ColumbiaX: Machine Learning

Lecture 19

Prof. John Paisley

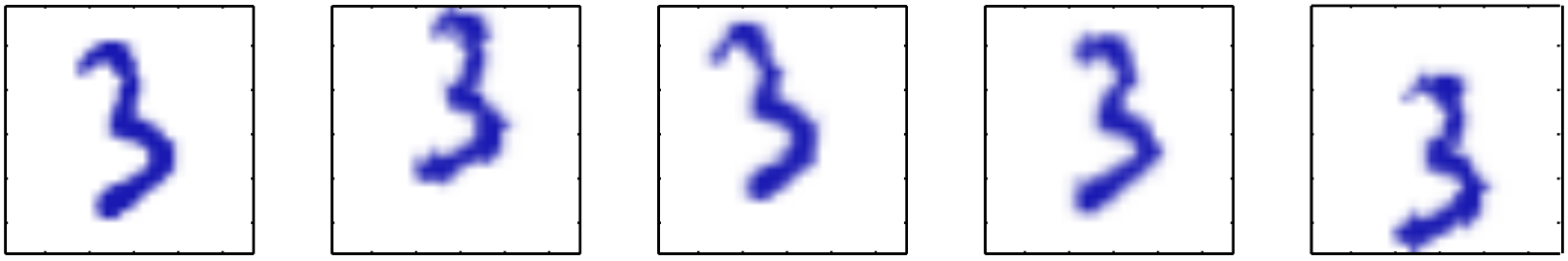
Department of Electrical Engineering
& Data Science Institute

Columbia University

Principal Component Analysis

DIMENSIONALITY REDUCTION

We're given data $x_1, \ldots, x_n$, where $x \in \mathbb{R}^d$. This data is often high-dimensional, but the "information" doesn't use the full d dimensions.



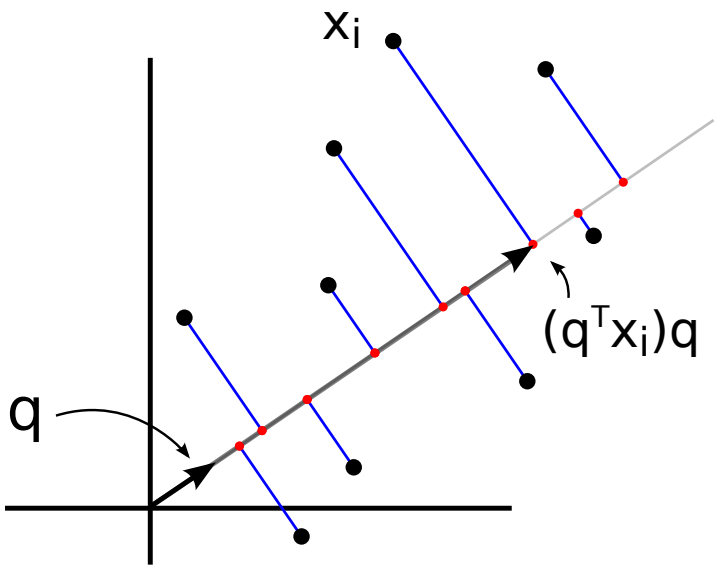
For example, we could represent the above images with three numbers since they have three degrees of freedom. Two for shifts and a third for rotation.

Principal component analysis can be thought of as a way of automatically mapping data x_i into some new low-dimensional coordinate system.

- It capture most of the information in the data in a few dimensions
- Extensions allow us to handle missing data, and "unwrap" the data.

PRINCIPAL COMPONENT ANALYSIS

Example: How can we approximate this data using a unit-length vector q?



q is a unit-length vector, $q^T q = 1$.

Red dot: The length, $q^T x_i$, to the axis after projecting x onto the line defined by q.

The vector $(q^T x_i)q$ takes q and stretches it to the corresponding red dot.

So what's a good q? How about minimizing the squared approximation error,

$$q = \arg\min_q \sum_{i=1}^{n} \|x_i - qq^T x_i\|^2 \qquad \text{subject to} \quad q^T q = 1$$

$qq^T x_i = (q^T x_i)q$: The approximation of x_i by stretching q to the "red dot."

PCA : THE FIRST PRINCIPAL COMPONENT

This is related to the problem of finding the largest eigenvalue,

$$\begin{aligned} q &= \arg\min_q \sum_{i=1}^{n} \|x_i - qq^T x_i\|^2 \qquad \text{s.t.} \quad q^T q = 1 \\ &= \arg\min_q \sum_{i=1}^{n} x_i^T x_i - q^T \underbrace{\left(\sum_{i=1}^{n} x_i x_i^T \right)}_{= XX^T} q \end{aligned}$$

We've defined $X = [x_1, \ldots, x_n]$. Since the first term doesn't depend on q and we have a negative sign in front of the second term, equivalently we solve

$$q = \arg\max_q \; q^T (XX^T) q \qquad \text{subject to} \quad q^T q = 1$$

This is the eigendecomposition problem:

- q is the first eigenvector of XX^T
- $\lambda = q^T (XX^T) q$ is the first eigenvalue

PCA: General

The general form of PCA considers K eigenvectors,

$$q = \arg\min_q \sum_{i=1}^{n} \|x_i - \underbrace{\sum_{k=1}^{K}(x_i^T q_k)q_k}_{\text{approximates } x}\|^2 \qquad \text{s.t. } q_k^T q_{k'} = \begin{cases} 1, & k = k' \\ 0, & k \neq k' \end{cases}$$

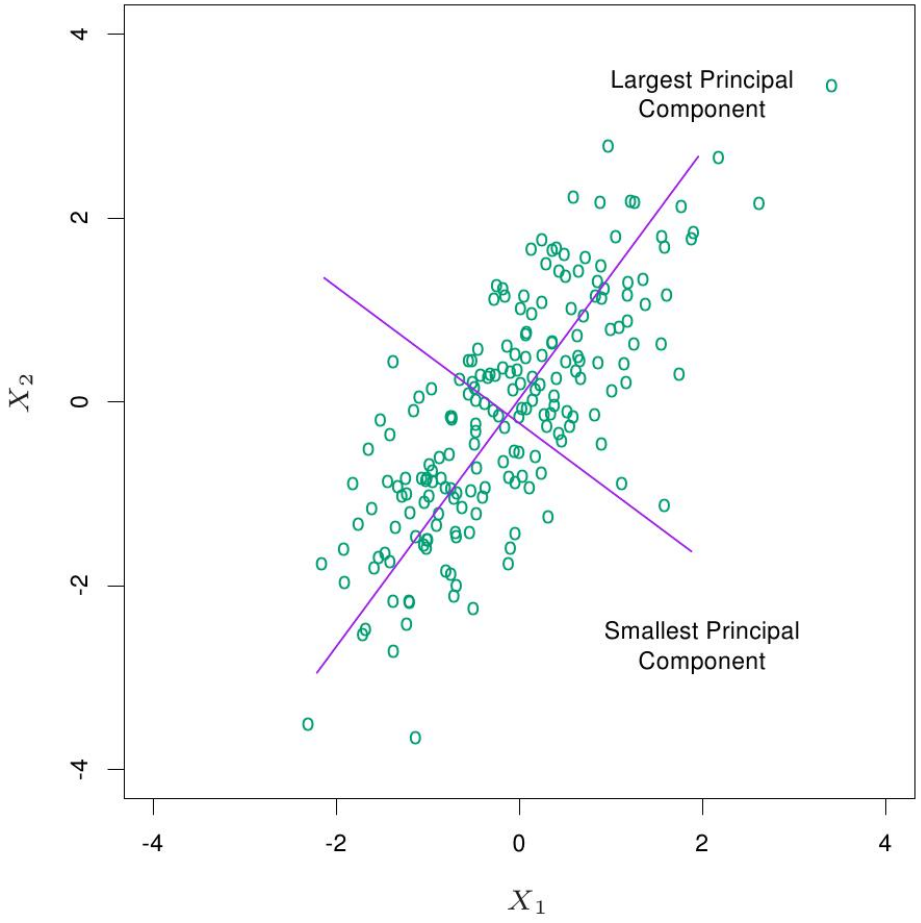
$$= \arg\min_q \sum_{i=1}^{n} x_i^T x_i - \sum_{k=1}^{K} q_k^T \underbrace{\left(\sum_{i=1}^{n} x_i x_i^T\right)}_{= XX^T} q_k$$

The vectors in $Q = [q_1, \ldots, q_K]$ give us a K dimensional subspace with which to represent the data:

$$x_{\text{proj}} = \begin{bmatrix} q_1^T x \\ \vdots \\ q_K^T x \end{bmatrix}, \qquad x \approx \sum_{k=1}^{K}(q_k^T x)q_k = Q x_{\text{proj}}$$

The eigenvectors of (XX^T) can be learned using built-in software.

