EECS E6690: SL for Bio & Info Lecture 10: Kernels, NN Nets, Unsupervised Learning: Clustering and Principal Components Analysis (PCA)

Prof. Predrag R. Jelenković
Time: Tuesday 4:10-6:40pm

Dept. of Electrical Engineering
Columbia University , NY 10027, USA
Office: 812 Schapiro Research Bldg.
Phone: (212) 854-8174
Email: predrag@ee.columbia.edu
URL: http://www.ee.columbia.edu/∼predrag

Last lecture: Unified Supervised Learning Theory

In general, all the supervised leaning problems can be written as

$$\hat{f}^* = \arg\min_{\hat{f}\in\mathcal{H}} \sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) + \lambda\Omega(\hat{f}) \tag{1}$$

where L is a general loss function, and $\lambda\Omega(\hat{f})$ is the penalty, and $\mathcal{H}$ is the space of approximation functions that we are considering.

- Example: linear functions $\hat{f}(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$
  - $\mathcal{H}$: set off all p-dimensional linear functions
  - Ridge penalty - square of the ℓ_2 norm
  $$\Omega(\hat{f}) =< \hat{f}, \hat{f} >= \|f\|^2 = \sum_{j=1}^{p} \beta_j^2$$
  - Lasso penalty - ℓ_1 norm
  $$\Omega(\hat{f}) = \|\hat{f}\|_1 = \sum_{j=1}^{p} |\beta_j|$$
  - Square loss: $L(y_i, \hat{f}(x_i)) = (y_i - \hat{f}(x_i))^2$

Last lecture: $\hat{f}$ in Kernel Hilbert Spaces

Already used it with SVM: here some more details

- Hilbert space $\{\mathcal{H}\}$: natural generalization of Euclidian spaces.
- It has an inner product: $< f, g >$, for any $f, g \in \mathcal{H}$, which allows computing angles between the elements of $\{\mathcal{H}\}$.
  In particular, $< f, g >= 0$, then f, g are orthogonal.
- Norm/distance: Norm $\|f\| = \sqrt{< f, f >}$, and
  distance $d(f, g) = \|f - g\| = \sqrt{< f - g, f - g >}, \qquad f, g \in \mathcal{H}$
- Euclidian example: if $x, y \in \mathbb{R}^d$, then $< x, y >= x_1 y_1 + \cdots + x_p y_p$

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}, \quad d(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2}$$

- Example - matrices: $< A, B >=$ trace$(A^\top B)$
- Example - random variable: $< X, Y >=$ cov(X, Y)

Reproducing Kernel Hilbert Spaces (RKHS)

RKHS - $\mathcal{H}_k$: Subset of Hilbert spaces with really nice properties

- Kernel is a positive definite function $k(x, y)$ (i.e., $\sum \alpha_i \alpha_j k(x_i, x_j) \geq 0$, for any $\alpha_i, \alpha_j, x_i, x_j$)
- **Reproducing property**: if $f \in \mathcal{H}_k$, then

$$< f, k(\cdot, x) >= f(x)$$

- Each kernel, $k(x, y)$, defines uniquely the Hilbert space, $\mathcal{H}_k$ Hence, in this space, $\mathcal{H}_k$, all we need to know if the kernel!
- Approximation functions in this space $f(x) \in \mathcal{H}_k$ can be written as

$$f(x) = \sum_i \beta_i k(x, x_i)$$

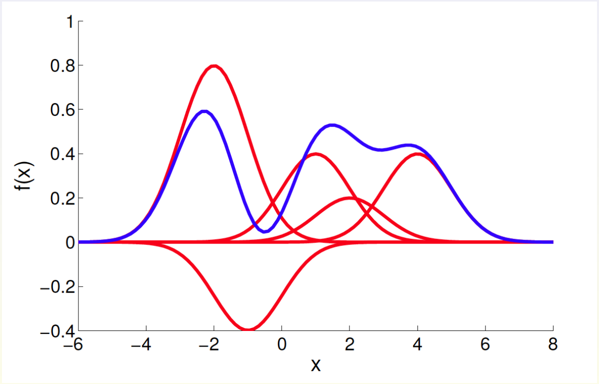
- Inner product of $f(x) = \sum_i \beta_i k(x, x_i)$ and $g(x) = \sum_j \alpha_j k(x, x_j)$

$$< f, g >= \sum_{i,j} \alpha_i \beta_j k(x_i, x_j)$$

Example: Gaussian (Radial) Kernel

Gaussian kernel in 1D: $k(x, y) = e^{\frac{-(x-y)^2}{2\sigma^2}}$
Typical function: sum of weighted Gaussian "blobs" - blue line below

$$f(x) = \sum_i \beta_i k(x, x_i) = \sum_i \beta_i e^{\frac{-(x-x_i)^2}{2\sigma^2}}$$



$\mathcal{H}_k$ is a linear space, but $f(x)$ is very nonlinear!

Generalized Learning in RKHS

Here we optimize over all approximation function in $\mathcal{H}_k$

$$\hat{f}^* = \arg\min_{\hat{f}\in\mathcal{H}_k} \sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) + \lambda\Omega(\|\hat{f}\|^2_{\mathcal{H}_k}) \tag{2}$$

Representer Theorem For any loss function L and any strictly increasing function $\Omega : \mathbb{R} \to \mathbb{R}$, an optima solution, $\hat{f}^*$, of Equation (2) has a form

$$\hat{f}^*(x) = \sum_{i=1}^{n} \beta_i^* k(x, x_i)$$

$\hat{f}^*$ has a finite representation (!) even though $\mathcal{H}_k$ can be (is) infinite.
(Puts much of the supervised learning under the same umbrella.
Drawback: works only with ℓ_2 norm, $\|\hat{f}\|_{\mathcal{H}_k}$ - doesn't work for Lasso.)

