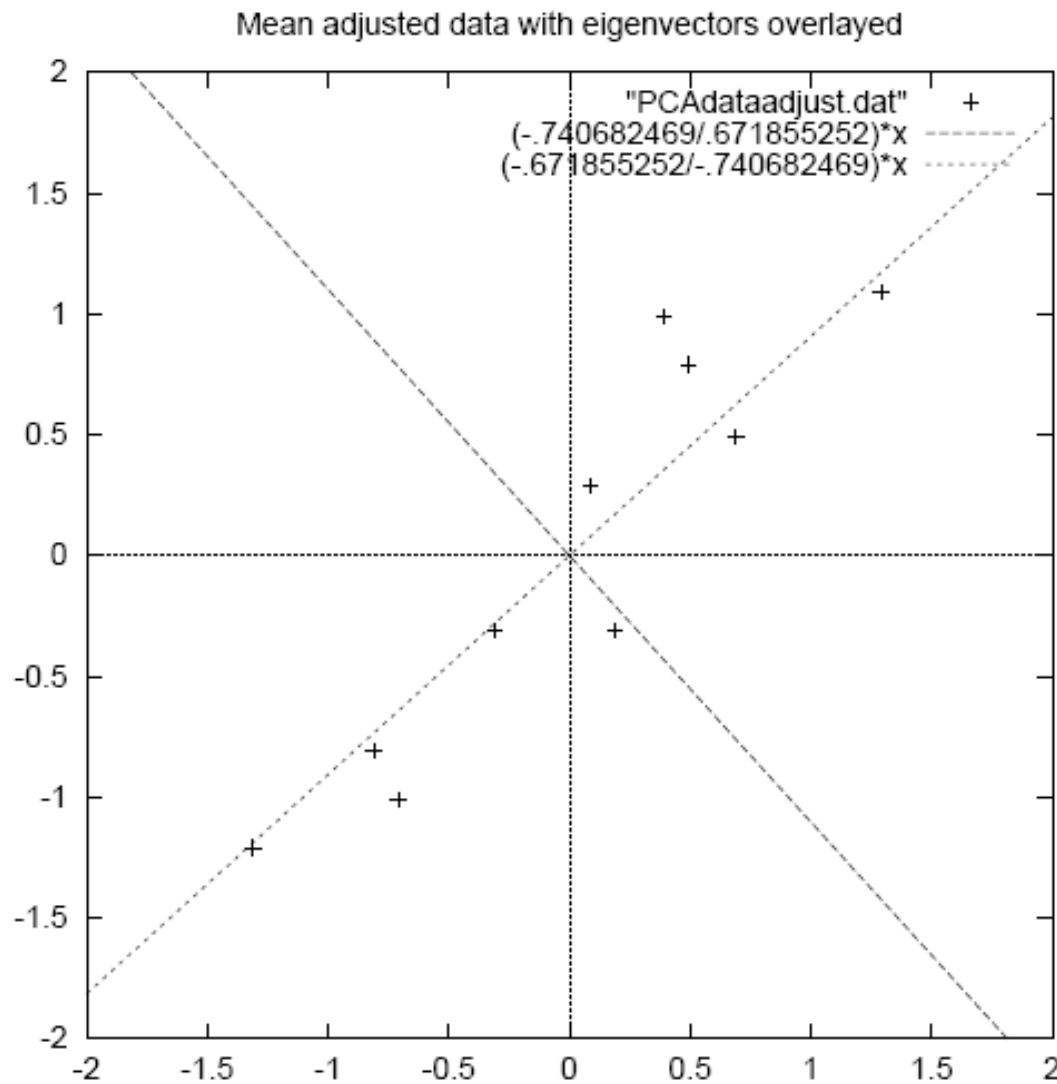
---

$$cov \begin{matrix} \mathbf{x} \\ \mathbf{y} \end{matrix} = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

3. Calculate the (unit) eigenvectors and eigenvalues of the covariance matrix:

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$



**Eigenvector with largest  
eigenvalue traces** ==  
**linear pattern in data**

Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlayed on top.

- 
4. Order eigenvectors by eigenvalue, highest to lowest.

$$\mathbf{v}_1 = \begin{pmatrix} -.677873399 \\ -.735178956 \end{pmatrix} \quad \lambda = 1.28402771$$

$$\mathbf{v}_2 = \begin{pmatrix} -.735178956 \\ .677873399 \end{pmatrix} \quad \lambda = .0490833989$$

In general, you get  $n$  components. To reduce dimensionality to  $p$ , ignore  $n-p$  components at the bottom of the list.

---

Construct new feature vector.

Feature vector =  $(\mathbf{v}_1, \mathbf{v}_2, \dots \mathbf{v}_p)$

$$FeatureVector1 = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

or reduced dimension feature vector :

$$FeatureVector2 = \begin{pmatrix} -.677873399 \\ -.735178956 \end{pmatrix}$$



---

## 5. Derive the new data set.

$$\textit{TransformedData} = \textit{RowFeatureVector} \times \textit{RowDataAdjust}$$

$$\textit{RowFeatureVector1} = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

$$\textit{RowFeatureVector2} = \begin{pmatrix} -.677873399 & -.735178956 \end{pmatrix}$$

$$\textit{RowDataAdjust} = \begin{pmatrix} .69 & -1.31 & .39 & .09 & 1.29 & .49 & .19 & -.81 & -.31 & -.71 \\ .49 & -1.21 & .99 & .29 & 1.09 & .79 & -.31 & -.81 & -.31 & -1.01 \end{pmatrix}$$

This gives original data in terms of chosen components (eigenvectors)—that is, along these axes.

	$x$	$y$
Transformed Data=	-.827970186	-.175115307
	1.77758033	.142857227
	-.992197494	.384374989
	-.274210416	.130417207
	-1.67580142	-.209498461
	-.912949103	.175282444
	.0991094375	-.349824698
	1.14457216	.0464172582
	.438046137	.0177646297
	1.22382056	-.162675287

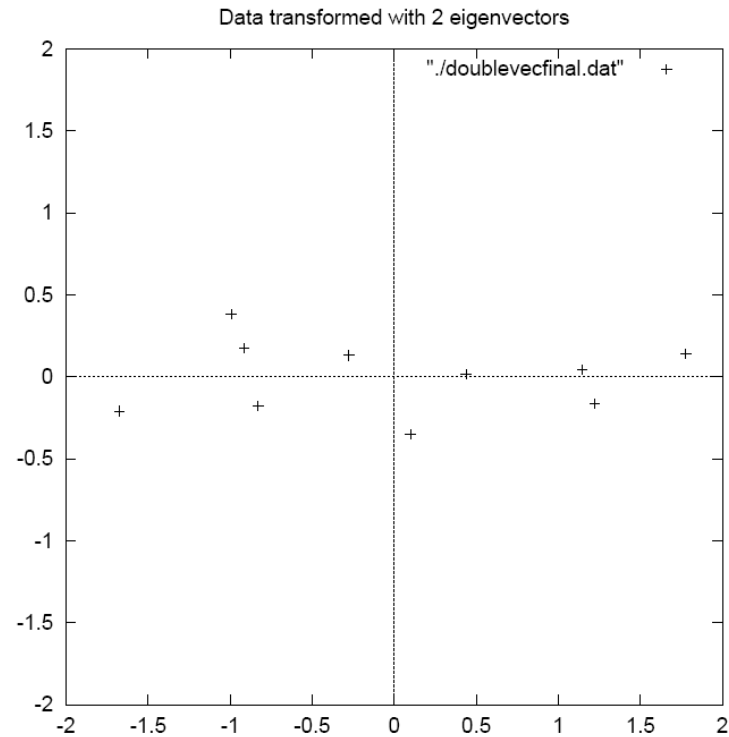
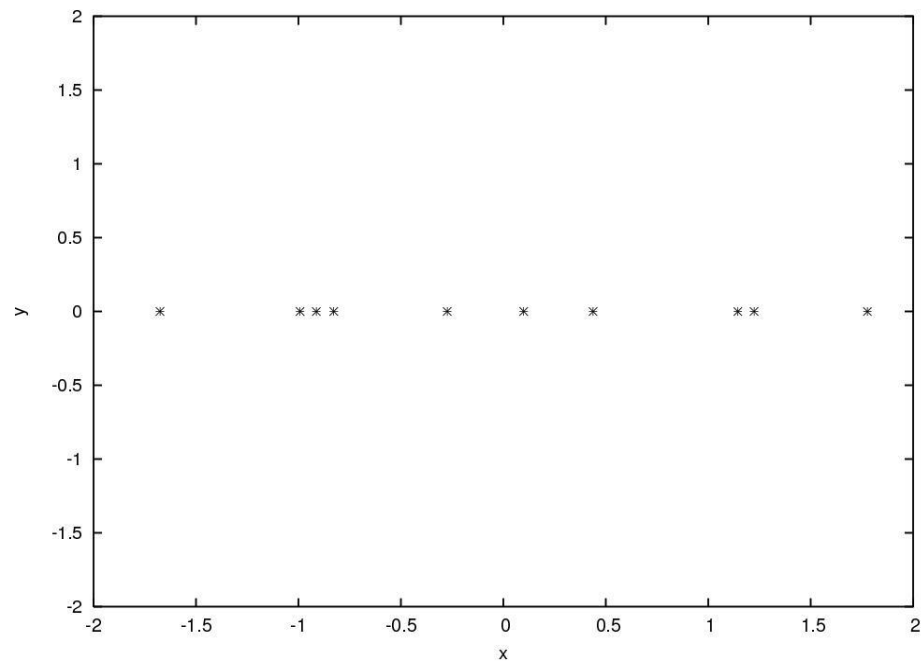


