BUSI97287: Advanced Machine Learning

Miscellaneous Topics

Martin Haugh

Imperial College Business School

Email: martin.b.haugh@gmail.com

References: Chapter 9 (online) of *Learning from Data* by Abu-Mostafa, Magdon-Ismail and Lin

Chapter 13 of *Elements of Statistical Learning Theory* by Hastie Tibshirani and Friedman

Outline

Learning Aids

The use of "learning aids" can lead to dramatic performance improvement in classification and regression models.

Such learning aids might include:

1. Appropriate preprocessing of data
   - e.g. standardizing features to have mean 0 and variance 1
   - e.g. "sphering" the data
2. Removing irrelevant features
3. Removing irrelevant dimensions
   - e.g. using PCA
4. Accounting for properties the target function / classifier is known to have
5. Identifying (and removing) unhelpful data-points

How to Sphere the Data

Suppose we are given centered data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with each $\mathbf{x}_i \in \mathbb{R}^d$.

Then sample covariance matrix is

$$\boldsymbol{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top = \frac{1}{n} \mathbf{X}^\top \mathbf{X}$$

where

$$\mathbf{X} = \begin{bmatrix} - & \mathbf{x}_1^\top & - \\ - & \mathbf{x}_2^\top & - \\ & \vdots & \\ - & \mathbf{x}_n^\top & - \end{bmatrix}$$

Assuming $\boldsymbol{\Sigma}$ is invertible we can construct its matrix square root $\boldsymbol{\Sigma}^{1/2}$ satisfying

$$\boldsymbol{\Sigma}^{1/2} \boldsymbol{\Sigma}^{1/2} = \boldsymbol{\Sigma}$$

We then transform the data to

$$\mathbf{z}_i = \boldsymbol{\Sigma}^{-1/2} \mathbf{x}_i \tag{1}$$

How to Sphere the Data

Remarks

1. Note that (1) is an invertible transformation since we can pre-multiply by $\boldsymbol{\Sigma}^{1/2}$ to obtain

$$\boldsymbol{\Sigma}^{1/2}\,\mathbf{z}_i = \mathbf{x}_i \tag{2}$$

2. The sample covariance matrix of the $\mathbf{z}_i$'s is

$$\boldsymbol{\Sigma}^{-1/2}\text{Cov}(\mathbf{x})\boldsymbol{\Sigma}^{-1/2} = \boldsymbol{\Sigma}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{-1/2} = \mathbf{I}_d$$

   where $\mathbf{I}_d$ is the $d \times d$ identity matrix. Hence the $\mathbf{z}_i$'s are uncorrelated.

Going from the $\mathbf{x}_i$'s to the $\mathbf{z}_i$'s is called sphering the data

- sometimes also called whitening the data.

When to Sphere the Data

The goal of data-sphering is to remove correlations from the data.

Often desirable since correlations can have a surprising (and negative) impact on learning.

Some learning approaches actually *require* sphering the data

- **e.g.** reduced-rank LDA.

Question: Should you sphere the data before doing PCA?

Using correlated input variables with regularization can make learning a target function difficult.

But transforming the data so that input features are uncorrelated can then make learning target function much easier – see next example.

Lasso: To Sphere or Not to Sphere

e.g. Suppose v_1 and v_2 are uncorrelated features and true model is

$$\begin{aligned} y &= \beta_1 v_1 + \beta_2 v_2 + \epsilon \\ &= 10 v_2 + \epsilon \end{aligned} \tag{3}$$

so $\beta_1 = 0$ and $\beta_2 = 10$.

- If we try to fit (3) then $||\boldsymbol{\beta}||_1$ of target function is 10.

Suppose now that $x_1 := v_1$ and $x_2 := v_1 + v_2$ and consider

$$y = \beta_1 x_1 + \beta_2 x_2 + \epsilon \tag{4}$$

- The features x_1 and x_2 are now correlated.
- If we try to fit (4) then target function has $\beta_2 = 10$ and $\beta_1 = -10$.
- So $||\boldsymbol{\beta}||_1$ of target function is now 20!

So Lasso will find it easier to fit target function using (3) rather than (4)

- hence a good idea to use uncorrelated features when regularizing!

Hints and Invariances

"Hints" are bits of information that you know about the target function $y = f(\mathbf{x})$ even without looking at any data.

These hints typically come from domain specific knowledge.

e.g. Suppose we want to fit a linear classifier

$$f(\mathbf{x}) = \text{sign}(\alpha + \beta x)$$

to predict the ultimate default (-1) or non-default ($+1$) of a borrower as function of income x.

Question: Without seeing any data, what (if anything) can you say about β?

Hints are useful because in general we don't have enough data to estimate the target function exactly.

Therefore makes sense to use the hints to try to improve the learning process

- can often result in dramatic improvements!

Common Examples of Hints and Invariances

1. Symmetry or anti-symmetry: $f(\mathbf{x}) = f(-\mathbf{x})$ or $f(\mathbf{x}) = -f(-\mathbf{x})$.
2. Rotational invariance: $f(\mathbf{x})$ only depends on $||\mathbf{x}||$
   - common in vision applications.
3. General invariance hints: for some transformation $\mathcal{T}$, $f(\mathbf{x}) = f(\mathcal{T}\mathbf{x})$.
4. Monotonicity: $f(\mathbf{x} + \Delta\mathbf{x}) \geq f(\mathbf{x})$ if $\Delta\mathbf{x} \geq \mathbf{0}$
   - e.g. borrower example from previous slide.
5. Convexity: $f(\lambda\mathbf{x} + (1-\lambda)\mathbf{x}') \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{x}')$.
6. Perturbation: f close to a known function f' so that $f = f' + \delta f$ where δf is small
   - e.g. f and f' denote prices of American and European option, respectively
   - in this case also know that $\delta f \geq \mathbf{0}$.

Remark: Also possible for these hints to apply to a strict subset of features.

How Should We Use These Hints / Invariances?

Problem: How to use these hints / invariance to improve our learning process?

Possible Answers

1. Add "hard constraint(s)" enforcing hints / invariances to optimisation problem that solves for optimal parameters
   - e.g. in borrower example add constraint $\beta \geq 0$ to optimisation problem.

