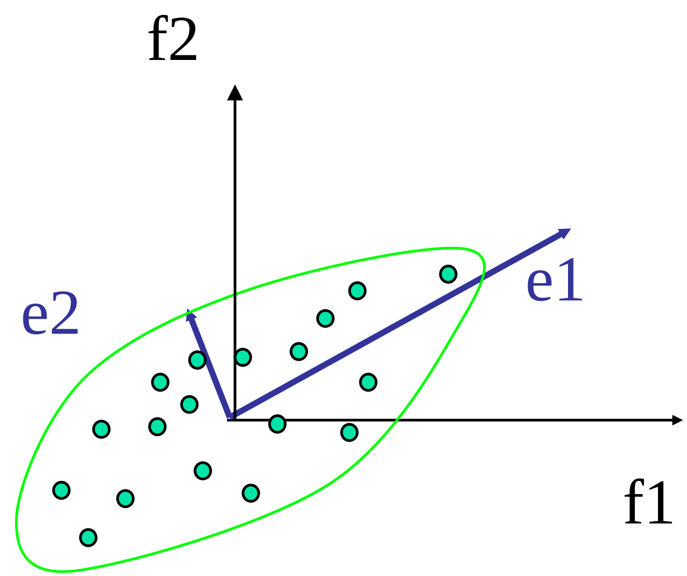
Dimensionality Reduction and Embeddings

SVD: The mathematical formulation

- Normalize the dataset by moving the origin to the center of the dataset
- Find the eigenvectors of the data (or covariance) matrix
- These define the new space
- Sort the eigenvalues in "goodness" order



Compute Approximate SVD efficiently

- Exact SVD is expensive: $O(\min\{n^2 m, n m^2\})$

So, we try to compute it approximately. We exploit the fact that, if $\mathbf{A} = \mathbf{U}\,\Lambda\,\mathbf{V}^T$ then: $\mathbf{A}\,\mathbf{A}^T = \mathbf{U}\,\Lambda^2\,\mathbf{U}^T$ and $\mathbf{A}^T\mathbf{A} = \mathbf{V}\,\Lambda^2\,\mathbf{V}^T$

1. Random projection + SVD.

   Cost $O(m\ n\ \log n)$

2. Random sampling (p rows) and then SVD on the samples.

   Cost $O(\max\{m\ p^2+p^3\})$ or $O(p^4)$!!

   (caution: constants can be high!)

Approximate SVD

- We can guarantee an approximation like the following:

$$|| A - P||^2_F <= || A - A_k||^2_F + \varepsilon \, ||A||^2_F$$

A randomized SVD

We pick s rows from A (m x n) and we create an s x n matrix S. Then we approximate the right singular vectors of A. λ

1. For t=1 to s do
   1. Pick an integer from {1..m}, with Prob(l) $= p_l$, $\sum_{l=0}^{m} p_l$=1
   2. Include row A(l) in S with values divided by $\sqrt{sp_l}$
2. Compute SS^T and its SVD. Now $SS^T = \sum_{t=1}^{s} \lambda_t{}^2\, w^{(t)}\, w^{(t)\,T}$ where λt are the singular values of S and $w^{(t)}$ its left singular vectors.
3. Return $h^{(t)} = S^T w^{(t)} / |S^T w^{(t)}|$, t=1,…, k. These are the approximations of the top k right singular values of A.

We can also create $P=AHH^T$ as a rank k approximation of A.

SVD Cont’d

- Advantages:
  - Optimal dimensionality reduction (for linear projections)

- Disadvantages:
  - Computationally expensive… but can be improved with random sampling
  - Sensitive to outliers and non-linearities

Embeddings

- Given a metric distance matrix D, embed the objects in a k-dimensional vector space using a mapping F such that
  - D(i,j) is close to D'(F(i),F(j))
- Isometric mapping:
  - exact preservation of distance
- Contractive mapping:
  - D'(F(i),F(j)) <= D(i,j)
- D' is some Lp measure

Multi-Dimensional Scaling (MDS)

- Map the items in a k-dimensional space trying to minimize the stress

$$stress = \sqrt{\frac{\sum_{i,j} (\hat{d}_{ij} - d_{ij})^2}{\sum_{i,j} d_{ij}^2}}, d_{ij} = | o_j - o_i | \quad and \quad \widehat{d}_{ij} = | \widehat{o}_j - \widehat{o}_i |$$

- Steepest Descent algorithm:
  - Start with an assignment
  - Minimize stress by moving points
- But the running time is $O(N^2)$ and $O(N)$ to add a new item
- Another method: stress iterative majorization