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

### Transformed Data (Single eigenvector)

$x$
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056



# Reconstructing the original data

---

We did:

$$\textit{TransformedData} = \textit{RowFeatureVector} \times \textit{RowDataAdjust}$$

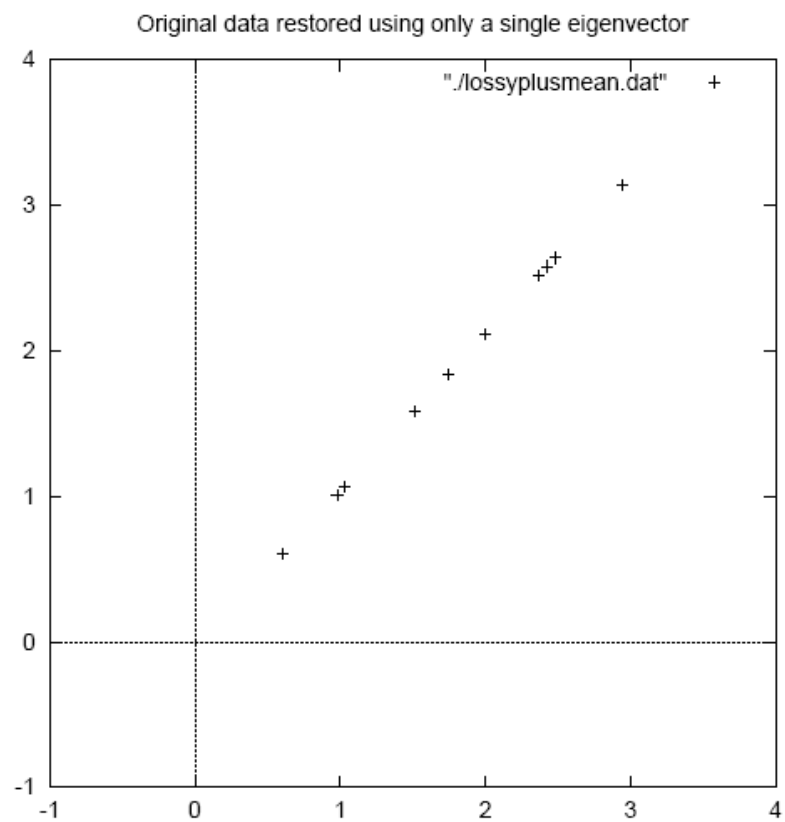
so we can do

$$\textit{RowDataAdjust} = \textit{RowFeatureVector}^{-1} \times \textit{TransformedData}$$

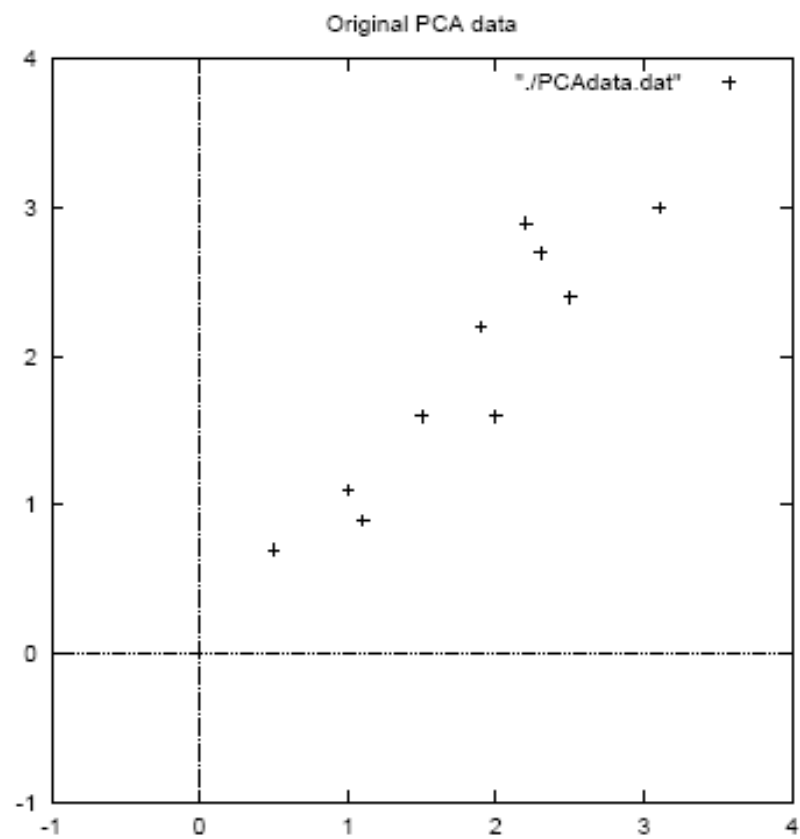
$$= \textit{RowFeatureVector}^T \times \textit{TransformedData}$$

