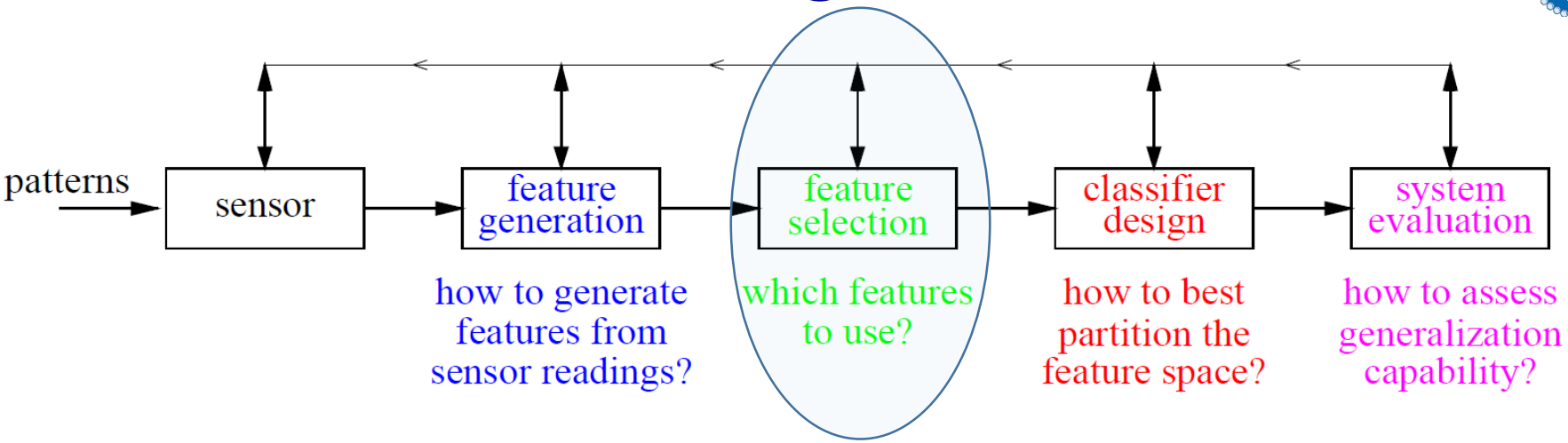
Machine learning

Dimensionality

Mohammad-Reza A. Dehaqani

dehaqani@ut.ac.ir

Feature conditioning



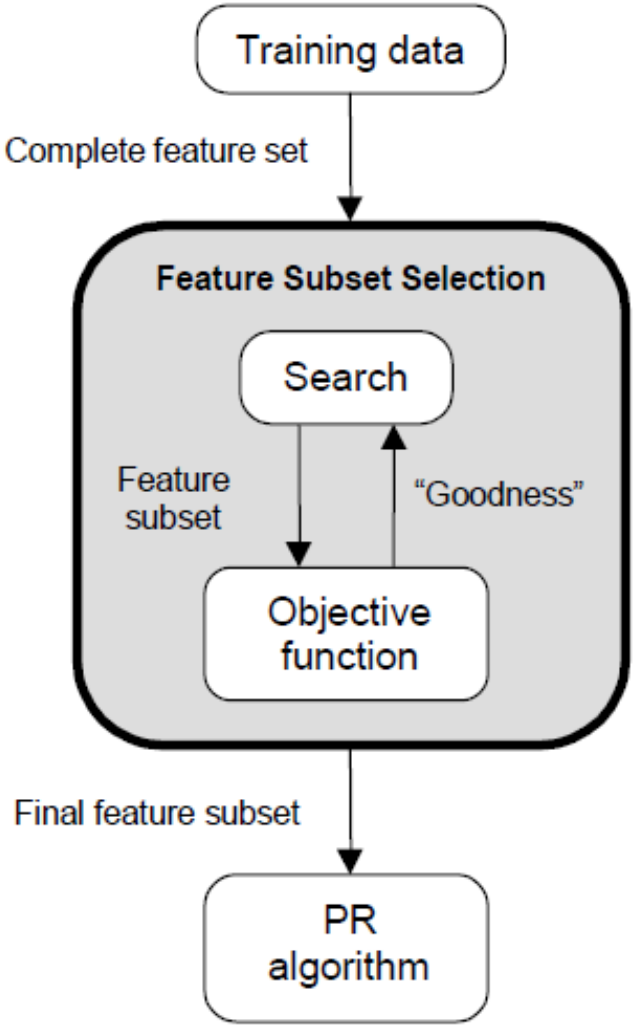
- In practical multicategory applications, it **is not unusual** to encounter problems involving **hundreds of features**.
- Feature selection:
  - Using a **criterion function** that is often a function of the classification error for feature selection (to select **discriminative** and **invariant** features)
- Feature reduction:
  - Using linear or non-linear **combinations of features** is feature selection that reduces dimensionality by selecting subsets of existing features.

Exhaustive search

- Examining all $\binom{d}{m}$ possible **subsets** of size m, and selecting the subset that **performs the best** according to the criterion function.
- The number of subsets grows **combinatorially**, making the exhaustive search **impractical**.
- Iterative procedures are **often used** but they **cannot guarantee** the selection of the **optimal** subset.

Feature Selection Steps

- Feature selection is an **optimization** problem.
  - **Step 1:** Search the **space of possible feature** subsets.
  - **Step 2:** **Pick** the subset that is optimal or near-optimal with respect to some **objective** function.