Generalized Ridge

For quadratic loss function

$$\hat{f}^* = \arg\min_{\hat{f}\in\mathcal{H}_k} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2 + \lambda \|\hat{f}\|^2_{\mathcal{H}_k} \tag{3}$$

The coefficients, β_i^*, of the optimizer

$$\hat{f}^*(x) = \sum_{i=1}^{n} \beta_i^* k(x, x_i)$$

are explicitly given by

$$\boldsymbol{\beta}^* = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

where $\boldsymbol{K}$ is a square matrix with elements $K_{ij} = k(x_i, x_j)$, $\boldsymbol{I}$ is an identity matrix, and $\boldsymbol{y}$ is the column vector $(y_1, \ldots, y_n)^\top$.

Neural Networks Verus Kernel Methods

- Similarity: both parametric methods
- Difference: fixed basis for kernels, while NN is discovering basis/features

Neural networks:

- Rich $\mathcal{H}$: Provides a versatile parametric class of functions
  - Universal function approximation
  - Difficult to train: non-convex optimization
  - Often accurate predictions, but difficult to interpret
- Automatic feature/basis extraction
  - Traditional Feature Engineering approach: expert constructs feature mapping $\phi : \mathcal{X} \to \Phi$. Then, apply machine learning to find a linear predictor on $\phi(\mathbf{x})$.
  - "Deep learning" approach: neurons in hidden layers can be thought of as features that are being learned automatically from data
  - Shallow neurons corresponds to low level features, while deep neurons correspond to high level features

General Parametric Supervised Learning

- $\mathcal{H}$ is a parametric class of functions $f(w, x), w \in \mathcal{W}, w = (w_1, \ldots, w_k)$
    - Examples: Generalized Ridge, or neural networks
- Finding $f \in \mathcal{H}$ is equivalent to finding $w \in \mathcal{W}$

Hence, our general supervised learning problem can be formulated in terms of w as (say, $x_i \in \mathbb{R}^p, y_i \in \mathbb{R}^m$,$\ell : \mathbb{R}^{p+m} \to \mathbb{R}$ - loss function)

$$\hat{w} = \arg \min_{w \in \mathcal{W}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(w, x_i)) + \lambda R(w) \tag{4}$$

How do we solve the preceding problem?

- When the problem is convex and we are lucky, we can find an explicit $\hat{w}$ by solving (e.g., generalized (Kernel) Ridge regression)

$$\frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(w, x_i)) + \lambda R(w) \right) = 0, \qquad i = 1, \ldots, k.$$

Parametric Supervised Learning: Numerical Optimization

Let us denote the objective function

$$F(w) := \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(w, x_i)) + \lambda R(w)$$

In general, we use the following numerical algorithm:

1. Initialization: Pick an initial value w_0, possibly random
2. Iteration: Keep updating w in small steps Δw

   $$w_{n+1} = w_n + \Delta w,$$

   such that $F(w_{n+1}) < F(w_n)$.
   Stoping criteria: $F(w_n) - F(w_{n+1}) < \epsilon$.

For the preceding procedure to find a local minimum

- We need to find a direction where F has a maximum decrease/steepest descent
- Avoid getting stuck on a flat surfaces, say flat saddle point

If F is convex, we can find a global minimum.

Recall Multi Calc: Gradient and Directional Derivatives

- Let $f(x) : \mathbb{R}^n \to \mathbb{R}, x = (x_1, \ldots, x_n)$
- **Gradient** is a vector of partial derivatives (assuming they exist)

$$\nabla f(x) := \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

- Let $u = (u_1, \ldots, u_n)$ be a unit vector ($\|u\|_2^2 = \sum u_i^2 = 1$)
- **Directional derivative** of $f(x)$ in direction u is

$$D_u f(x) := \frac{d}{dt} f(x+ut) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} u_i = \nabla f(x) \cdot u = \|\nabla f(x)\|_2 \cos\theta,$$

  where $y \cdot z = \langle y, z \rangle$ is the dot product and θ is the angle between $\nabla f(x)$ and u.
- Hence, $-\|\nabla f(x)\|_2 \leq D_u f(x) \leq \|\nabla f(x)\|_2$, i.e.,
  $\nabla f(x)$: direction of steepest ascent/max increase of $f(x)$
  $-\nabla f(x)$: direction of steepest descent/max decrease of $f(x)$
  $\nabla f(x)$ is perpendicular to level curves $f(x) = c$ (prove this)

Gradient Descent Algorithm

GD Algorithm is a discrete linear approximation to Equation (**??**)

$$\frac{\mathbf{x}(t+\Delta t)-\mathbf{x}(t)}{\Delta t} \approx \frac{d\mathbf{x}(t)}{dt} = -\eta \nabla f(\mathbf{x}(t))$$

or equivalently (with a small abuse of notation), discrete $t = 0, 1, 2, \ldots$,

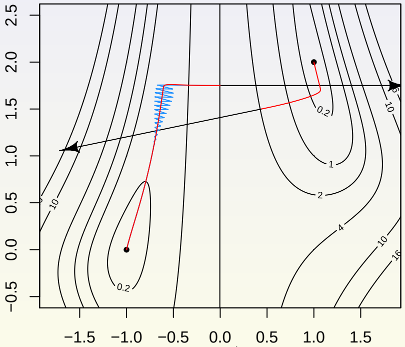
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla f(\mathbf{x}^{(t)})$$

- Hence, after initialization at $\mathbf{x}^{(0)}$, the GD Algorithm follows the preceding iteration to a local minimum
- Stopping criterion (could be): $|f(\mathbf{x}^{(t+1)}) - f(\mathbf{x}^{(t)})| < \epsilon$