FastMap

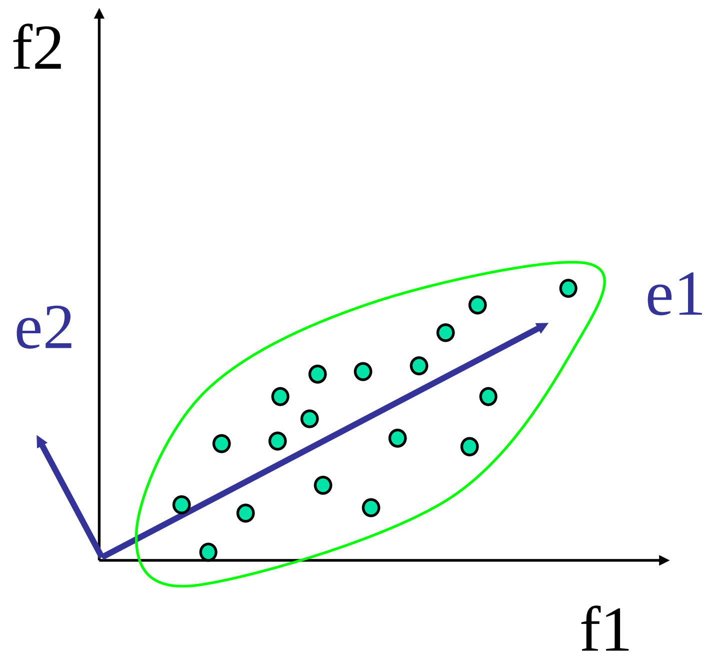
What if we have a finite metric space (X, d)?

Faloutsos and Lin (1995) proposed FastMap as metric analogue to the PCA. Imagine that the points are in a Euclidean space.

- Select two **pivot points** x_a and x_b that are far apart.
- Compute a **pseudo-**projection of the remaining points along the "line" $x_a x_b$.
- **"Project"** the points to an orthogonal subspace and **recurse**.

FastMap

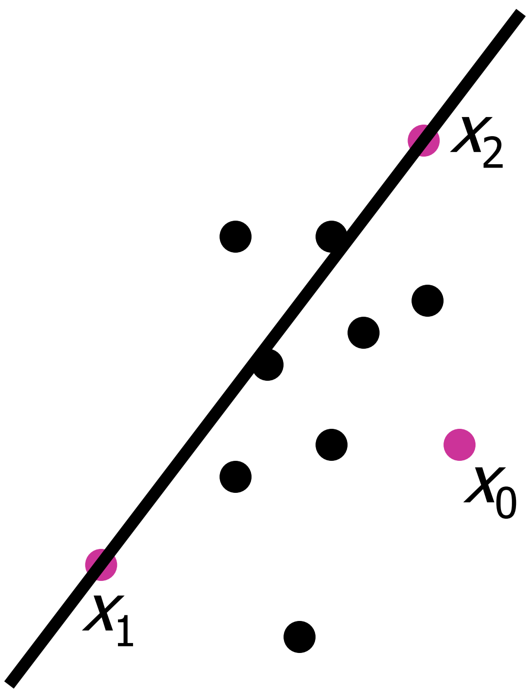
- We want to find e1 first



Selecting the Pivot Points

The pivot points should lie along the principal axes, and hence should be far apart.

- Select any point x_0.
- Let x_1 be the furthest from x_0.
- Let x_2 be the furthest from x_1.
- Return (x_1, x_2).



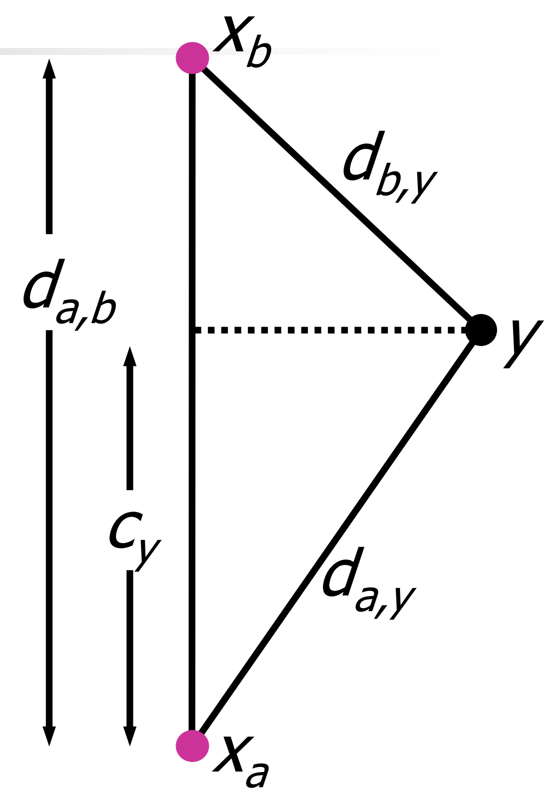
Pseudo-Projections

Given pivots (x_a , x_b), for any third point y, we use the **law of cosines** to determine the relation of y along $x_a x_b$.