- **Search strategies**
  - Exhaustive
  - Heuristic
  - Randomized
- **Evaluation** strategies
  - Filter methods
  - Wrapper methods

Evaluation Strategies

- **Filter Methods**
  - Evaluation is **independent of the classification algorithm**.
  - The **objective function** evaluates feature subsets by their **information content**, typically **interclass distance**, statistical **dependence** or **information-theoretic** measures (e.g., **mutual information**).
  - Fast, general, and tendency to select big subsets
- **Wrapper Methods**
  - Evaluation **uses criteria related to the classification** algorithm.
  - The objective function is a pattern classifier, which **evaluates feature subsets by their predictive accuracy** (recognition rate on test data) by statistical **resampling or cross-validation**.
  - Slow, accurate, and lack of generality (over fit problem)

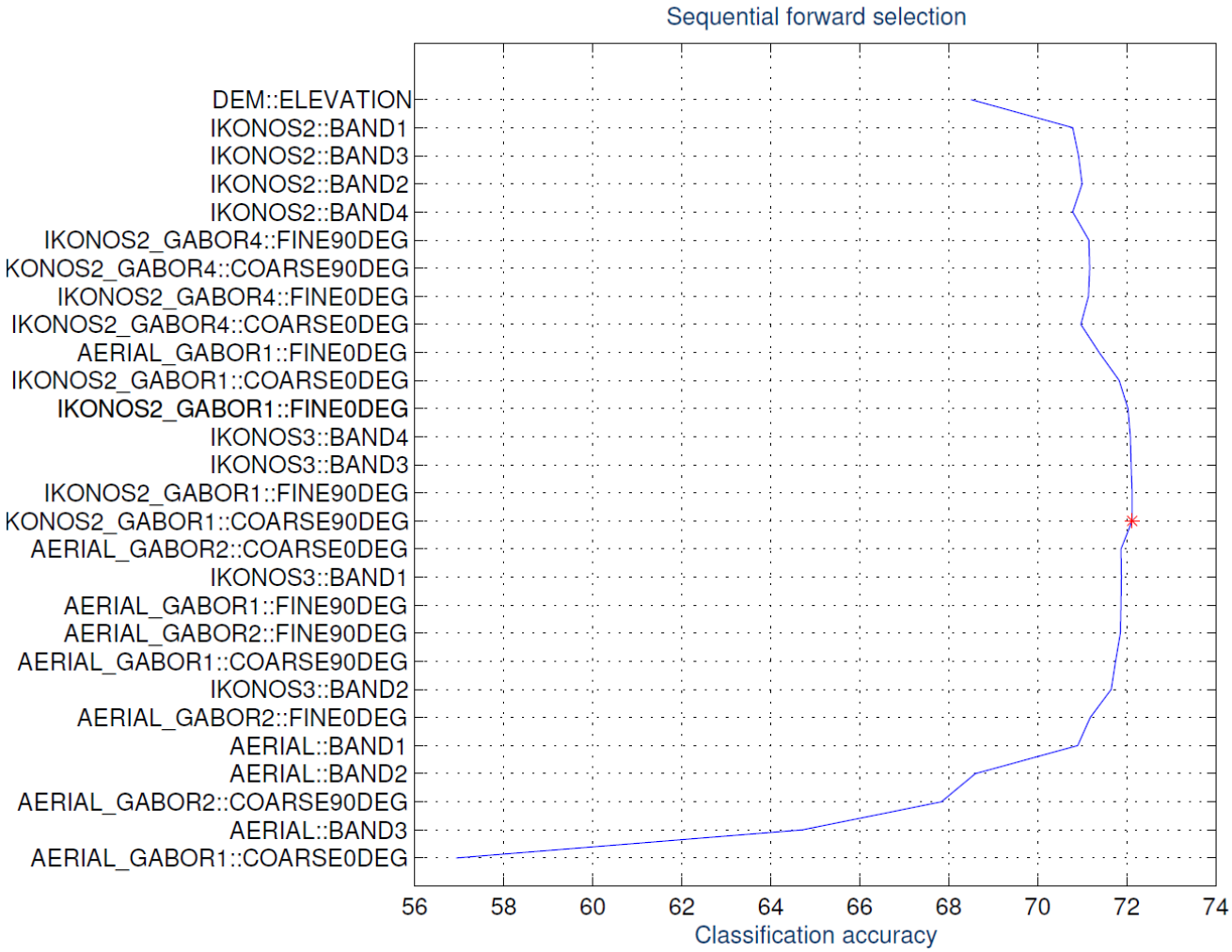
Naïve Search; (heuristic search)

- Given n features, **sort** them in order of their "goodness" based on some objective function.
- Select the **top d features** from this sorted list.
- Disadvantage
  - **Correlation** among features is not considered.
  - The best subset of features may not even contain **the best individual feature.**

Sequential forward selection: (heuristic search)

- First, the **best single feature** is selected.
- Then, **pairs** of features are formed using one of the remaining features and this best feature, and the **best pair is selected**.
- Next, **triplets** of features are formed using one of the **remaining** features and these **two best features**, and the **best triplet** is selected.
- This procedure continues until **all or a predefined** number of features are selected.

Sequential forward selection:

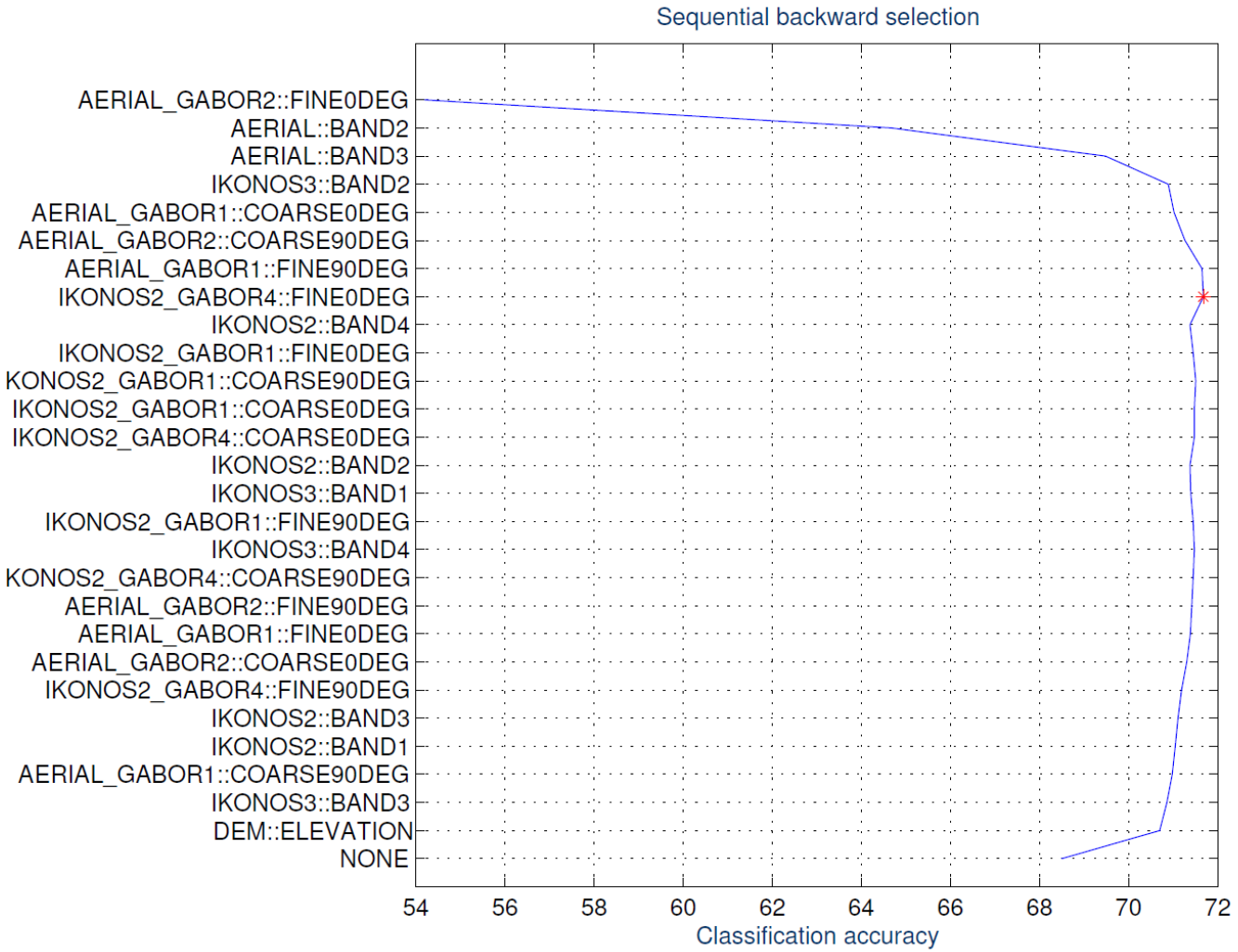


- classification of a satellite image using **28 features**.
- x-axis shows the classification accuracy (%) and y-axis shows the features added at **each iteration**
- (the first iteration is at the **bottom**).
- The **highest** accuracy value is shown with a **star**.

Sequential backward selection: (heuristic search)

- First, the criterion function is computed for **all d features**.
- Then, each feature is deleted one at a time, the criterion function is computed for **all subsets with d − 1** features, and the **worst feature is discarded**.
- Next, each feature among the **remaining d − 1** is deleted one at a time, and the **worst feature** is discarded to form a subset with d − 2 features.
- This procedure continues until **one feature or a predefined** number of features are left.

Sequential backward selection



- y-axis shows the features **removed** at each iteration (the first iteration is at the bottom). The highest accuracy value is shown with a **star**.

Bidirectional Search (BDS)

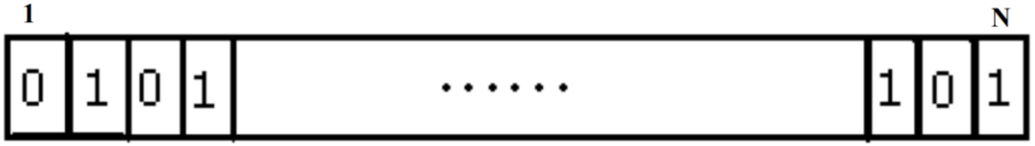
- BDS applies SFS and SBS **simultaneously**:
  - SFS is performed from the empty set.
  - SBS is performed from the full set.
- To guarantee that SFS and SBS converge to the same solution:
  - Features already selected by SFS are **not removed** by SBS.
  - Features already removed by SBS are **not added** by SFS.

Floating techniques

- The main limitation of:
  - **SFS** is that it is unable to remove features that become non useful **after the addition** of other features.
  - **SBS** is its inability to reevaluate the usefulness of a feature after it has been **discarded**.
- **Sequential floating forward selection (SFFS):**
  - Sequential floating forward selection (SFFS) starts from the **empty** set.
  - After each forward step, SFFS performs backward steps as long as the objective function increases.
- **Sequential floating backward selection (SFBS)**
  - Sequential floating backward selection (SFBS) starts from **the full set**.
  - After each backward step, SFBS performs forward steps as long as the objective function increases

Randomized search :Genetic Algorithms (GAs)

- Search probabilistically using a **population** of possible solutions.
- Each solution is **encoded** as a string of symbols.
- Use an objective (or **fitness**) function to evaluate the "goodness" of each solution.
- Do not require **derivatives**. More effective to **escape local minima**
- **Binary encoding**: 1 means "choose feature" and 0 means "do not choose" feature