Adaptive GD (AdaGrad): modifies the learning rate η_t in each iteration t

More Interesting Landscape

- $f(x) = (x_1^2 - 1)^2 + (x_1^2 x_2 - x_1 - 1)^2$
- Gradient
$$\nabla f(x) = \begin{bmatrix} 4x_1(x_1^2 - 1) + 2(2x_1 x_2 - 1)(x_1^2 x_2 - x_1 - 1) \\ 2x_1^2(x_1^2 x_2 - x_1 - 1) \end{bmatrix}$$
- Oscillations in "narrow valleys"

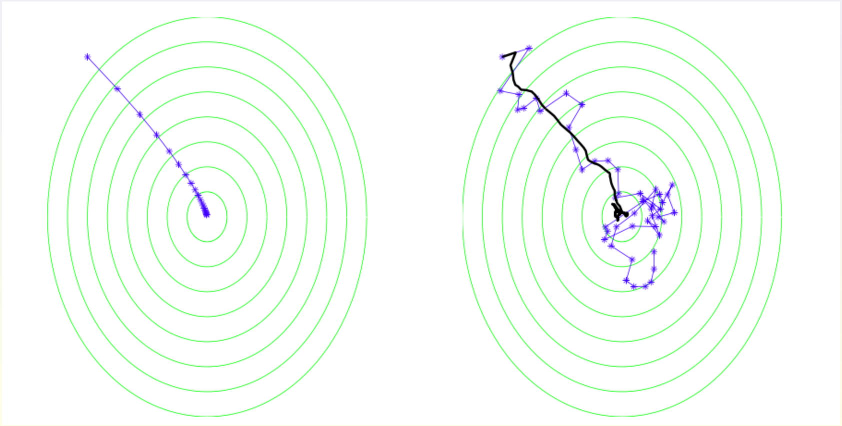


Motivation for momentum: remembers/averages previous Δx

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla f(\mathbf{x}^{(t)}) + \mu_t(\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)})$$

Stochastic Gradient Descent

- Dates back to Robbins and Monroe (1951).
- Stochastic gradient is an unbiased estimator of the gradient
- Stochastic versus regular gradient descent



Stochastic Gradient Descent

- Stochastic approximation of gradient descent
- Function (typically encountered in learning)

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

- Computationally expensive gradient for large n
- Approximation: pick a random subset $\mathcal{S} \in [1, n]$

$$\frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \nabla f_i(x)$$

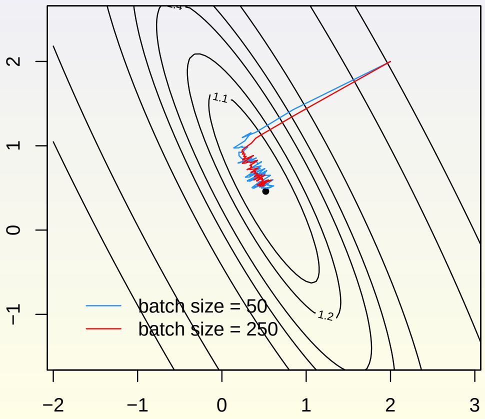
- $\mathcal{S}$ - called batch/mini-batch
- Example
  - n scalar data points $x_1, x_2, \cdots, x_n$
  - objective

$$\min_c \frac{1}{n}\sum_{i=1}^{n}(x_i - c)^2$$

SGD Example: Linear Regression

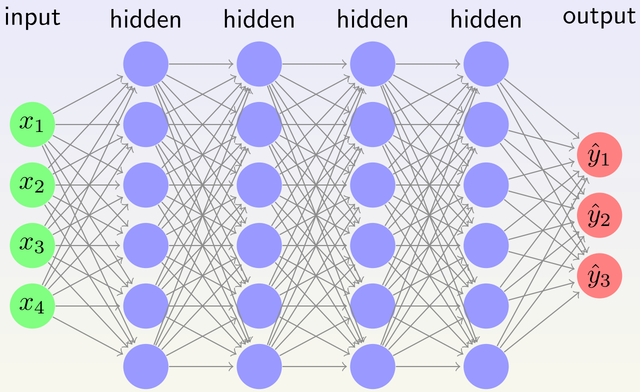
- Data: $(x_i, y_i)_{i=1}^n$, $n = 10^5$
- Loss function: $L(\beta, x, y) = \frac{1}{n}\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$
- Stochastic gradient

$$\nabla_\beta \hat{L}(\beta, x, y) = \frac{2}{|\mathcal{S}|}\sum_{i \in \mathcal{S}} \begin{bmatrix} (\beta_0 + \beta_1 x_i - y_i) \\ x_i(\beta_0 + \beta_1 x_i - y_i) \end{bmatrix}$$



Deep Neural Networks, a.k.a. Multilayer Perceptrons

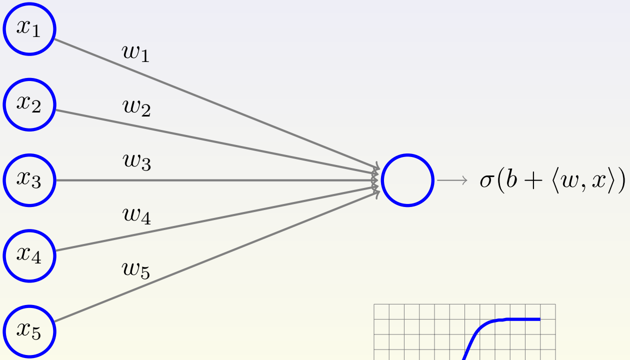
- Feed-forward network



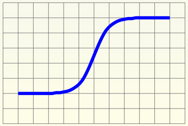
- Designed to mimic the function of neurons
- Blue nodes: activation functions/neurons
- Depth $=$ lengths of a longest path
- Deep network: depth ≥ 3
- Very successful in solving practical problems