and

$$\textit{RowDataOriginal} = \textit{RowDataAdjust} + \textit{OriginalMean}$$



=

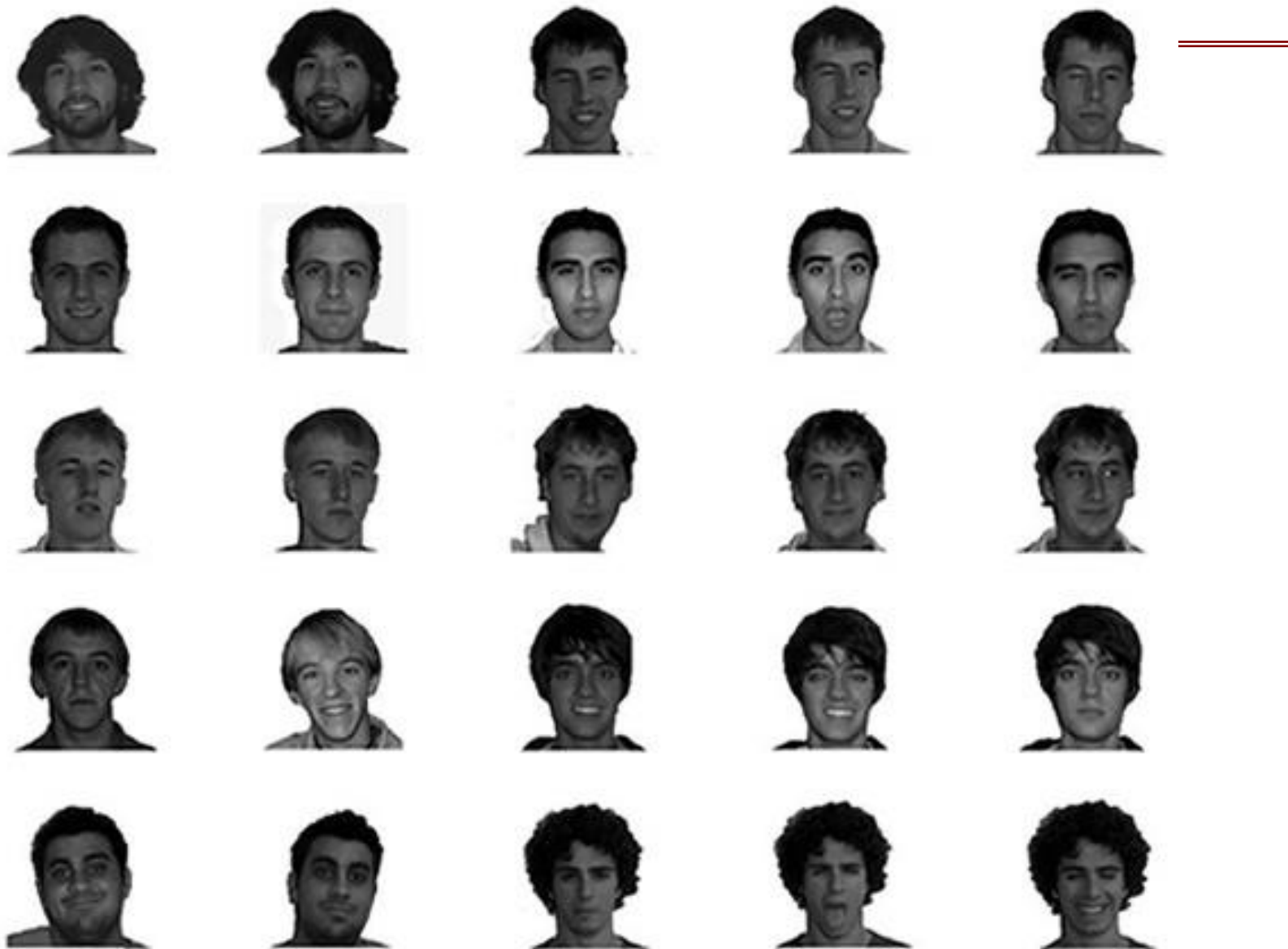


# Example: Linear discrimination using PCA for face recognition

---

## 1. Preprocessing: “Normalize” faces

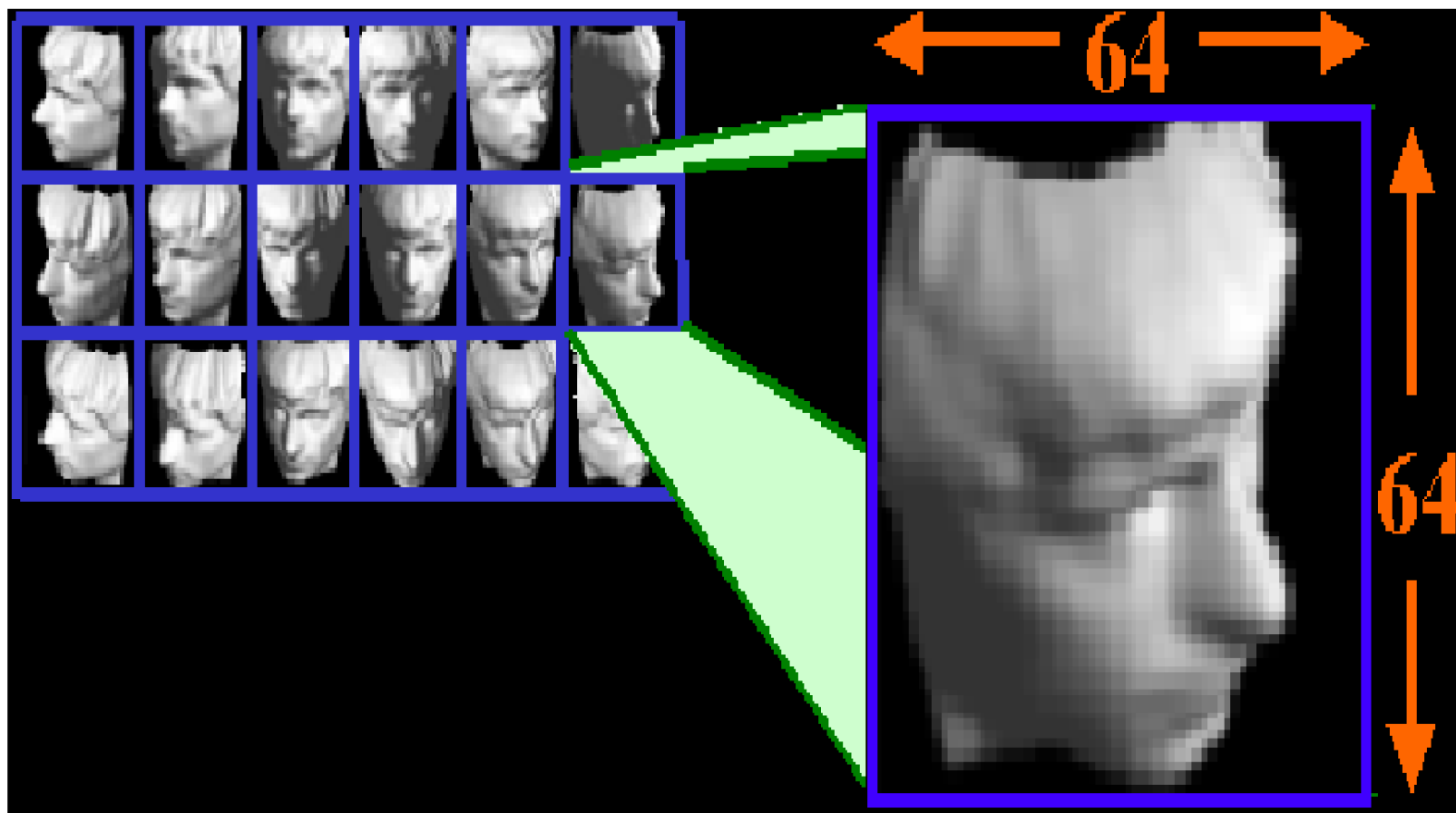
- Make images the same size
- Line up with respect to eyes
- Normalize intensities



- 
2. Raw features are pixel intensity values (2061 features)
  3. Each image is encoded as a vector  $\Gamma_i$  of these features
  4. Compute “mean” face in training set:

$$\Psi = \frac{1}{M} \sum_{i=1}^M \Gamma_i$$

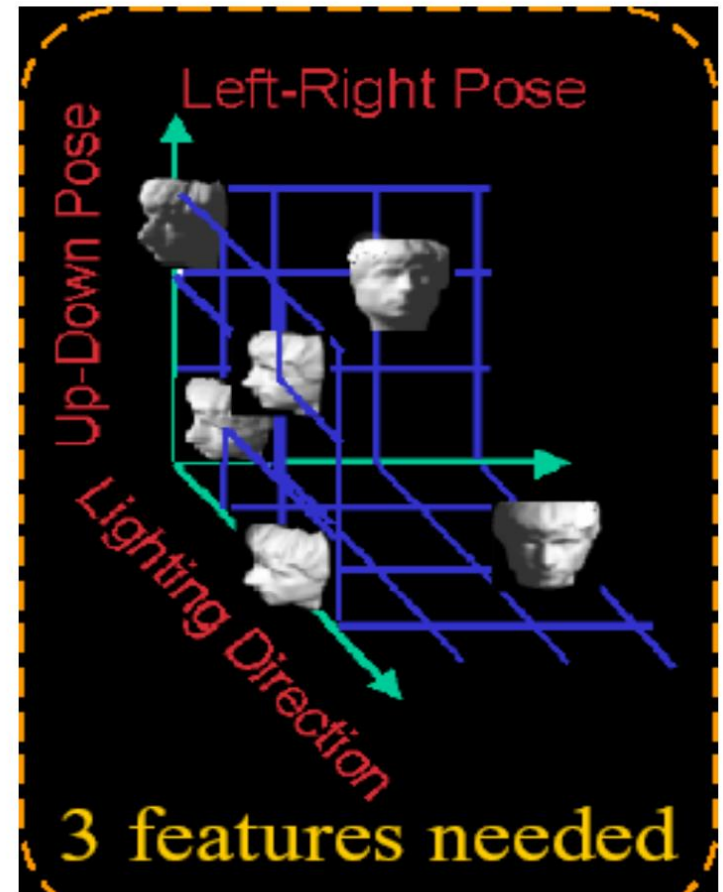




- 
- Every pixel?
  - Or perceptually meaningful structure?

- Up-down pose
- Left-right pose
- Lighting direction

So, your brain successfully reduced the high-dimensional inputs to an intrinsically 3-dimensional manifold!





The average face and first four eigenfaces

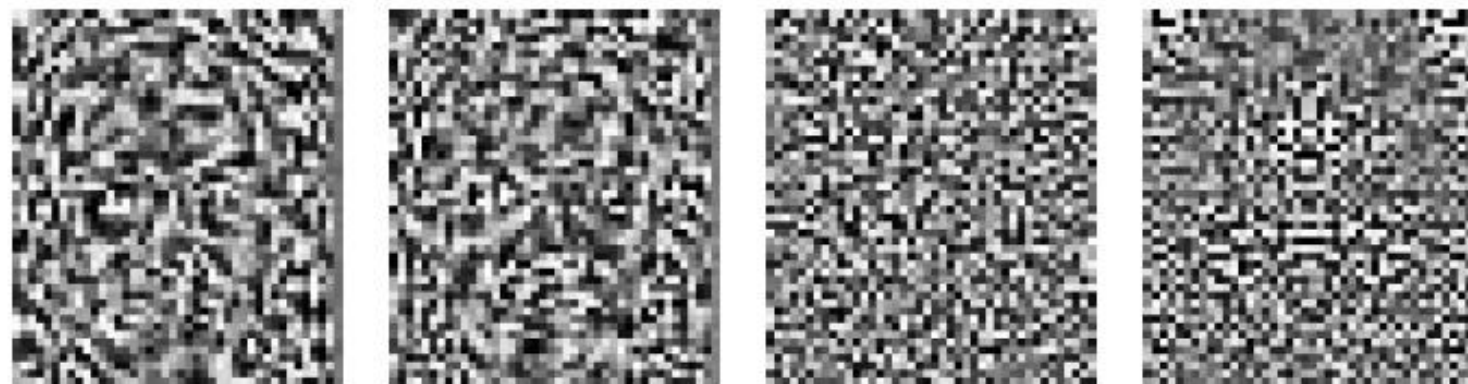
- ❑ Subtract the mean face from each face vector

$$\Phi_i = \Gamma_i - \Psi$$

- ❑ Compute the covariance matrix  $\mathbf{C}$
- ❑ Compute the (unit) eigenvectors  $\mathbf{v}_i$  of  $\mathbf{C}$
- ❑ Keep only the first  $K$  principal components (eigenvectors)



Eigenfaces 15, 100, 200, 250, 300



Eigenfaces 400, 450, 1000, 2000

---

The eigenfaces encode the principal sources of variation in the dataset (e.g., absence/presence of facial hair, skin tone, glasses, etc.).