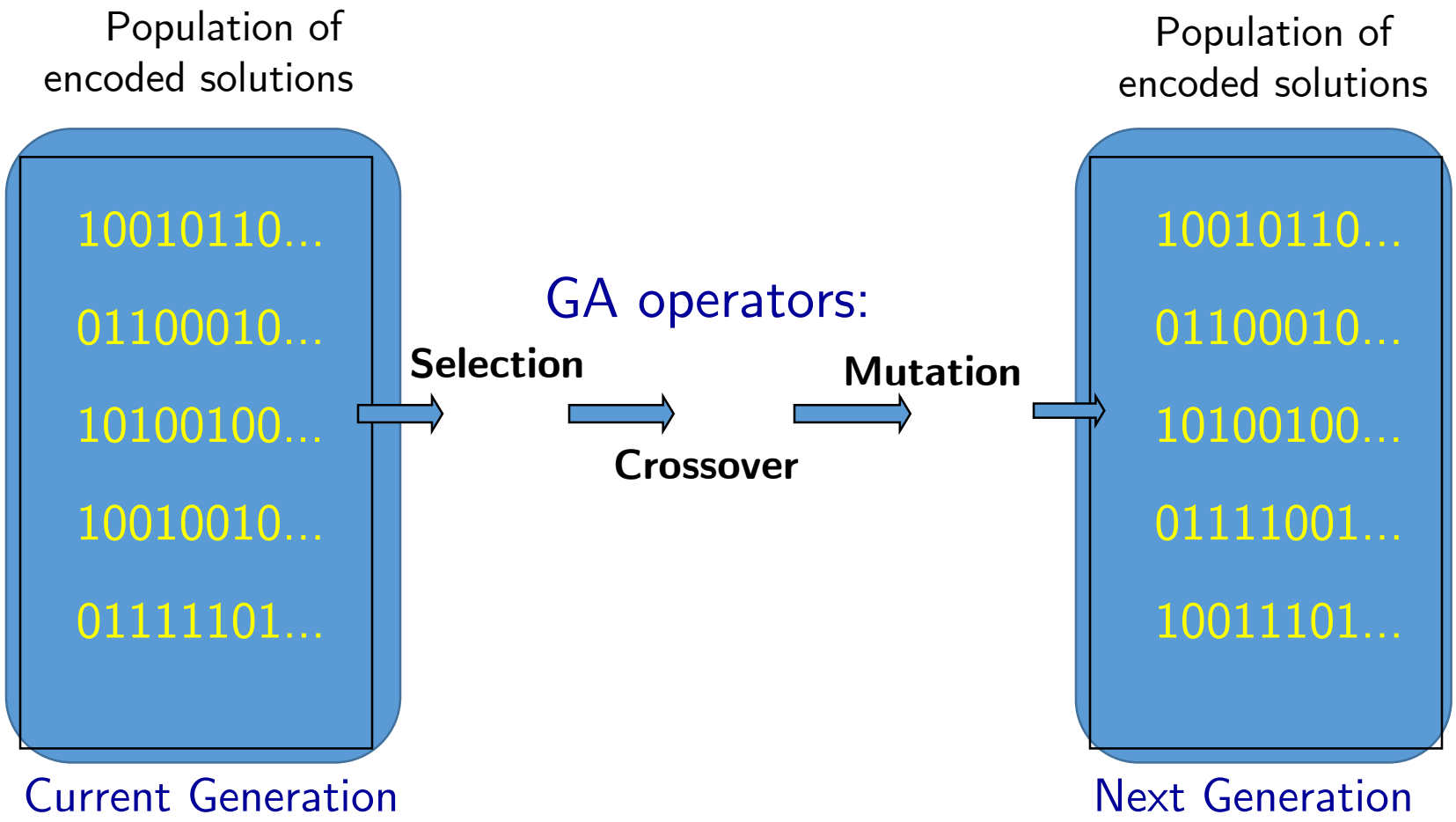


- Sample of fitness evaluation (to be maximized)

$$\text{Fitness} = w_1 \times \text{accuracy} + w_2 \times \#\text{zero}$$

Classification accuracy using a **validation** set and $w_1 > w_2$

Randomized search :Genetic Algorithms (GAs)

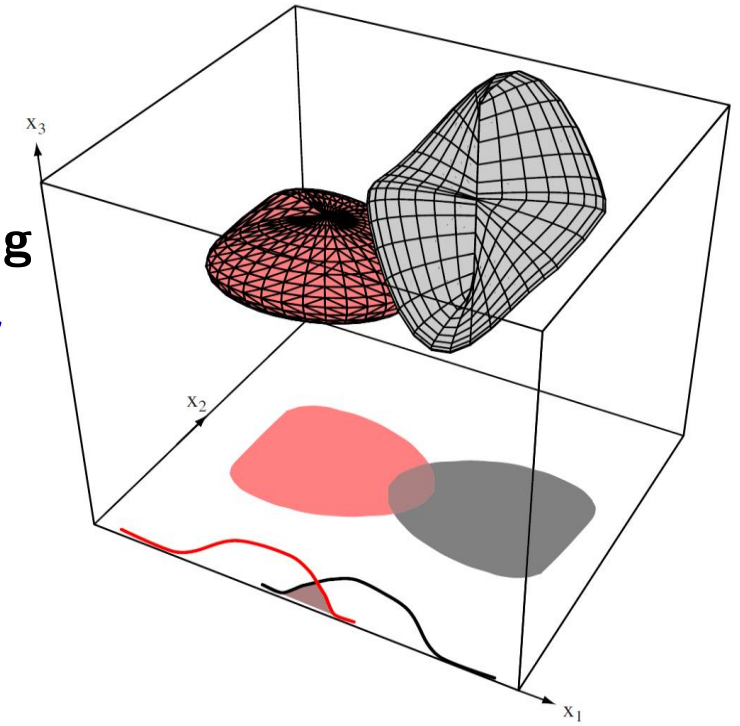


Adding feature and dimension reduction

- Intuitively, it may seem that each feature is useful for at least **some of the discriminations**.
- In general, if the performance obtained with a given set of features is **inadequate**, it is **natural** to consider adding new features. Even though **increasing the number of features** increases the complexity of the classifier.
- Unfortunately, it has frequently been observed in practice that, beyond a certain point, adding new features leads to **worse** rather than **better** performance (i.e., curse of dimensionality).