A Single Artificial Neuron

- A single neuron function: $\mathbf{x} \mapsto \sigma(b + \langle \mathbf{w}, \mathbf{x} \rangle)$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is called the activation function of the neuron. Inner/dot product: $\langle \mathbf{w}, \mathbf{x} \rangle = \sum x_i w_i$.
- More compact notation $\langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}} \rangle$, where $\tilde{\mathbf{x}} = (1, \mathbf{x}), \tilde{\mathbf{w}} = (b, \mathbf{w})$



- E.g., σ is a sigmoidal function



Common Activation Functions/Perceptrons

- step function

$$\sigma(x) = 1_{\{x>0\}} \qquad \sigma'(x) = 0,\ x \neq 0$$

- logistic

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

- rectified linear unit (ReLU)

$$\sigma(x) = \max\{x, 0\} \qquad \sigma'(x) = 1_{\{x>0\}},\ x \neq 0$$

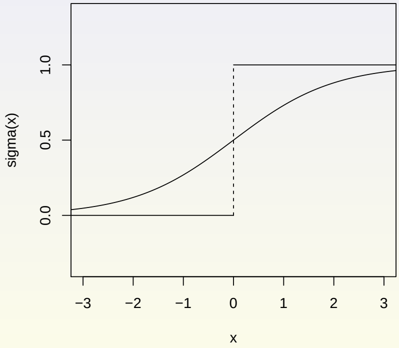
- soft-plus

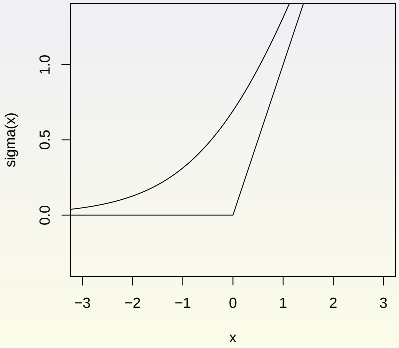
$$\sigma(x) = \log(1+e^{x}) \qquad \sigma'(x) = \frac{1}{1+e^{-x}}$$

Comparing Activation Functions

Logistic and soft-plus have continuous derivatives in comparison to step function and ReLU

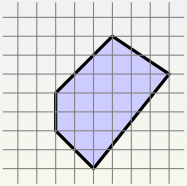
- Positive derivative avoids vanishing gradient





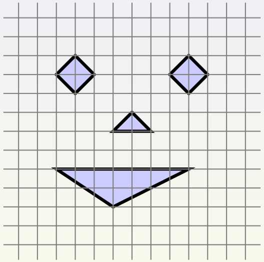
Example

- Single neuron is a binary half-space classifier: $\mathrm{sign}(w \cdot x + b)$
- 2 layer networks can express intersection of halfspaces



Example

- 3 layer networks can express unions of intersection of halfspaces



How to train neural network?

- Neural nets: excellent hypothesis class, but difficult to train
- Main technique: Stochastic Gradient Descent (SGD)
- Not convex, no guarantees, can take a long time, but:
  - Often still works fine, finds a good solution
  - With some luck:)

Stochastic Gradient Descent (SGD) for Neural Networks

Common Training Ideas:

- Random initialization: rule of thumb, $w[u \to v] \sim U[-c, c]$ where $c = \sqrt{3/|\{(u', v) \in E\}|}$ (or small Gaussian instead of $U[-c, c]$)
- Update step with Nesterov's momentum: Initialize $\theta = 0$ and:

$$\theta_{t+1} = \mu_t \theta_t - \eta_t \tilde{\nabla} L(w_t + \mu_t \theta_t)$$
$$w_{t+1} = w_t + \theta_{t+1}$$

  where:
  μ_t is momentum parameter (e.g. $\mu_t = 0.9$ for all t)
  η_t is learning rate (e.g. $\eta_t = 0.01$ for all t)
  $\tilde{\nabla} L$ is an estimate of the gradient of L based on a small set of random examples (often called a "minibatch")

- Efficient gradient calculation: Backpropagation

Interesting Questions in Deep Learning

Why does it work well?

- Mathematically, it is not well understood.

This motivated the development of a new class, called

- E6699: Mathematics of Deep Learning

I thought it in Spring'19, Spring'21, Spring'22, and will teach again in Spring'23.
Here, some interesting topics/questions that the course addressed:
(Maybe I'll discuss further some of the topics next week.)

- Expressiveness of neural nets
- Expressive power of deep learning: why is depth good, depth separation results
- Global versus local optimality
- Generalization error: Why DL models generalize well?
- Random initialization
- Wide nets and connection to Kernels
- Etc.

Unsupervised Learning

There is no input-output relationship, $y = f(x)$, i.e. there is no **y**.

- We only have a bunch of points $(x_1, x_2, \ldots, x_n)$
- The problem has less structure
- We are trying to discover a structure - maybe more interesting

Typical questions and approaches

- Clustering - grouping data points
- Principal Component Analysis (PCA): used of preprocessing and visualization
- Association Rules - discovering relationships between data points
- Community Detection or Graph Clustering: e.g., discovering communities on Face book
- Ranking: e.g., Google's PageRank algorithm

Clustering

Clustering looks to find homogeneous subgroups among the observations

Approaches

- Flat: we set in advance how many clusters are there
  - K-means
- Hierarchical: bottom up (agglomerative) merging or top down splitting
  - Bottom up: tree-based representation - dendogram

K-Means Clustering

K-Means - simple and intuitive
Notation

- n - number of data points
- K - fixed and predetermined number of clusters (weakness of this procedure)
- $C_1, \ldots, C_K$: sets containing the indices of the observations in each cluster.
- These sets satisfy two properties:
  1. $C_1 \cup C_2 \cup \ldots \cup C_K = \{1, ..., n\}$ - each observation in at least one clusters.
  2. $C_1 \cap C_2 \cap \ldots \cap C_K = \emptyset$ - clusters are nonoverlapping

K-Means Clustering

Must have a measure of closeness/distance between points.
Examples of distances (we have seen them before)

