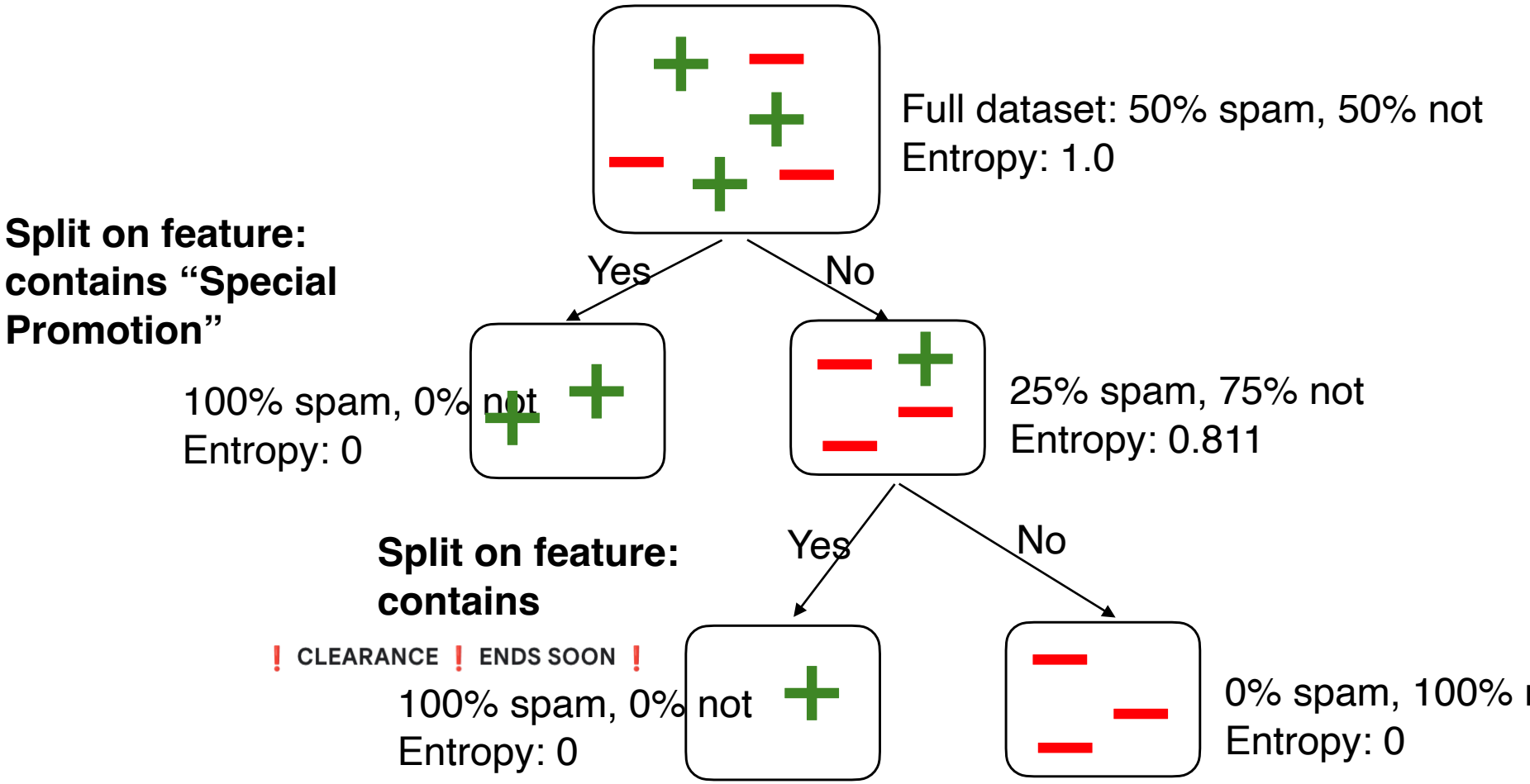
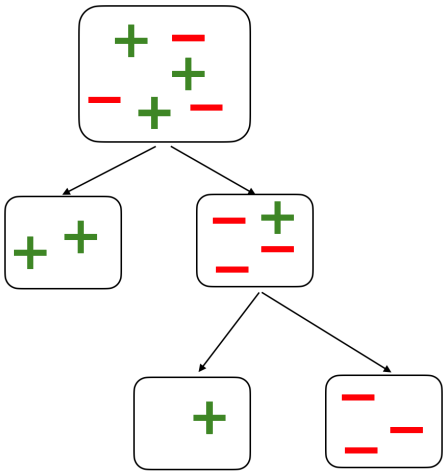
Trees, continued

Decision trees: How to build them



Generic Tree-Building algorithm



Overall goal: choose features which can best sort positive and negative examples. For all the samples in a region R_m, predict the majority class c_m

Loss func?

0/1 error / binary classification error

$$f(x) = \sum_{m=1}^{M} c_m 1\{x \in R_m\}$$

split on $x_j \leq s, \quad x_j > s$

Same

Same for illustrative purposes. In reality you can also have categorical features. E.g. “Islet Antibody Positive Test”, or “Blood Type”

Define “impurity” measure I (measures how well positive class examples are sorted from negative)
Find split that minimizes impurity

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m).$$

$$R_1(j,s) = \{X | X_j \leq s\} \text{ and } R_2(j,s) = \{X | X_j > s\}.$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,\, s}\Big[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2\Big].$$

Generic Tree-Building algorithm

1. Start from empty decision tree
2. Recursively, for each node:
   - Iterate through all features and compute how good it'd be to split on each feature
   - Split on the "best" feature
3. Prune

Design choices:

- Termination condition (max depth, entropy, train/val error)
- Tree complexity
- Splitting criterion
- Pruning

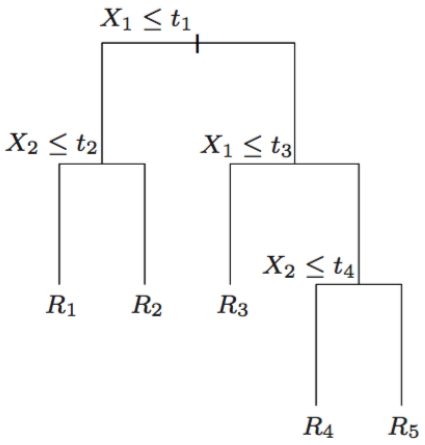
Axis-aligned trees

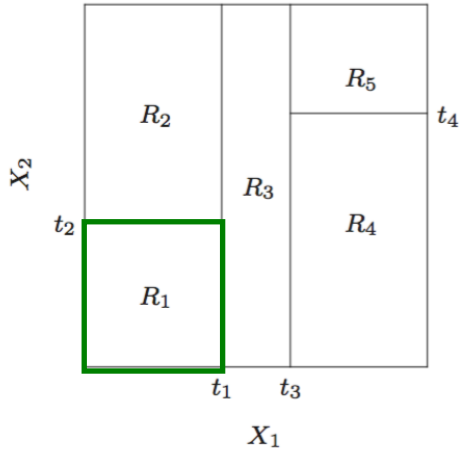
Regression Trees

Overall goal: carve feature space X into M regions, $R_1 \ldots R_M$. For all the samples in a region R_m, predict some value c_m

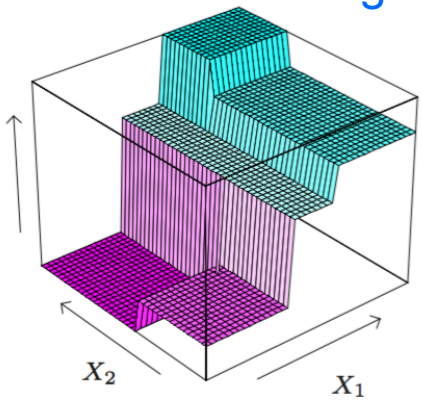
$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Actual output value for regression





Decision boundary



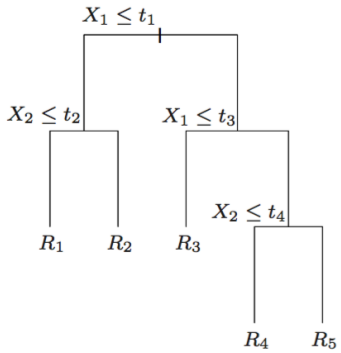
When to use?

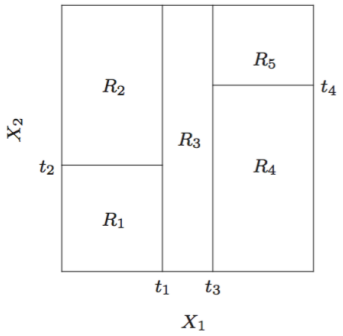
- Can be interpretable

When not to use?

- Complex, high-dimensional data (e.g. pixels)

Regression Trees: what to predict at the leaves





Loss func? MSE

For region m, which has samples with labels $y_1, y_2 \dots y_n$

$$\min_{c_m \in \mathbb{R}} \sum_i (y_i - c_m)^2$$ # How do we find c_m?

$$\frac{d}{dc} \sum_i (y_i - c)^2 = 0$$ # Take the derivative and set it to zero!

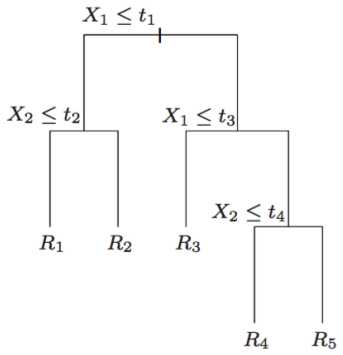
$$= -2 \sum_i (y_i - c) = 0$$

$$nc = \sum_i y_i \quad \rightarrow \boxed{c = \frac{1}{n} \sum_i y_i}$$ # Just the mean of the y values

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m).$$

Regression Trees: when to split a node



Loss func? MSE

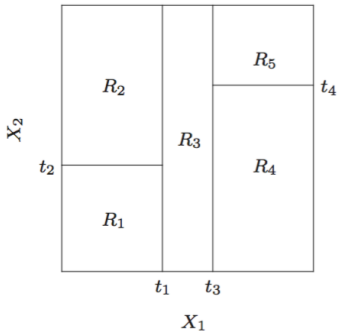
$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m).$$

$$R_1(j,s) = \{X | X_j \le s\} \text{ and } R_2(j,s) = \{X | X_j > s\}.$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,\, s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$



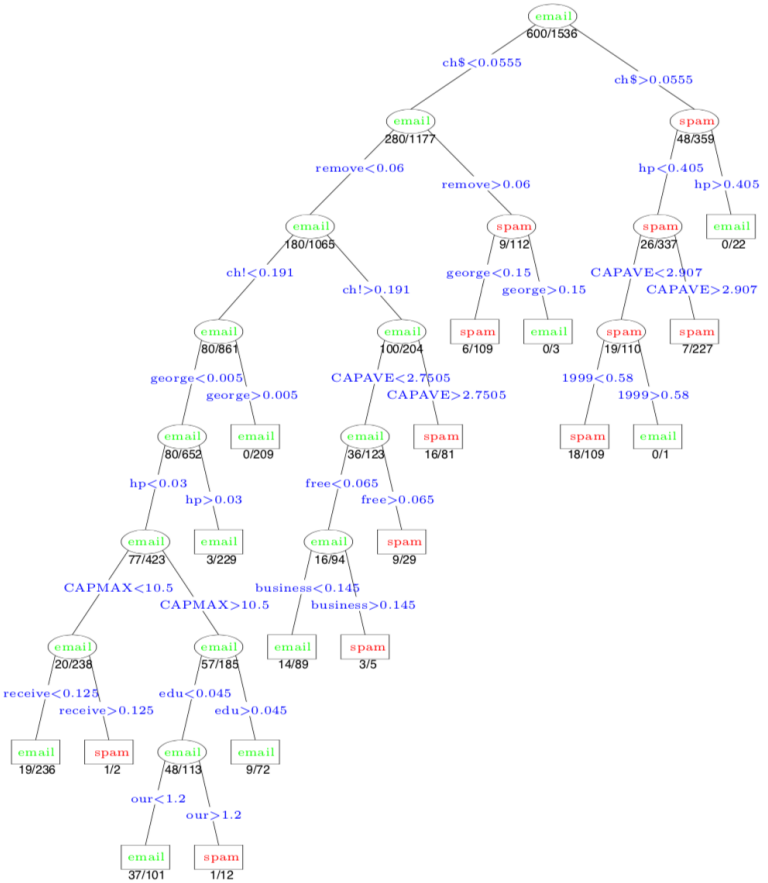
Trees: Interpretable (?)

Trees are “easy” to interpret:

- You can explain <u>how</u> the classifier came to the conclusion it did

But they can be complex

- Small changes in data can result in large difference in trees



Summary of trees

- Trees have low bias, high variance
- Deal with categorial variables well # Where don't they work?
- Intuitive, "interpretable" Many related continuous features (pixels)
- Good software exists
- Some theoretical guarantees

Why low bias?