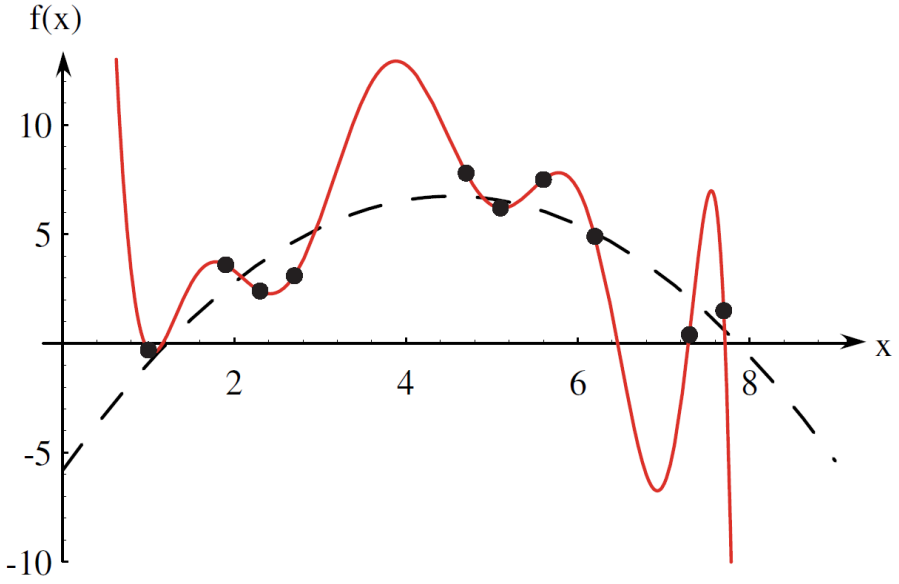
From a **theoretical point of view**, increasing the number of features can lead to better performance.

- In three dimensions the Bayes error **vanishes**; in 2 and 1 D there are **greater overlap** of the projected distributions, and hence greater **Bayes errors**.
- However in practice, the **number of training examples** required increases **exponentially** with dimensionality.
- There are two issues that we must be **careful** about:
  - How is the **classification accuracy** affected by the dimensionality (relative to the amount of training data)?
  - How is the **complexity of the classifier** affected by the dimensionality?



Two source of errors

- Potential **reasons** for increase in error include
  - Wrong assumptions in **model selection**,
  - Estimation errors due to the **finite number** of training samples for high-dimensional observations (overfitting).
- **Potential solutions** include
  - **Reducing** the dimensionality,
  - **Simplifying** the estimation n.



Problem of **insufficient** data is analogous to problems in **curve fitting**. The training data (black dots) are **selected from a quadratic function** plus Gaussian noise. A **tenth-degree** polynomial fits the data perfectly but we prefer a second-order polynomial for **better generalization**

Number of samples

- All of the commonly used classifiers can **suffer** from the **curse of dimensionality**.
- While an **exact** relationship between the probability of error, the number of training samples, the number of features, and the number of parameters is very **difficult** to establish, some **guidelines** have been suggested.
  - It is generally accepted that using at least **ten times** as many training samples **per class** as the number of features (n/d>10) is a good practice.
  - The **more complex** the classifier, the **larger ratio of sample size**.

Feature reduction

- One approach for coping with the problem **of high dimensionality** is to **reduce** the dimensionality by **combining features**.
- **Issues** in feature reduction:
  - **Linear** vs. **non-linear** transformations (combination of features).
  - Use of class **labels** or not (depends on the availability of training data).
  - Training objective:
    - minimizing **classification error** (discriminative training),
    - minimizing **reconstruction error** (PCA),
    - maximizing class **separability** (LDA),
    - making features **as independent as possible** (ICA).

Why do dimensionality reduction?

- Computational:
  - **compress** data ⇨ time/space efficiency
- Statistical:
  - fewer dimensions ⇨ better **generalization**
- Visualization:
  - understand **structure** of data
- Anomaly detection:
  - describe normal data, detect **outliers**

Dimensionality reduction setup
Using a new **basis** for the data

- Given n data points in d dimensions: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots\cdots & \mathbf{x}_n \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times n}$$

- Want to **reduce** dimensionality from d to k

Choose k **directions** $\mathbf{u}_1, \ldots, \mathbf{u}_k$

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_k \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times k}$$

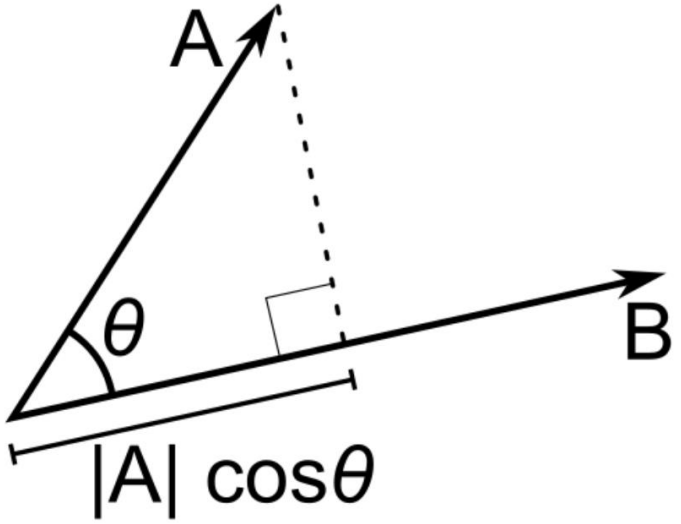
For each $\mathbf{u}_j$, compute "**similarity**" $z_j = \mathbf{u}_j^\top \mathbf{x}$

Project **x** down to $\mathbf{z} = (z_1, \ldots, z_k)^\top = \mathbf{U}^\top \mathbf{x}$

How to **choose** **U**?