- Square of Euclidian distance - ℓ_2 norm
$$d(x_i, x_j) = \sum_{l=1}^{p} (x_{il} - x_{jl})^2$$
- ℓ_1 norm - also known Manhattan street norm
$$d(x_i, x_j) = \sum_{l=1}^{p} |x_{il} - x_{jl}|$$
- Correlation
$$\rho(x_i, x_j) = \frac{\sum_{l=1}^{p} (x_{il} - \bar{x}_i)(x_{jl} - \bar{x}_j)}{\sqrt{\sum_{l=1}^{p} (x_{il} - \bar{x}_i)^2 \sum_{l=1}^{p} (x_{jl} - \bar{x}_j)^2}}$$
- Many other distance measures could work, e.g. mutual information, or those we have seen in Hilbert spaces
- Carefully choosing the distance measure can make a difference

K-Means Clustering

Next, define a weight of a cluster: the closeness of pints in a cluster

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,j \in C_k} d(x_i, x_j)$$

where $|C_k|$ denotes the number of points in C_k.
$W(C_k)$ is the average distance in the cluster.

Now, the clustering problem is to find $C_1, \ldots, C_K$ that minimize

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k)$$

K-Means Clustering

Euclidian example

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,j \in C_k} \sum_{l=1}^{p} (x_{il} - x_{jl})^2$$

where $|C_k|$ denotes the number of points in C_k.
$W(C_k)$ is the average Euclidean distance

Now, the clustering problem is explicitly given as

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,j \in C_k} \sum_{l=1}^{p} (x_{il} - x_{jl})^2$$

K-Means Algorithm

Algorithm for Euclidean distance

1. Initialization: Randomly assign a number, from 1 to K, to each of the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster centroid. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
   (b) Assign each observation to the cluster whose centroid is closest in Euclidean distance.

Properties

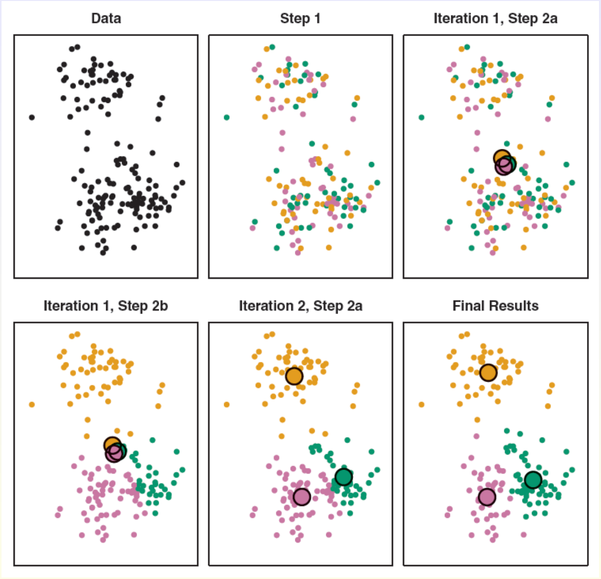
- It converges due to this identity

$$\frac{1}{|C_k|}\sum_{i,j\in C_k}\sum_{l=1}^{p}(x_{il}-x_{jl})^2 = 2\sum_{i\in C_k}\sum_{l=1}^{p}(x_{il}-\bar{x}_{kl})^2$$

  $\bar{x}_{kl}=\frac{1}{|C_k|}\sum_{i\in C_k} x_{il}$ is the l-coordinate of the cluster centroid
- Problem: finds only a local minimum
- Find good starting points and repeat a few times

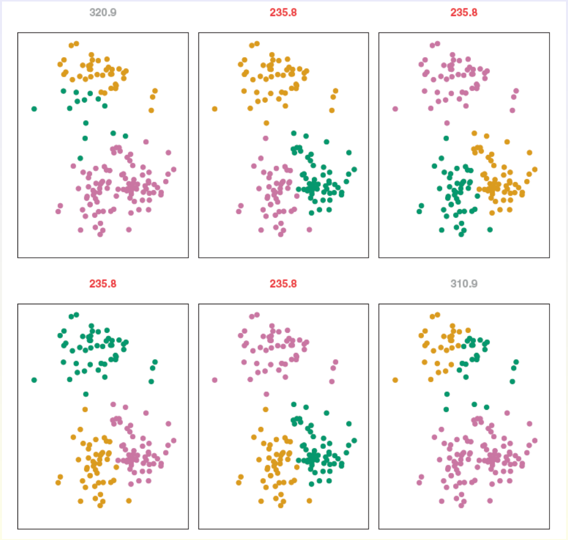
K-Means: Example

$K = 3$



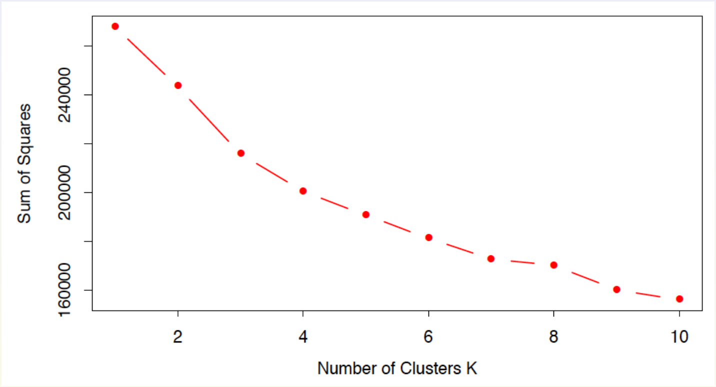
K-Means: Example

$K = 3$ - Different, possibly bad, local minima



K-Means: What is the best K?

No good analytical procedure
Heuristics: Look for a **kink** on the plot of the total within-cluster sum of squares. **May not work** in general.