If you allow enough depth / splits, you can fit anything

How to regularize?

We will say more, but you can limit the number of nodes, the depth, or the requirements on entropy for leaves

Ensembling methods

(Some slides from Natasha Jaques)

UNIVERSITY of WASHINGTON

So far

1. Talked regression (linear, mostly)
2. Classification
    1. Logistic Regression
    2. SVMs
    3. Kernelized versions
    4. k-NN
    5. Neural Nets
    6. Trees

Most of these can be used for regression too

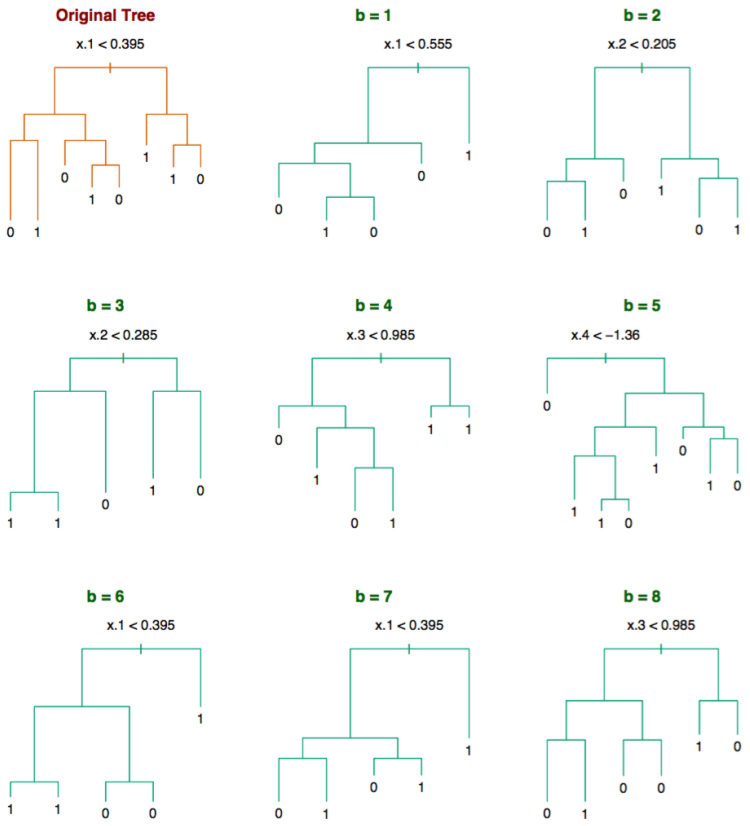
Was going to more complex model classes so really necessary?
- If we have sufficiently complex kernels, or 1-NN, or high depth trees, we can get very low bias, high variance…
- What about combining weak learners into better learners?

Random Forests, aka “bagging” of trees

- Forest = many trees
- Tree methods have low bias but high variance
- We can reduce variance by constructing many “lightly correlated” trees and averaging them

- Bagging: Bootstrap aggregating

Each tree is trained on new bootstrap sample of data



Random Forests…

Random Forest learning algorithm

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

with replacement

b = train tree

but you're using a random subset of the features

Average

Other ensembling methods? Boosting!

Boosting!!! A personal favorite.

Take advantage of the fact that you can learn what parts of the space you are bad at, and do better there.

- 1988 Kearns and Valiant: "Can **weak learners** be combined to create a **strong learner?**"

An algorithm $\mathcal{A}$ is a *weak learner* for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $\mathcal{A}$ correctly classifies h with error at most $1/2 - \gamma$

1990 Robert Schapire: "Yup!"
1995 Schapire and Freund: "Practical for 0/1 loss" AdaBoost
2001 Friedman: "Practical for arbitrary losses"
2014 Tianqi Chen: "Scale it up!" XGBoost

Additive models

- Consider the first algorithm we used to get good classification for MNIST. Given: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$
- Generate **random** functions: $\phi_t : \mathbb{R}^d \to \mathbb{R} \quad t = 1, \ldots, p$
- Learn some weights: $$\widehat{w} = \arg\min_w \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$$
- Classify new data: $$f(x) = \text{sign}\left(\sum_{t=1}^p \widehat{w}_t \phi_t(x)\right)$$

An interpretation:
Each $\phi_t(x)$ is a classification rule that we are assigning some weight $\widehat{w}_t$

$$\widehat{w}, \widehat{\phi}_1, \ldots, \widehat{\phi}_t = \arg\min_{w, \phi_1, \ldots, \phi_p} \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$$

is in general computationally hard

Adaboost

$b(x, \gamma)$ is a function with parameters γ

$$b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}}$$

$$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

AdaBoost: $b(x, \gamma)$: classifiers to $\{-1, 1\}$

$$L(y, f(x)) = \exp(-yf(x))$$

Boosted Regression Trees

$b(x, \gamma)$ is a function with parameters γ

$$b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}}$$

$$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

Boosted Regression Trees: $L(y, f(x)) = (y - f(x))^2$

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \qquad r_{im} = y_i - f_{m-1}(x_i) \end{aligned}$$

Efficient: No harder than learning regression trees!

Boosted Logistic Trees

$b(x, \gamma)$ is a function with parameters γ

$$b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}}$$

$$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

Boosted Logistic Trees: $L(y, f(x)) = y \log(f(x)) + (1 - y) \log(1 - f(x))$

$b(x, \gamma)$: regression trees

$$r_{im} = y_i - f_{m-1}(x_i)$$

Computationally hard to update

Gradient Boosting

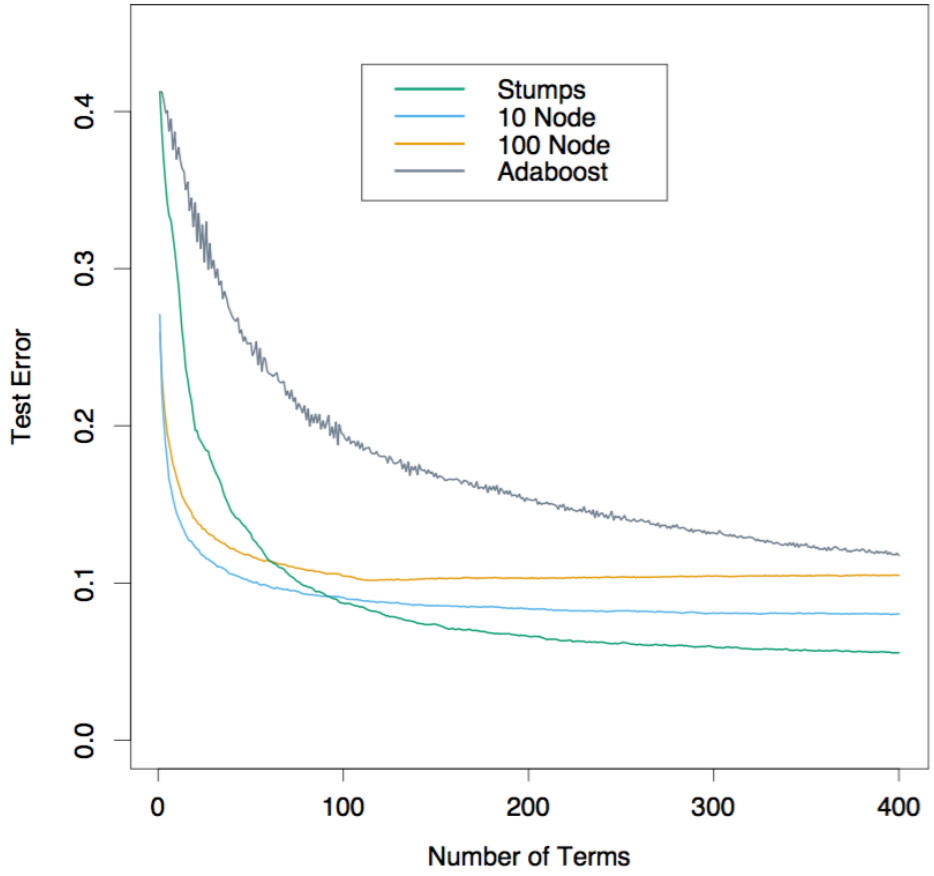
Least squares, exponential loss easy. But what about cross entropy? Huber?

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

LS fit regression tree to n-dimensional gradient, take a step in that direction

Gradient Boosting, empirical performance



AdaBoost uses 0/1 loss,
all other trees are minimizing
binomial deviance

Boosting: everyone's favorite algorithm

- Boosting is popular at parties: Invented by theorists, heavily adopted by practitioners.
- Computationally efficient with "weak" learners. But can also use trees! Boosting can scale.
- Kind of like sparsity?
- Gradient boosting generalization with good software packages (e.g., *XGBoost)*. Effective on Kaggle
- Robust to overfitting and can be dealt with with "shrinkage" and "sampling"

Bagging versus Boosting

- Bagging *averages* many **low-bias**, **lightly dependent** classifiers to reduce the variance
- Boosting *learns* linear combination of **high-bias**, **highly dependent** classifiers to reduce error
- Empirically, boosting appears to outperform bagging

Unsupervised Learning

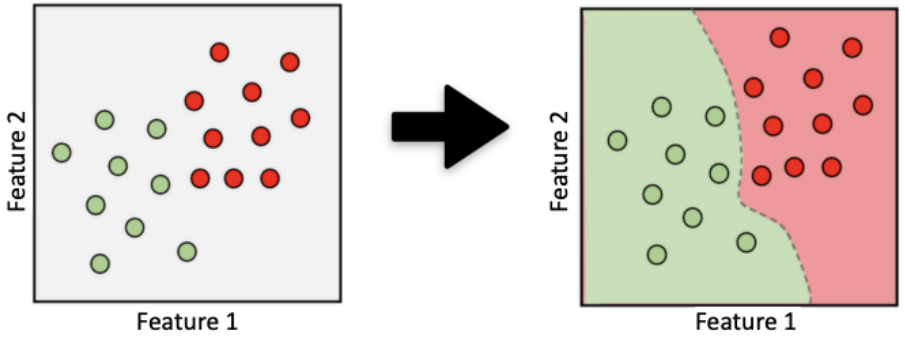
UNIVERSITY of WASHINGTON

Unsupervised vs. Supervised Learning

Previously: Supervised Learning

- Each data point x_i has a corresponding label y_i.
- Dataset is $\{x_i, y_i\}_{i=1}^n$,

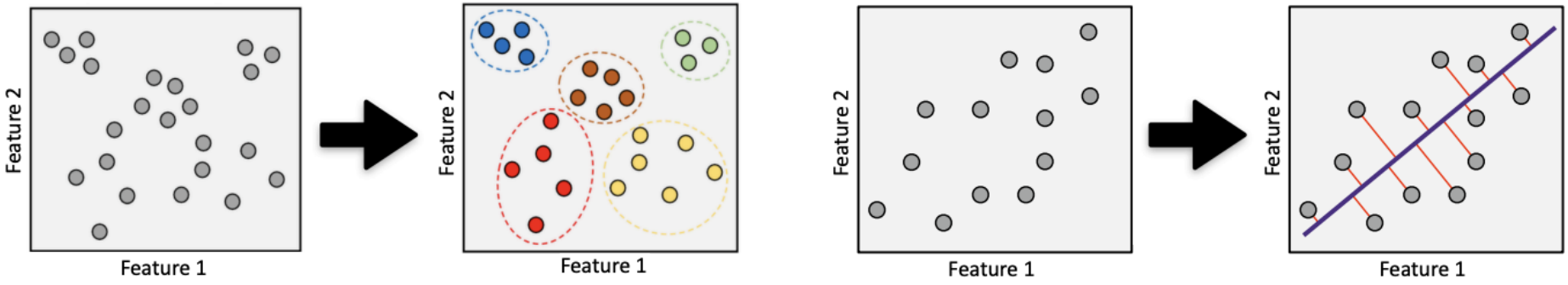
  where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.



Next few lectures: Unsupervised Learning

- No labels. Only data: $\{x_i\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$. Goal is to model the data distribution $P(X)$, potentially by finding:

clusters in the data or low-dimensional structure



Principal Component Analysis

Motivation for Dimensionality Reduction

Goal: find a $k < d$-dimensional representation of X which captures "most" information

(Since many datasets have features with lots of shared/mutual information)

Linear projections

Given $x_1, \ldots, x_n \in \mathbb{R}^d$, for $q << d$, find a compression $z_1, \ldots, z_n \in \mathbb{R}^q$ such that

$x_i \approx \mu + V_q z_i$ and $\mathbf{V}_q^\top V_q = I$.

$$\min_{V_q, \{z_i\}} \sum_{i=1}^{n} ||x_i - \mu - V_q z_i||^2. \qquad \mathbf{V}_q \in \mathbb{R}^{d \times q}, \qquad \mathbf{V}_q^\top \mathbf{V}_q = I_q$$

$\mathbf{V}_q$ is orthonormal, (in columns)

$$y \qquad Xw$$

Fix V_q, solve for $\{z_i\}$

$$\hat{w} = (X^\top X)^{-1} X^\top y$$

$$v_i^\top v_j = \begin{cases} 0, & i \neq j, \\ 1, & i = j, \end{cases}$$

$$\|v_i\|_2^2 = v_i^\top v_i = 1.$$

$$\hat{z}_i = (\mathbf{V}_q^\top \mathbf{V}_q)^{-1} \mathbf{V}_q^\top (x_i - \mu)$$

$$= \mathbf{V}_q^\top (x_i - \mu)\,.$$

$V_q \mathbf{V}_q^\top$ projects de-meaned data onto a q-dimensional subspace embedded in $\mathbb{R}^d$.