Reminder: Vector Projections

- Basic definitions:
  - A.B = |A||B|cosθ
  - cosθ=|adj|/|hyp|



- Assume |B|=1 (unit vector)
  - A.B=|A|cosθ
  - So, dot product is **length** of projection
  - Projection of x on u_j

$$z_j = \mathbf{u}_j^\top \mathbf{x}$$

PCA objective 1: reconstruction error

- **U** serves two **functions**:

  Encode: $\mathbf{z} = \mathbf{U}^\top \mathbf{x}, \quad z_j = \mathbf{u}_j^\top \mathbf{x}$

  Decode: $\tilde{\mathbf{x}} = \mathbf{U}\mathbf{z} = \sum_{j=1}^{k} z_j \mathbf{u}_j$

- Want **reconstruction error** $\|\mathbf{x} - \tilde{\mathbf{x}}\|$ to be small

- **Objective**:

  minimize total squared reconstruction error (in the l**east-squares** sense)

$$\min_{\mathbf{U} \in \mathbb{R}^{d \times k}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{U}\mathbf{U}^\top \mathbf{x}_i\|^2$$

PCA objective 2: projected variance

- Empirical distribution: **uniform** over $\mathbf{x}_1, \ldots, \mathbf{x}_n$
- **Expectation** (think sum over data points):

$$\hat{\mathbb{E}}[f(\mathbf{x})] = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i)$$

- **Variance** (think sum of squares if centered):

$$\widehat{\text{var}}[f(\mathbf{x})] + (\hat{\mathbb{E}}[f(\mathbf{x})])^2 = \hat{\mathbb{E}}[f(\mathbf{x})^2] = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i)^2$$

- Assume data is **centered**: $\hat{\mathbb{E}}[\mathbf{x}] = 0$ (what is $\hat{\mathbb{E}}[\mathbf{U}^\top \mathbf{x}]$?)
- **Objective**: maximize **variance of projected** data

$$\max_{\mathbf{U}\in\mathbb{R}^{d\times k},\, \mathbf{U}^\top\mathbf{U}=I} \hat{\mathbb{E}}[\|\mathbf{U}^\top \mathbf{x}\|^2]$$

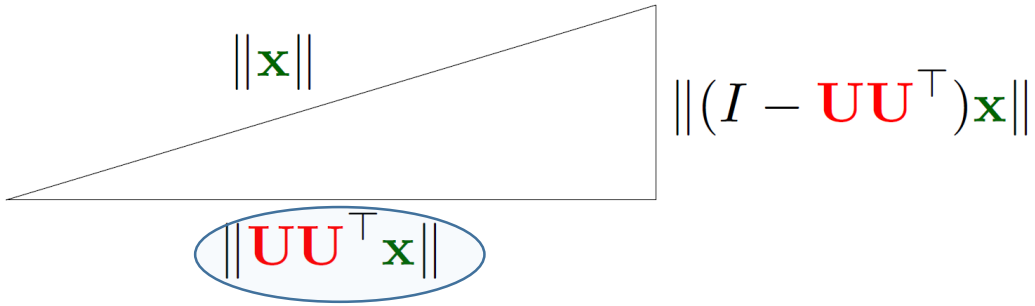
orthogonal and has unit norm

Equivalence in two objectives

- Key intuition:

$$\underbrace{\text{variance of data}}_{\text{fixed}} = \underbrace{\text{captured variance}}_{\text{want large}} + \underbrace{\text{reconstruction error}}_{\text{want small}}$$

- Pythagorean decomposition: $\mathbf{x} = \mathbf{U}\mathbf{U}^\top\mathbf{x} + (I - \mathbf{U}\mathbf{U}^\top)\mathbf{x}$

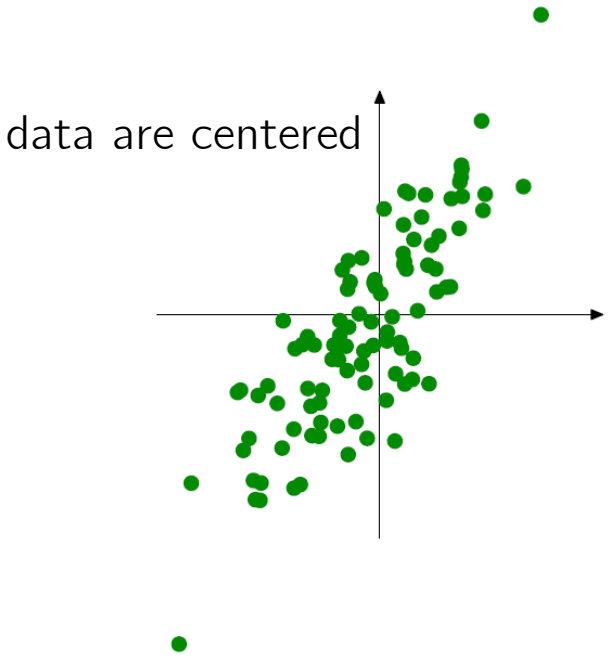


- Take expectations; note rotation U doesn't affect length:

$$\hat{\mathbb{E}}[\|\mathbf{x}\|^2] = \hat{\mathbb{E}}[\|\mathbf{U}^\top\mathbf{x}\|^2] + \hat{\mathbb{E}}[\|\mathbf{x} - \mathbf{U}\mathbf{U}^\top\mathbf{x}\|^2]$$

- **Minimize** reconstruction error ↔ **Maximize** captured variance

Finding one principal component



Objective: maximize variance of projected data

$$= \max_{\|\mathbf{u}\|=1} \hat{\mathbb{E}}[(\mathbf{u}^\top \mathbf{x})^2]$$

$$= \max_{\|\mathbf{u}\|=1} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{u}^\top \mathbf{x}_i)^2$$

$$= \max_{\|\mathbf{u}\|=1} \frac{1}{n} \|\mathbf{u}^\top \mathbf{X}\|^2$$

$$= \max_{\|\mathbf{u}\|=1} \mathbf{u}^\top \left(\frac{1}{n} \mathbf{X}\mathbf{X}^\top\right) \mathbf{u}$$

Input data:

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \dots & \mathbf{x}_n \\ | & & | \end{pmatrix}$$

Finding one principal component

$$\max_{\|\mathbf{u}\|=1} \mathbf{u}^\top \left(\frac{1}{n}\mathbf{X}\mathbf{X}^\top\right)\mathbf{u}$$

$$\max_{u} u^T X X^T u \text{ s.t. } u^t u = 1$$

Lagrangian (wrap constrain into the objective function)

$$\max_{u} u^T X X^T u - \lambda u^t u = 1$$

$$\frac{\partial}{\partial u} = 0 \qquad (XX^T - \lambda I)u = 0$$

$$(XX^T)u = \lambda u$$

$= \text{largest eigenvalue of } \Sigma \overset{\text{def}}{=} \frac{1}{n}\mathbf{X}\mathbf{X}^\top$

(Σ is covariance matrix of data)

Other components

$$(XX^T)u = \lambda u$$

- Therefore, u is the eigenvector of **sample correlation**/ **covariance** matrix XX^T
- Sample **variance** of **projection**:

$$u^T XX^T u = \lambda u^T u = \lambda$$

- Thus, the eigenvalue λ denotes the **amount of variability** captured along that dimension (aka **amount of energy** along that dimension).
- Eigenvalues λ1> λ2> λ3> . . .