Total within-cluster sum of squares for human tumor microarray data

Hierarchical Clustering

- Bottom up: Start with each point being a cluster
- Decide on a distance of dissimilarity
- Compute the distance between all pairs of points, and merge the two closest into one class.
- **Q**: How do we measure the distance between sets?
  - This is called linkage

Types of linkage

- **Complete**: Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between 2 clusters and pick the largest
- **Single**: Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between 2 clusters and pick the largest
- **Average**: Mean intercluster dissimilarity. Compute all pairwise dissimilarities between record the average
- **Centroid**: Distance between the centroids of two clusters

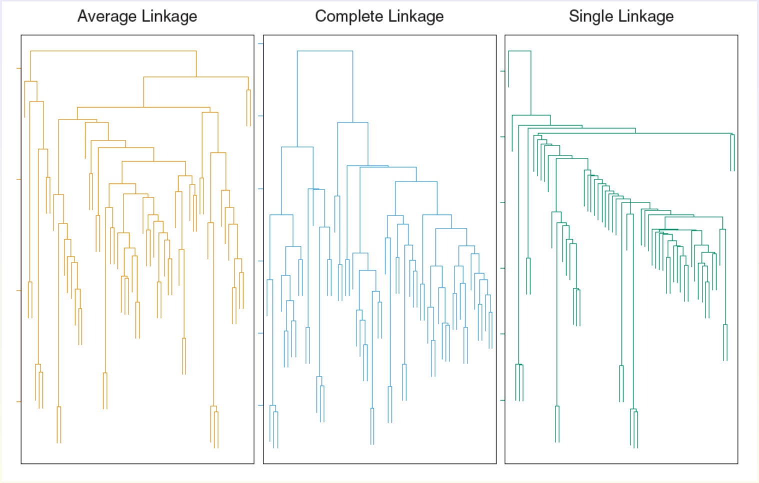
Hierarchical Clustering Algorithm

Algorithm

1. Begin with n observations and a measure (such as Euclidean distance) of all the $n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.
2. For $i = n, n-1, ..., 2$:
   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters.
   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

Hierarchical Clustering: Impact of Linkage

Average and complete linkage tend to yield more balanced clusters.



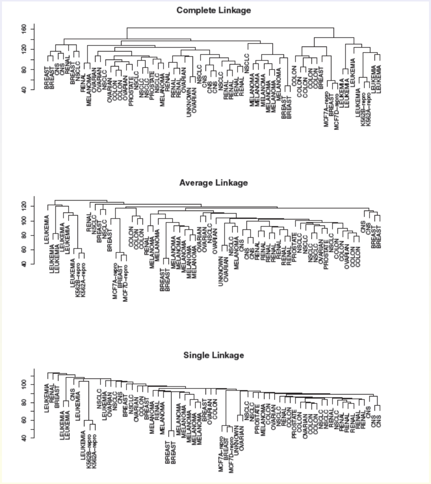
NCI60 Cancer Data

- 64 cancers: 6,830 gene
- >library(ISLR)
  >nci.labs=NCI60$labs
  >nci.data=NCI60$data

```
> nci.labs[1:4]
[1] "CNS"    "CNS"    "CNS"    "RENAL"
> table(nci.labs)
nci.labs
     BREAST         CNS       COLON K562A-repro K562B-repro
          7           5           7           1           1
   LEUKEMIA MCF7A-repro MCF7D-repro    MELANOMA       NSCLC
          6           1           1           8           9
    OVARIAN    PROSTATE       RENAL     UNKNOWN
          6           2           9           1
```

NCI60 Dendogram with Euclidean Distance

Complete and average linkage tend to yield evenly sized clusters whereas single linkage tends to yield extended clusters to which single leaves are fused one by one.



Principal Components Analysis (PCA)

Principal Components Analysis - unsupervised approach (no use of Y)

- Finding directions along which data is located - direction of largest variance
- These directions define lines/subspaces that are close to the "data cloud"
- Can be used for visualization/understanding data
- Can be used as preprocessing for supervised learning

First Principal Components

- Look for the linear combination of the sample feature of the form

$$z_{i1} = \phi_{11} x_{i1} + \phi_{21} x_{i2} + \cdots + \phi_{p1} x_{ip}$$

- *First loading vector*: $\boldsymbol{\phi}_1 = (\phi_{11}, \phi_{21}, \ldots \phi_{p1})$
- Assume x_i-s are centered ($\sum x_i = 0$)
- Look for $\boldsymbol{\phi}_1$ that has the largest sample variance, i.e.

$$\max_{\boldsymbol{\phi}_1} \frac{1}{n} \sum_{i=1}^{n} z_{i1}^2 = \max_{\boldsymbol{\phi}_1} \frac{1}{n} \sum_{i=1}^{n} (\phi_{11} x_{i1} + \phi_{21} x_{i2} + \cdots + \phi_{p1} x_{ip})^2$$

subject to $\sum_{j=1}^{p} \phi_{j1}^2 = 1$ (i.e., $\boldsymbol{\phi}_1$ is a unit vector)

- z_{i1} are the *scores*
- This optimization problem can be solved via eigen-decomposition