Linear projections: geometrically

Given $x_1, \ldots, x_n \in \mathbb{R}^d$, for $q << d$, find a compression $z_1, \ldots, z_n \in \mathbb{R}^q$ such that

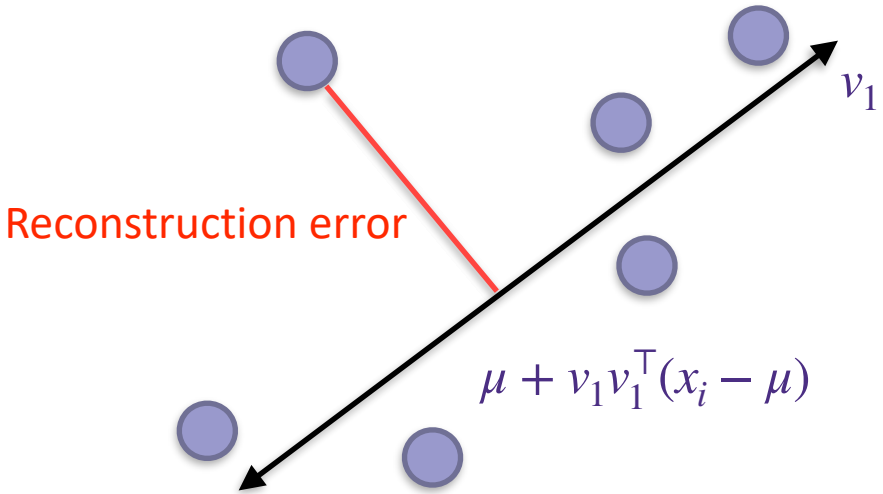
$x_i \approx \mu + V_q z_i$ and $\mathbf{V}_q^\top V_q = I$.

$$\min_{V_q, \{z_i\}} \sum_{i=1}^{n} ||x_i - \mu - V_q z_i||^2.$$

Fix V_q, solve for $\{z_i\}$

$$\hat{z}_i = (\mathbf{V}_q^\top \mathbf{V}_q)^{-1} \mathbf{V}_q^\top (x_i - \mu)$$

$$= \mathbf{V}_q^\top (x_i - \mu)\,.$$



$\mathbf{V}_q \mathbf{V}_q^T$ is a *projection matrix* that minimizes error in basis of size q

PCA

Data dependent dimensionality reduction

- Useful for
  - Visualization
  - Interpretation
  - Compression
  - Understanding “intrinsic dimension”

	kale	taco bell	sashimi	pop tarts
Alice	10	1	2	7
Bob	7	2	1	10
Carolyn	2	9	7	3
Dave	3	6	10	2

Figure credit:
Karlin
Roughgarden + Va
Benedetto
Novembre et al
Alex Williams
Sandipan Dey
Victor Lavenko

PCA

Claim:

Each row can be expressed approximately as

$$x_i \approx \bar{x} + a_{i1}\vec{v_1} + a_{i2}\vec{v_2}$$

$$\vec{v_1} = [3 \quad -3 \quad -3 \quad 3]$$

$$\vec{v_2} = [1 \quad -1 \quad 1 \quad -1]$$

	kale	taco bell	sashimi	pop tarts
Alice	10	1	2	7
Bob	7	2	1	10
Carolyn	2	9	7	3
Dave	3	6	10	2

$$\bar{x} = [5.5 \quad 4.5 \quad 5 \quad 5.5]$$

Figure credit:
Karlin
Roughgarden + Va
Benedetto
Novembre et al
Alex Williams
Sandipan Dey
Victor Lavenko

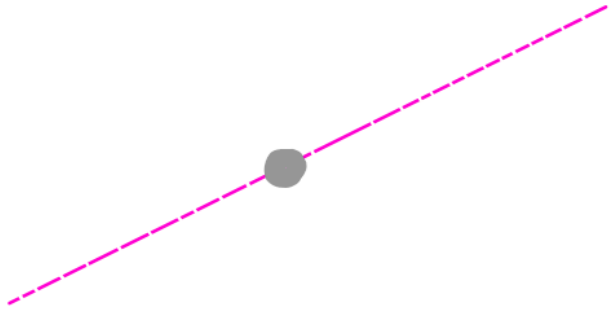
PCA in one dimension

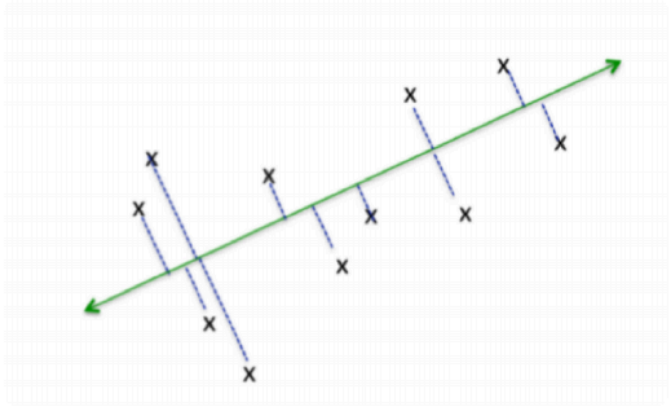
Goal: find a k < d-dimensional representation of X

For k = 1:

Choose $\vec{v} \in \mathbb{R}^d, ||v|| = 1$
to minimize

$$\frac{1}{n}\sum_{i=1}^{n} dist(x_i, \text{line defined by } \vec{v})$$





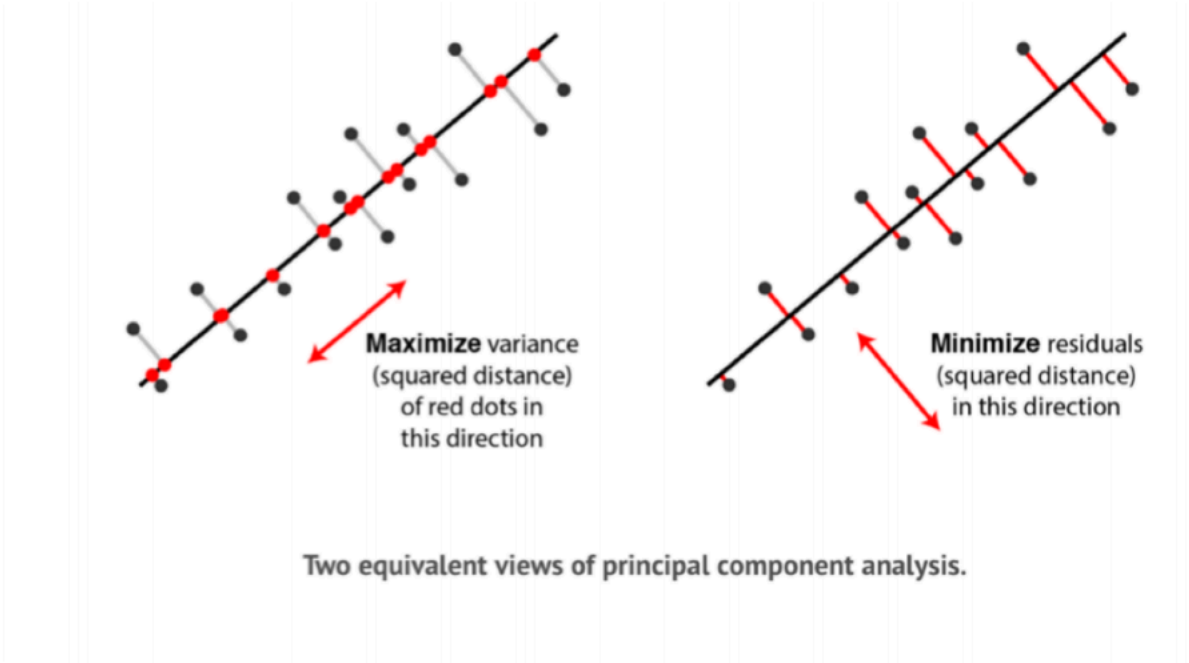
PCA in one dimension, 2 equivalent views

Goal: find a k < d-dimensional representation of X

For k = 1:

Choose $\vec{v} \in \mathbb{R}^d, ||v|| = 1$
to minimize

$$\frac{1}{n}\sum_{i=1}^{n} dist(x_i, \text{line defined by } \vec{v})$$



Two equivalent views of principal component analysis.

PCA: a high-fidelity linear projection

$$\sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})||_2^2$$

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T$$

$$\mathbf{V}_q^T \mathbf{V}_q = I_q$$

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})||_2^2 = \min_{\mathbf{V}_q} Tr(\Sigma) - Tr(\mathbf{V}_q^T \Sigma \mathbf{V}_q)$$

Eigenvalue decomposition

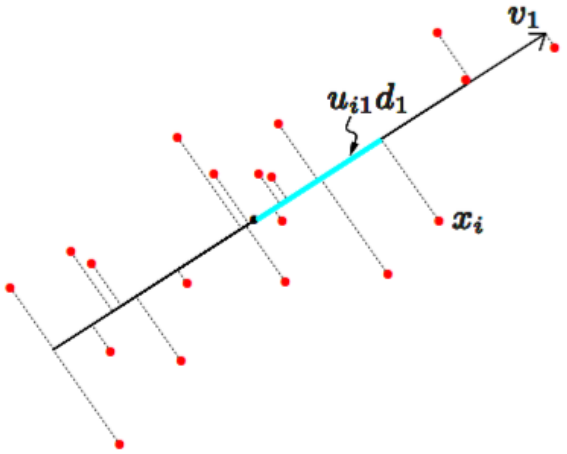
$\mathbf{V}_q$ are the first q eigenvectors of Σ

Minimize reconstruction error and capture the most variance in your data.

PCA: a high-fidelity linear projection

Given $x_i \in \mathbb{R}^d$ and some $q < d$ consider

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})||^2.$$



where $\mathbf{V}_q = [v_1, v_2, \ldots, v_q]$ is orthonormal:

$$\mathbf{V}_q^T \mathbf{V}_q = I_q$$

$\mathbf{V}_q$ are the first q eigenvectors of Σ

$\mathbf{V}_q$ are the first q *principal components*

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T$$

Principal Component Analysis (PCA) projects $(\mathbf{X} - \mathbf{1}\bar{x}^T)$ down onto $\mathbf{V}_q$

$$(\mathbf{X} - \mathbf{1}\bar{x}^T)\mathbf{V}_q = \mathbf{U}_q \mathrm{diag}(d_1, \ldots, d_q) \qquad \mathbf{U}_q^T \mathbf{U}_q = I_q$$

How do we compute the principal components?