   Seems like a reasonable thing to do but:
   - May be hard to solve optimisation problem with these new constraints.
   - Imposing hard constraints gives up flexibility and possibly ignores very good approximations that only slightly violate the constraints

2. Instead impose "soft constraint(s)" that allow the hint / invariance to be violated but penalizes such violations in the objective.
   - A more flexible approach that can also be easy to implement.
   - e.g. via virtual examples.

Virtual Examples

Virtual examples augment the data-set so that hint becomes represented by the data.

The learning algorithm (ridge, lasso, logistic etc.) then naturally trades off:

1. Satisfying the hint — by attempting to fit the virtual examples

with

2. Approximating f — by attempting to fit the real examples

Virtual Examples

e.g. Suppose we know that $f(\mathbf{x}) = f(-\mathbf{x})$. Then can:

- Augment data-set with virtual pairs $\{\mathbf{v}_m, -\mathbf{v}_m\}$ for $m = 1, \ldots, M$.
- Then define the hint error

$$E_{\text{hint}}(\hat{f}) = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}(\mathbf{v}_m) - \hat{f}(-\mathbf{v}_m) \right)^2$$

  where $\hat{f}$ is the approximation to the target function.

- Now define an augmented error function

$$E_{\text{aug}}(\hat{f}) = E_{\text{in}}(\hat{f}) + \lambda E_{\text{hint}}(\hat{f}) \tag{5}$$

  where $E_{\text{in}}(\hat{f})$ is the usual loss function, e.g. sum-of-squared errors in regression.

Virtual Examples for the Symmetry Hint

Questions

1. How should we choose λ?

   Answer: Cross-validation!

2. How might we choose the virtual examples?

   Answer: Just take $\{\mathbf{x}_i, -\mathbf{x}_i\}$ for $i = 1, \ldots, n$, i.e. use the actual data to create virtual data.

3. When might we want to use more hint examples than those provided by the $\{\mathbf{x}_i, -\mathbf{x}_i\}$ pairs?

   Answer: ??

Virtual Examples

Easy to create virtual examples for other types of hints / invariances.

e.g. Suppose we know f is monotonic increasing. Then can:

- Create (how?) virtual examples $\{\mathbf{x}_i, \mathbf{x}_i'\}$ where $\mathbf{x}_i' = \mathbf{x}_i + \Delta\mathbf{x}_i$ for some $\Delta\mathbf{x}_i \geq \mathbf{0}$.
- Then use a hint error function of the form

$$E_{\text{hint}}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{f}(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i')\right)^2 \mathbf{1}_{\{\hat{f}(\mathbf{x}_i') < \hat{f}(\mathbf{x}_i)\}}$$

Question: How might you include virtual examples for a target function that is rotationally invariant?

Remarks

1. Can of course also include a regularization term to $E_{\text{aug}}(\hat{f})$ in (5).
2. Motivation for regularization is to reduce variance at cost of (hopefully) small increase in bias.
3. Motivation for $E_{\text{hint}}(\hat{f})$ term is to reduce bias.

Data Cleaning

Sometimes can be beneficial to remove data – even though that data may be "correct".

e.g. A noisy "outlier" may be very misleading and removing it may help the learning process.

Identifying misleading data-points is difficult – of course! So how do we proceed?

Two Approaches (See Section 9.4 of Learning from Data)

1. Using a simpler model to identify noisy data
2. Computing a validation leverage score

Using a Simpler Model to Identify Noisy Data

Misleading data is often data that is hard to classify correctly

- so let's look for such points.

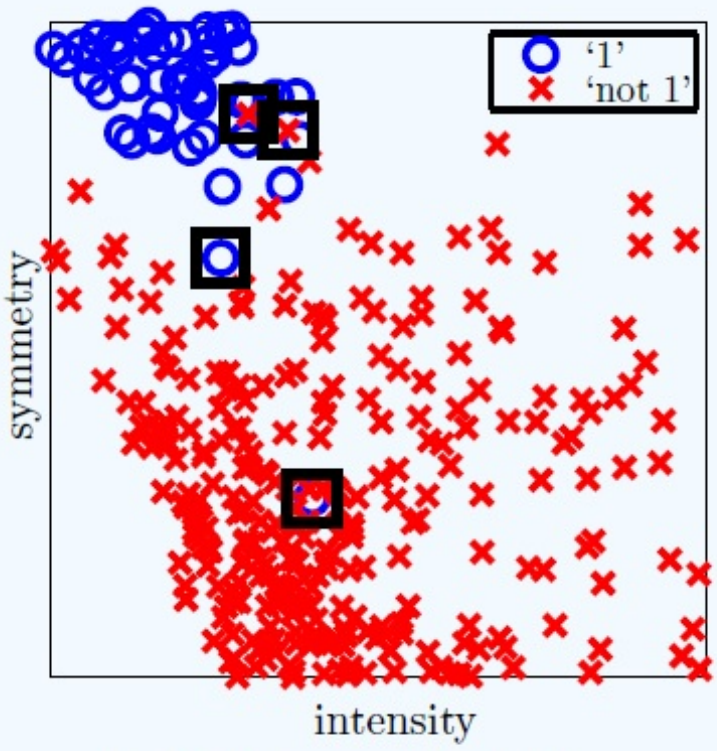
But complex classifiers can more easily (correctly) classify data-points

- So let's use simple classifiers to identify hard-to-classify points
- In fact we can use several simple classifiers.
- If a point is misclassified by a majority of the simple classifiers then can declare that data-point to be "hard"
  - and should then consider omitting that data-point from the data-set.