First Principal Components: Geometric interpretation

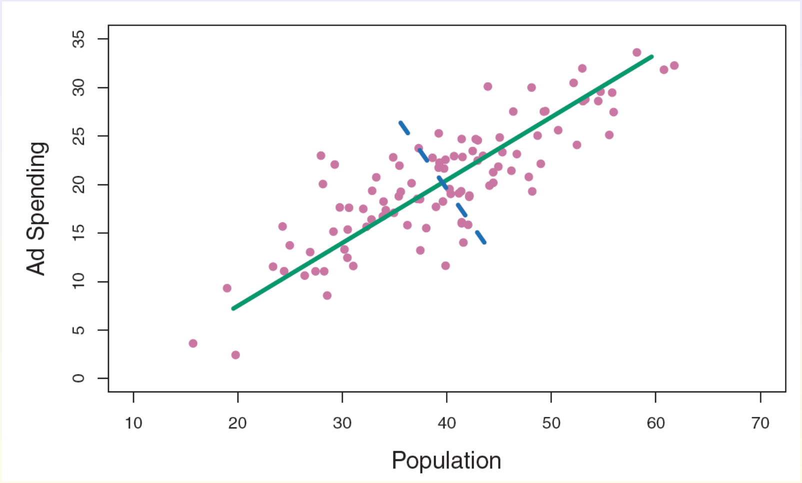
- Loading vector ϕ_1 represents the direction along which the data varies the most
- If we project $x_1, \dots, x_n$ onto ϕ_1, the projected values are the PC scores z_{i1} since
$$z_{i1} =< x_i, \phi_1 >$$

Second and higher principal components

- After ϕ_1 has been determined, we look for ϕ_2 in a similar way, but with the additional constraint that ϕ_1, ϕ_2 are uncorelated
$$< \phi_1, \phi_2 >= 0$$
- We continue this procedure until we find as many PC as we want

PCA: Example

Two PC-s: Solid line: First PC; Dashed line: Second PC



PCA: Good for High Dimensional Data - Large p

- Dimensionality reduction - e.g. gene expression data
- Assume we compute M principal components
- The best M-dimensional approximation to x_{ij}

$$x_{ij} = \sum_{m=1}^{M} z_{im}\phi_{jm}$$

How good is PC approximation?

- Proportion of Variance Explained (PVE)
- For each component, m, PVE is equal to

$$\frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{jm}x_{ij}\right)^2}{\sum_{ij}x_{ij}^2}$$

PCA is an Eigen-Decomposition Problem

- In general finding k-principal components is equivalent to finding a $k \times p$ matrix such that
$$\boldsymbol{z}_i = W\boldsymbol{x}_i$$
- We also want linear recovery, i.e., to find a matrix U, such that
$$\hat{\boldsymbol{x}}_i = U\boldsymbol{z}_i = UW\boldsymbol{x}_i \approx \boldsymbol{x}_i$$
- Hence, this reduces to an optimization problem
$$\arg\min_{W,U} \sum_{i=1}^{n} \|\boldsymbol{x}_i - UW\boldsymbol{x}_i\|^2 \tag{5}$$

Theorem Let $A = \sum_{i=1}^{n} \boldsymbol{x}_i\boldsymbol{x}_i^\top$ and let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ be the k leading eigenvectors of A. Then, the solution to the PCA problem is to set the columns of U to be $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ and to set $W = U^\top$.

Other types of linear dimensionality reduction:

- **Compressed sensing**: Also finds a matrix W with special properties and compresses $\boldsymbol{x}$ into $W\boldsymbol{x}$. Reconstruction solves a linear program.

PCA is an Eigen-Decomposition Problem: Proof

Lemma Let (U, W) be a solution to Equation (5). Then, the columns of U are orthonormal: namely, $U^\top U = I$ and $W = U^\top$.

Proof:

- Consider space $R = \{UW\boldsymbol{x} : \boldsymbol{x} \in \mathbb{R}^p\} \subset \mathbb{R}^p$
- Let V be orthonormal matrix ($V^\top V = I$) whose columns are the basis of R
- Hence, each vector in R can be written as $V\boldsymbol{z}, \boldsymbol{z} \in \mathbb{R}^k$ and

  $$\|\boldsymbol{x} - V\boldsymbol{z}\|_2^2 = \|\boldsymbol{x}\|_2^2 + \boldsymbol{z}^\top V^\top V\boldsymbol{z} - 2\boldsymbol{z}^\top(V^\top\boldsymbol{x})$$

  which, by taking derivative w.r.t. $\boldsymbol{z}$ is minimized for $\boldsymbol{z} = V^\top\boldsymbol{x}$, implying

  $$VV^\top\boldsymbol{x} = \arg\min_{\tilde{\boldsymbol{x}} \in R} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2$$

  and furthermore

  $$\arg\min_{U,W} \sum_{i=1}^{n} \|\boldsymbol{x}_i - UW\boldsymbol{x}_i\|_2^2 \geq \arg\min_{V} \sum_{i=1}^{n} \|\boldsymbol{x}_i - VV^\top\boldsymbol{x}_i\|_2^2$$

PCA: Proof of Theorem

- Based on the preceding lemma, optimization in Equation (5) can be written as
$$\arg \min_{U \in \mathbb{R}^{p,k} : U^\top U = I} \sum_{i=1}^{n} \|\boldsymbol{x}_i - UU^\top \boldsymbol{x}_i\|_2^2$$
- Next, since $U^\top U = I$, it follows that
$\|\boldsymbol{x} - UU^\top \boldsymbol{x}\|_2^2 = \|\boldsymbol{x}\|_2^2 - \text{trace}(U^\top \boldsymbol{x}\boldsymbol{x}^\top U), \quad (y^\top y = \text{trace}(yy^\top))$
which means that one can equivalently solve
$$\arg \max_{U : U^\top U = I} \text{trace}(U^\top \sum_{i=1}^{p} \boldsymbol{x}_i \boldsymbol{x}_i^\top U)$$
- Let $V \Lambda V^\top = A := \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^\top$, be the spectral decomposition of A with $VV^\top = V^\top V = I$, and Λ a diagonal matrix whose diagonal elements are the eigenvalues of A; assume that $\lambda_1 \geq \cdots \geq \lambda_p$. Then,
$$U^\top A U = U^\top V \Lambda V^\top U =: B^\top \Lambda B, \qquad B := V^\top U$$