1. Power iteration
2. **Solving for a singular value decomposition (SVD)**

Singular Value Decomposition (SVD)

Theorem (SVD): Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with rank $r \leq \min\{m, n\}$. Then $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ where $\mathbf{S} \in \mathbb{R}^{r \times r}$ is diagonal with positive entries, $\mathbf{U}^T\mathbf{U} = I$, $\mathbf{V}^T\mathbf{V} = I$.

$$
\begin{aligned}
\mathbf{A}^T\mathbf{A}v_i = \quad & (USV^\top)^\top(USV^\top)v_i && = VS^\top U^\top USV^\top v_i \\
& && = VS^2V^\top v_i \\
& && = VS^2 e_i \\
\mathbf{A}\mathbf{A}^T u_i = \quad & = S_{ii}^2 u_i\,. && = S_{ii}^2 v_i\,.
\end{aligned}
$$

Singular Value Decomposition (SVD)

Theorem (SVD): Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with rank $r \leq \min\{m, n\}$. Then $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ where $\mathbf{S} \in \mathbb{R}^{r \times r}$ is diagonal with positive entries, $\mathbf{U}^T\mathbf{U} = I$, $\mathbf{V}^T\mathbf{V} = I$.

$$\mathbf{A}^T\mathbf{A}v_i = \mathbf{S}_{i,i}^2 v_i$$

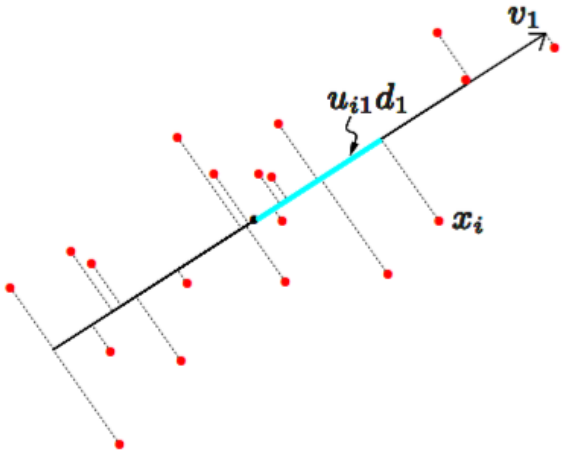
$$\mathbf{A}\mathbf{A}^T u_i = \mathbf{S}_{i,i}^2 u_i$$

$\mathbf{V}$ are the first r eigenvectors of $\mathbf{A}^T\mathbf{A}$ with eigenvalues $\text{diag}(\mathbf{S})$
$\mathbf{U}$ are the first r eigenvectors of $\mathbf{A}\mathbf{A}^T$ with eigenvalues $\text{diag}(\mathbf{S})$

Linear projections

Given $x_i \in \mathbb{R}^d$ and some $q < d$ consider

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})||^2.$$



where $\mathbf{V}_q = [v_1, v_2, \ldots, v_q]$ is orthonormal:

$$\mathbf{V}_q^T \mathbf{V}_q = I_q$$

$\mathbf{V}_q$ are the first q eigenvectors of Σ

$\mathbf{V}_q$ are the first q *principal components*

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T$$

Principal Component Analysis (PCA) projects $(\mathbf{X} - \mathbf{1}\bar{x}^T)$ down onto $\mathbf{V}_q$

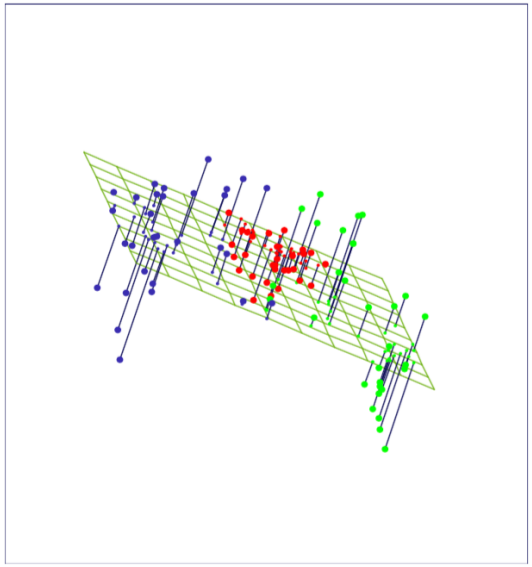
$$(\mathbf{X} - \mathbf{1}\bar{x}^T)\mathbf{V}_q = \mathbf{U}_q \mathrm{diag}(d_1, \ldots, d_q) \qquad \mathbf{U}_q^T \mathbf{U}_q = I_q$$

Singular Value Decomposition defined as

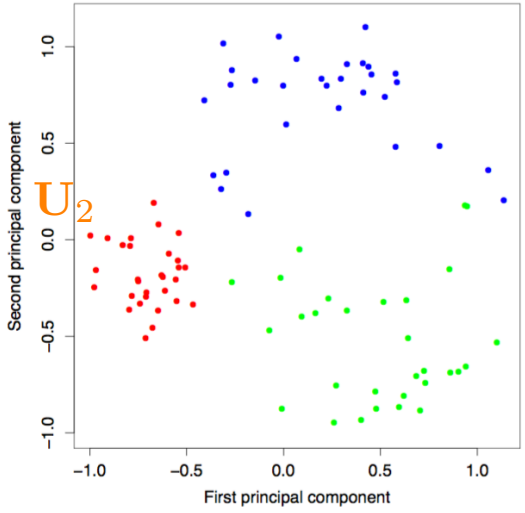
$$\mathbf{X} - \mathbf{1}\bar{x}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

Dimensionality reduction

$\mathbf{V}_q$ are the first q eigenvectors of Σ and *SVD* $\mathbf{X} - \mathbf{1}\bar{x}^T = \mathbf{USV}^T$



$\mathbf{X} - \mathbf{1}\bar{x}^T$



$\mathbf{U}_2$

$\mathbf{U}_1$

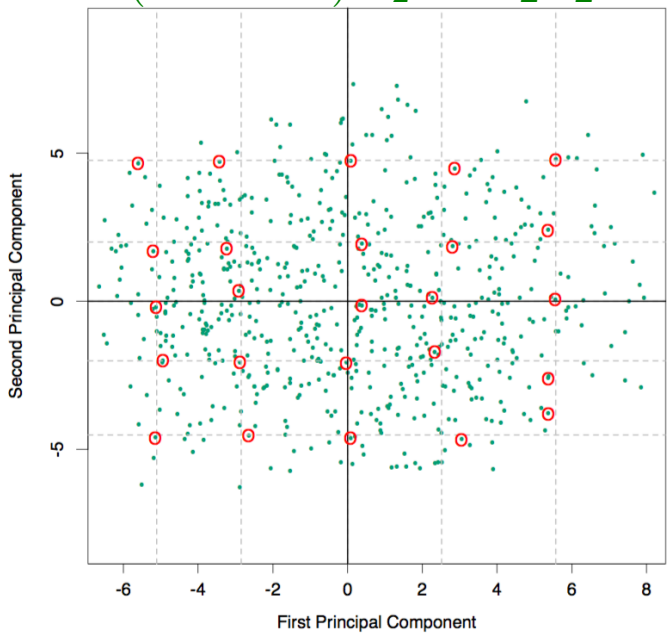
Dimensionality reduction

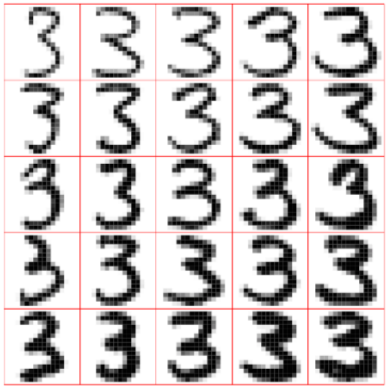
$\mathbf{V}_q$ are the first q eigenvectors of Σ and *SVD* $\mathbf{X} - \mathbf{1}\bar{x}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T$

Handwritten 3's, 16x16 pixel image so that $x_i \in \mathbb{R}^{256}$

$$\begin{aligned}\hat{f}(\lambda) &= \bar{x} + \lambda_1 v_1 + \lambda_2 v_2 \\ &= \bar{x} + \lambda_1 \cdot v_1 + \lambda_2 \cdot v_2 .\end{aligned}$$

$(\mathbf{X} - \mathbf{1}\bar{x}^T)\mathbf{V}_2 = \mathbf{U}_2\mathbf{S}_2 \in \mathbb{R}^{n\times 2}$





$\mathrm{diag}(\mathbf{S})$

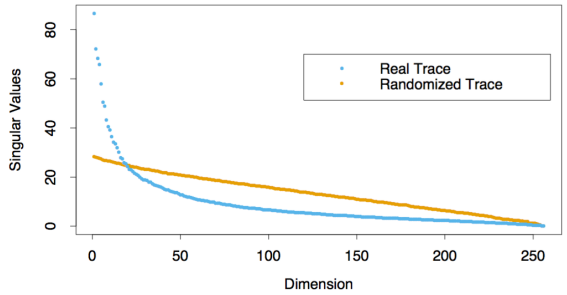


FIGURE 14.24. *The 256 singular values for the digitized threes, compared to those for a randomized version of the data (each column of* $\mathbf{X}$ *was scrambled).*

PCA Algorithm

PCA

input

A matrix of m examples $X \in \mathbb{R}^{m,d}$

number of components n

if $(m > d)$

$A = X^\top X$

Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be the eigenvectors of A with largest eigenvalues

else

$B = XX^\top$

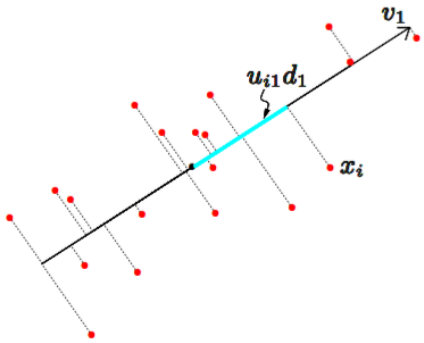
Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the eigenvectors of B with largest eigenvalues

for $i = 1, \ldots, n$ set $\mathbf{u}_i = \frac{1}{\|X^\top \mathbf{v}_i\|} X^\top \mathbf{v}_i$

output: $\mathbf{u}_1, \ldots, \mathbf{u}_n$

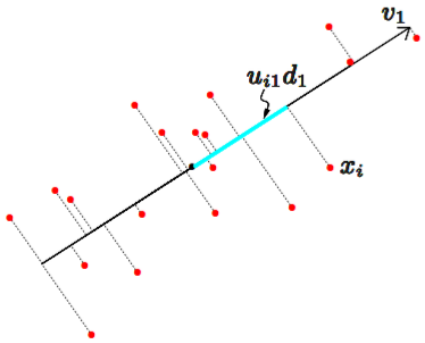
Power method - one at a time

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \qquad v_* = \arg\max_{v} \; v^T \Sigma v$$



Power method - one at a time

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \qquad v_* = \arg\max_{v} \; v^T \Sigma v$$



$$z_0 \sim \mathcal{N}(0, I) \qquad \text{Iterate:} \quad z_{t+1} = \frac{\Sigma z_t}{\|\Sigma z_t\|_2}$$

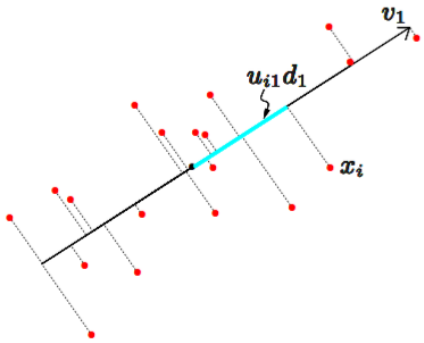
To analyze write: $\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T \qquad z_t =: \mathbf{V}\alpha_t$

$$\alpha_{t+1} = \mathbf{V}^T z_{t+1} = \frac{\mathbf{V}^T \Sigma z_t}{\|\Sigma z_t\|} = \frac{\mathbf{D}\alpha_t}{\|\mathbf{D}\alpha_t\|} = \frac{\mathbf{D}^2\alpha_{t-1}}{\|\mathbf{D}^2\alpha_{t-1}\|} = \frac{\mathbf{D}^t\alpha_0}{\|\mathbf{D}^t\alpha_0\|}$$

$$\mathbf{D}^t = (\mathbf{D}_{1,1})^t(\mathbf{D}/\mathbf{D}_{1,1})^t \to (\mathbf{D}_{1,1})^t \mathbf{e}_1\mathbf{e}_1^T \text{ since } \mathbf{D}_{i,i}/\mathbf{D}_{1,1} < 1$$

Power method - one at a time

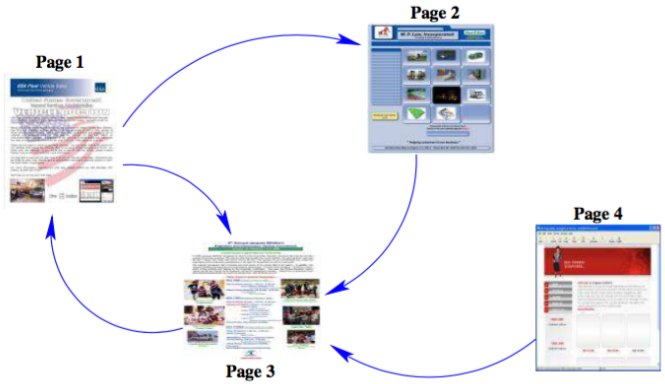
$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \qquad v_* = \arg\max_{v} \; v^T \Sigma v$$



$z_0 \sim \mathcal{N}(0, I)$ Iterate: $z_{t+1} = \dfrac{\Sigma z_t}{\|\Sigma z_t\|_2}$

To analyze write: $\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T \qquad z_t =: \mathbf{V}\alpha_t$

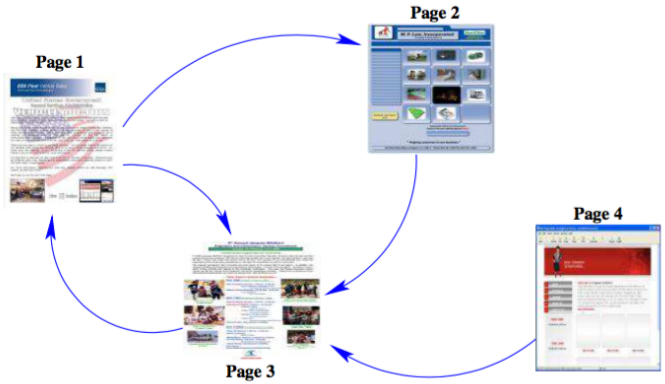
Markov chains - PageRank



Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



Google PageRank of page i:

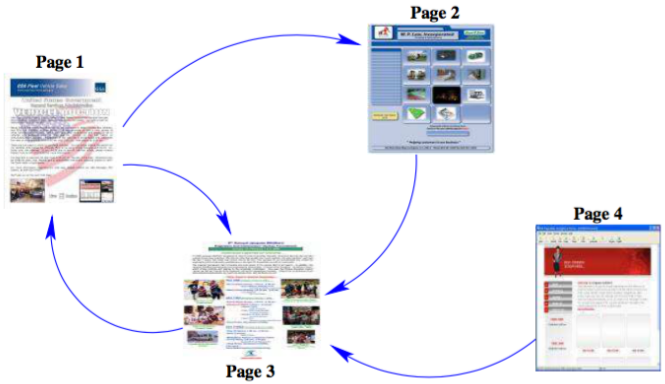
$$p_i = (1-\lambda) + \lambda \sum_{j=1}^{n} \frac{L_{i,j}}{c_j} p_j$$

$$c_j = \sum_{k=1}^{n} L_{j,k}$$

Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



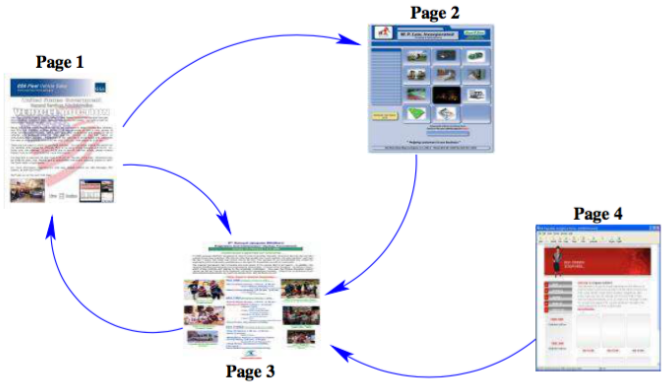
Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Set arbitrary normalization: $\mathbf{1}^T\mathbf{p} = n$ so that

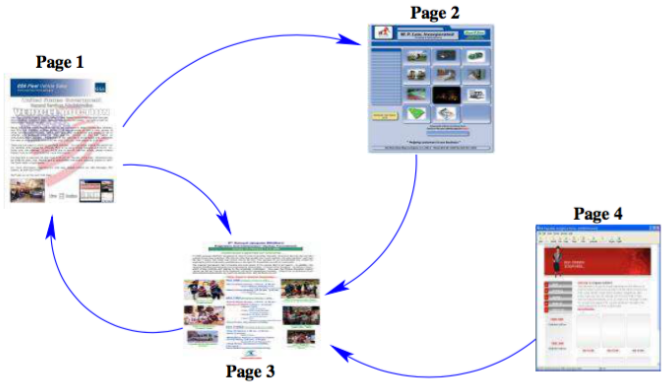
$$\mathbf{p} = \left((1-\lambda)\mathbf{1}\mathbf{1}^T/n + \lambda \mathbf{L}\mathbf{D}_c^{-1}\right)\mathbf{p}$$

$$=: \mathbf{A}\mathbf{p}$$

Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Set arbitrary normalization: $\mathbf{1}^T\mathbf{p} = n$ so that

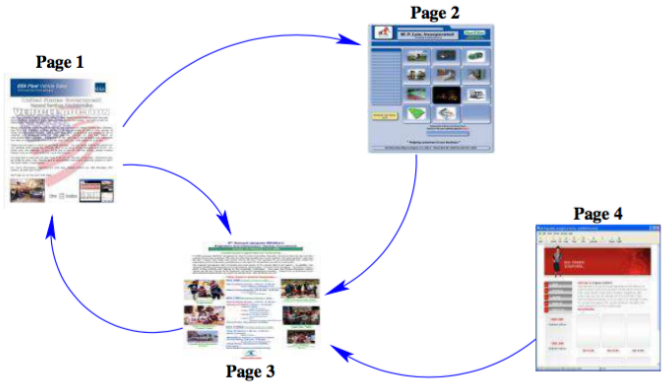
$$\begin{aligned} \mathbf{p} &= \left((1-\lambda)\mathbf{1}\mathbf{1}^T/n + \lambda \mathbf{L}\mathbf{D}_c^{-1}\right)\mathbf{p} \\ &=: \mathbf{A}\mathbf{p} \end{aligned}$$

$\mathbf{p}$ is an eigenvector of $\mathbf{A}$ with eigenvalue 1! And by the properties stochastic matrices, it corresponds to the *largest* eigenvalue

Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Set arbitrary normalization: $\mathbf{1}^T\mathbf{p} = n$ so that

$$\mathbf{p} = \left((1-\lambda)\mathbf{1}\mathbf{1}^T/n + \lambda \mathbf{L}\mathbf{D}_c^{-1}\right)\mathbf{p}$$
$$=: \mathbf{A}\mathbf{p}$$

$\mathbf{p}$ is an eigenvector of $\mathbf{A}$ with eigenvalue 1! And by the properties stochastic matrices, it corresponds to the *largest* eigenvalue

Solve using power method:

$$\mathbf{p}_{k+1} = \frac{\mathbf{A}\mathbf{p}_k}{\mathbf{1}^T\mathbf{A}\mathbf{p}_k/n} \qquad \mathbf{p}_0 \sim \text{uniform}([0,1]^n)$$

Matrix completion

Given historical data on how users rated movies in past:

NETFLIX

17,700 movies, 480,189 users, 99,072,112 ratings (Sparsity: 1.2%)

Predict how the same users will rate movies in the future (for $1 million prize)

Matrix completion

n movies, m users, $|S|$ ratings

$$\underset{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}}{\arg\min} \sum_{(i,j,s) \in \mathcal{S}} ||(UV^T)_{i,j} - s_{i,j}||_2^2$$

How do we solve it? With full information?

Matrix completion

n movies, m users, $|S|$ ratings

$$\underset{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}}{\arg\min} \sum_{(i,j,s) \in \mathcal{S}} ||(UV^T)_{i,j} - s_{i,j}||_2^2$$

Random projections

PCA finds a low-dimensional representation that reduces population variance

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})||^2.$$

$\mathbf{V}_q \mathbf{V}_q^T$ is a *projection matrix* that minimizes error in basis of size q

$\mathbf{V}_q$ are the first q eigenvectors of Σ

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T$$

But what if I care about the reconstruction of the *individual* points?

$$\min_{\mathbf{W}_q} \max_{i=1,\ldots,n} ||(x_i - \bar{x}) - \mathbf{W}_q \mathbf{W}_q^T (x_i - \bar{x})||^2$$

Random projections

$$\min_{\mathbf{W}_q} \max_{i=1,\ldots,n} ||(x_i - \bar{x}) - \mathbf{W}_q \mathbf{W}_q^T (x_i - \bar{x})||^2$$

Johnson-Lindenstrauss (1983)

Theorem 1.1. *(Johnson-Lindenstrauss) Let* $\epsilon \in (0, 1/2)$. *Let* $Q \subset \mathbb{R}^d$ *be a set of* n *points and* $k = \frac{20 \log n}{\epsilon^2}$. *There exists a Lipshcitz mapping* $f : \mathbb{R}^d \to \mathbb{R}^k$ *such that for all* $u, v \in Q$:

(independent of d)

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2$$

Random projections

$$\min_{\mathbf{W}_q} \max_{i=1,\ldots,n} ||(x_i - \bar{x}) - \mathbf{W}_q \mathbf{W}_q^T (x_i - \bar{x})||^2$$

Johnson-Lindenstrauss (1983)

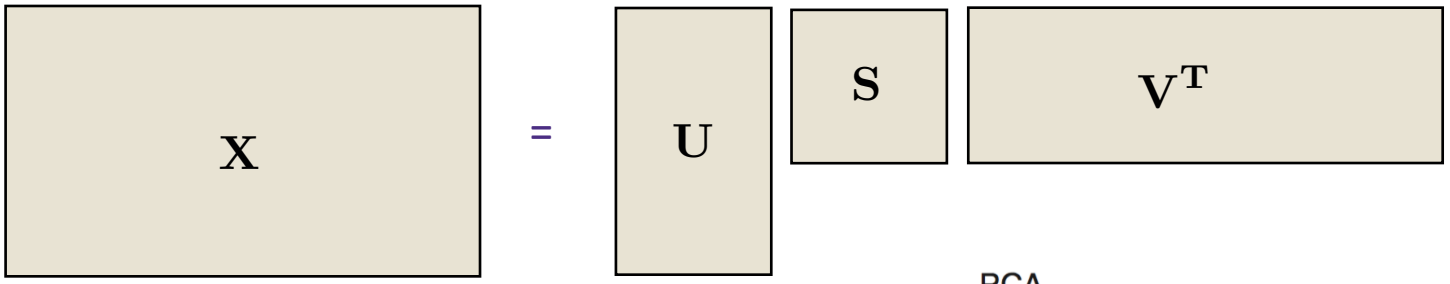
Theorem 1.1. *(Johnson-Lindenstrauss) Let* $\epsilon \in (0, 1/2)$. *Let* $Q \subset \mathbb{R}^d$ *be a set of* n *points and* $k = \frac{20 \log n}{\epsilon^2}$. *There exists a Lipshcitz mapping* $f : \mathbb{R}^d \to \mathbb{R}^k$ *such that for all* $u, v \in Q$: (independent of d)

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2$$

Theorem 1.2. *(Norm preservation) Let* $x \in \mathbb{R}^d$. *Assume that the entries in* $A \subset \mathbb{R}^{k \times d}$ *are sampled independently from* $N(0, 1)$. *Then,*

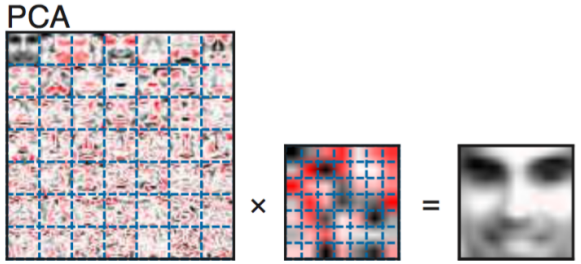
$$\Pr((1 - \epsilon)\|x\|^2 \leq \|\frac{1}{\sqrt{k}} Ax\|^2 \leq (1 + \epsilon)\|x\|^2) \geq 1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}$$

Other matrix factorizations



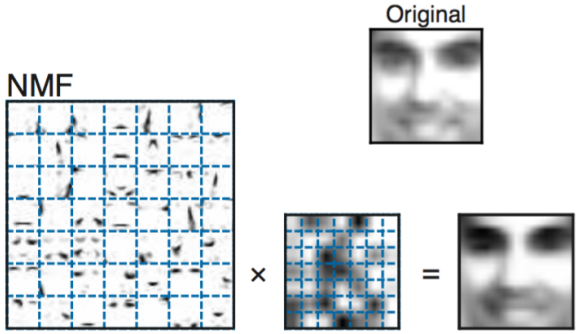
Singular value decomposition

Elements of $\mathbf{U}, \mathbf{S}, \mathbf{V}$ in $\mathbb{R}$



Nonnegative matrix factorization (NMF)