EIGENVALUES, EIGENVECTORS AND THE SVD



An equivalent formulation of the problem is to find (λ, q) such that

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

Since (XX^T) is a PSD matrix, there are $r \leq \min\{d, n\}$ pairs,

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0,$$

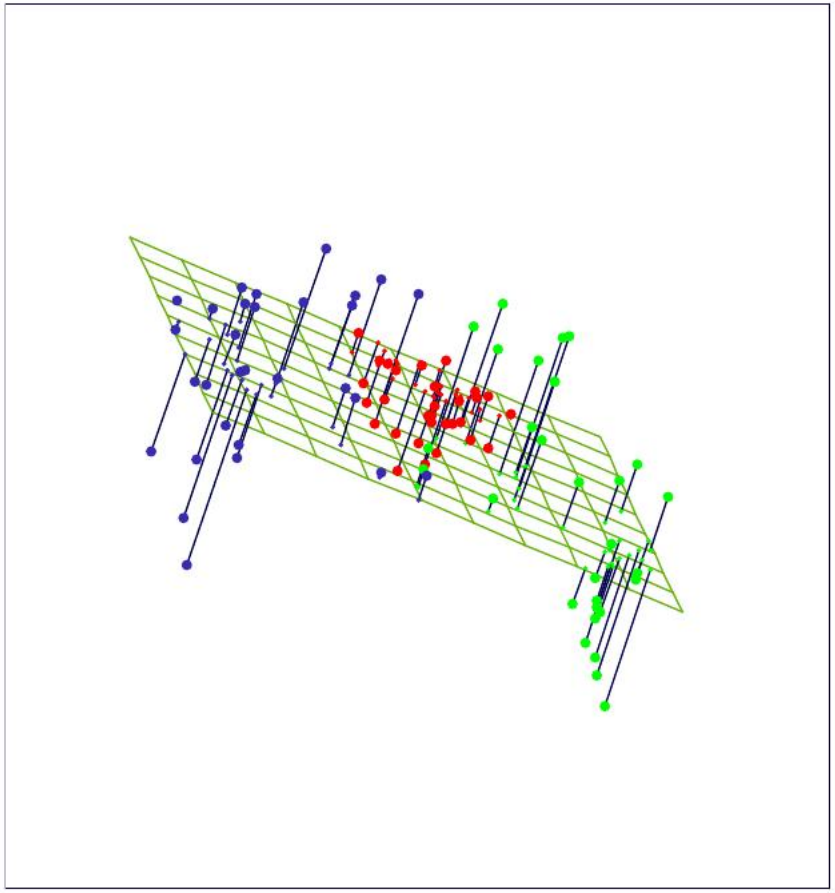
$$q_k^T q_k = 1, \quad q_k^T q_{k'} = 0$$

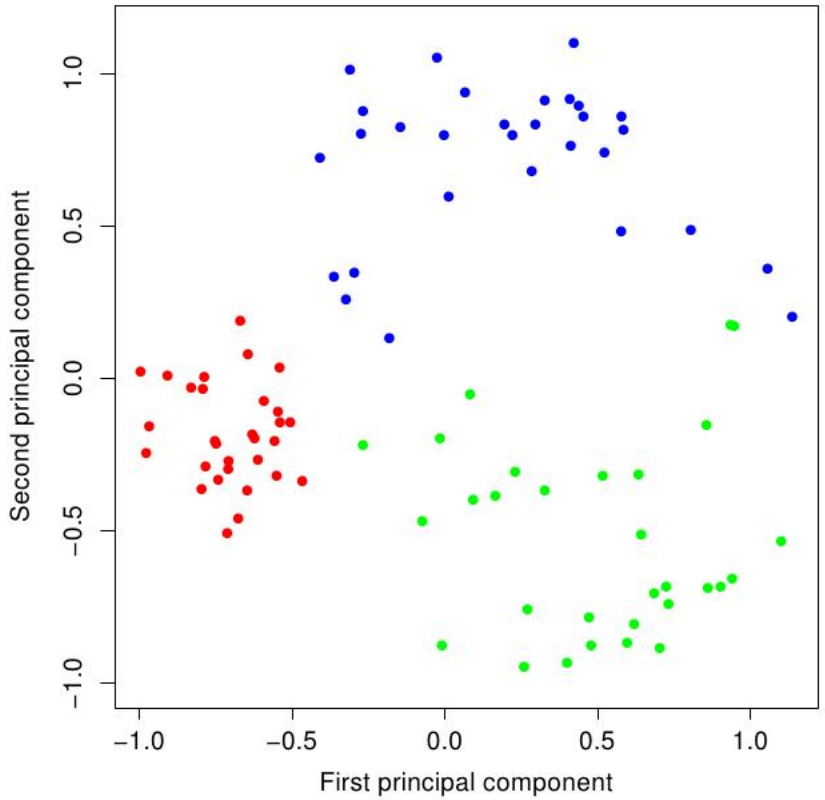
Why is (XX^T) PSD? Using the SVD, $X = USV^T$, we have that

$$(XX^T) = US^2U^T \quad \Rightarrow \quad Q = U, \quad \lambda_i = (S^2)_{ii} \geq 0$$

Preprocessing: Usually we first subtract off the mean of each dimension of x.

PCA: Example of projecting from $\mathbb{R}^3$ to $\mathbb{R}^2$





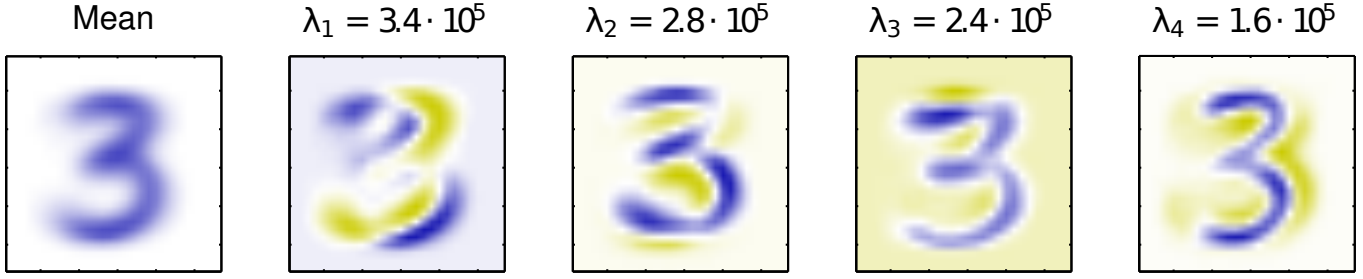
For this data, most information (structure in the data) can be captured in $\mathbb{R}^2$.

(left) The original data in $\mathbb{R}^3$. The hyperplane is defined by q_1 and q_2.

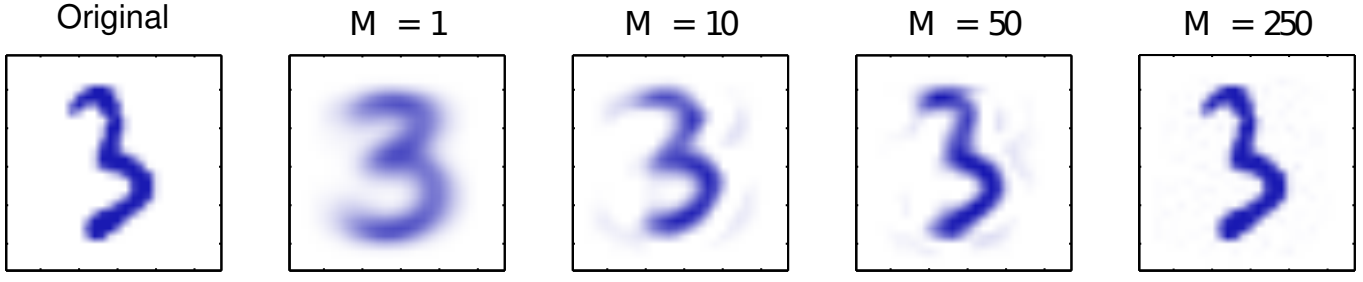
(right) The new coordinates for the data: $x_i \rightarrow x_i^{proj} = \begin{bmatrix} x_i^T q_1 \\ x_i^T q_2 \end{bmatrix}$.