PCA: Proof of Theorem

- Hence,

$$\text{trace}(U^\top A U) = \sum_{j=1}^{p} \lambda_j \sum_{i=1}^{k} B_{j,i}^2$$

- Also, $B^\top B = I$, which implies

$$\sum_{j=1}^{p} \sum_{i=1}^{k} B_{j,i}^2 = k.$$

- Next, let $\tilde{B} \in \mathbb{R}^{p,p}$ be a matrix whose first k columns are the columns of B. Then,

$$\beta_j := \sum_{i=1}^{k} B_{j,i}^2 \leq \sum_{i=1}^{p} \tilde{B}_{j,i}^2 = 1$$

PCA: Proof of Theorem

- Therefore

$$\text{trace}(U^\top AU) \leq \max_{\boldsymbol{\beta}\in[0,1]^p:\|\boldsymbol{\beta}\|_1\leq k} \sum_{j=1}^{p} \lambda_j \beta_j \leq \sum_{j=1}^{k} \lambda_j$$

- Furthermore, this upper bound is achieved when U is chosen to be the matrix whose columns are equal to the k leading eigenvectors of A, i.e., direct calculation yields

$$\text{trace}(U^\top AU) = \sum_{j=1}^{k} \lambda_j,$$

  which completes the proof.

More details can be found in Chapter 23 of ML book by Shalev-Shwartz & Ben-David, 2014: `https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf`

Kernel PCA

- PCA is based on orthogonal projections
- So we can use Kernels and Hilbert spaces
- Recall that our approximation functions are of the form

$$g_1(x) = \sum_{i=1}^{n} \phi_{i1} k(x, x_i)$$

- Hence, the optimization is

$$\max_{g_1 \in \mathcal{H}_k} \text{Sample Var}(g_1(X)) \quad \text{subject to} \quad \|g_1\|_{\mathcal{H}_k} = 1$$

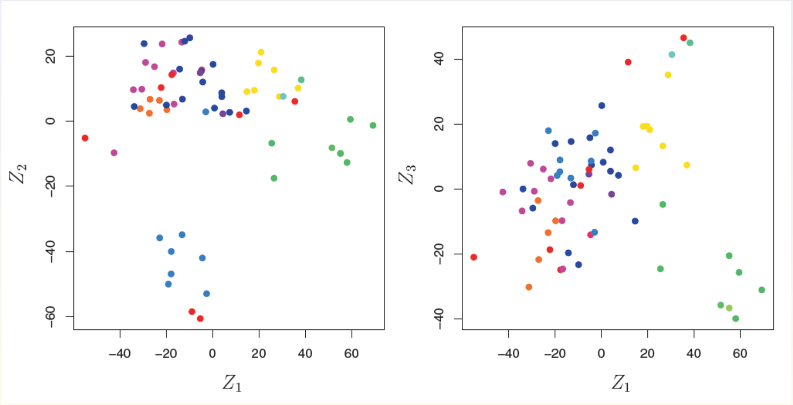
- Now the approximation "directions" are not lines any more
- This is also eigen-decomposition problem

PCA on the NCI60 Data

- First *scale* the data
  ```
  > pr.out=prcomp(nci.data, scale=TRUE)
  ```
- Then, **assign a color** to each of the 64 cancer cell lines
  ```
  Cols=function(vec){
  + cols=rainbow(length(unique(vec)))
  + return(cols[as.numeric(as.factor(vec))]) }
  ```
- We now can plot the principal component score vectors
  ```
  > par(mfrow=c(1,2))
  > plot(pr.out$x[,1:2], col=Cols(nci.labs), pch=19,
    xlab="Z1",ylab="Z2")
  > plot(pr.out$x[,c(1,3)], col=Cols(nci.labs), pch=19,
    xlab="Z1",ylab="Z3")
  ```

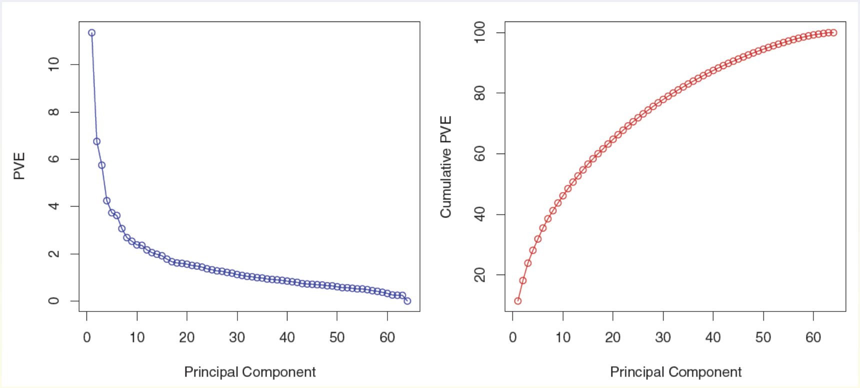
PCA on the NCI60 Data

- Observations belonging to a single cancer type tend to lie near each other in this low-dimensional space.
- Not possible to visualize data without PCA, $\binom{6832}{2}$ scatter plots (!)



Principal Value Explained (PVE) on NCI60 Data

- First 7 PC-s explain 40% of data
- 70 PC-s explain 100% of data
- Compare 70 to $p = 6832(!)$



Reading

- ESL: Chapter 11, 14; ISL: Chapter 10

Homework: Work on the final project. Today is the deadline for group formation and project selection.

Reading on Deep Learning:

- Chapters 14& 20 in: Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014.
- Chapter 6 in: Deep Learning, I. Goodfellow and Y. Bengio and A. Courville, MIT Press, 2016. `http://www.deeplearningbook.org`
- Software - Tensor Flow in R: `https://tensorflow.rstudio.com`

Optional reading on Kernels (RKHS):
(These books are available online through CU Library)

1. Chapters 1, 2, 13-16 in
   B. Schölkopf and A. J. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.
2. Chapters 1, 5.3, 6, 7 in (this book is mathematically advanced)
   A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics.* Kluwer, 2004.