We can represent any face as a linear combination of these “basis” faces.

Use this representation for:

- Face recognition  
(e.g., Euclidean distance from known faces)
- Linear discrimination  
(e.g., “glasses” versus “no glasses”,  
or “male” versus “female”)



## Finding Similar Sets

# Finding Similar Sets

---

- ❑ Applications
- ❑ Shingling
- ❑ Minhashing
- ❑ Locality-Sensitive Hashing

# Finding Similar Items

---

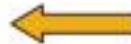
- **Many problems can be expressed as finding “similar” sets:**
  - Find near-neighbors in high-dimensional space
- **Examples:**
  - **Pages with similar words**
    - For duplicate detection, classification by topic
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Images with similar features**
  - Users who visited the similar websites





# Finding Similar Items

---



# Entity Resolution

---

- The *entity-resolution* problem is to examine a collection of records and determine which refer to the same entity.
  - *Entities* could be people, events, etc.
- Typically, we want to merge records if their values in corresponding fields are similar.

# Matching Customer Records

---

- ❑ Company B had about 1 million records of all its customers.
- ❑ Company A had about 1 million records describing customers, some of whom it had signed up for B.
- ❑ Records had name, address, and phone, but for various reasons, they could be different for the same person.

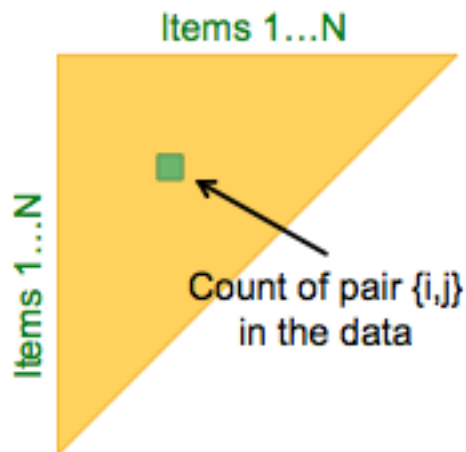
# Matching Customer Records

---

- ❑ Step 1: Design a measure (“score”) of how similar records are:
  - ❑ E.g., deduct points for small misspellings (“Jeffrey” vs. “Jeffery”) or same phone with different area code.
- ❑ Step 2: Score all pairs of records; report high scores as matches.

# Apriori Algorithm

---

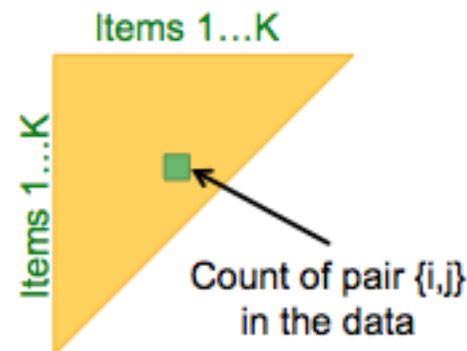


## Naïve solution:

Single pass but requires space quadratic in the number of items

N ... number of distinct items

K ... number of items with support  $\geq s$



## A-priori:

First pass: Find frequent singletons

For a pair to be a **candidate for a frequent pair**, its singletons have to be frequent!

Second pass:

**Count only candidate pairs!**

# Apriori Algorithm

---

## □ Apriori Algorithm

- Counts only for candidates for frequent itemsets

## □ Finding Similar Items

- Compute similarity only between good candidates for similar items

# Problem: Comparing Documents

---

- ❑ Common text, not common topic.
- ❑ Special cases are easy, e.g., identical documents, or one document contained character-by-character in another.
- ❑ General case, where many small pieces of one doc appear out of order in another, is very hard.

# Similar Documents, Cont.

---

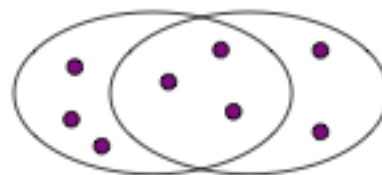
- Given a body of documents, e.g., the Web, find pairs of documents with a lot of text in common, e.g.:
  - Mirror sites, or approximate mirrors.
    - Application: Don't want to show both in a search.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    - Application: Cluster articles by “same story.”



# Similarity Measure

---

- **Goal: Find near-neighbors in high-dim. space**
  - We formally define “near neighbors” as points that are a “small distance” apart
- For each application, we first need to define what “**distance**” means
- **Today: Jaccard distance (/similarity)**
  - The *Jaccard Similarity/Distance* of two **sets** is the size of their intersection / the size of their union:
  - $sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
  - $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



3 in intersection

8 in union

Jaccard similarity =  $3/8$

Jaccard distance =  $5/8$

# Duplicate Document Detection

---

- **Goal:** Given a large number ( $N$  in the millions or billions) of text documents, find pairs that are “near duplicates”
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don’t want to show both in a search
  - Similar news articles at many news sites
    - Cluster articles by “same story”
- **Problems:**
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory

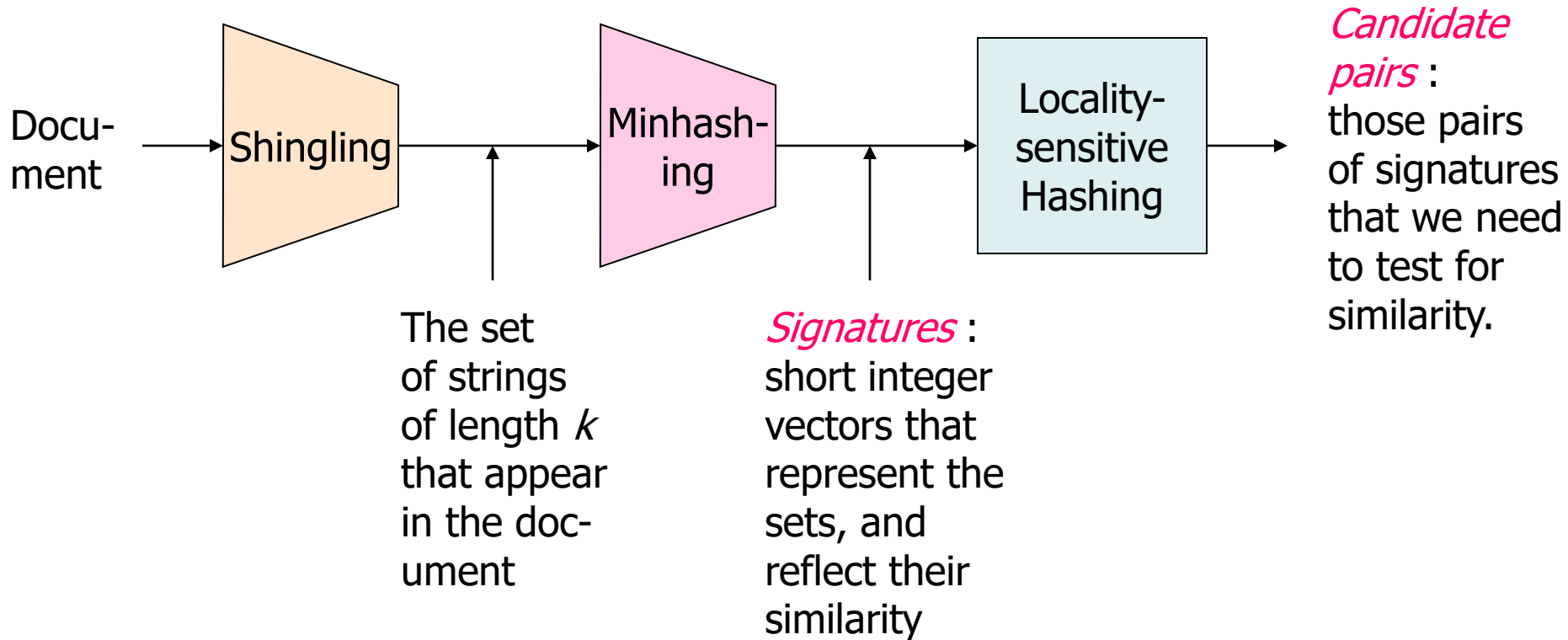
# Deduplication Process

---

1. **Shingling:** Convert documents to sets
2. **Minhashing:** Convert large sets to short signatures, while preserving similarity
3. **Locality-sensitive hashing:** Focus on pairs of signatures likely to be from similar documents
  - **Candidate pairs!**

# The Big Picture

---



# Shingling

---

- **Step 1: *Shingling*:** Convert documents to sets
- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of “important” words
  - Don’t work well for this application. *Why?*
- **Need to account for ordering of words!**
- A different way: **Shingles!**

# Shingling

---

- ❑ A  $k$ -shingle (or  $k$ -gram) for a document is a sequence of  $k$  characters that appears in the document.
- ❑ Example:  $k=2$ ; doc = abcab. Set of 2-shingles = {ab, bc, ca}.
  - ❑ Option: regard shingles as a bag, and count ab twice.
- ❑ Represent a doc by its set of  $k$ -shingles.