EXAMPLE: DIGITS

Data: 16×16 images of handwritten 3's (as vectors in $\mathbb{R}^{256}$)



Above: The first four eigenvectors q and their eigenvalues λ.



Above: Reconstructing a 3 using the first $M-1$ eigenvectors plus the mean, and approximation

$$x \approx \text{mean} + \sum_{k=1}^{M-1} (x^T q_k) q_k$$

PROBABILISTIC PCA

PCA AND THE SVD

We've discussed how any matrix X has a singular value decomposition,

$$X = USV^T, \quad U^T U = I, \quad V^T V = I$$

and S is a diagonal matrix with non-negative entries.

Therefore,

$$XX^T = US^2U^T \quad \Leftrightarrow \quad (XX^T)U = US^2$$

U is a matrix of eigenvectors, and S^2 is a diagonal matrix of eigenvalues.

A MODELING APPROACH TO PCA

Using the SVD perspective of PCA, we can also derive a probabilistic model for the problem and use the EM algorithm to learn it.

This model will have the advantages of:

- Handling the problem of missing data
- Allowing us to learn additional parameters such as noise
- Provide a framework that could be extended to more complex models
- Gives distributions used to characterize uncertainty in predictions
- etc.

PROBABILISTIC PCA

In effect, this is a new matrix factorization model.

- With the SVD, we had $X = USV^T$.
- We now approximate $X \approx WZ$, where
  - W is a $d \times K$ matrix. In different settings this is called a "factor loadings" matrix, or a "dictionary." It's like the eigenvectors, but no orthonormality.
  - The ith column of Z is called $z_i \in \mathbb{R}^K$. Think of it as a low-dimensional representation of x_i.

The generative process of Probabilistic PCA is

$$x_i \sim N(Wz_i, \sigma^2 I), \qquad z_i \sim N(0, I).$$

In this case, we don't know W or any of the z_i.

Maximum likelihood

Our goal is to find the maximum likelihood solution of the matrix W under the marginal distribution, i.e., with the z_i vectors integrated out,

$$W_{\text{ML}} = \arg\max_W \ln p(x_1, \ldots, x_n | W) = \arg\max_W \sum_{i=1}^{n} \ln p(x_i | W).$$

This is intractable because $p(x_i|W) = N(x_i|0, \sigma^2 I + WW^T)$,

$$N(x_i|0, \sigma^2 I + WW^T) = \frac{1}{(2\pi)^{\frac{d}{2}} |\sigma^2 I + WW^T|^{\frac{1}{2}}} e^{-\frac{1}{2} x^T (\sigma^2 I + WW^T)^{-1} x}$$

We can set up an EM algorithm that uses the vectors $z_1, \ldots, z_n$.

EM FOR PROBABILISTIC PCA

Setup

The marginal log likelihood can be expressed using EM as

$$\sum_{i=1}^{n} \ln \int p(x_i, z_i|W)\, dz_i = \sum_{i=1}^{n} \int q(z_i) \ln \frac{p(x_i, z_i|W)}{q(z_i)}\, dz_i \quad \leftarrow \quad \mathcal{L}$$

$$+ \sum_{i=1}^{n} \int q(z_i) \ln \frac{q(z_i)}{p(z_i|x_i, W)}\, dz_i \quad \leftarrow \quad \mathrm{KL}$$

EM Algorithm: Remember that EM has two iterated steps

1. Set $q(z_i) = p(z_i|x_i, W)$ for each i (making $\mathrm{KL} = 0$) and calculate $\mathcal{L}$
2. Maximize $\mathcal{L}$ with respect to W

Again, for this to work well we need that

- we can calculate the posterior distribution $p(z_i|x_i, W)$, and
- maximizing $\mathcal{L}$ is easy, i.e., we update W using a simple equation

EM for Probabilistic PCA

Given: Data $x_{1:n}$, $x_i \in \mathbb{R}^d$ and model $x_i \sim N(Wz_i, \sigma^2)$, $z_i \sim N(0, I)$, $z \in \mathbb{R}^K$

Output: Point estimate of W and posterior distribution on each z_i

E-Step: Set each $q(z_i) = p(z_i|x_i, W) = N(z_i|\mu_i, \Sigma_i)$ where

$$\Sigma_i = (I + W^T W/\sigma^2)^{-1}, \quad \mu_i = \Sigma_i W^T x_i/\sigma^2$$

M-Step: Update W by maximizing the objective $\mathcal{L}$ from the E-step

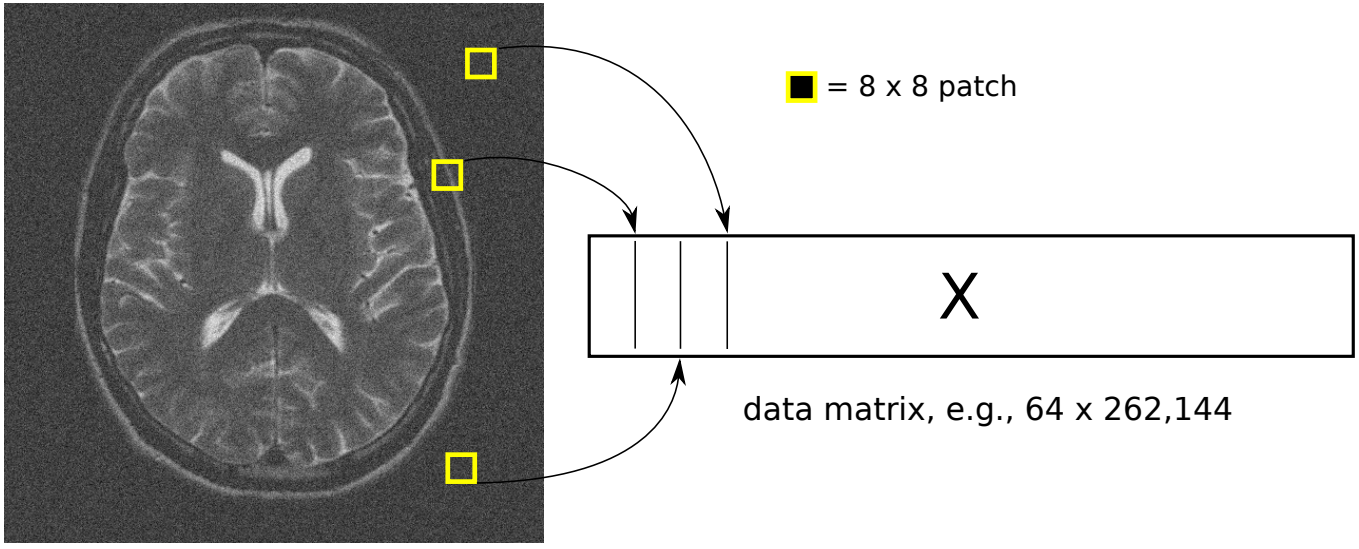
$$W = \left[\sum_{i=1}^{n} x_i \mu_i^T\right] \left[\sigma^2 I + \sum_{i=1}^{n} (\mu_i \mu_i^T + \Sigma_i)\right]^{-1}$$

Iterate E and M steps until increase in $\sum_{i=1}^{n} \ln p(x_i|W)$ is "small."

Comment:

- The probabilistic framework gives a way to learn K and σ^2 as well.