Elements of $\mathbf{U}, \mathbf{S}, \mathbf{V}$ in $\mathbb{R}_+$

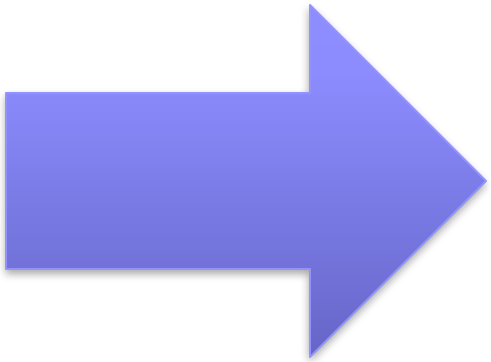


Clustering
K-means

©Kevin Jamieson 2018

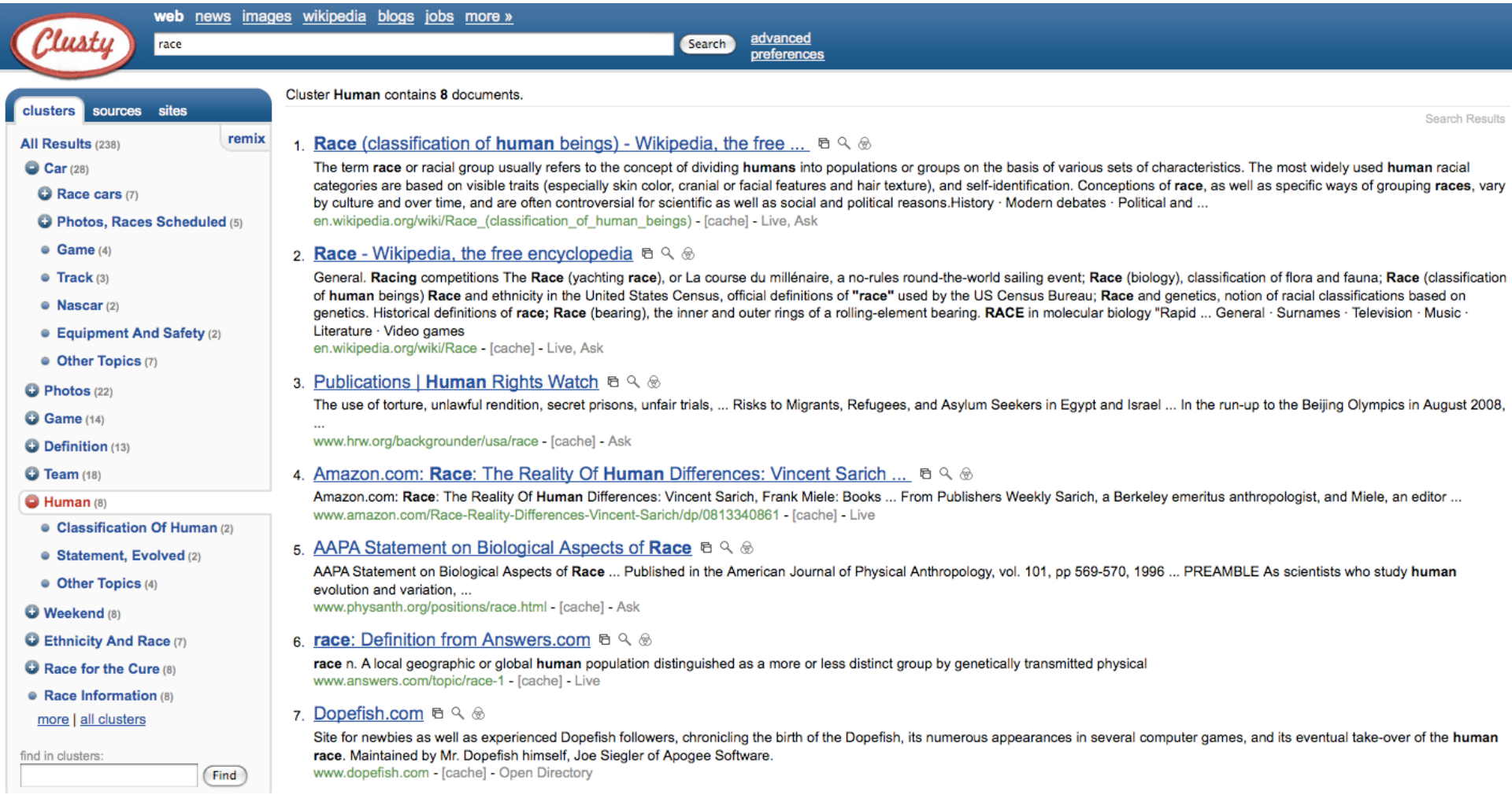
Clustering images

Set of Images

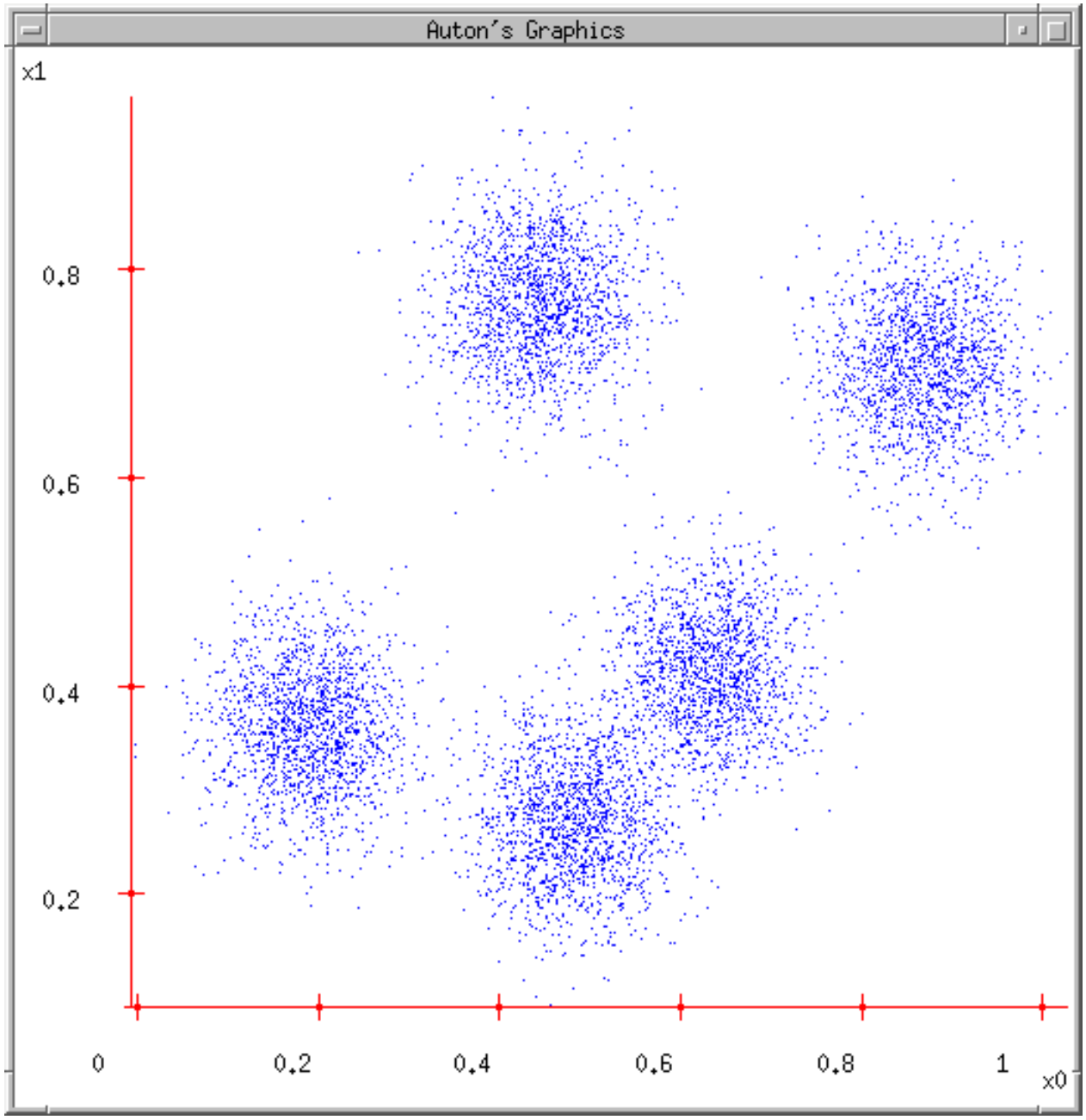


[Goldberger et al.]

Clustering web search results

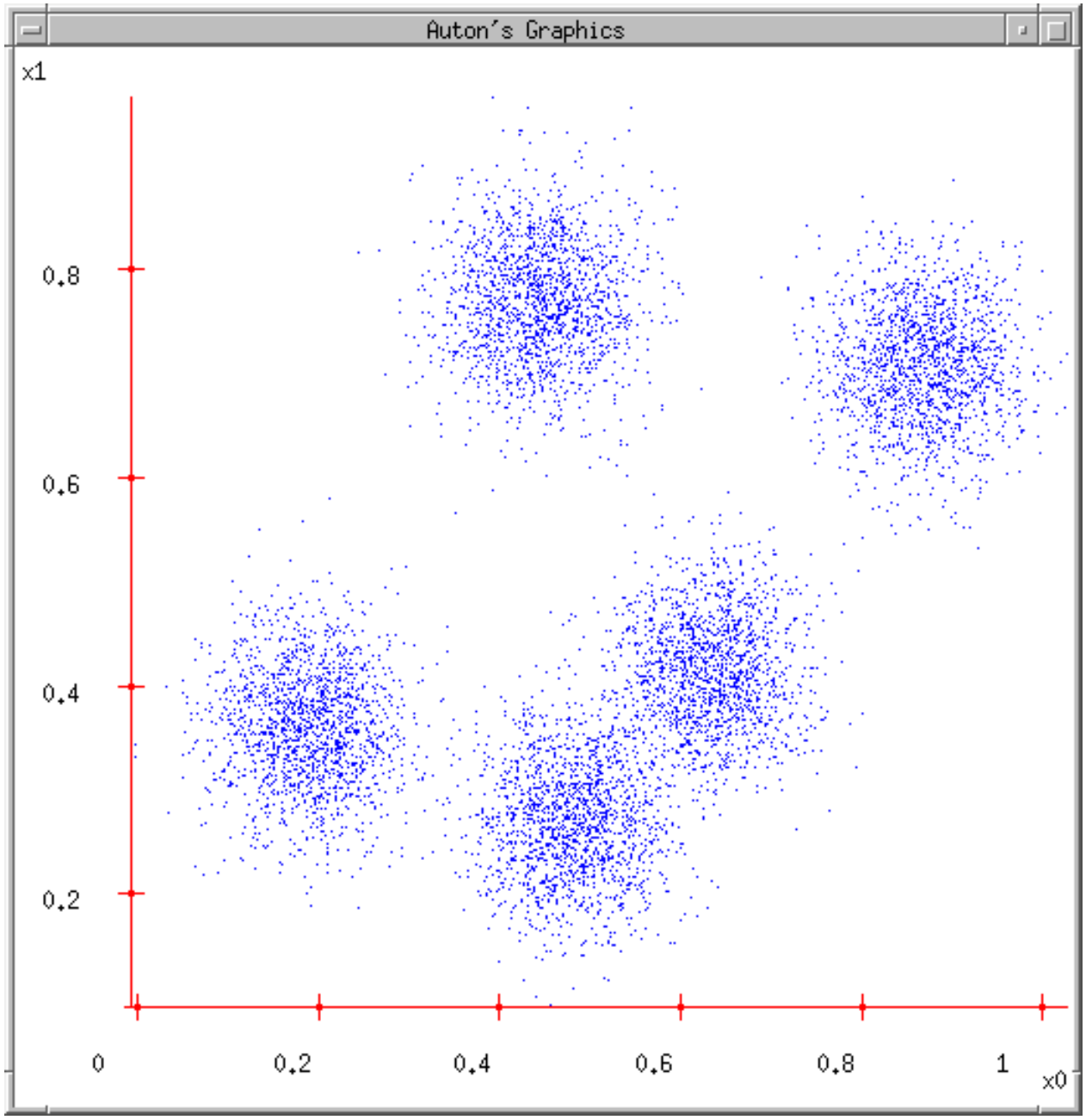


Some Data



Clustering

1. Ask user how many clusters they'd like.
   (e.g. k=5)
2. Pick clusters to minimize some objective fn.



Clustering

1. Fix a # of clusters *(e.g. k=5)*
2. Choose/Assign each point x_j to $C(j) \in \{1, \ldots, k\}$
    1. Sometimes, pick centers $\mu_1, \ldots \mu_k$

    To minimize

$$F(\mu, C)$$

K-means refers to optimizing this objective:

$$F(\mu, C) = \sum_{j=1}^{m} ||\mu_{C(j)} - x_j||^2$$

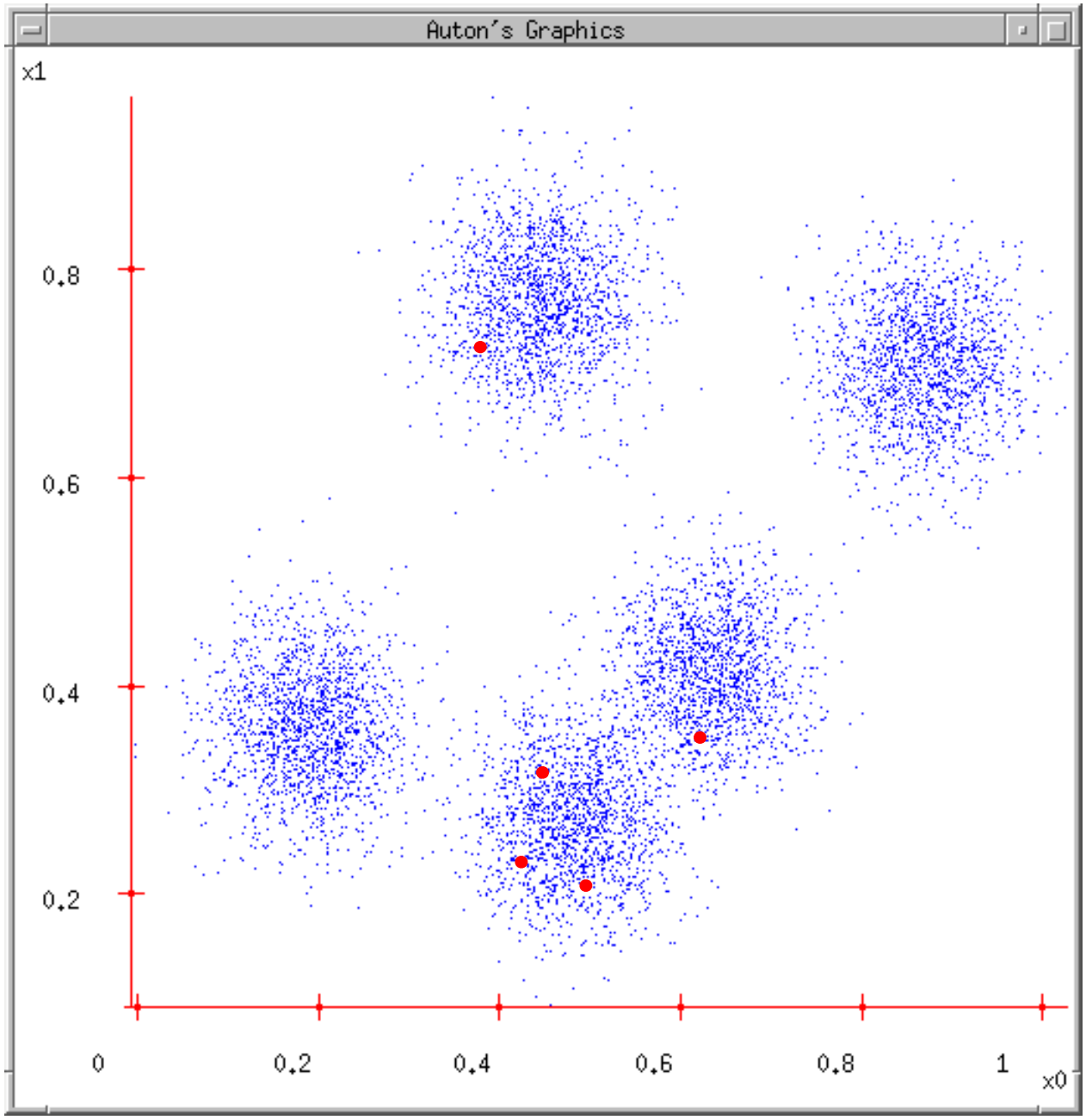
How to minimize this quantity?

NP-Hard to minimize exactly. :(

But, several natural algorithms work well in practice!

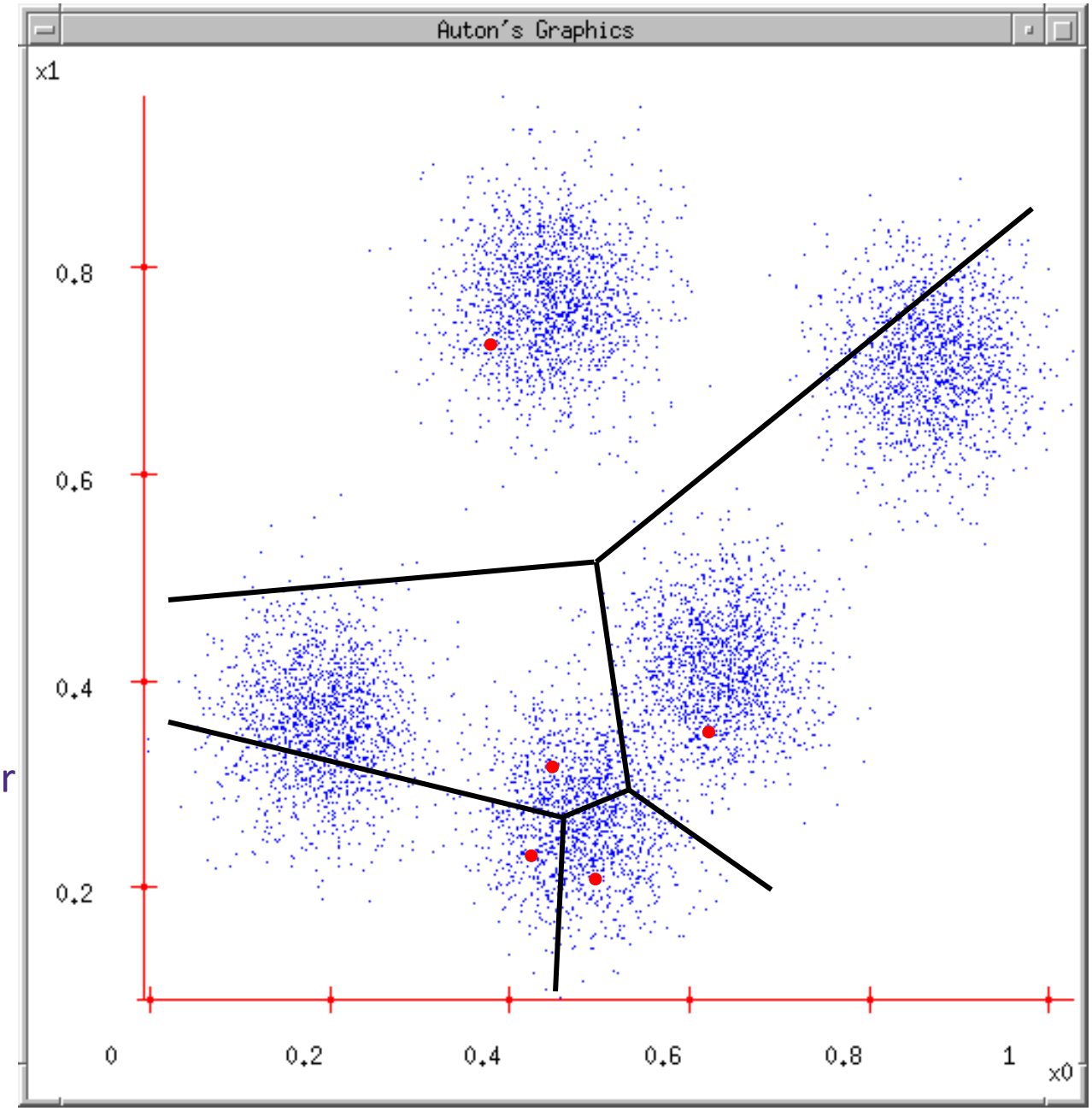
Lloyd's algorithm

1. Ask user how many clusters they'd like.
   (e.g. k=5)
2. Randomly guess k cluster Center locations



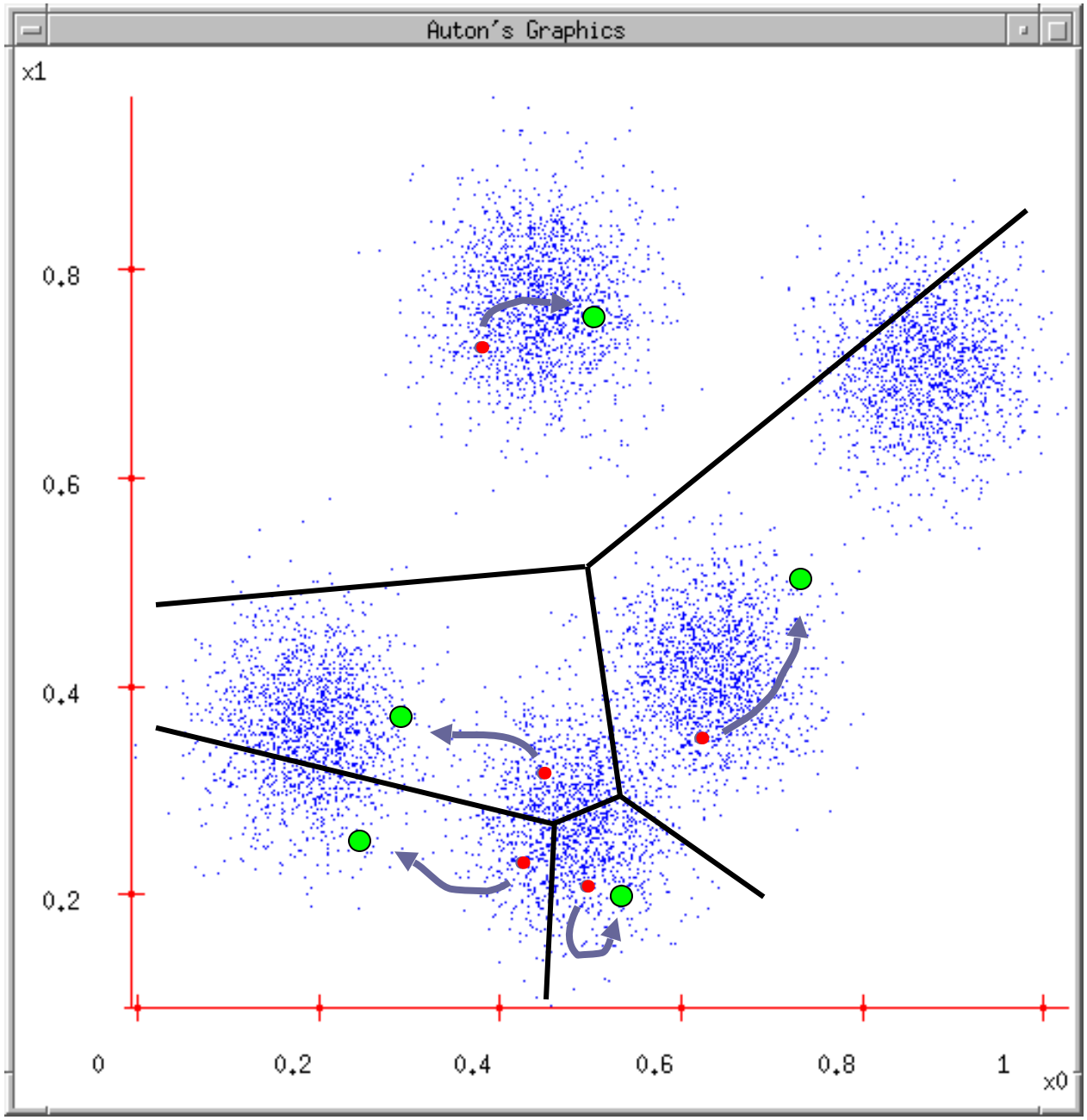
Lloyd's algorithm

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Each center "owns" a set of datapoints)



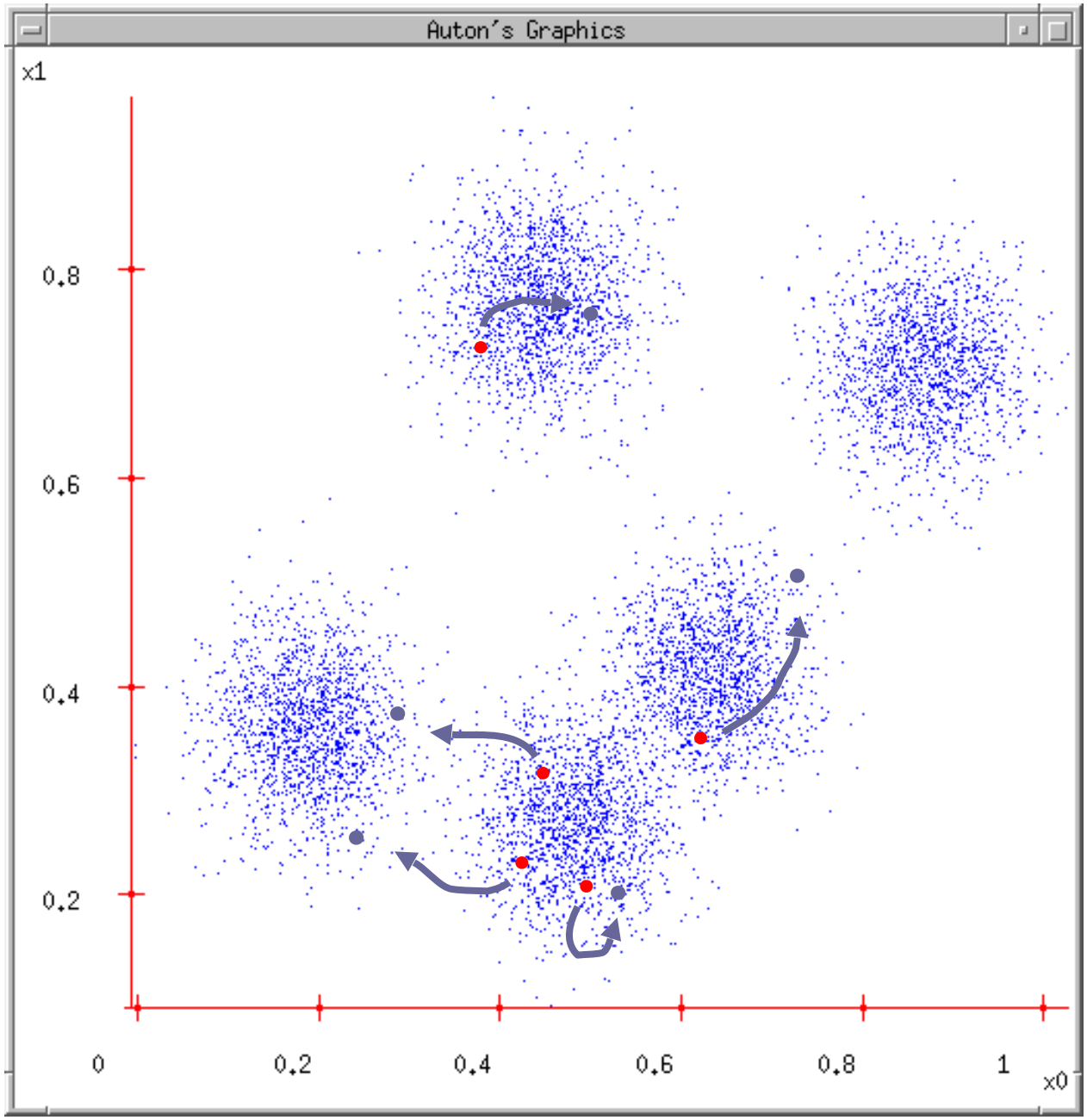
Lloyd's algorithm

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



Lloyd's algorithm

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



Lloyd's algorithm

$$F(\mu, C) = \sum_{j=1}^{m} ||\mu_{C(j)} - x_j||^2$$

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!