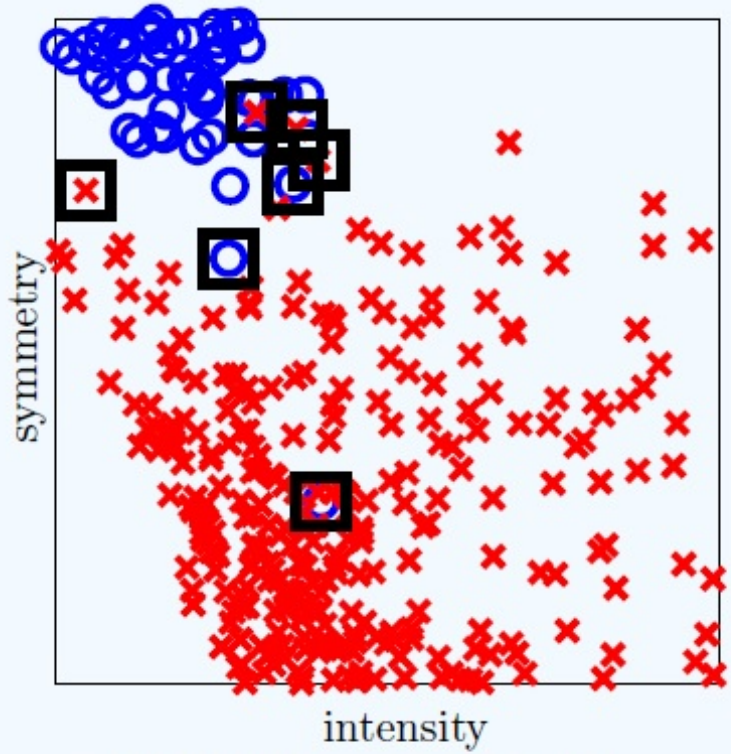
Examples of simple classifiers are linear models for regression or k-NN with k large for classification.

Question: How can we construct several simple classifiers?

Answer: Use bootstrap data-sets and train same simple model on each one.



(a) 5-Nearest Neighbor



(b) 101-Nearest Neighbor

Figure 9.5: Identifying noisy data using the 'simple' nearest neighbor rule. The data in black boxes are identified as noisy (a) using 5-Nearest Neighbor and (b) using 101-Nearest Neighbor. 101-Nearest Neighbor, the simpler of the two models, identifies more noisy data as expected.

Figure taken from Abu-Mostafa, Magdon-Ismail and Lin's *Learning from Data*

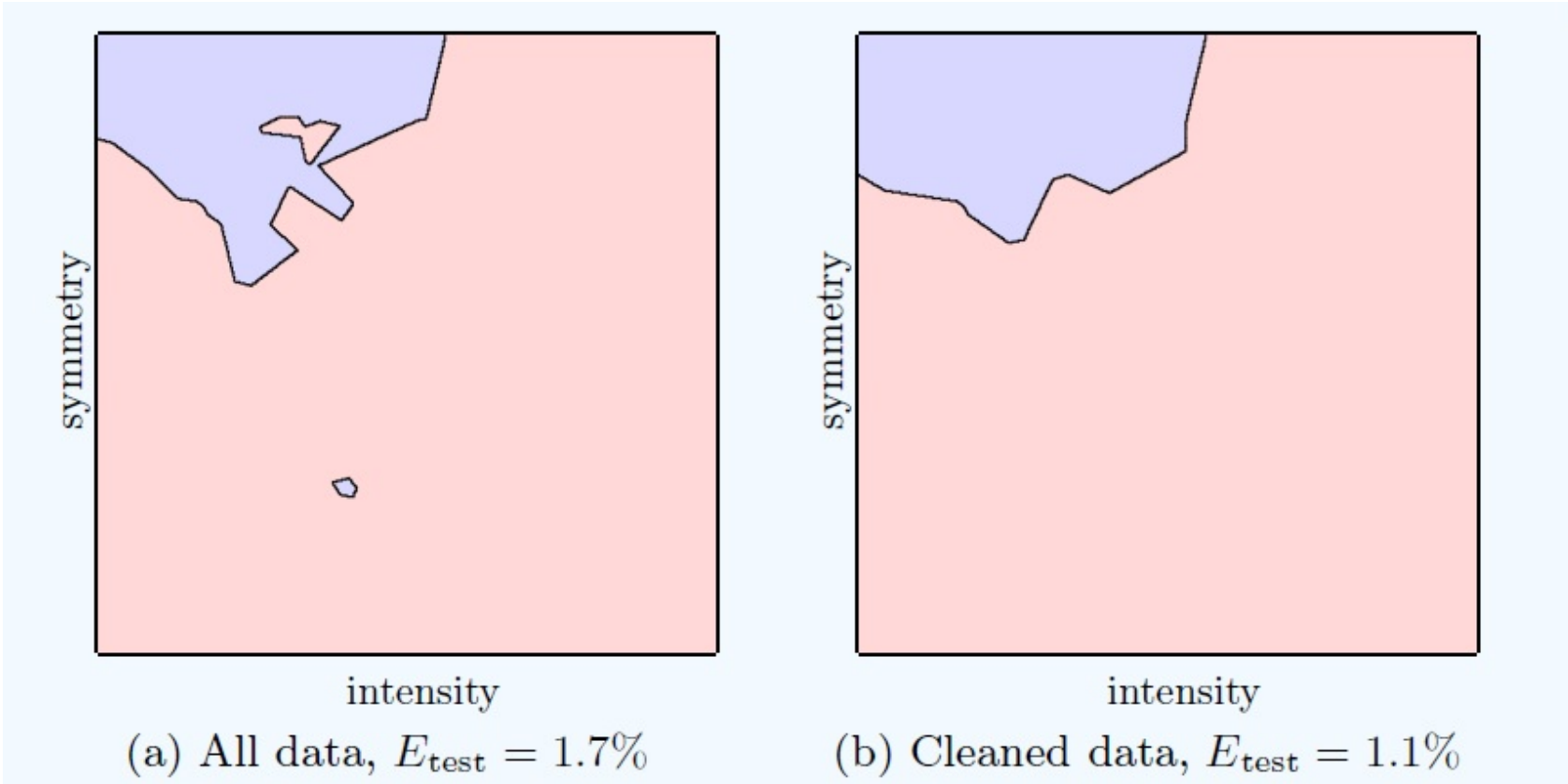


Figure 9.6: The 1-Nearest Neighbor classifier (a) using all the data and (b) after removing the bad examples as determined using 101-Nearest Neighbor. Removing the bad data gives a more believable decision boundary.

Figure taken from Abu-Mostafa, Magdon-Ismail and Lin's *Learning from Data*

- 1-NN is the "complex" model ultimately chosen to construct the classifier
- 5/101-NN are "simple" models used to identify hard-to-classify points.