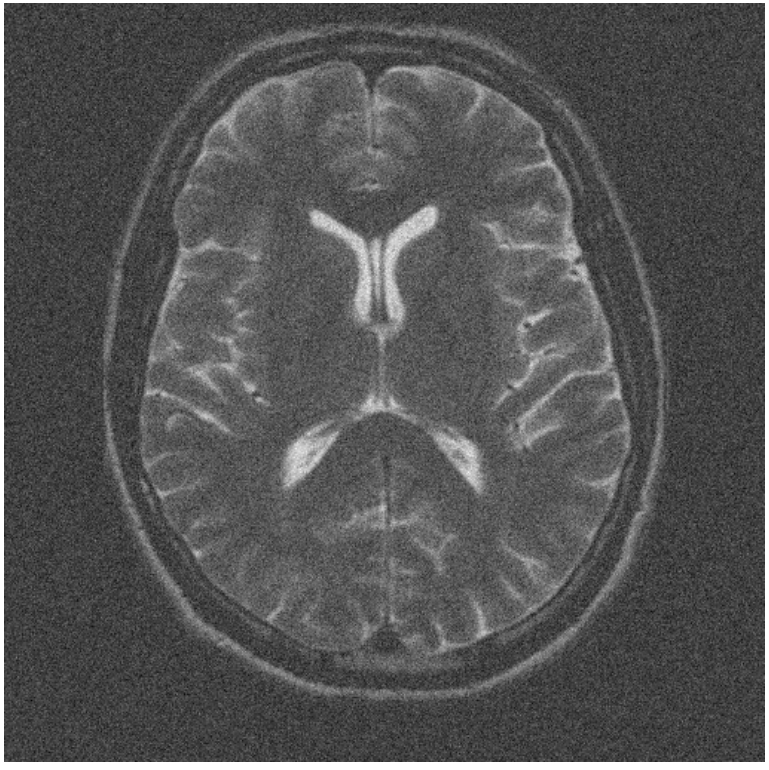
EXAMPLE: IMAGE PROCESSING

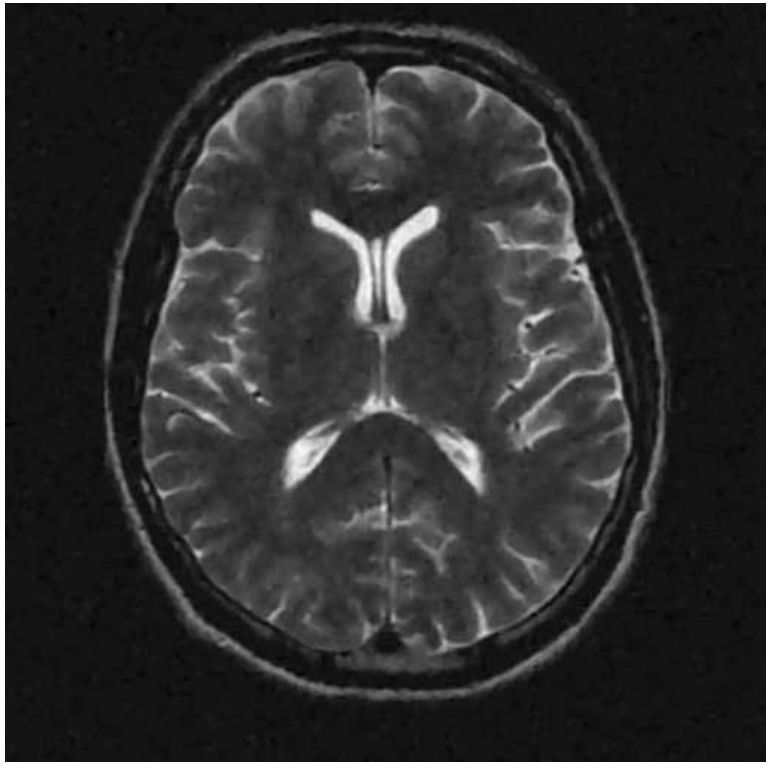


For image problems such as denoising or inpainting (missing data)

- Extract overlapping patches (e.g., 8×8) and vectorize to construct X
- Model with a factor model such as Probabilistic PCA
- Approximate $x_i \approx W\mu_i$, where μ_i is the posterior mean of z_i
- Reconstruct the image by replacing x_i with $W\mu_i$ (and averaging)

EXAMPLE: DENOISING





Noisy image on left, denoised image on right. The noise variance parameter σ^2 was learned for this example.

EXAMPLE: MISSING DATA



Another somewhat extreme example:

- Image is $480 \times 320 \times 3$ (RGB dimension)
- Throw away 80% at random
- (left) Missing data, (middle) reconstruction, (right) original image

KERNEL PCA

KERNEL PCA

We've seen how we can take an algorithm that uses dot products, $x^T x$, and generalize with a nonlinear kernel. This generalization can be made to PCA.

Recall: With PCA we find the eigenvectors of the matrix $\sum_{i=1}^{n} x_i x_i^T = XX^T$.

- Let $\phi(x)$ be a feature mapping from $\mathbb{R}^d$ to $\mathbb{R}^D$, where $D \gg d$
- We want to solve the eigendecomposition

$$\left[\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T\right] q_k = \lambda_k q_k$$

  without having to work in the higher dimensional space.

- That is, how can we do PCA without explicitly using $\phi(\cdot)$ and q?

KERNEL PCA

Notice that we can reorganize the operations of the eigendecomposition

$$\sum_{i=1}^{n} \phi(x_i) \underbrace{\left(\phi(x_i)^T q_k\right)/\lambda_k}_{=a_{ki}} = q_k$$

That is, the eigenvector $q_k = \sum_{i=1}^{n} a_{ki}\phi(x_i)$ for some vector $\boldsymbol{a}_k \in \mathbb{R}^n$.

The trick is that instead of learning q_k, we'll learn $\boldsymbol{a}_k$.

Plug this equation for q_k back into the first equation:

$$\sum_{i=1}^{N} \phi(x_i) \sum_{j=1}^{n} a_{kj} \underbrace{\phi(x_i)^T \phi(x_j)}_{=K(x_i,x_j)} = \lambda_k \sum_{i=1}^{n} a_{ki}\phi(x_i)$$

and multiply both sides by $\phi(x_l)^T$ for each $l \in \{1, \ldots, n\}$.

KERNEL PCA

When we multiply the following by $\phi(x_l)^T$ for $l = 1 \ldots, n$:

$$\sum_{i=1}^{N} \phi(x_i) \sum_{j=1}^{n} a_{kj} \underbrace{\phi(x_i)^T \phi(x_j)}_{= K(x_i, x_j)} = \lambda_k \sum_{i=1}^{n} a_{ki} \phi(x_i)$$

we get a new set of linear equations

$$K^2 \boldsymbol{a}_k = \lambda_k K \boldsymbol{a}_k \quad \Longleftrightarrow \quad K \boldsymbol{a}_k = \lambda_k \boldsymbol{a}_k$$

where K is the $n \times n$ *kernel matrix* constructed on the data.

Because K is guaranteed to be PSD because it is a matrix of dot-products, the LHS and RHS above share a solution for $(\lambda_k, \boldsymbol{a}_k)$.

Now perform "regular" PCA, but on the kernel matrix K instead of the data matrix XX^T. We summarize the algorithm on the following slide.

Kernel PCA

Given: Data $x_1, \ldots, x_n$, $x \in \mathbb{R}^d$, and a kernel function $K(x_i, x_j)$.

Construct: The kernel matrix on the data, e.g., $K_{ij} = b \exp\left\{-\frac{\|x_i - x_j\|^2}{c}\right\}$.

Solve: The eigendecomposition

$$K\boldsymbol{a}_k = \lambda_k \boldsymbol{a}_k$$

for the first $r \ll n$ eigenvector/eigenvalue pairs $(\lambda_1, \boldsymbol{a}_1), \ldots, (\lambda_r, \boldsymbol{a}_r)$.

Output: A new coordinate system for x_i by (implicitly) mapping $\phi(x_i)$ and then projecting $q_k^T \phi(x_i)$

$$x_i \quad \xrightarrow{\text{projection}} \quad \begin{bmatrix} \lambda_1 a_{1i} \\ \vdots \\ \lambda_r a_{ri} \end{bmatrix}$$

where a_{ki} is the ith dimension of the kth eigenvector $\boldsymbol{a}_k$.

KERNEL PCA AND NEW DATA

Q: How do we handle new data, x_0? Before, we could take the eigenvectors q_k and project $x_0^T q_k$, but $\boldsymbol{a}_k$ is different here.