# Shingle Hash

---

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a doc by the set of hash values of its  $k$ -shingles**
  - **Idea:** Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared
- **Example:**  $k=2$ ; document  $D_1 = \text{ab cab}$   
Set of 2-shingles:  $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$   
Hash the singles:  $h(D_1) = \{1, 5, 7\}$

# Basic Data Model: Sets

---

- Many similarity problems can be couched as finding subsets of some universal set that have significant intersection.
- Examples include:
  - Documents represented by their sets of shingles (or hashes of those shingles).
  - Similar customers or products.



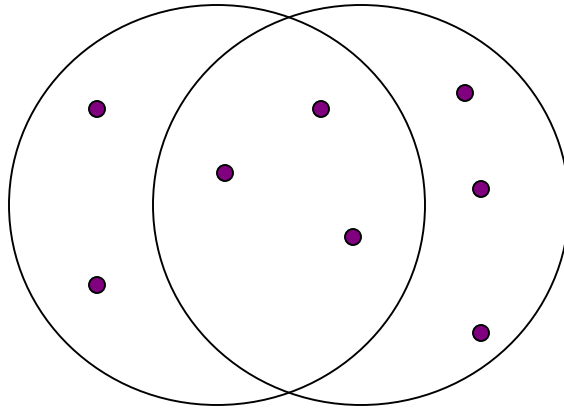
# Jaccard Similarity of Sets

---

- The Jaccard similarity of two sets is the size of their intersection divided by the size of their union.
  - $\text{Sim}(C1, C2) = |C1 \cap C2| / |C1 \cup C2|.$

# Jaccard Similarity

---



3 in intersection.

8 in union.

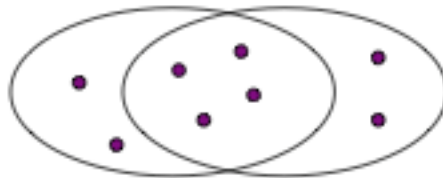
Jaccard similarity  
 $= 3/8$

# Similarity Metric for Shingles

---

- Document  $D_1$  = set of  $k$ -shingles  $C_1 = S(D_1)$
- Equivalently, each document is a 0/1 vector in the space of  $k$ -shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse
- A natural similarity measure is the **Jaccard similarity**:

$$Sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



# Shingle-Based Similarity

---

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.
- Careful: you must pick  $k$  large enough, or most documents will have most shingles.
  - $k = 5$  is OK for short documents;  $k = 10$  is better for long documents.

# MinHashing

---

- ❑ Data as Sparse Matrices
- ❑ Jaccard Similarity Measure
- ❑ Constructing Signatures

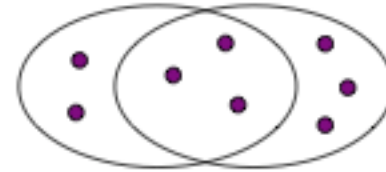
# MinHashing Motivation

---

- Suppose we need to find near-duplicate documents among  $N=1$  million documents
- Naïvely, we'd have to compute **pairwise Jaccard similarities** for every pair of docs
  - i.e,  $N(N-1)/2 \approx 5 \cdot 10^{11}$  comparisons
  - At  $10^5$  secs/day and  $10^6$  comparisons/sec, it would take 5 days
- For  $N = 10$  million, it takes more than a year...

# Sets as Bit Vectors

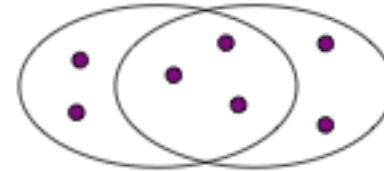
---



- Many similarity problems can be formalized as **finding subsets that have significant intersection**
- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret **set intersection as bitwise AND**, and **set union as bitwise OR**
- **Example:**  $C_1 = 10111$ ;  $C_2 = 10011$ 
  - Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) =  $3/4$
  - $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 1/4$

# From Sets to Boolean Matrices

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 1 in row  $e$  and column  $s$  if and only if  $e$  is a member of  $s$
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**
- **Each document is a column:**
  - **Example:**  $\text{sim}(C_1, C_2) = ?$ 
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) =  $3/6$
    - $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 3/6$



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0



## Example: Jaccard Similarity of Columns

---

$C_1$	$C_2$		
0	1		*
1	0		*
1	1	*	*
0	0		
1	1		
0	1	*	*
			*

$$\text{Sim}(C_1, C_2) = \frac{2}{5} = 0.4$$

## Aside

---

- ❑ We might not really represent the data by a boolean matrix.
- ❑ Sparse matrices are usually better represented by the list of places where there is a non-zero value.
- ❑ But the matrix picture is conceptually useful.

# Finding Similar Columns

---

- **So far:**
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- **Next Goal:** Find similar columns, Small signatures
- **Approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures & columns are related
  - **3) Optional:** Check that columns with similar signatures are really similar
- **Warnings:**
  - Comparing all pairs may take too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

# Hashing Columns

---

- **Key idea:** “hash” each column  $C$  to a small *signature*  $h(C)$ , such that:
  - (1)  $h(C)$  is small enough that the signature fits in RAM
  - (2)  $\text{sim}(C_1, C_2)$  is the same as the “similarity” of signatures  $h(C_1)$  and  $h(C_2)$
- **Goal: Find a hash function  $h(\cdot)$  such that:**
  - if  $\text{sim}(C_1, C_2)$  is high, then with high prob.  $h(C_1) = h(C_2)$
  - if  $\text{sim}(C_1, C_2)$  is low, then with high prob.  $h(C_1) \neq h(C_2)$
- **Hash documents into buckets, and expect that “most” pairs of near duplicate docs hash into the same bucket!**

# Min-Hashing

---

- **Goal: Find a hash function  $h(\cdot)$  such that:**
  - if  $\text{sim}(C_1, C_2)$  is high, then with high prob.  $h(C_1) = h(C_2)$
  - if  $\text{sim}(C_1, C_2)$  is low, then with high prob.  $h(C_1) \neq h(C_2)$
- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function
- **There is a suitable hash function for Jaccard similarity: Min-hashing**

# Min-Hashing

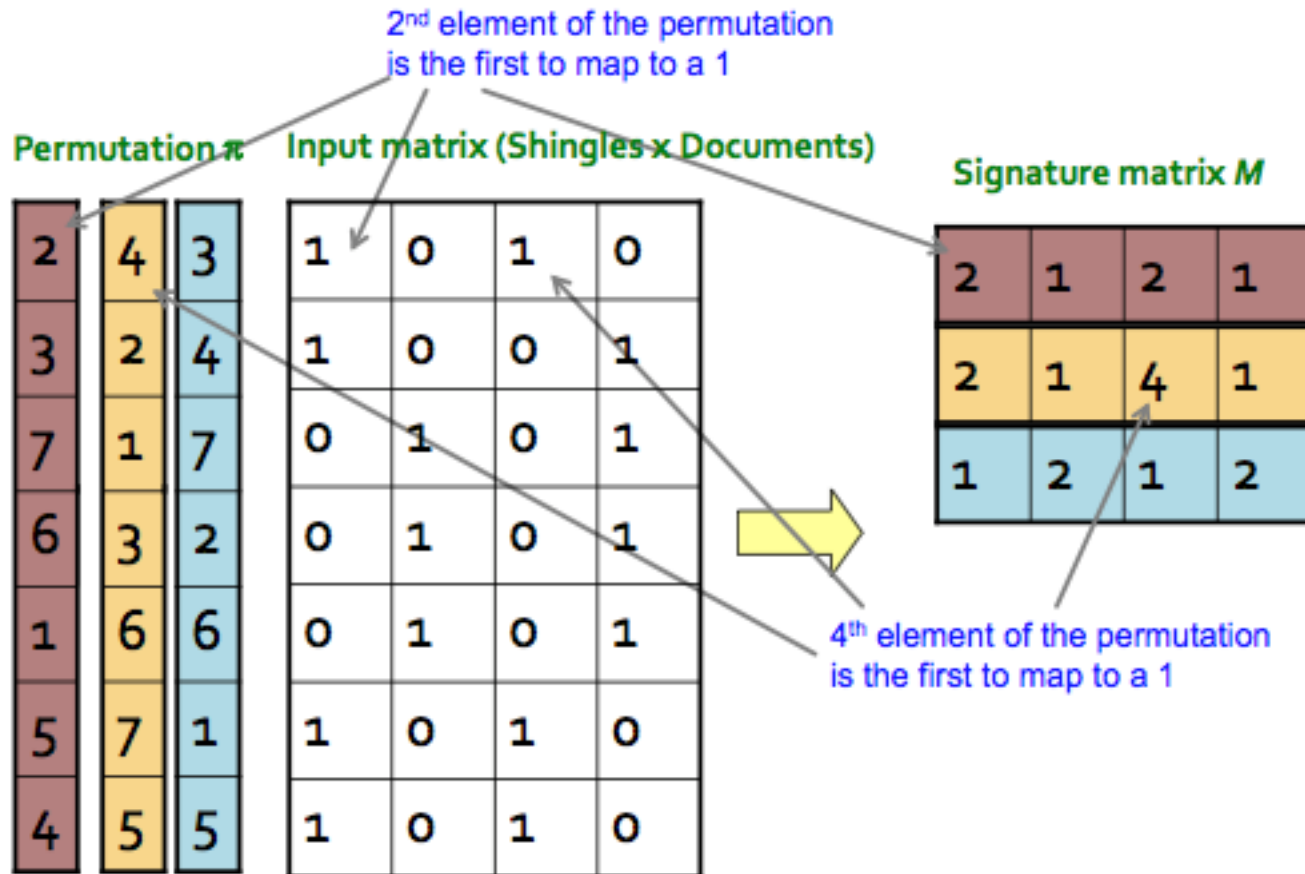
---

- Imagine the rows of the boolean matrix permuted under **random permutation**  $\pi$
- Define a **“hash” function**  $h_{\pi}(C)$  = the number of the **first** (in the permuted order  $\pi$ ) row in which column  $C$  has value 1:

$$h_{\pi}(C) = \min_{\pi} \pi(C)$$

- Use several (e.g., 100) independent hash functions to create a signature of a column