$$d_{by}^2 = d_{ay}^2 + d_{ab}^2 - 2c_y d_{ab}$$

The **pseudo-projection** for y is

$$c_y = \frac{d_{ay}^2 + d_{ab}^2 - d_{by}^2}{2d_{ab}}$$

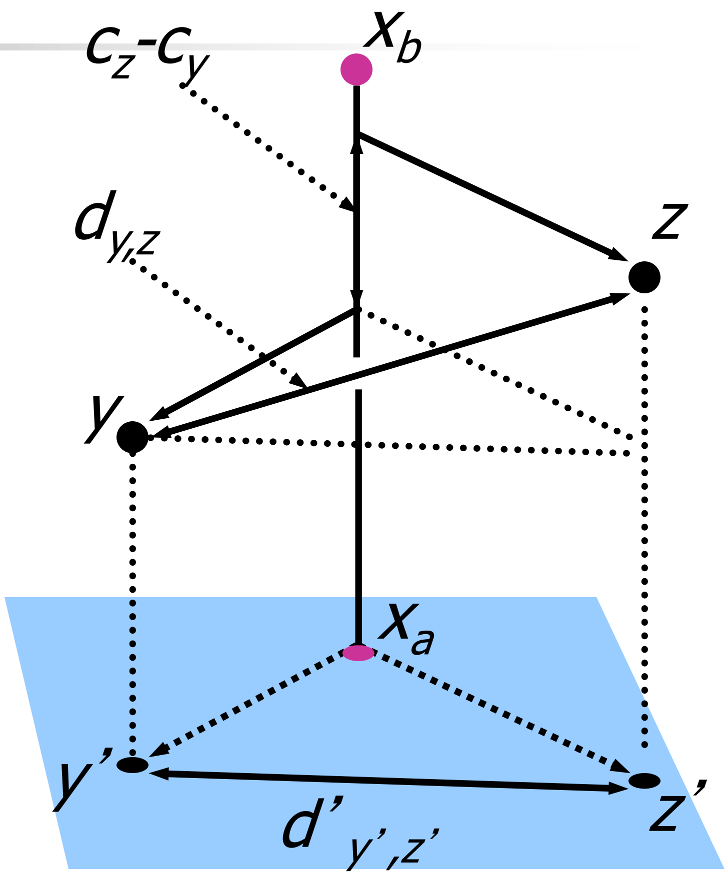


"Project to orthogonal plane"

Given distances along $x_a x_b$ we can compute distances within the "orthogonal hyperplane" using the Pythagorean theorem.

$$d'(y',z') = \sqrt{d^2(y,z) - (c_z - c_y)^2}$$

Using d'(.,.), recurse until k features chosen.



Compute the next coordinate

- Now, we have projected all objects into a subspace orthogonal to first dimension (line x_a, x_b)
- We can apply recursively FastMap on the new projected dataset:

  FastMap(k-1, d’, D)

Random Projections

- Based on the Johnson-Lindenstrauss lemma:
- For:
  - $0 < \varepsilon < 1/2$,
  - any (sufficiently large) set $\boldsymbol{S}$ of M points in R_n
  - $k = O(\varepsilon^{-2} \ln M)$
- There exists a linear map $f: \boldsymbol{S} \rightarrow R_k$, such that
  - $(1 - \varepsilon)\, D(S,T) < D(f(S),f(T)) < (1+ \varepsilon) D(S,T)$ for S,T in $\boldsymbol{S}$
- Random projection is good with constant probability

Random Projection: Application

- Set k = $O(\varepsilon^{-2} \ln M)$
- Select k random n-dimensional vectors
  - (an approach is to select k gaussian distributed vectors with variance 1 and mean value 0: N(0,1))
- Project the original points into the k vectors.
- The resulting k-dimensional space approximately preserves the distances with high probability

Database Friendly Random Projection

- For each point (vector) $\boldsymbol{x}$ in d-dimensions need to find the projection to point $\boldsymbol{y}$ in k-dimensions
- For n points, using the naive approach, I need to perform ndk operations.
- this can be large for large datasets and dimensionalities.
- A better approach is the following [Achlioptas 2003]:
  - Create a matrix **A** such that:

$$\mathbf{A[i,j]} = \begin{cases} \mathbf{1} & \text{with prob } 1/6 \\ \mathbf{0} & \text{with prob } 2/3 \\ \mathbf{-1} & \text{with prob } 1/6 \end{cases}$$

Then, we can compute each $\boldsymbol{y}$ as: $\boldsymbol{y} = \boldsymbol{x}\,\mathbf{A}$

Why this is better?

Random Projection

- A very useful technique,
- Especially when used in conjunction with another technique (for example SVD)
- Use Random projection to reduce the dimensionality from thousands to hundred, then apply SVD to reduce dimensionality farther

References:

[Achlioptas 2003] Dimitris Achlioptas: Database-friendly random projections: Johnson-Lindenstrauss with binary coins. J. Comput. Syst. Sci. 66(4): 671-687 (2003)