A: Recall the relationship of $\boldsymbol{a}_k$ to q_k in kernel PCA is

$$q_k = \sum_{i=1}^{n} a_{ki} \phi(x_i).$$

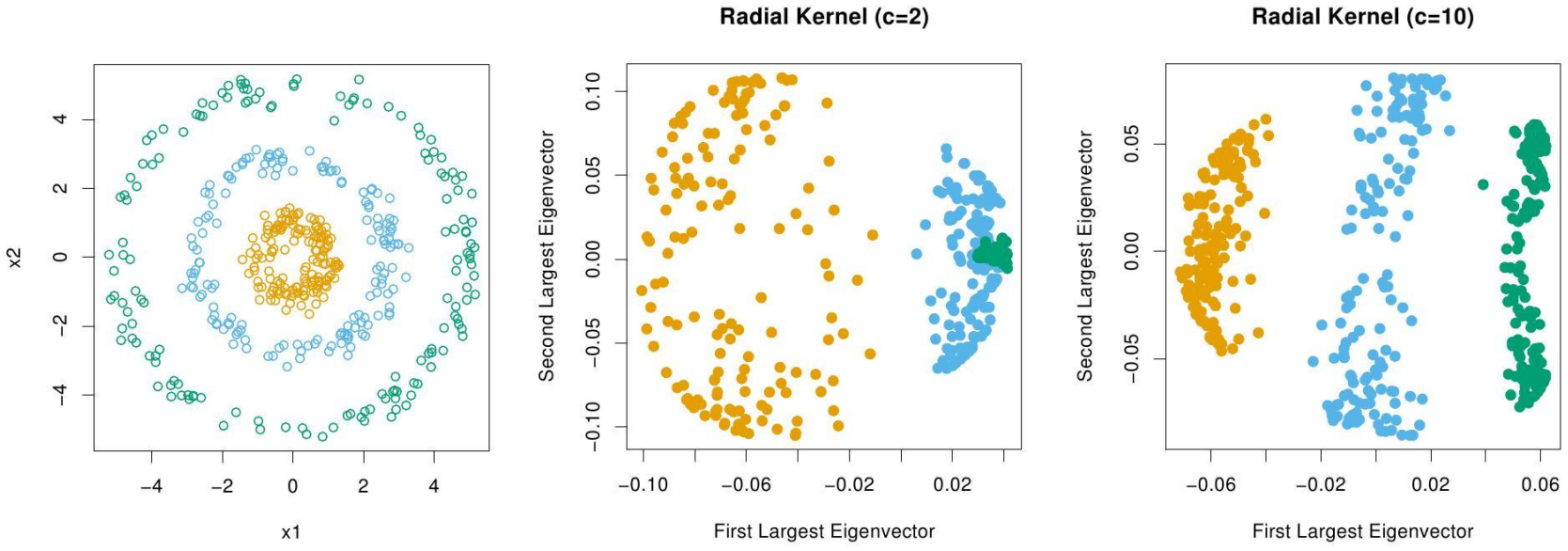
We used the "kernel trick" to avoid working with or even defining $\phi(x_i)$.

As with regular PCA, after mapping x_0 we want to project onto eigenvectors

$$x_0 \quad \xrightarrow{\text{projection}} \quad \begin{bmatrix} \phi(x_0)^T q_1 \\ \vdots \\ \phi(x_0)^T q_r \end{bmatrix}$$

Plugging in for q_k: $\quad \phi(x_0)^T q_k = \sum_{i=1}^{n} a_{ki} \phi(x_0)^T \phi(x_i) = \sum_{i=1}^{n} a_{ki} K(x_0, x_i).$

EXAMPLE RESULTS



An example of kernel PCA using the Gaussian kernel.

(left) Original data, colored for reference (but may be classes)

(middle) New coordinates using kernel width $c = 2$

(right) New coordinates using kernel width $c = 10$

Terminology: What we are doing is closely related to "spectral clustering" and can be considered an instance of "manifold learning."

ColumbiaX: Machine Learning

Lecture 20

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

SEQUENTIAL DATA

So far, when thinking probabilistically we have focused on the i.i.d. setting.

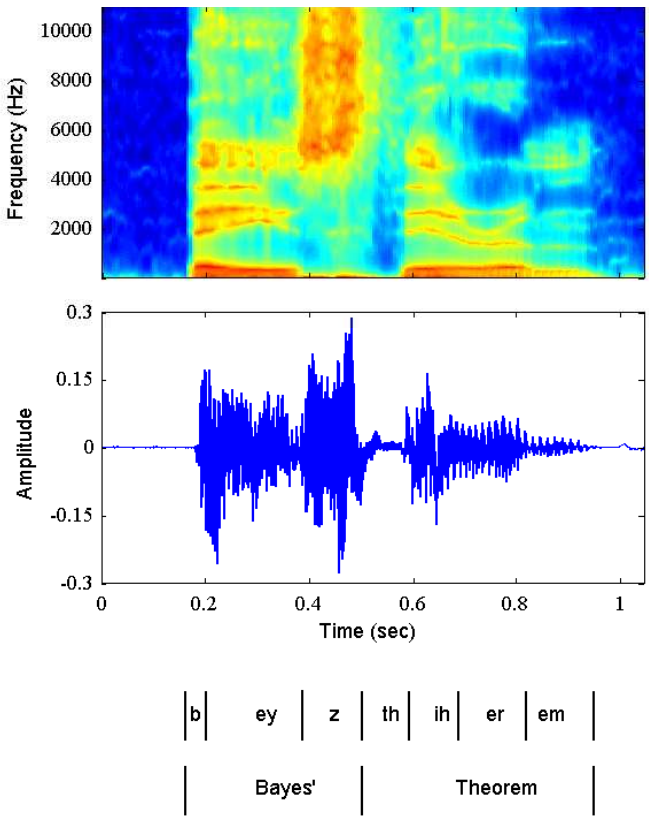
- All data are independent given a model parameter.
- This is often a reasonable assumption, but was also done for convenience.

In some applications this assumption is bad:

- Modeling rainfall as a function of hour
- Daily value of currency exchange rate
- Acoustic features of speech audio

The distribution on the next value clearly depends on the previous values.

A basic way to model sequential information is with a discrete, first-order Markov chain.



MARKOV CHAINS

EXAMPLE: ZOMBIE WALKER[1]

Imagine you see a zombie in an alley. Each time it moves forward it steps

$$(\text{left, straight, right}) \quad \text{with probability} \quad (p_l, p_s, p_r),$$

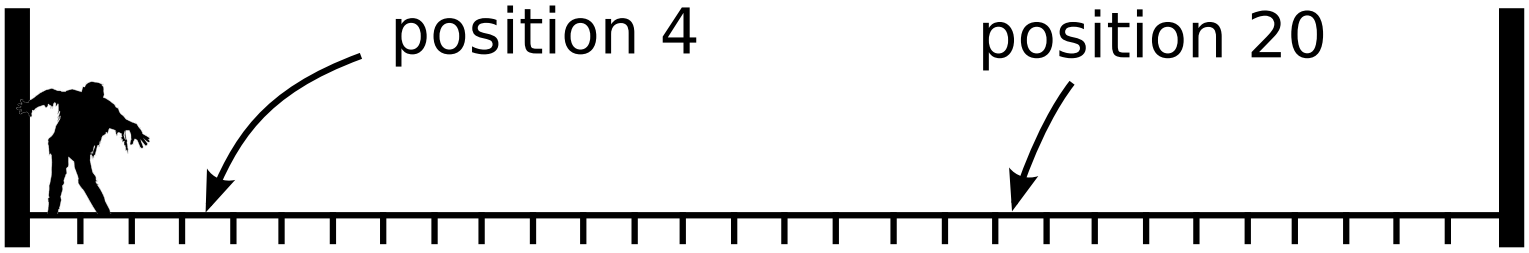
unless it's next to the wall, in which case it steps straight with probability p_s^w and toward the middle with probability p_m^w.

The distribution on the next location only depends on the current location.

[1] This problem is often introduced with a "drunk," so our maturity is textbook-level.

RANDOM WALK NOTATION

We simplify the problem by assuming there are only a finite number of positions the zombie can be in, and we model it as a random walk.



The distribution on the next position only depends on the current position. For example, for a position i away from the wall,

$$s_{t+1} \,|\, \{s_t = i\} \quad = \quad \begin{cases} i+1 & \text{w.p.} \quad p_r \\ i & \text{w.p.} \quad p_s \\ i-1 & \text{w.p.} \quad p_l \end{cases}$$

This is called the *first-order Markov property*. It's the simplest type. A second-order model would depend on the previous two positions.