Principal Component Analysis (PCA)

- The 1st principal component u_1 is the **eigenvector** of the **sample covariance matrix** XX^T associated with the largest eigenvalue λ_1
- The 2nd principal component u_2 is the **eigenvector** of the sample covariance matrix XX^T associated with the second largest eigenvalue λ_2
- And so on ...

Computing the PCs

- Eigenvectors are **solutions** of the following equation:

$$(XX^T)u = \lambda u \qquad (XX^T - \lambda I)u = 0$$

- **Non-zero solution** $u \neq 0$ possible only if

$$\det(XX^T - \lambda I) = 0 \qquad \textbf{Characteristic equation}$$

- This is a d^{th} **order equation in $\boldsymbol{\lambda}$**, can have at most d **distinct solutions** (roots of the characteristic equation)
- Once eigenvaluesare computed, solve for eigenvectors (Principal Components) using

$$(XX^T - \lambda I)u = 0$$

- For **symmetric** matrices, eigenvectors for **distinct** eigenvalues are **orthogonal (exercise: proof it!)**

Properties of the Covariance Matrix

- The covariance matrix of a random vector $\mathbf{x} \in R^n$ with mean vector μ is:

$$\Sigma = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T]$$

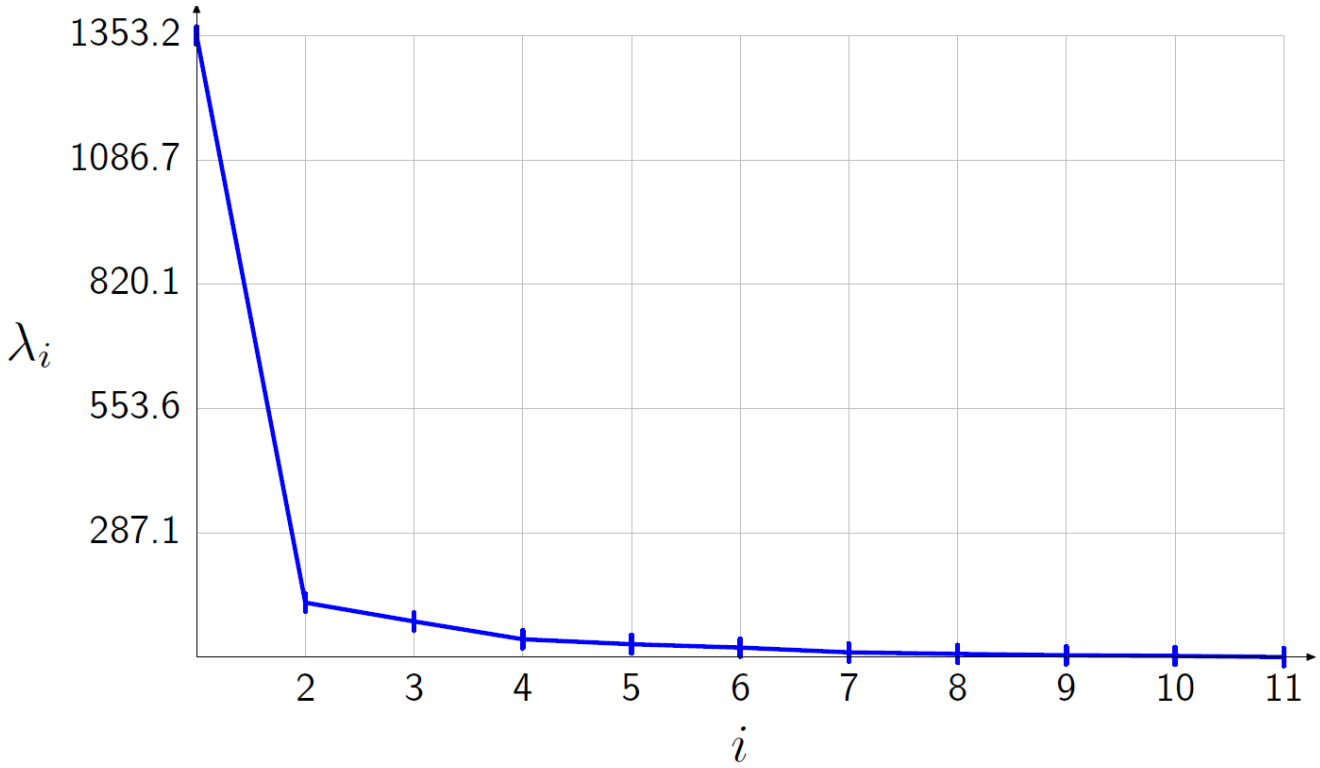
- **Sample covariance** matrix elements is:

$$\sigma_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (xi_j - \mu_j)(x_{ik} - \mu_k)$$

- It is a **square matrix** giving the **covariance** between each pair of elements of a given random vector
- It is **symmetric** so its **eigenvalues** are **all real** and **positive** and the **eigenvectors that belong to distinct eigenvalues are orthogonal**. As a consequence, the **determinant** of the covariance matrix is **positive**
- **Positive semidefinite**, i.e., for $x \in R^n$; $x^T \Sigma x \geq 0$

How many principal components?

- Only keep data projections onto principal components with **large** eigenvalues
- Magnitude of eigenvalues indicate fraction of variance captured.
- Features that cover 90% of variances.