# Min-Hashing



## Surprising Property

- Choose a random permutation  $\pi$
- **Claim:**  $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- **Why?**
  - Let  $X$  be a document (set of shingles)
  - **Then:**  $\Pr[\pi(x) = \min(\pi(X))] = 1/|X|$ 
    - It is equally likely that any  $x \in X$  is mapped to the *min* element
  - Let  $x$  be s.t.  $\pi(x) = \min(\pi(C_1 \cup C_2))$
  - **Then either:**  $\pi(x) = \min(\pi(C_1))$  if  $x \in C_1$ , **or**  
 $\pi(x) = \min(\pi(C_2))$  if  $x \in C_2$
  - So the prob. that **both** are true is the prob.  $x \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1)) = \min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)$

0	0
0	0
1	1
0	0
0	1
1	0

One of the two  
cols had to have  
1 at position  $x$



# Min-Hashing

---

- Given cols  $C_1$  and  $C_2$ , rows may be classified as:

	<u><math>C_1</math></u>	<u><math>C_2</math></u>
A	1	1
B	1	0
C	0	1
D	0	0

- $a$  = # rows of type A, etc.
- **Note:**  $\text{sim}(C_1, C_2) = a/(a+b+c)$
- **Then:**  $\Pr[h(C_1) = h(C_2)] = \text{Sim}(C_1, C_2)$ 
  - Look down the cols  $C_1$  and  $C_2$  until we see a 1
  - If it's a type-A row, then  $h(C_1) = h(C_2)$   
If a type-B or type-C row, then not

# Similarity for Signatures

---

- We know:  $\Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = \text{sim}(C_1, C_2)$
- Now generalize to multiple hash functions
- The *similarity of two signatures* is the fraction of the hash functions in which they agree
- **Note:** Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

# Min-Hashing

Permutation  $\pi$

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix  $M$

2	1	2	1
2	1	4	1
1	2	1	2



Similarities:

Col/Col  
Sig/Sig

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

# MinHash Signatures

---

- **Pick  $K=100$  random permutations of the rows**
- Think of  $\text{sig}(\mathbf{C})$  as a column vector
- $\text{sig}(\mathbf{C})[i]$  = according to the  $i$ -th permutation, the index of the first row that has a 1 in column  $C$

$$\text{sig}(\mathbf{C})[i] = \min(\pi_i(\mathbf{C}))$$

- **Note:** The sketch (signature) of document  $C$  is small --  $\sim 100$  bytes!
- **We achieved our goal! We “compressed” long bit vectors into short signatures**

# Implementation Issues

---

- **Permuting rows even once is prohibitive**
- **Row hashing!**
  - Pick  $K = 100$  hash functions  $k_i$
  - Ordering under  $k_i$  gives a random row permutation!
- **One-pass implementation**
  - For each column  $C$  and hash-func.  $k_i$  keep a “slot” for the min-hash value
  - Initialize all  $\text{sig}(C)[i] = \infty$
  - **Scan rows looking for 1s**
    - Suppose row  $j$  has 1 in column  $C$
    - Then for each  $k_i$ :
      - If  $k_i(j) < \text{sig}(C)[i]$ , then  $\text{sig}(C)[i] \leftarrow k_i(j)$

How to pick a random hash function  $h(x)$ ?

**Universal hashing:**

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$

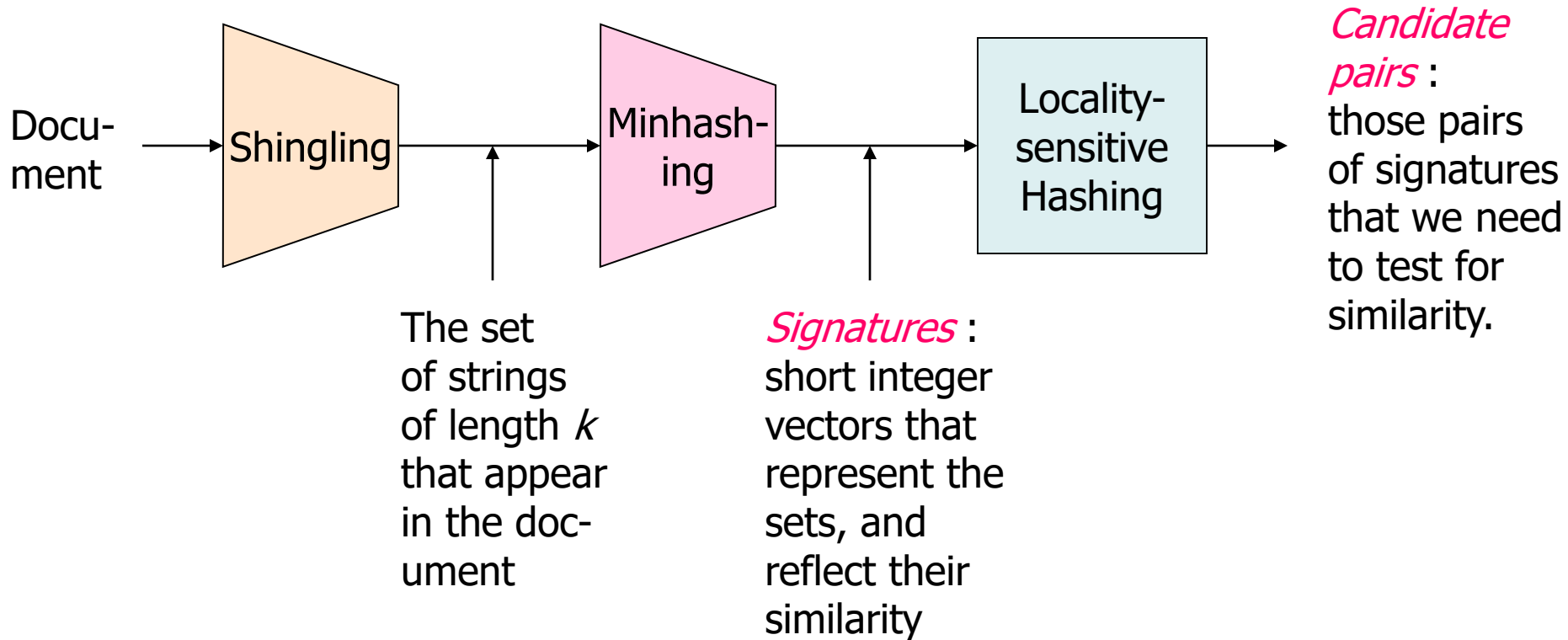
where:

$a, b \dots$  random integers

$p \dots$  prime number ( $p > N$ )

# The Big Picture

---



# Locality Sensitive Hashing

---

- **Goal:** Find documents with Jaccard similarity at least  $s$  (for some similarity threshold, e.g.,  $s=0.8$ )
- **LSH – General idea:** Use a function  $f(x,y)$  that tells whether  $x$  and  $y$  is a **candidate pair**: a pair of elements whose similarity must be evaluated
- **For minhash matrices:**
  - Hash columns of **signature matrix  $M$**  to many buckets
  - Each pair of documents that hashes into the same bucket is a **candidate pair**

# Locality Sensitive Hashing

---

- Pick a similarity threshold  $s$  ( $0 < s < 1$ )
- Columns  $\mathbf{x}$  and  $\mathbf{y}$  of  $\mathbf{M}$  are a **candidate pair** if their signatures agree on at least fraction  $s$  of their rows:  
 $M(i, \mathbf{x}) = M(i, \mathbf{y})$  for at least frac.  $s$  values of  $i$ 
  - We expect documents  $\mathbf{x}$  and  $\mathbf{y}$  to have the same (Jaccard) similarity as is the similarity of their signatures