MATRIX NOTATION

A more compact notation uses a matrix.

For the random walk problem, imagine we have 6 different positions, called *states*. We can write the *transition matrix* as

$$M = \begin{bmatrix} p_s^w & p_m^w & 0 & 0 & 0 & 0 \\ p_l & p_s & p_r & 0 & 0 & 0 \\ 0 & p_l & p_s & p_r & 0 & 0 \\ 0 & 0 & p_l & p_s & p_r & 0 \\ 0 & 0 & 0 & p_l & p_s & p_r \\ 0 & 0 & 0 & 0 & p_m^w & p_s^w \end{bmatrix}$$

M_{ij} is the probability that the next position is j given the current position is i.

Of course we can jumble this matrix by moving rows and columns around in a correct way, as long as we can map the rows and columns to a position.

First-order Markov chain (general)

Let $s \in \{1, \ldots, S\}$. A sequence $(s_1, \ldots, s_t)$ is a *first-order Markov chain* if

$$p(s_1, \ldots, s_t) \overset{(a)}{=} p(s_1) \prod_{u=2}^{t} p(s_u | s_1, \ldots, s_{u-1}) \overset{(b)}{=} p(s_1) \prod_{u=2}^{t} p(s_u | s_{u-1})$$

From the two equalities above:

(a) This equality is *always* true, regardless of the model (chain rule).

(b) This simplification results from the Markov property assumption.

Notice the difference from the i.i.d. assumption

$$p(s_1, \ldots, s_t) = \begin{cases} p(s_1) \prod_{u=2}^{t} p(s_u | s_{u-1}) & \text{Markov assumption} \\ \prod_{u=1}^{t} p(s_u) & \text{i.i.d. assumption} \end{cases}$$

From a modeling standpoint, this is a significant difference.

First-order Markov chain (general)

Again, we encode this more general probability distribution in a matrix:

$$M_{ij} \;=\; p(s_t = j | s_{t-1} = i)$$

We will adopt the notation that rows are distributions.

- M is a *transition matrix*, or *Markov matrix*.
- M is $S \times S$ and each row sums to one.
- M_{ij} is the probability of transitioning to state j given we are in state i.

Given a starting state, s_0, we generate a sequence $(s_1, \ldots, s_t)$ by sampling

$$s_t \,|\, s_{t-1} \;\sim\; \text{Discrete}(M_{s_{t-1},:}).$$

We can model the starting state with its own separate distribution.

MAXIMUM LIKELIHOOD

Given a sequence, we can approximate the transition matrix using ML,

$$M_{\text{ML}} = \arg\max_M \; p(s_1, \ldots, s_t | M) \;=\; \arg\max_M \sum_{u=1}^{t-1} \sum_{i,j}^{S} \mathbb{1}(s_u = i, s_{u+1} = j) \ln M_{ij}.$$

Since each row of M has to be a probability distribution, we can show that

$$M_{\text{ML}}(i,j) \;=\; \frac{\sum_{u=1}^{t-1} \mathbb{1}(s_u = i,\; s_{u+1} = j)}{\sum_{u=1}^{t-1} \mathbb{1}(s_u = i)}.$$

Empirically count how many times we observe a transition from $i \to j$ and divide by the total number of transitions from i.

Example: Model probability it rains (r) tomorrow given it rained today with observed fraction $\frac{\#\{r \to r\}}{\#\{r\}}$. Notice that $\#\{r\} = \#\{r \to r\} + \#\{r \to \textit{no-r}\}$.

PROPERTY: STATE DISTRIBUTION

Q: Can we say at the beginning what state we'll be in at step $t+1$?

A: Imagine at step t that we have a probability distribution on which state we're in, call it $p(s_t = u)$. Then the distribution on s_{t+1} is

$$p(s_{t+1} = j) = \sum_{u=1}^{S} \underbrace{p(s_{t+1} = j | s_t = u) p(s_t = u)}_{p(s_{t+1}=j,\ s_t=u)}.$$

Represent $p(s_t = u)$ with the row vector w_t (the state distribution). Then

$$\underbrace{p(s_{t+1} = j)}_{w_{t+1}(j)} = \sum_{u=1}^{S} \underbrace{p(s_{t+1} = j | s_t = u)}_{M_{uj}} \underbrace{p(s_t = u)}_{w_t(u)}.$$

We can calculate this for all j with the matrix-vector product $w_{t+1} = w_t M$. Therefore, $w_{t+1} = w_1 M^t$ and w_1 can be indicator if starting state is known.

PROPERTY: STATIONARY DISTRIBUTION

Given current state distribution w_t, the distribution on the next state is

$$w_{t+1}(j) = \sum_{u=1}^{S} M_{uj} w_t(u) \quad \iff \quad w_{t+1} = w_t M$$

What happens if we project an infinite number of steps out?

Definition: Let $w_\infty = \lim_{t\to\infty} w_t$. Then w_∞ is the *stationary distribution*.

- There are many technical results that can be proved about w_∞.
- Property: If the following are true, then w_∞ is the same vector for all w_0
  1. We can eventually reach any state starting from any other state,
  2. The sequence doesn't loop between states in a pre-defined pattern.
- Clearly $w_\infty = w_\infty M$ since w_t is converging and $w_{t+1} = w_t M$.

This last property is related to the first eigenvector of M^T:

$$M^T q_1 = \lambda_1 q_1 \quad \implies \quad \lambda_1 = 1, \quad w_\infty = \frac{q_1}{\sum_{u=1}^{S} q_1(u)}$$

A RANKING ALGORITHM

EXAMPLE: RANKING OBJECTS

We show an example of using the stationary distribution of a Markov chain to rank objects. The data are pairwise comparisons between objects.

For example, we might want to rank

- Sports teams or athletes competing against each other
- Objects being compared and selected by users

Our goal is to rank objects from "best" to "worst."

- We will construct a random walk matrix on the objects.
- The stationary distribution will give us the ranking.
- Notice: We don't consider the sequential information in the data itself. The Markov chain is an artificial modeling construct.

Example: Team rankings

Problem setup

We want to construct a Markov chain where each team is a state.

- We encourage transitions from teams that lose to teams that win.
- Predicting the "state" (i.e., team) far in the future, we can interpret a more probable state as a better team.

One specific approach to this specific problem:

- Transitions only occur between teams that play each other.
- If Team A beats Team B, there should be a high probability of transitioning from B$\rightarrow$A and small probability from A$\rightarrow$B.
- The strength of the transition can be linked to the score of the game.

EXAMPLE: TEAM RANKINGS

How about this?

Initialize $\hat{M}$ to a matrix of zeros. For a particular game, let j_1 be the index of Team A and j_2 the index of Team B. Then update

$$\hat{M}_{j_1 j_1} \leftarrow \hat{M}_{j_1 j_1} + \mathbb{1}\{\texttt{Team A wins}\} + \frac{\texttt{points}_{j_1}}{\texttt{points}_{j_1} + \texttt{points}_{j_2}},$$

$$\hat{M}_{j_2 j_2} \leftarrow \hat{M}_{j_2 j_2} + \mathbb{1}\{\texttt{Team B wins}\} + \frac{\texttt{points}_{j_2}}{\texttt{points}_{j_1} + \texttt{points}_{j_2}},$$

$$\hat{M}_{j_1 j_2} \leftarrow \hat{M}_{j_1 j_2} + \mathbb{1}\{\texttt{Team B wins}\} + \frac{\texttt{points}_{j_2}}{\texttt{points}_{j_1} + \texttt{points}_{j_2}},$$

$$\hat{M}_{j_2 j_1} \leftarrow \hat{M}_{j_2 j_1} + \mathbb{1}\{\texttt{Team A wins}\} + \frac{\texttt{points}_{j_1}}{\texttt{points}_{j_1} + \texttt{points}_{j_2}}.$$

After processing all games, let M be the matrix formed by normalizing the rows of $\hat{M}$ so they sum to 1.