Computing a Validation Leverage Score

No guarantee that removing a data-point will improve test error.

Can estimate adverse impact or leverage that a data-point has on final classifier using validation.

Notation

- $\mathcal{D}$ denotes entire training data-set
- $\mathcal{D}_i$ denotes entire training data-set except data-point $(\mathbf{x}_i, y_i)$
- $\hat{f}$ denotes final fitted model using $\mathcal{D}$
- $\hat{f}_i^-$ denotes final fitted model using $\mathcal{D}_i$

Computing a Validation Leverage Score

Then say the leverage score of data-point $(\mathbf{x}_i, y_i)$ is

$$l_i = E_{\text{out}}(\hat{f}) - E_{\text{out}}(\hat{f}_i^-)$$

where $E_{\text{out}}(\hat{f})$ and $E_{\text{out}}(\hat{f}_i^-)$ denote out-of-sample errors of $\hat{f}$ and $\hat{f}_i^-$.

- they can be estimated using cross-validation.

Leverage score measures how valuable $(\mathbf{x}_i, y_i)$: if l_i large and positive then $(\mathbf{x}_i, y_i)$ is harmful and should be discarded from the training data.

But estimating l_i for all i is computationally expensive so not always feasible

- but can be done for a subset of points like those identified using simpler model approach described earlier.

Adaptive k-Nearest Neighbors

Non-parametric methods such as k-NN can work very poorly in high-dimensions

- follows because neighborhood of a test point $\mathbf{x}_0$ will typically have very few (if any) training points

Can partly address this problem by allowing metric used at a given point to depend on the point

- yields adaptive k-nearest-neighbors.

Can motivate adaptive k-NN by considering Figure 13.13 from HTF:

- 2 classes and 2 features but clearly only one feature is related to class.
- Specifically, class probabilities vary only in the x-direction and not in the y-direction
- So it makes sense to stretch the neighborhood in the y-direction.

Can generalize this insight to apply in higher dimensions by adapting the metric at each point.

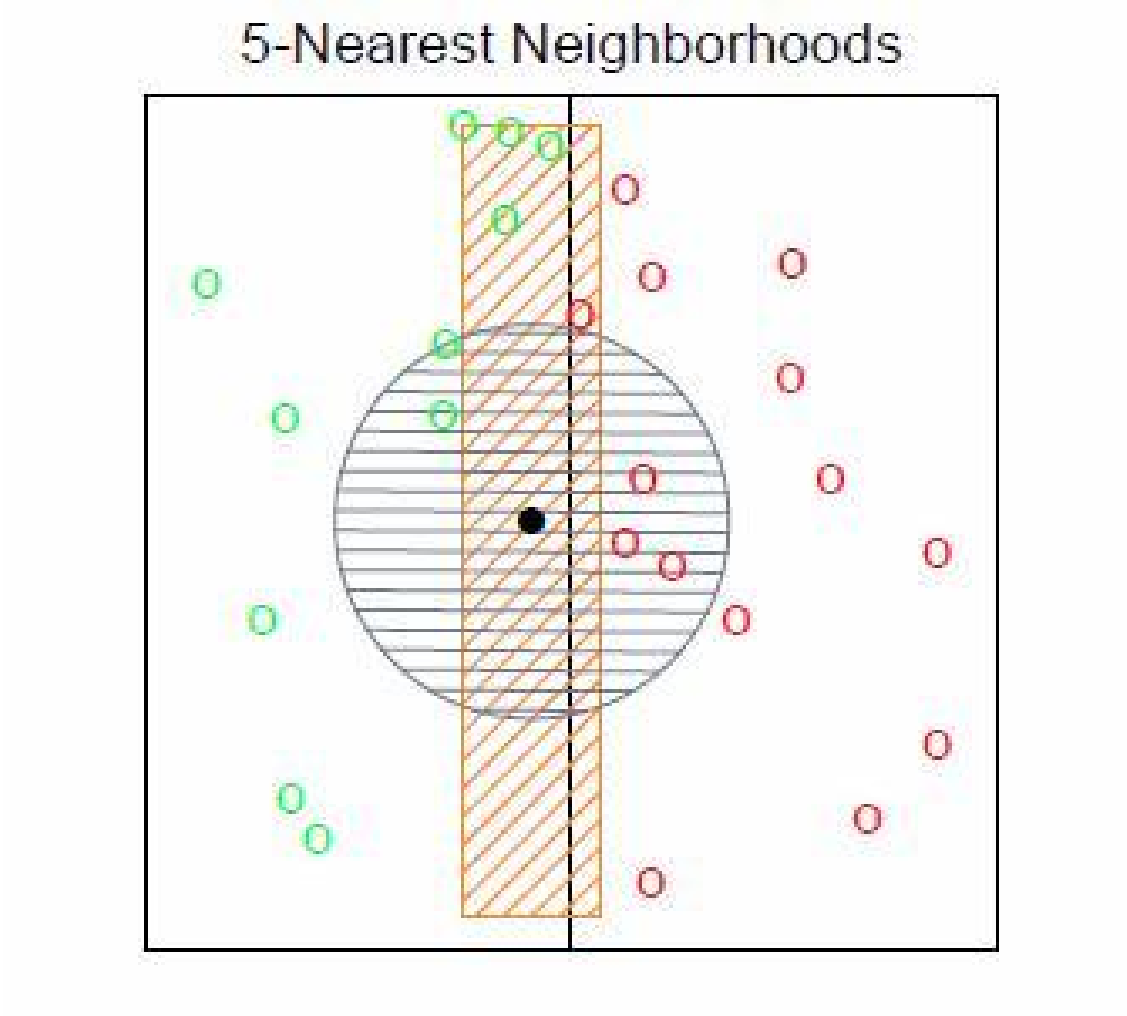


Figure 13.13 from HTF: The points are uniform in the cube, with the vertical line separating class red and green. The vertical strip denotes the 5-nearest-neighbor region using only the horizontal coordinate to find the nearest-neighbors for the target point (solid dot). The sphere shows the 5-nearest-neighbor region using both coordinates, and we see in this case it has extended into the class-red region (and is dominated by the wrong class in this instance).