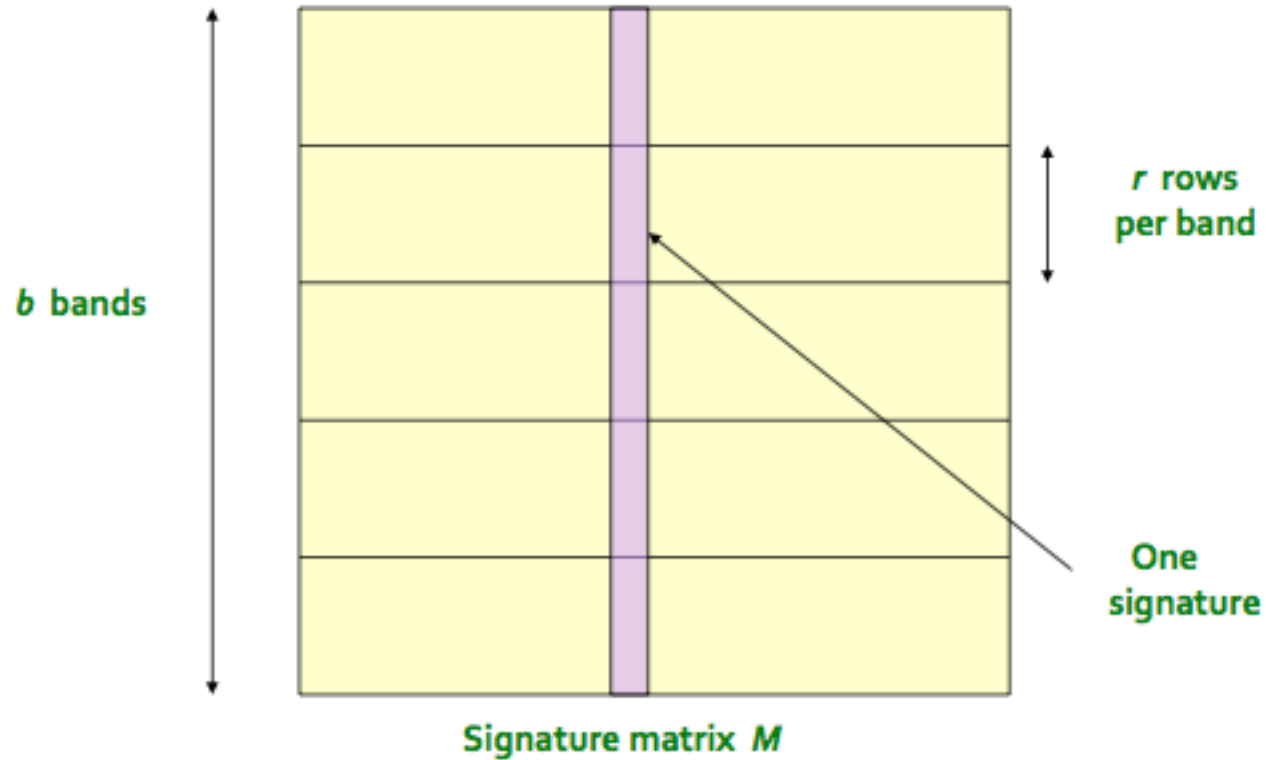
# Locality Sensitive Hashing

---

- **Big idea:** Hash columns of signature matrix  $M$  several times
- Arrange that (only) **similar columns** are likely to **hash to the same bucket**, with high probability
- **Candidate pairs are those that hash to the same bucket**

# Partition M into Bands

---

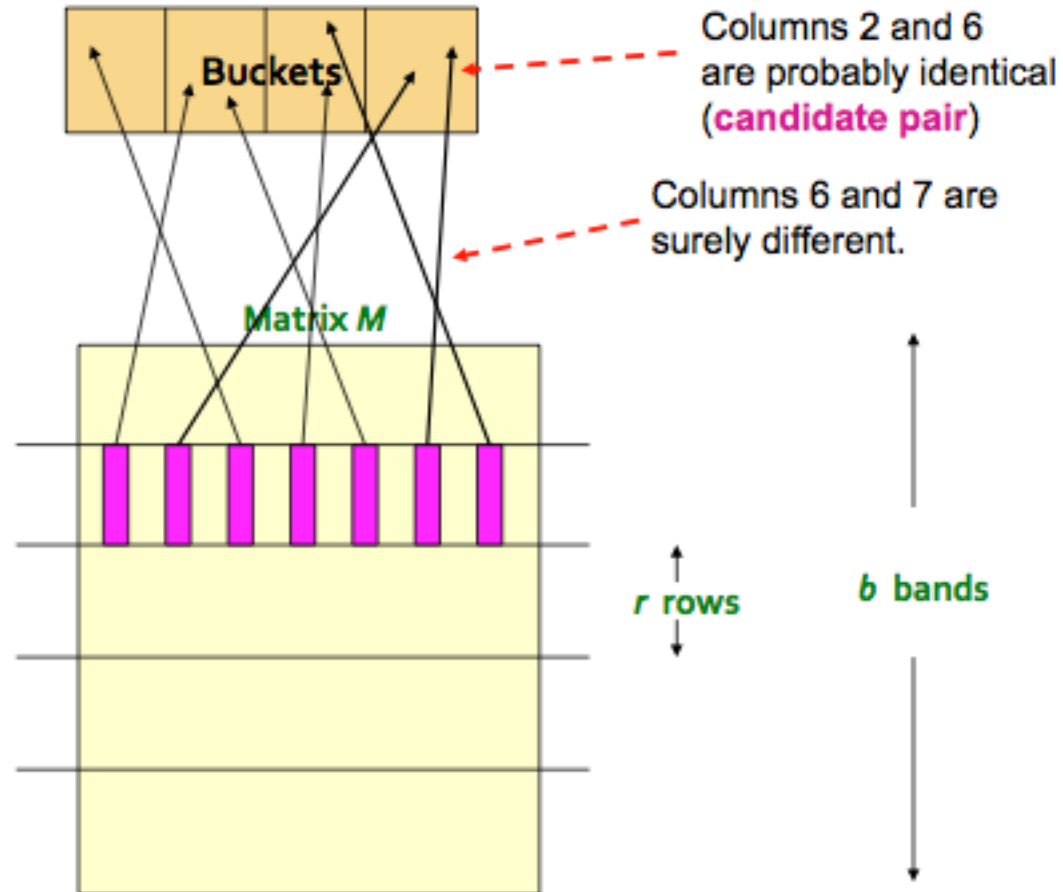


# Partition $M$ into Bands

---

- Divide matrix  $M$  into  $b$  bands of  $r$  rows
- For each band, hash its portion of each column to a hash table with  $k$  buckets
  - Make  $k$  as large as possible
- **Candidate** column pairs are those that hash to the same bucket for  $\geq 1$  band
- Tune  $b$  and  $r$  to catch most similar pairs, but few non-similar pairs

# Hashing Bands



# Assumption

---

- There are **enough buckets** that columns are unlikely to hash to the same bucket unless they are **identical** in a particular band
- Hereafter, we assume that “**same bucket**” means “**identical in that band**”
- Assumption needed only to simplify analysis, not for correctness of algorithm

# Example of Bands

---

## Assume the following case:

- Suppose 100,000 columns of  $M$  (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take 40Mb
- Choose  $b = 20$  bands of  $r = 5$  integers/band
- **Goal:** Find pairs of documents that are at least  $s = 0.8$  similar

# C1, C2 are Similar

---

- Find pairs of  $\geq s=0.8$  similarity, set  $b=20$ ,  $r=5$
- Assume:  $\text{sim}(C_1, C_2) = 0.8$ 
  - Since  $\text{sim}(C_1, C_2) \geq s$ , we want  $C_1, C_2$  to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- Probability  $C_1, C_2$  identical in one particular band:  
 $(0.8)^5 = 0.328$
- Probability  $C_1, C_2$  are **not** similar in all of the 20 bands:  $(1-0.328)^{20} = 0.00035$ 
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

# C1, C2 are Similar

---

- Find pairs of  $\geq s=0.8$  similarity, set  $b=20$ ,  $r=5$
- Assume:  $\text{sim}(C_1, C_2) = 0.3$ 
  - Since  $\text{sim}(C_1, C_2) < s$  we want  $C_1, C_2$  to hash to **NO common buckets** (all bands should be different)
- Probability  $C_1, C_2$  identical in one particular band:  
 $(0.3)^5 = 0.00243$
- Probability  $C_1, C_2$  identical in at least 1 of 20 bands:  
 $1 - (1 - 0.00243)^{20} = 0.0474$ 
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming **candidate pairs**
    - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold  $s$