EXAMPLE: 2014-2015 COLLEGE BASKETBALL SEASON

USA Today Coaches Poll

RK	TEAM	RECORD	PTS
1	Villanova (30)	35-5	750
2	North Carolina	33-7	720
3	Kansas	33-5	657
4	Oklahoma	29-8	643
5	Virginia	29-8	631
6	Oregon	31-7	596
7	Michigan State	29-6	488
8	Miami	27-8	480
9	Indiana	27-8	456
10	Syracuse	23-14	446
11	Xavier	28-6	361
12	Texas A&M	28-9	358
12	Maryland	27-9	358
14	West Virginia	26-9	331
15	Iowa State	23-12	315
16	Kentucky	27-9	297
17	Notre Dame	24-12	285
18	Duke	25-11	263
19	Purdue	26-9	184
20	Utah	27-9	170
21	Gonzaga	28-8	157
22	Arizona	25-9	154
23	Wisconsin	22-13	149
24	Baylor	22-12	105
25	Iowa	22-11	82

Markov chain ranking

RK	SCORE	TEAM
1	0.0090112	Villanova
2	0.0079282	Kansas
3	0.0074781	North Carolina
4	0.0067752	Virginia
5	0.0065791	Oklahoma
6	0.0063760	Oregon
7	0.0058095	Michigan St
8	0.0056623	Xavier OH
9	0.0055031	Miami FL
10	0.0049979	West Virginia
11	0.0047690	Utah
12	0.0047131	Kentucky
13	0.0046578	Indiana
14	0.0046482	Seton Hall
15	0.0046097	Texas A&M
16	0.0045635	Duke
17	0.0042596	Maryland
18	0.0042441	Purdue
19	0.0041866	Iowa St
20	0.0041599	St Joseph's PA
21	0.0040336	Notre Dame
22	0.0040017	Arizona
23	0.0039594	George Wash
24	0.0039369	Louisville
25	0.0039273	Providence RI

1,549 total teams

22,033 total games

SCORE = stationary distribution

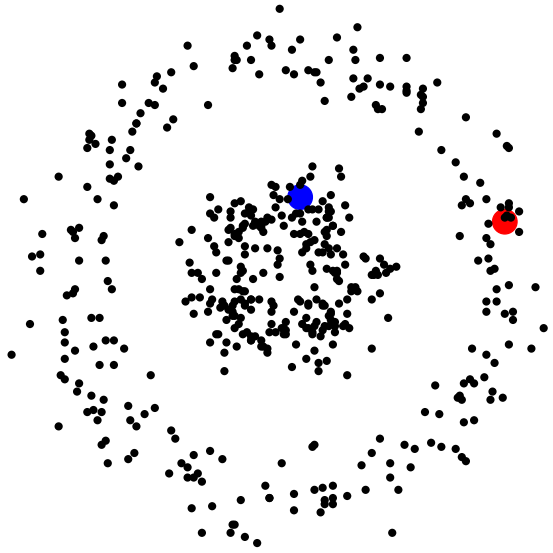
A CLASSIFICATION ALGORITHM

SEMI-SUPERVISED LEARNING

Imagine we have data with very few labels.

We want to use the structure in the dataset to help classify the unlabeled data.

We can do this with a Markov chain.



Semi-supervised learning uses partially labeled data to do classification.

- Many or most y_i will be missing in the pair (x_i, y_i).
- Still, there is structure in $x_1, \ldots, x_n$ that we don't want to throw away.
- In the example above, we might want the inner ring to be one class (blue) and the outer ring another (red).

A RANDOM WALK CLASSIFIER

We will define a classifier where, starting from any data point x_i,

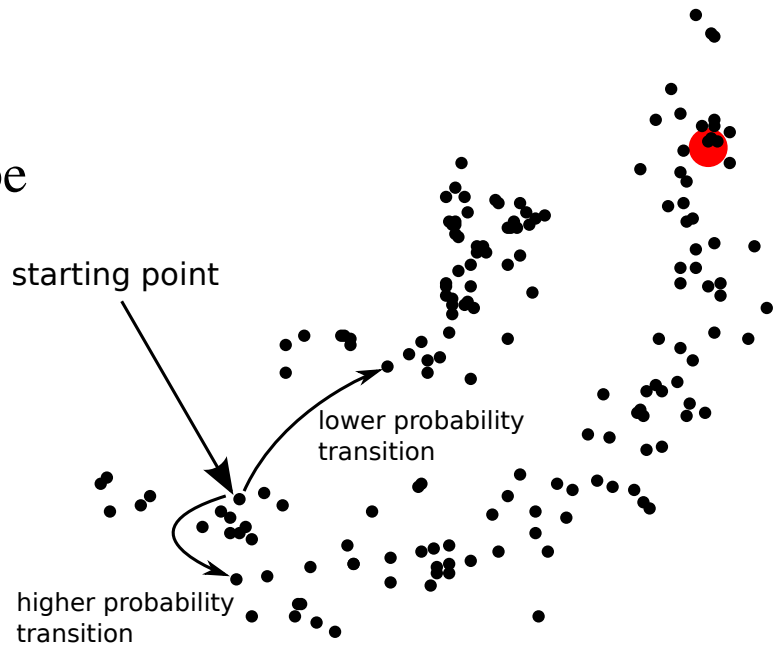
- A "random walker" moves around from point to point
- A transition between nearby points has higher probability
- A transition to a labeled point terminates the walk
- The label of a point x_i is the label of the terminal point

One possible random walk matrix

1. Let the *unnormalized* transition matrix be

$$\hat{M}_{ij} = \exp\left\{-\frac{\|x_i - x_j\|^2}{b}\right\}$$

2. Normalize rows of $\hat{M}$ to get M
3. If x_i has label y_i, re-define $M_{ii} = 1$



PROPERTY: ABSORBING STATES

Imagine we have S states. If $p(s_t = i | s_{t-1} = i) = 1$, then the ith state is called an **absorbing state** since we can never leave it.

Q: Given initial state $s_0 = j$ and set of absorbing states $\{i_1, \ldots, i_k\}$, what is the probability a Markov chain terminates at a particular absorbing state?

- Aside: For the semi-supervised classifier, the answer gives the probability on the label of x_j.

A: Start a random walk at j and keep track of the distribution on states.

- w_0 is a vector of 0's with a 1 in entry j because we know $s_0 = j$
- If M is the transition matrix, we know that $w_{t+1} = w_t M$.
- So we want $w_\infty = w_0 M^\infty$.

PROPERTY: ABSORBING STATE DISTRIBUTION

Group the absorbing states and break up the transition matrix into quadrants:

$$M = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}$$

The bottom half contains the self-transitions of the absorbing states.

Observation: $w_{t+1} = w_t M = w_{t-1} M^2 = \cdots = w_0 M^{t+1}$

So we need to understand what's going on with M^t. For the first two we have

$$M^2 = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A^2 & AB + B \\ 0 & I \end{bmatrix}$$

$$M^3 = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^2 & AB + B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A^3 & A^2B + AB + B \\ 0 & I \end{bmatrix}$$

GEOMETRIC SERIES

Detour: We will use the matrix version of the following scalar equality.

Definition: Let $0 < r < 1$. Then $\sum_{u=0}^{t-1} r^u = \frac{1-r^t}{1-r}$ and so $\sum_{u=0}^{\infty} r^u = \frac{1}{1-r}$.

Proof: First define the top equality and create the bottom equality

$$\begin{array}{rcccccccccc} C_t & = & 1 & + & r & + & r^2 & + & \cdots & + & r^{t-1} \\ r\,C_t & = & & & r & + & r^2 & + & \cdots & + & r^{t-1} & + & r^t \end{array}$$

and so

$$C_t - r\,C_t \;=\; 1 - r^t.$$

Therefore

$$C_t \;=\; \sum_{u=0}^{t-1} r^u \;=\; \frac{1-r^t}{1-r} \qquad \text{and} \qquad C_\infty \;=\; \frac{1}{1-r}.$$

PROPERTY: ABSORBING STATE DISTRIBUTION

A matrix version of the geometric series appears here. We see the pattern

$$M^t = \begin{bmatrix} A^t & \left(\sum_{u=0}^{t-1} A^u\right) B \\ 0 & I \end{bmatrix}.$$