Adaptive k-Nearest Neighbors

Suppose there are K classes. The discriminant adaptive nearest neighbor (DANN) rule works as follows:

1. At a query point $\mathbf{x}_0$, form the neighborhood of the n nearest points
   - will use these n points to determine the metric used at $\mathbf{x}_0$.
2. Let π_k denote percentage of these n points from class k.
3. Using these n points compute:
   (i) $\boldsymbol{W} = \sum_{k=1}^{K} \pi_k \boldsymbol{W}_k$, the pooled within-class covariance matrix.
   (ii) $\boldsymbol{B} = \sum_{k=1}^{K} \pi_k (\bar{\mathbf{x}}_k - \bar{\mathbf{x}})(\bar{\mathbf{x}}_k - \bar{\mathbf{x}})^\top$, the between-class covariance matrix.
4. Define
$$\boldsymbol{\Sigma} \;:=\; \boldsymbol{W}^{-1/2} \Big[\boldsymbol{W}^{-1/2} \boldsymbol{B}\, \boldsymbol{W}^{-1/2} \;+\; \epsilon \boldsymbol{I} \Big] \boldsymbol{W}^{-1/2}. \tag{6}$$
5. Now classify $\mathbf{x}_0$ using k-NN with $k < n$ and the metric
$$D(\mathbf{x}, \mathbf{x}_0) \;=\; (\mathbf{x} - \mathbf{x}_0)^\top \boldsymbol{\Sigma} (\mathbf{x} - \mathbf{x}_0).$$

HTF suggest using $n = 50$ and $\epsilon = 1$.

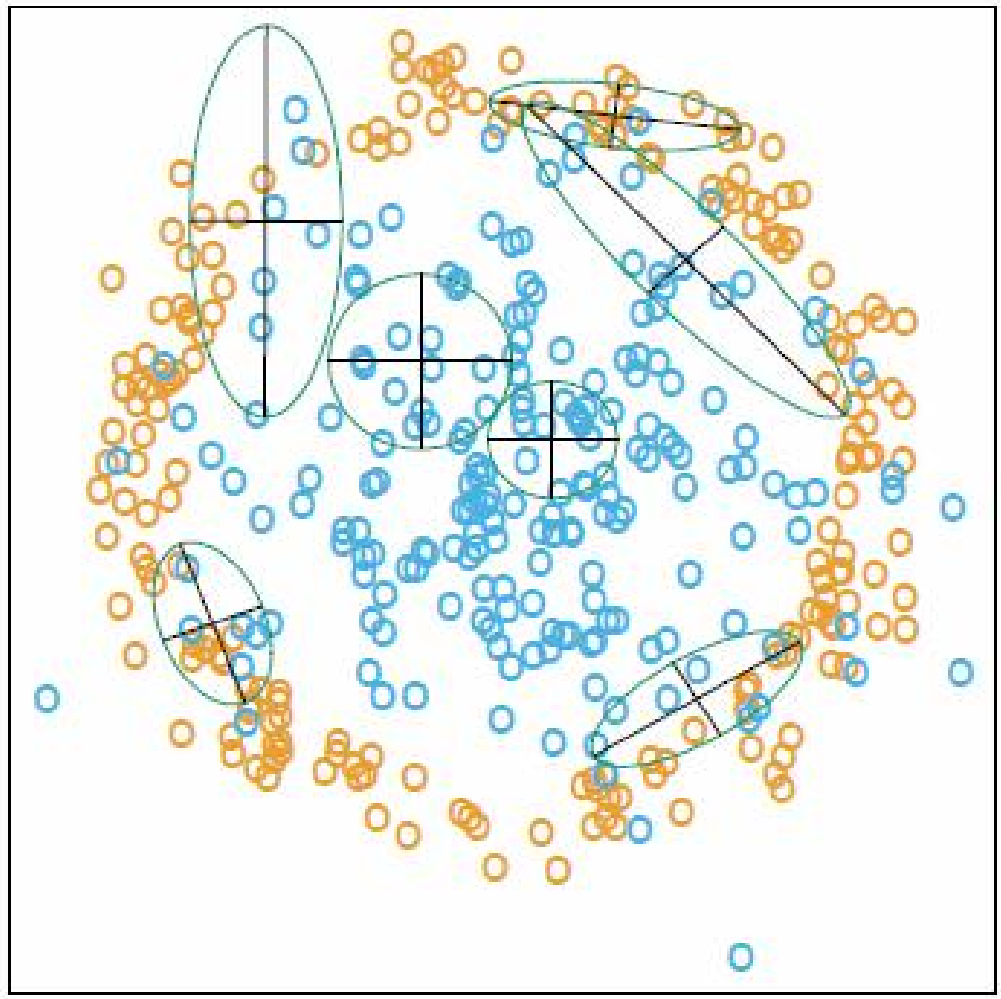


Figure 13.14 from HTF: Neighborhoods found by the DANN procedure, at various query points (centers of the crosses). There are two classes in the data, with one class surrounding the other. 50 nearest-neighbors were used to estimate the local metrics. Shown are the resulting metrics used to form 15-nearest-neighborhoods.

Adaptive k-Nearest Neighbors

Question: What is the geometric interpretation behind (6)?

Solution: (6) arises naturally from the following steps:

1. First sphere the data using the pooled within-class variance $\mathbf{W}$ so that

$$\mathbf{x}_i^* := \mathbf{W}^{-1/2}\mathbf{x}_i$$

   for each point $\mathbf{x}_i$ in the n-neighborhood of $\mathbf{x}_0$.

2. Now note that $\mathbf{B}^* := \mathbf{W}^{-1/2}\mathbf{B}\mathbf{W}^{-1/2}$ is the between-class covariance matrix under these new coordinates.

3. The metric in (6) now reduces to

$$\begin{aligned} D(\mathbf{x}, \mathbf{x}_0) &= (\mathbf{x} - \mathbf{x}_0)^\top \boldsymbol{W}^{-1/2} \left[\boldsymbol{W}^{-1/2} \boldsymbol{B} \boldsymbol{W}^{-1/2} + \epsilon \boldsymbol{I} \right] \boldsymbol{W}^{-1/2} (\mathbf{x} - \mathbf{x}_0) \\ &= (\mathbf{x}^* - \mathbf{x}_0^*)^\top \left[\boldsymbol{B}^* + \epsilon \boldsymbol{I} \right] (\mathbf{x}^* - \mathbf{x}_0^*). \end{aligned} \tag{7}$$

Adaptive k-Nearest Neighbors

Can now make some intuitive sense of (7). For example ...