$$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

$$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$

Does Lloyd's algorithm converge??? Part 1

$$\min_{\mu} \min_{C} F(\mu, C) = \min_{\mu} \min_{C} \sum_{i=1}^{k} \sum_{j:C(j)=i} ||\mu_i - x_j||^2$$

First, fix $\boldsymbol{\mu}$

minimize w.r.t C

Does Lloyd's algorithm converge??? Part 2

$$\min_{\mu} \min_{C} F(\mu, C) = \min_{\mu} \min_{C} \sum_{i=1}^{k} \sum_{j:C(j)=i} ||\mu_i - x_j||^2$$

First, fix $\boldsymbol{\mu}$
minimize w.r.t C

Then, fix C
minimize w.r.t $\boldsymbol{\mu}$

$F(\mu, C)$ decreases each step $\Rightarrow$ the algorithm doesn't cycle
Only $\binom{n}{k} \approx n^k$ configuarations $\Rightarrow$ converges in finite # iterations

A cool application of k-means clustering: compression

Vector Quantization, Fisher Vectors

Vector Quantization (for compression)

1. Represent image as grid of patches
2. Run k-means on the patches to build code book
3. Represent each patch as a code word.

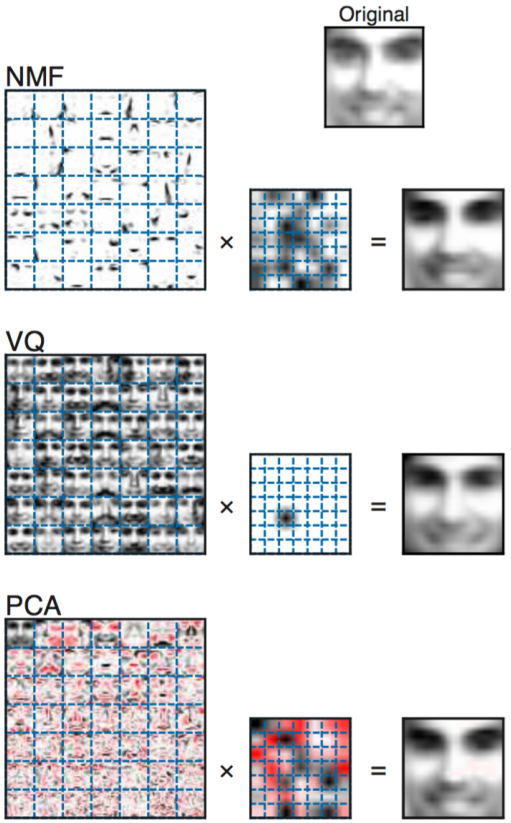
FIGURE 14.9. *Sir Ronald A. Fisher (1890 – 1962) was one of the founders of modern day statistics, to whom we owe maximum-likelihood, sufficiency, and many other fundamental concepts. The image on the left is a 1024×1024 grayscale image at 8 bits per pixel. The center image is the result of 2 × 2 block VQ, using 200 code vectors, with a compression rate of 1.9 bits/pixel. The right image uses only four code vectors, with a compression rate of 0.50 bits/pixel*

Vector Quantization, Fisher Vectors

Vector Quantization (for compression)

1. Represent image as grid of patches
2. Run k-means on the patches to build code book
   (k = # of codewords, center is code!)
3. Represent each patch as a code word.

FIGURE 14.9. *Sir Ronald A. Fisher (1890 – 1962) was one of the founders of modern day statistics, to whom we owe maximum-likelihood, sufficiency, and many other fundamental concepts. The image on the left is a 1024×1024 grayscale image at 8 bits per pixel. The center image is the result of 2 × 2 block VQ, using 200 code vectors, with a compression rate of 1.9 bits/pixel. The right image uses only four code vectors, with a compression rate of 0.50 bits/pixel*

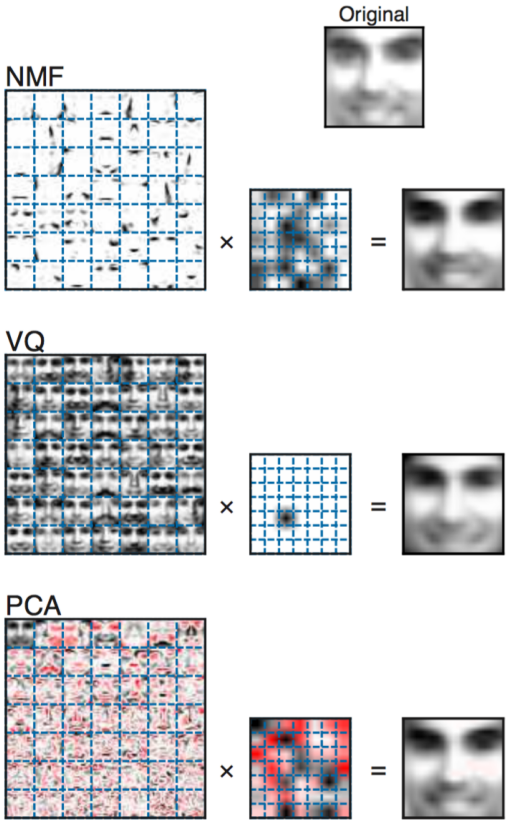


Vector Quantization, Fisher Vectors

Vector Quantization (for compression)

1. Represent image as grid of patches
2. Run k-means on the patches to build code book
   (k = # of codewords, center is code!)
3. Represent each patch as a code word.

FIGURE 14.9. *Sir Ronald A. Fisher (1890 – 1962) was one of the founders of modern day statistics, to whom we owe maximum-likelihood, sufficiency, and many other fundamental concepts. The image on the left is a 1024×1024 grayscale image at 8 bits per pixel. The center image is the result of 2 × 2 block VQ, using 200 code vectors, with a compression rate of 1.9 bits/pixel. The right image uses only four code vectors, with a compression rate of 0.50 bits/pixel*



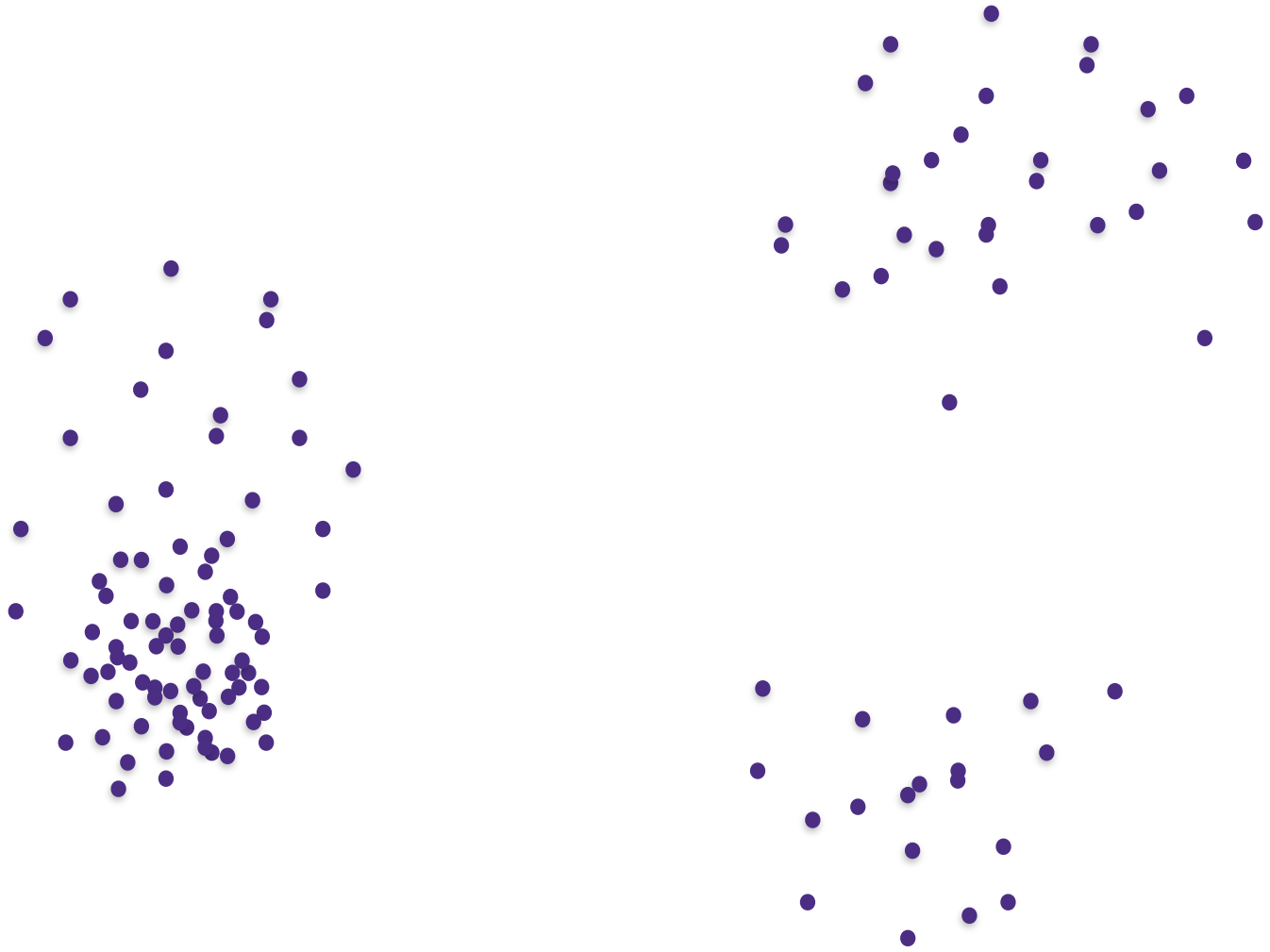
Coates, Ng, *Learning Feature Representations with K-means*, 2012

When to use K-means, or something else?

What sort of groupings are desired?

- Nonoverlapping, similar diameter clusters: k-means may work well
- Otherwise, might want another objective function (spectral, k-median, k-mode, …)

One bad case for k-means



K-means summary

A clustering objective

- minimize average L2 distance to centers of clusters

Lloyd's algorithm: a greedy heuristic for minimizing it

- Will converge in finite time, may not find global minimum

Good for finding similar width, nonoverlapping clusters

Sensitive to initial center selection, and random may not be the best a priori

- See k-means++, *The Advantages of Careful Seeding,* Arthur and Vassilvitskii

Expectation Maximization: an algorithmic template

Recall Lloyd's algorithm

$$F(\mu, C) = \sum_{j=1}^{m} ||\mu_{C(j)} - x_j||^2$$

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it is closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!

$$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

$$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$

Recall Lloyd's algorithm

$$F(\mu, C) = \sum_{j=1}^{m} ||\mu_{C(j)} - x_j||^2$$

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it is closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!

Fixing centers, assign points to "most probable" cluster

$$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

$$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$

Fixing assignment, compute "most likely" center

Recall Lloyd's algorithm

$$F(\mu, C) = \sum_{j=1}^{m} ||\mu_{C(j)} - x_j||^2$$

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it is closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!

Expectation: Fixing centers, assign points to "most probable" cluster

$$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

$$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$

Maximization: Fixing assignment, compute "most likely" center

What likelihood function?

There are k truncated Gaussians each generating data

Expectation: Fixing centers, assign points to "most probable" cluster

$$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

$$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$

Maximization: Fixing assignment, compute "most likely" center

The Expectation Maximization template

Expectation:

Fix parameters, estimate unobserved data

Maximization:

Fix unobserved data, find MLE for parameters

Expectation: Fixing centers, assign points to "most probable" cluster

$$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

$$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$

Maximization: Fixing assignment, compute "most likely" center

Why use this template?

Expectation:

Fix parameters,
estimate unobserved
data

Maximization:

Fix unobserved data,
find MLE for parameters

Usually, the joint optimization problem is hard to solve (e.g., finding the global optimum to k-means)