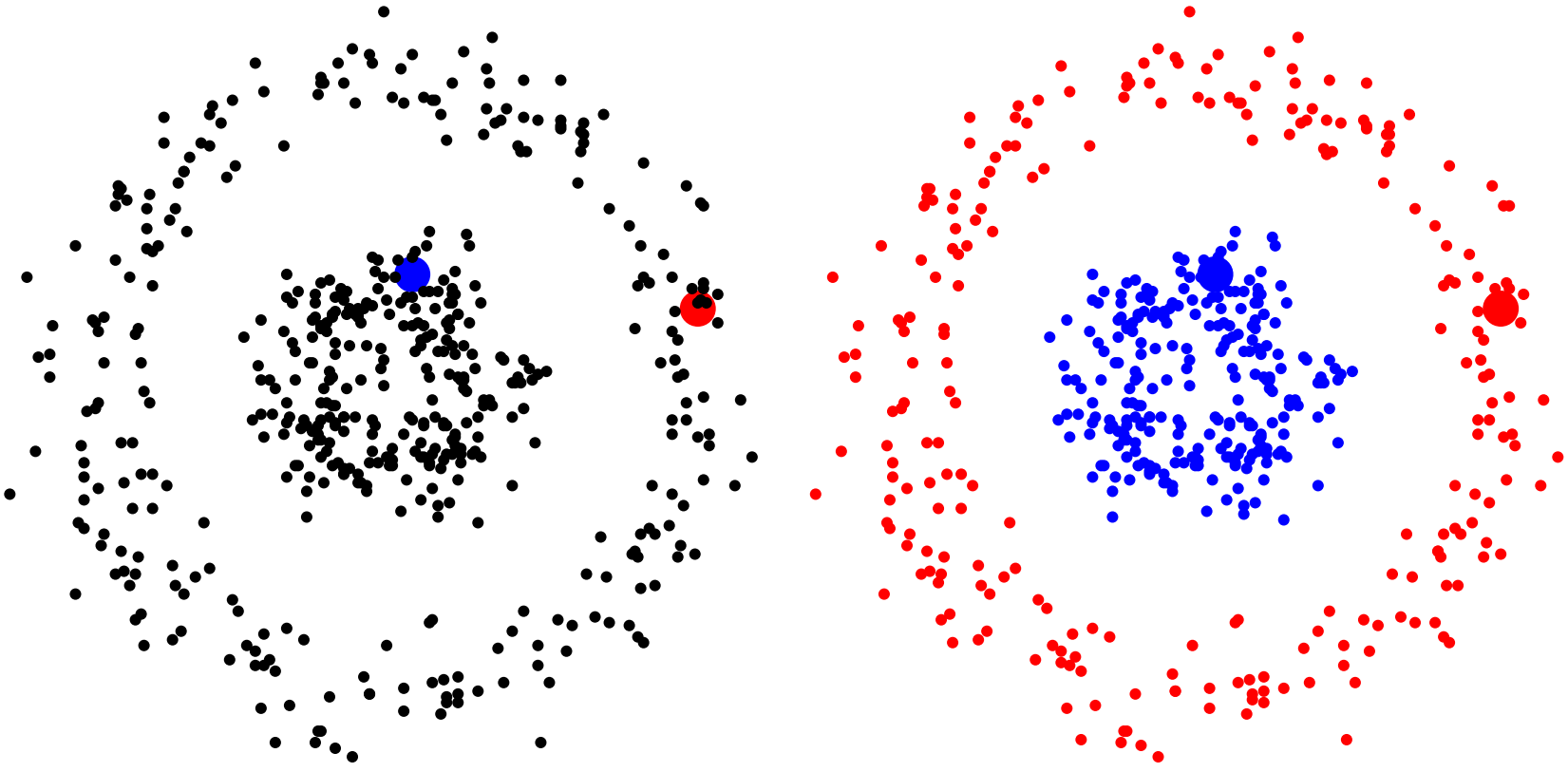
Two key things that can be shown are:

$$A^\infty = 0, \qquad \sum_{u=0}^{\infty} A^u = (I - A)^{-1}$$

Summary:

- After an infinite # of steps, $w_\infty = w_0 M^\infty = w_0 \begin{bmatrix} 0 & (I-A)^{-1}B \\ 0 & I \end{bmatrix}$.
- The non-zero dimension of w_0 picks out a row of $(I - A)^{-1}B$.
- The probability that a random walk started at x_j terminates at the ith absorbing state is $[(I - A)^{-1}B]_{ji}$.

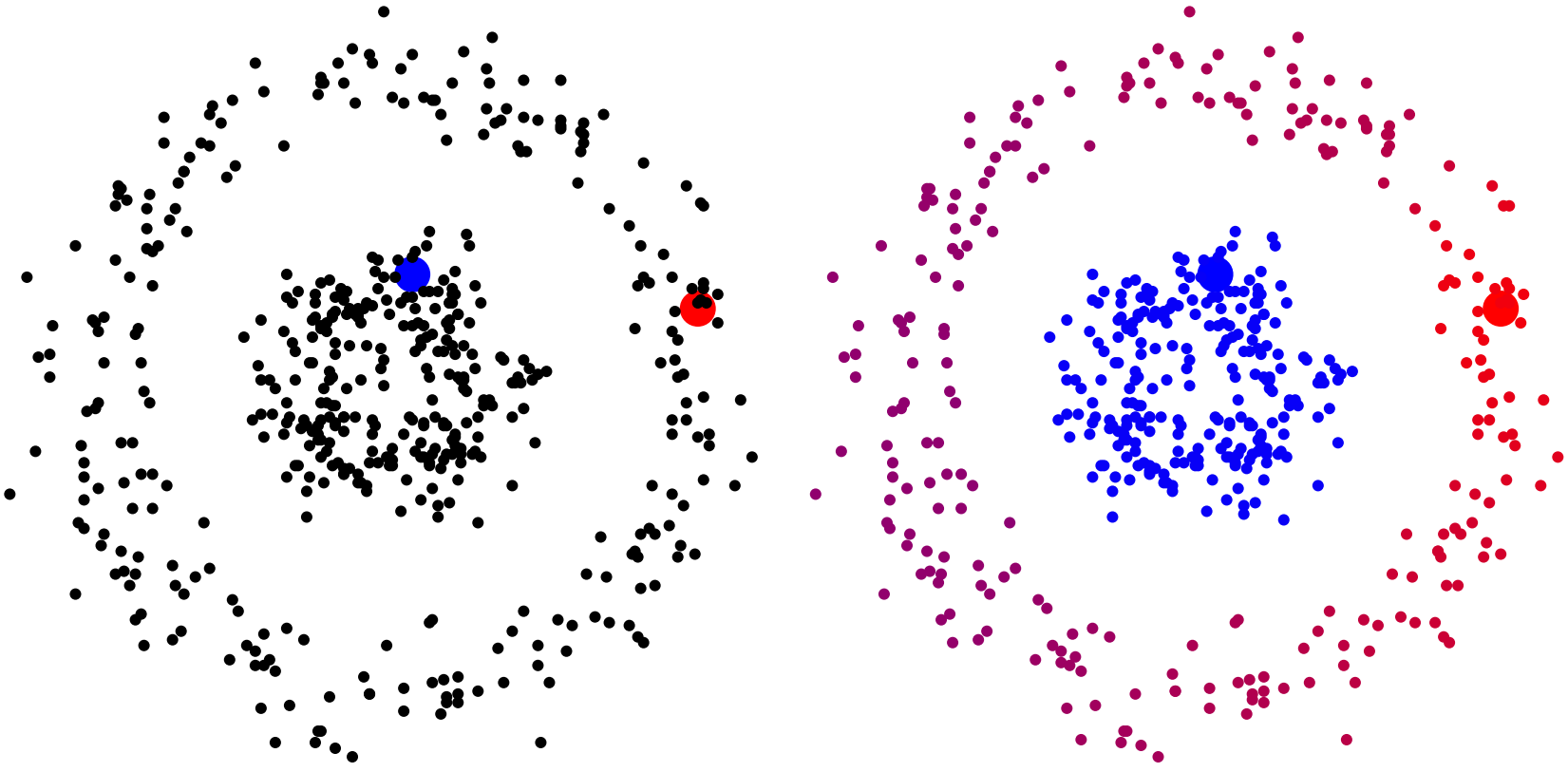
CLASSIFICATION EXAMPLE



Using a Gaussian kernel normalized on the rows. The color indicates the distribution on the terminal state for each starting point.

Kernel width was tuned to give this result.

CLASSIFICATION EXAMPLE



Using a Gaussian kernel normalized on the rows. The color indicates the distribution on the terminal state for each starting point.

Kernel width is larger here. Therefore, purple points may leap to the center.