- Let $\mathbf{e}$ be an eigen vector of $\mathbf{B}^*$ so that $\mathbf{e}^\top \mathbf{e} = 1$

$$\mathbf{B}^* \mathbf{e} = \lambda \mathbf{e}$$

- Suppose λ is small.
- Suppose also for some constant a

$$\mathbf{x}^* - \mathbf{x}_0^* \approx a\mathbf{e}$$

- Then (ignoring $\epsilon \boldsymbol{I}$ term) we see that (7) will be small because

$$(\mathbf{x}^* - \mathbf{x}_0^*)^\top \boldsymbol{B}^* (\mathbf{x}^* - \mathbf{x}_0^*) \quad \approx \quad \lambda a^2$$

- So $\mathbf{x}^*$ more likely to be included among the k nearest neighbors
  - which is desirable since $\mathbf{x}^*$ more likely (why?) to have same class as $\mathbf{x}_0^*$

Role of $\epsilon \boldsymbol{I}$ is simply to round any infinite strips to ellipses in new coordinates.

Reduced-Rank LDA

Recall our LDA analysis where

$$\log \frac{\mathrm{P}(G=k|\mathbf{X}=\mathbf{x})}{\mathrm{P}(G=l|\mathbf{X}=\mathbf{x})} = \log \frac{\hat{\pi}_k}{\hat{\pi}_l} - \frac{1}{2}(\mathbf{x}-\hat{\boldsymbol{\mu}}_k)^\top \hat{\boldsymbol{\Sigma}}^{-1}(\mathbf{x}-\hat{\boldsymbol{\mu}}_k) + \frac{1}{2}(\mathbf{x}-\hat{\boldsymbol{\mu}}_l)^\top \hat{\boldsymbol{\Sigma}}^{-1}(\mathbf{x}-\hat{\boldsymbol{\mu}}_l) \quad (8)$$

Now let $\hat{\boldsymbol{\Sigma}} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^\top$ be the eigen decomposition of $\hat{\boldsymbol{\Sigma}}$ where:

- $\boldsymbol{U}$ is an $m \times m$ orthornormal matrix
- $\boldsymbol{D}$ is a diagonal matrix of the positive eigen values of $\hat{\boldsymbol{\Sigma}}$.

Consider now a change of variable with $\mathbf{x}^* := \boldsymbol{D}^{-1/2}\boldsymbol{U}^\top\mathbf{x}$

- so that variance-covariance matrix of $\mathbf{x}^*$ is the identity matrix

Also implies $\hat{\boldsymbol{\mu}}_k^* := \boldsymbol{D}^{-\frac{1}{2}}\mathbf{U}^\top\hat{\boldsymbol{\mu}}_k$ are now the estimated class centroids and

$$(\mathbf{x}-\hat{\boldsymbol{\mu}}_k)^\top \hat{\boldsymbol{\Sigma}}^{-1}(\mathbf{x}-\hat{\boldsymbol{\mu}}_k) = \|\mathbf{x}^*-\hat{\boldsymbol{\mu}}_k^*\|_2^2 .$$

Reduced-Rank LDA

Can then rewrite (8) as

$$\begin{aligned}\log\frac{\mathsf{P}(G=k|\mathbf{X}=\mathbf{x})}{\mathsf{P}(G=l|\mathbf{X}=\mathbf{x})} &= \log\frac{\hat{\pi}_k}{\hat{\pi}_l} - \frac{1}{2}(\mathbf{x}^*-\hat{\boldsymbol{\mu}}_k^*)^\top(\mathbf{x}^*-\hat{\boldsymbol{\mu}}_k^*) \\ &\quad + \frac{1}{2}(\mathbf{x}^*-\hat{\boldsymbol{\mu}}_l^*)^\top(\mathbf{x}^*-\hat{\boldsymbol{\mu}}_l^*) \\ &= \left(\log\hat{\pi}_k - \frac{1}{2}\|\mathbf{x}^*-\hat{\boldsymbol{\mu}}_k^*\|_2^2\right) \\ &\quad - \left(\log\hat{\pi}_\ell - \frac{1}{2}\|\mathbf{x}^*-\hat{\boldsymbol{\mu}}_\ell^*\|_2^2\right). \qquad (9)\end{aligned}$$

From (9) it follows that the LDA classifier satisfies

$$k^* = \underset{1\le k\le K}{\operatorname{argmax}}\left\{\log\hat{\pi}_k - \frac{1}{2}\|\mathbf{x}^*-\hat{\boldsymbol{\mu}}_k^*\|_2^2\right\}$$

Reduced-rank LDA now does classification by working in the $K-1$-dimensional subspace spanned by the $\hat{\boldsymbol{\mu}}_k^*$'s – or sub-spaces of this sub-space.