Computing PCA;
Method 1: **eigendecomposition**

- Because covariance matrices are **symmetric** and **positive semidefinite**

$$\Sigma = U \Lambda U^T$$

$$= \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ & & \cdots & \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix}^T$$

- U are **eigenvectors** of covariance matrix (Σ)
- Computing Σ already takes $O(nd^2)$ time (very **expensive**)

Method 2:
singular value decomposition (SVD)

- For $X \in R^{d \times n}$ and **rank r**; find

$$X = U_{d \times r} S_{r \times r} V^T_{n \times r}$$

- Where $U^\top U = I_{d \times d}$, $V^\top V = I_{n \times n}$ and **S** is diagonal r

$$X = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix} \begin{bmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & s_r \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_r^T \end{bmatrix}$$

$$= USV^T$$

- Computing top **k** singular vectors takes only O(ndk)

SVD

- The **singular values** $s_1 \geq s_2 \geq \ldots \geq s_r$ are positive real numbers (relationship between s_i to eigenvalues of Σ, $\lambda_i = s_i^2/(n-1)$ $\quad \Sigma = U(S^2/n\text{-}1)U^T$)
- The **left singular vectors** $u_1, u_2, \ldots u_r$ form an orthonormal set and are a **basis** for the **column space**
- The right singular vectors $v_1, v_2, \ldots, v_r$ also form an orthonormal set and a **basis** for the **row space**
- The SVD is **unique** if all the singular values are **different**
- The SVD decomposes the action of a matrix X on a vector a into **Rotation**, **Scaling**, and **Rotation**

$$Xa = USV^T a$$

Rotation Scaling Rotation

Relationship between eigendecomposition and SVD:

$$\Sigma = XX^T / (n\text{-}1)$$

$$= USV^T(USV^T)^T/(n\text{-}1)$$

$$= USV^TVSU^T/(n\text{-}1)$$

$$= US^2U^T/(n\text{-}1)$$

Case study; Eigen-faces [Turk and Pentland, 1991]

- d = number of pixels
- Each $\mathbf{x}_i \in R^d$ is a face image
- $\mathbf{x}_{ji}$ = intensity of the j^{th} pixel in image i

$$\mathbf{X}_{d\times n} \approx \mathbf{U}_{d\times k} \mathbf{Z}_{k\times n}$$

- Idea: z_i more "**meaningful**" **representation** of i^{th} face than x_i
- Can use z_i for nearest-neighbor **classification**
- Much faster: when n>>d and k

Case study; Latent Semantic Analysis [Deerwater, 1990]

- d = number of words in the vocabulary
- Each $\mathbf{x}_i \in R^d$ is a vector of word counts
- x_{ji} = frequency of word j in document I

$$\mathbf{X}_{d\times n} \approx \mathbf{U}_{d\times k}\,\mathbf{Z}_{k\times n}$$

$$\begin{pmatrix} \text{stocks: } 2 \cdots 0 \\ \text{chairman: } 4 \cdots 1 \\ \text{the: } 8 \cdots 7 \\ \cdots \vdots \cdots \vdots \\ \text{wins: } 0 \cdots 2 \\ \text{game: } 1 \cdots 3 \end{pmatrix} \approx \begin{pmatrix} 0.4 \cdots -0.001 \\ 0.8 \cdots 0.03 \\ 0.01 \cdots 0.04 \\ \vdots \cdots \vdots \\ 0.002 \cdots 2.3 \\ 0.003 \cdots 1.9 \end{pmatrix} \begin{pmatrix} | & & | \\ \mathbf{z}_1 & \cdots & \mathbf{z}_n \\ | & & | \end{pmatrix}$$

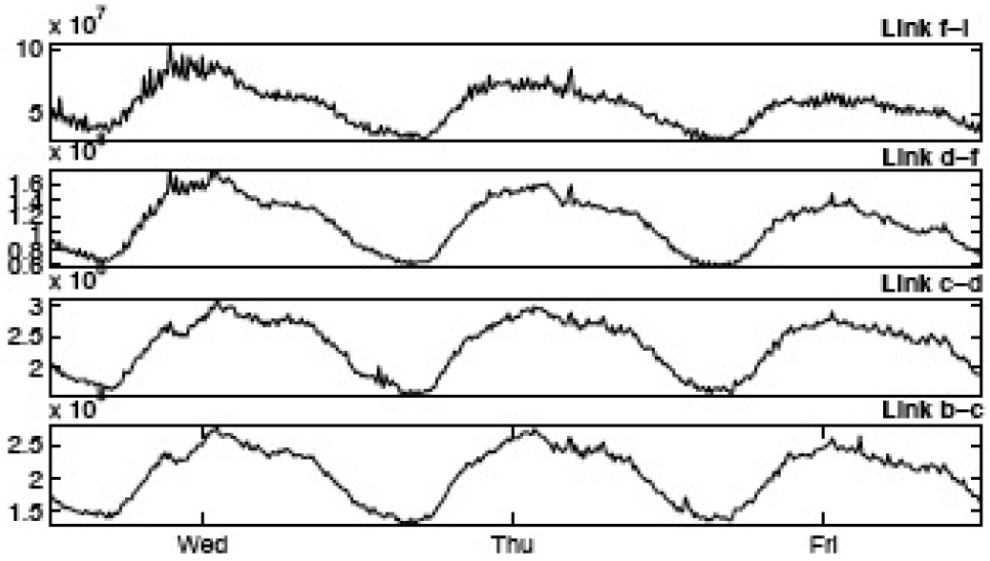
- How to measure **similarity** between two documents?(**Cosine similarity**)

  $z_1^T z_2$ is probably better than $x_1^T x_2$

- Applications: information retrieval
- Note: no computational savings; original x is already sparse

Network anomaly detection [Lakhina, '05]

- x_{ji} = amount of traffic on **link j** in the network during each **time interval** i



Model assumption: total traffic is sum of flows along a few "**paths**"

Apply PCA: each **principal** component intuitively represents a "path"

Anomaly when traffic **deviates** from first **few principal** components