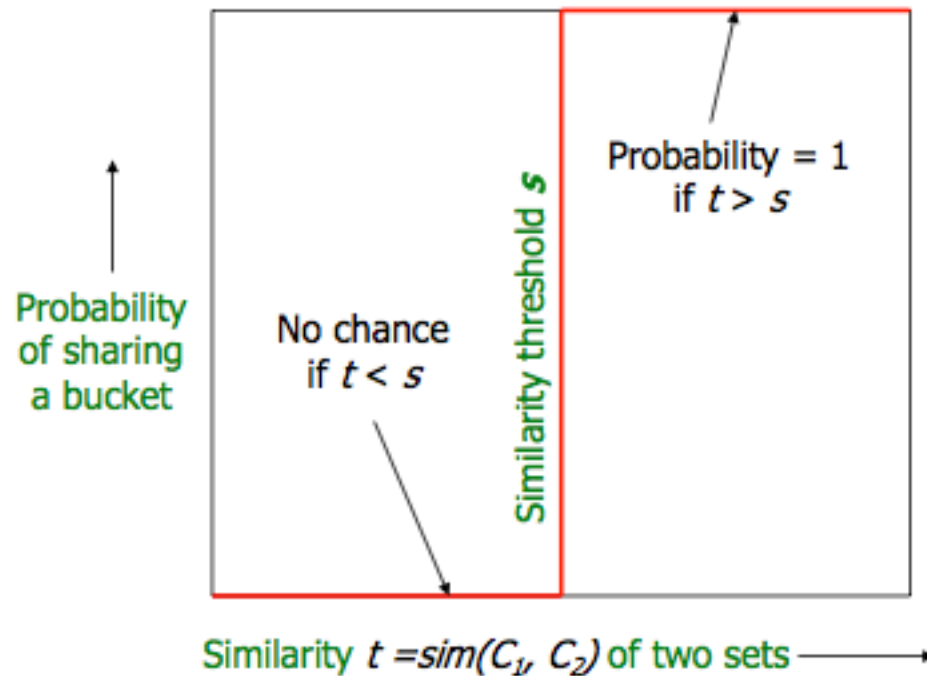
# LHS Trade-Off

---

- **Pick:**
  - the number of minhashes (rows of  $M$ )
  - the number of bands  $b$ , and
  - the number of rows  $r$  per bandto balance false positives/negatives
- **Example:** if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

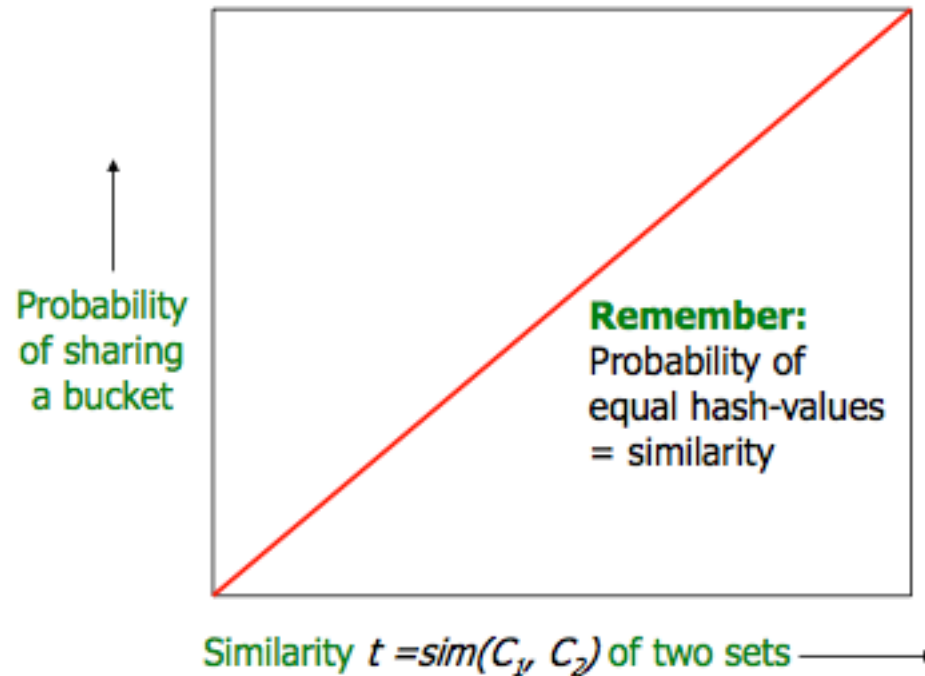
# Analysis of LSH: Desired Result

---



# Analysis of LSH: 1 Band

---



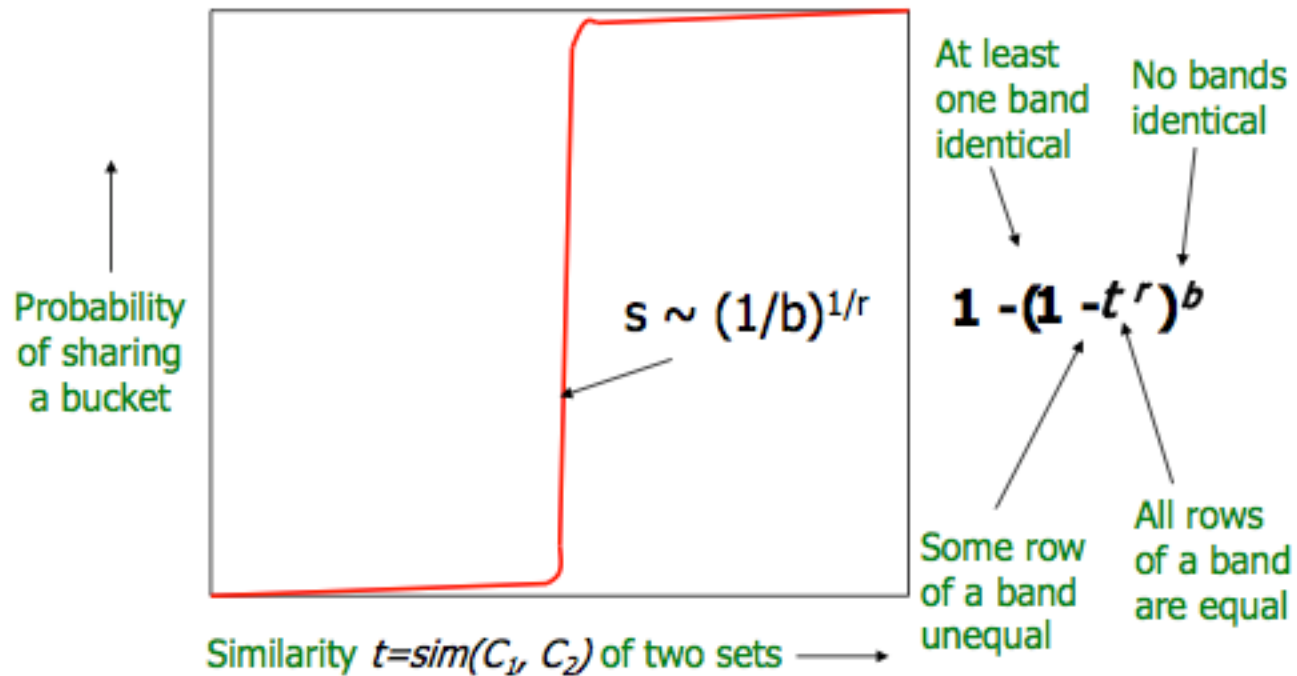
## B Bands, R Rows

---

- Columns  $C_1$  and  $C_2$  have similarity  $t$
- Pick any band ( $r$  rows)
  - Prob. that all rows in band equal =  $t^r$
  - Prob. that some row in band unequal =  $1 - t^r$
- Prob. that no band identical =  $(1 - t^r)^b$
- Prob. that at least 1 band identical =  
 $1 - (1 - t^r)^b$

# B Bands, R Rows

---



$$B = 20, r = 5$$

---

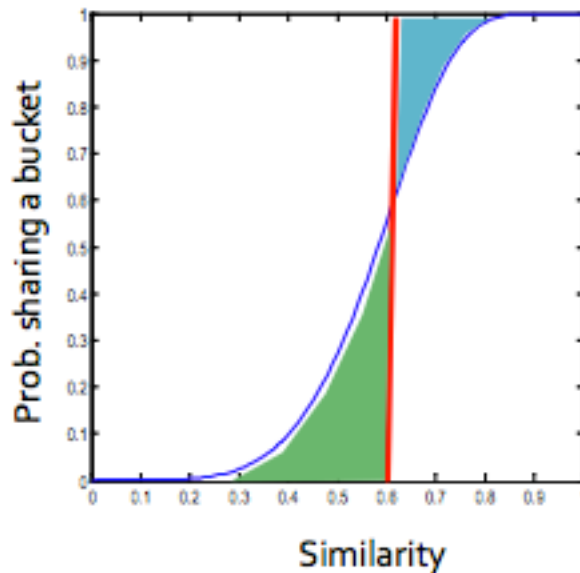
- **Similarity threshold  $s$**
- **Prob. that at least 1 band is identical:**

<b><math>s</math></b>	<b><math>1-(1-s^r)^b</math></b>
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

$$B = 20, r = 5$$

---

- Picking  $r$  and  $b$  to get the best S-curve
  - 50 hash-functions ( $r=5, b=10$ )



Blue area: False Negative rate  
Green area: False Positive rate

# LSH Summary

---

- Tune  $M, b, r$  to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
- Check in main memory that **candidate pairs** really do have **similar signatures**
- **Optional:** In another pass through data, check that the remaining candidate pairs really represent similar documents



# LSH Summary

---

- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property  $\Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = \text{sim}(C_1, C_2)$
  - We used hashing to get around generating random permutations
- **Locality-sensitive hashing:** Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find **candidate pairs** of similarity  $\geq s$

# The Big Picture

---