LDA in Spherical Coordinates

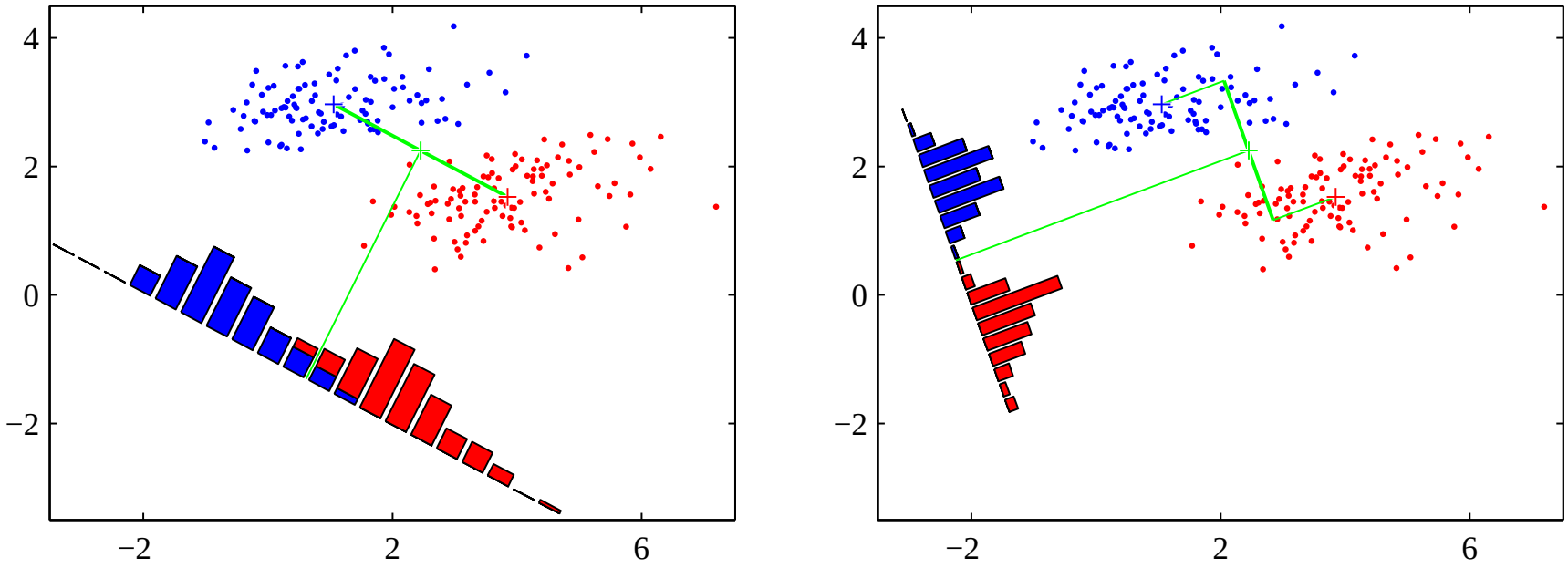


Figure 4.6 from Bishop: The left plot shows samples from two classes (depicted in red and blue) along with the histograms resulting from projection onto the line joining the class means. Note that there is considerable class overlap in the projected space. The right plot shows the corresponding projection based on the Fisher linear discriminant, showing the greatly improved class separation.

Reputation Systems

The ever increasing amount of online activity and data has been accompanied by the decentralization of control.

For example:

1. `IMDB` and `Rotten Tomatoes` help us choose what movies to watch.
2. How do we decide what sellers to trust on `Ebay`?
3. Who should we follow on `Twitter`?
4. How does `Google` figure out the importance of web-pages?
5. Why is `Yelp` so popular?

Reputation Systems

An important problem is figuring out who or what pages are the "best"

- clearly this is also a dimension reduction problem!

Here we will consider `Google`'s PageRank system:

- clearly of enormous importance
- has applications to networks beyond evaluating the importance of web-pages
- can use it to highlight the ongoing challenges with reputation systems.

Question: How do we define reputation? How about this definition?

> "You're great if other great people think you're great."

Any problems with this? Yes ... but how do reputations in the real world work?

Google's Naive PageRank Algorithm

Consider the web and suppose there are a total of d pages.

Say $i \to j$ if page i links to page j and define $c(i) := \#$ of out-links from page i.

Let $X_t \in \{1, \ldots, d\}$ denote the page that a web-surfer visits at time t.

Initial model assumes X_t a Markov chain with transition matrix $\mathbf{Q}$ where

$$\begin{aligned} Q_{i,j} &:= \mathsf{P}\left(X_{t+1} = j \mid X_t = i\right) \\ &= \begin{cases} \frac{1}{c(i)}, & \text{if } i \to j \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Would like to find a stationary distribution, $\boldsymbol{\mu}$, of the Markov chain so that

$$\boldsymbol{\mu} = \boldsymbol{\mu}\mathbf{Q}.$$

Could then use $\boldsymbol{\mu}(i)$ to measure the importance of the i^{th} page. Why?

But $\boldsymbol{\mu}$ will not be unique unless X is irreducible

- and X will not be irreducible in general because (for example) many web-pages will not have out-links.

Naive PageRank

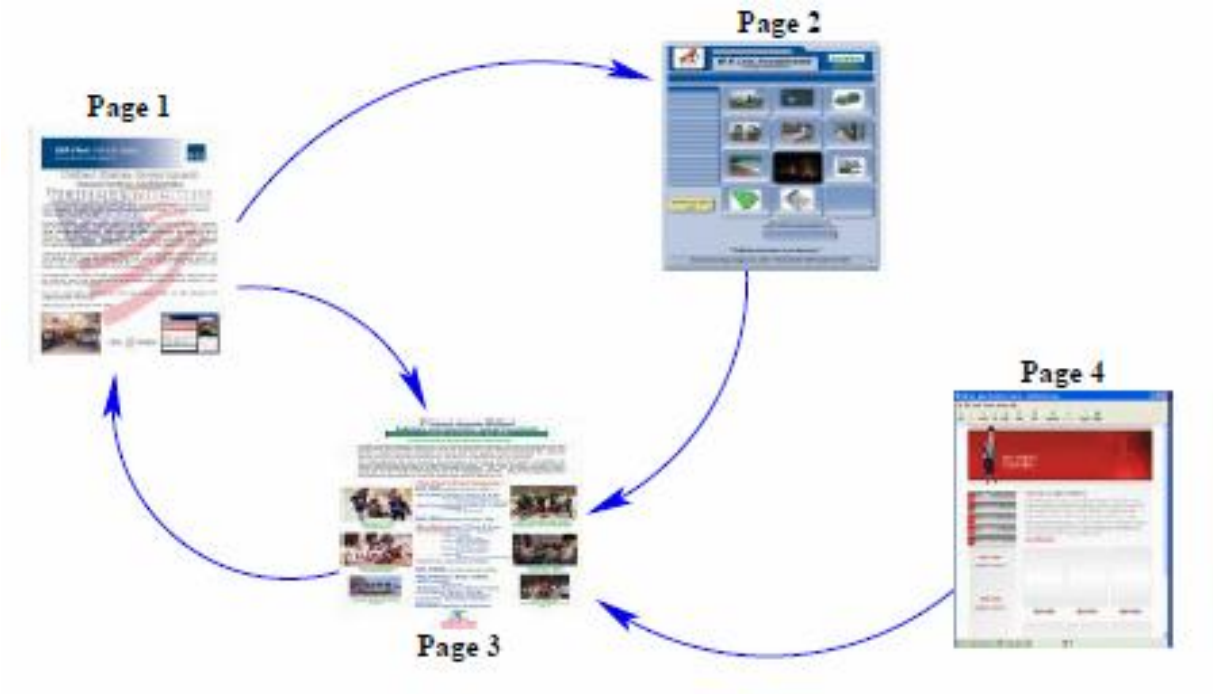


Figure 14.46 from HTF: Page-Rank algorithm: example of a small network

Another (less important) issue with defining web-page reputations in this way is that it is easily "gamed" by just a couple of web-pages that collude.

See for example "Making Eigenvector-Based Reputation Systems Robust to Collusion" (2004) by Zhang, Goel, Govindan, Mason and Van Roy.

Google's PageRank Algorithm

We resolve the uniqueness problem (and make collusion harder!) by instead assuming a transition matrix $\bar{\mathbf{Q}}$ where

$$\bar{Q}_{i,j} \quad := \quad (1-\epsilon)\, Q_{i,j} \; + \; \frac{\epsilon}{d} > 0 \tag{10}$$

for $0 < \epsilon < 1$. Can write (10) equivalently as

$$\bar{\mathbf{Q}} \quad := \quad (1-\epsilon)\mathbf{Q} \; + \; \frac{\epsilon}{d}\, \mathbf{1}\mathbf{1}^{\top} \tag{11}$$

where $\mathbf{1}$ is a $d \times 1$ vector of 1's.

Note that (11) implies that $\bar{\mathbf{Q}}$ is irreducible and therefore has a unique stationary distribution, $\boldsymbol{\mu}$, satisfying

$$\boldsymbol{\mu} \; = \; \boldsymbol{\mu}\bar{\mathbf{Q}}. \tag{12}$$

Can interpret the resulting Markov chain as one where with probability ϵ we choose a new web-page randomly from the entire universe of web-pages and with probability $1-\epsilon$ we click randomly on a link on the current page

- the random walk interpretation.

Google's PageRank Algorithm

$\boldsymbol{\mu}$ gives the page-ranks of all the web-pages.

If we run the Markov chain for a sufficiently long time then proportion of time we spend on page i is $\boldsymbol{\mu}(i)$.

Page-rank or "importance" of a page is therefore captured by importance of the pages that link to it

- so still a circular definition of reputation!

PageRank has been generalized, manipulated, defended, etc. and reputation systems in general have been the cause of much litigation

- should give you an idea of just how important they are!

Computing PageRank is computationally intensive. Rather than solving (12) directly there are two popular approaches:

1. Power iteration – the most commonly used approach.
2. Monte-Carlo which exploits our earlier random walk interpretation
    - an approach that has been proposed for the fast updating of PageRank.

Using Power Iteration to Estimate PageRank

Power iteration is very simple:

1. Choose an initial vector, $\boldsymbol{\mu}_0$. (Easy choice is $\boldsymbol{\mu}_0 = \mathbf{1}/d$.)
2. Then iterate $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t \bar{\mathbf{Q}}$ for $t = 1, \ldots$ until convergence.

Question: Is convergence guaranteed?

Answer: Yes!

Proof: Let $\Delta_t := \sum_{i=1}^{d} |\boldsymbol{\mu}_t(i) - \boldsymbol{\mu}(i)|$ where $\boldsymbol{\mu}$ is the solution to (12).
Using the triangle inequality can see that

$$|\boldsymbol{\mu}_{t+1}(i) - \boldsymbol{\mu}(i)| \leq (1-\epsilon) \sum_{j:\, j \text{ links to } i} \frac{|\boldsymbol{\mu}_t(j) - \boldsymbol{\mu}(j)|}{c(j)} \tag{13}$$

Summing (13) over i yields $\Delta_{t+1} \leq (1-\epsilon)\Delta_t$.
Therefore (why?) $\Delta_t \to 0$ as $t \to \infty$ and so $\boldsymbol{\mu}_t(i)$ converges to $\boldsymbol{\mu}(i)$.$\square$

Question: Is the convergence fast?

PageRank

`Google` originally used PageRank to index the web and to provide an ordering of web-pages according to their reputation / importance.

This ordering was vital to search and advertising

- but also provided an obvious motive for companies to try and increase the PageRank of their web-pages.

Over the years the PageRank system has adapted and heuristics added

- in part to counter attempts at collusion and manipulation.

While the specific details are private it is very likely that PageRank still plays the key role in `Google`'s search algorithms.

PageRank and Search

Question: How might you use PageRank as part of a search system?

Here's how ...

1. First compute the PageRank of all web-pages and store it in descending order of importance
   - just like a book with the 1^{st} page having the highest PageRank, the 55^{th} page having the 55^{th} highest PageRank etc.
2. A reverse index for all commonly searched terms is also computed and stored
   - this is exactly like an index at the back of a book telling you what pages important terms appear on.

Steps 1 and 2 are updated periodically.

3. When a user searches for "dog" say, the reverse index tells you exactly what web-pages the term "dog" appears on and returns the first n of them.
   - This is easy and "correct". Why?

Question: How do you handle a multiple term search, e.g. "dog" *and* "pony"?

Other Applications / Extensions of PageRank

The ordering returned by PageRank can easily be adapted to reflect a preference for a specific subset, $\mathcal{C}$ say, of web-pages.

Can do this by changing the algorithm so that w.p. ϵ we choose a new web-page randomly from $\mathcal{C}$ and w.p. $1-\epsilon$ we click randomly on a link on the current page

- is known as personalized PageRank.

PageRank can be applied in most network settings where links between nodes indicate some sort of preference. Other examples include:

1. **Recommendation systems:** form a bipartite graph with movies on one side and users on the other. Add a link between a movie and a user if the user liked that movie. Now PageRank gives an ordering of movies *and* users.

   Question: How would you recommend movies to a specific user?

2. Networks can also be created out of users and tweets on `Twitter` with links added to identify creators of tweets and followers. Can use such a system to identify influential people and make recommendations to users.

3. Ranking of sports teams!