# Introduction

Health care analytics in the presence of big data: Case studies in Python and Apache Spark

colorado school of public health

## Welcome!

- Thank you to our hosts for allowing us the opportunity to give this workshop.
- Thank you to the participants for your interest!
- Let us introduce ourselves…

## Debashis Ghosh

- Professor and Chair, Department of Biostatistics and Informatics, ColoradoSPH
- UW grad (yeah!)
- Interests in: machine learning, causal inference, integrative genomics
- Dabbler in big data/data science
- After-work interests: reading, playing violin, running

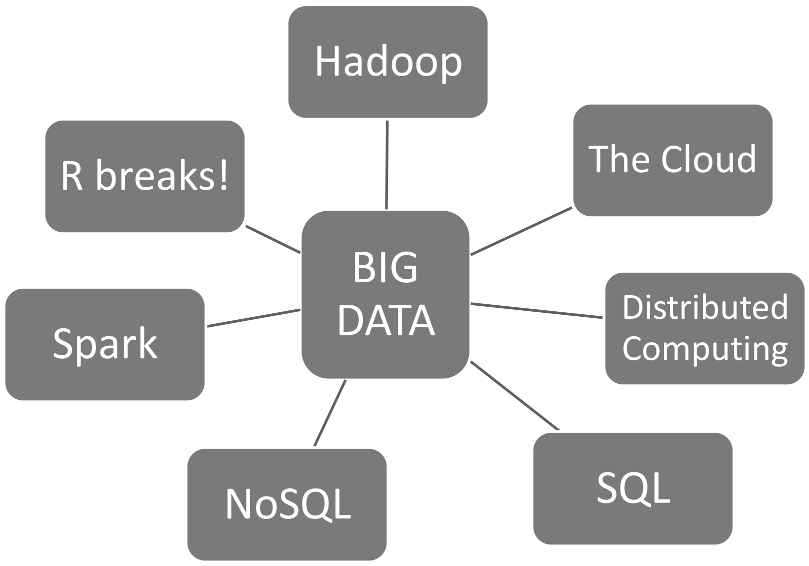
## Evan.Carey@va.gov

- Work
  - Statistician / Data Scientist
  - 5 years working in large administrative datasets with Veterans Healthcare Administration
- Coding
  - R, SAS, Python, SQL variants, Spark
- Education
  - Masters of Science in Applied Biostatistics from CSPH
  - PhD Candidate in Epidemiology
- Play
  - Tennis, Volleyball, drums ☺

## Big Data, right?

- Can someone define big data for me?
- When hear the phrase big data, what other words come to mind?

## (Collaborative Filtering Algorithm)

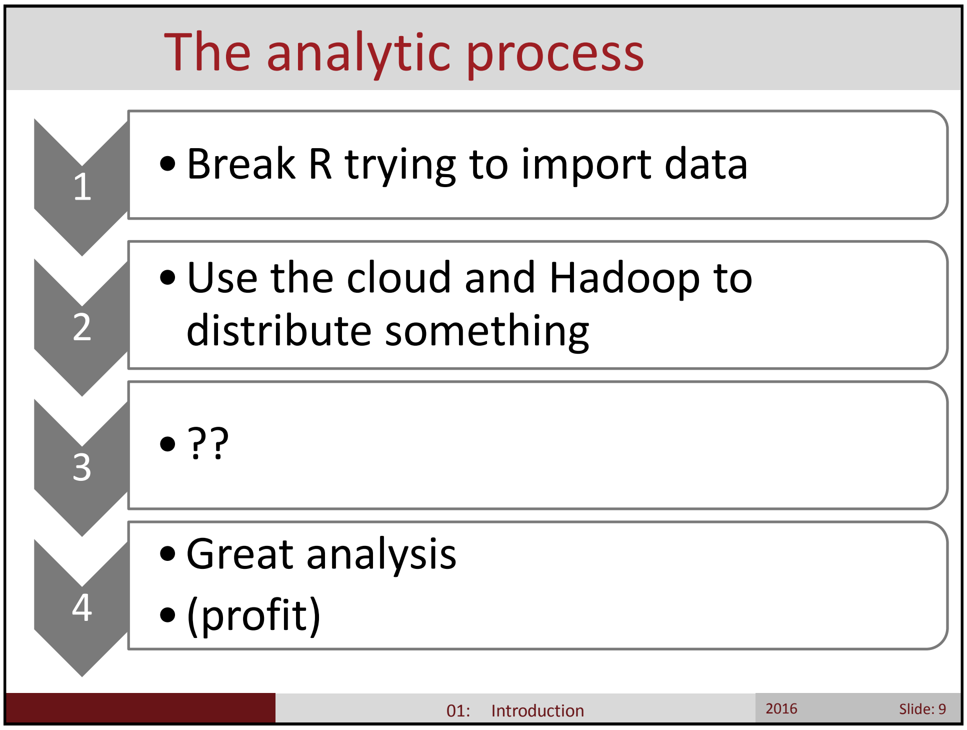


## This new thing, big data ☺

- September 1994: BusinessWeek publishes a cover story on "Database Marketing"
  - An earlier flush of enthusiasm prompted by the spread of checkout scanners in the 1980s ended in widespread disappointment: Many companies were too overwhelmed by the sheer quantity of data to do anything useful with the information... Still, many companies believe they have no choice but to brave the database-marketing frontier."

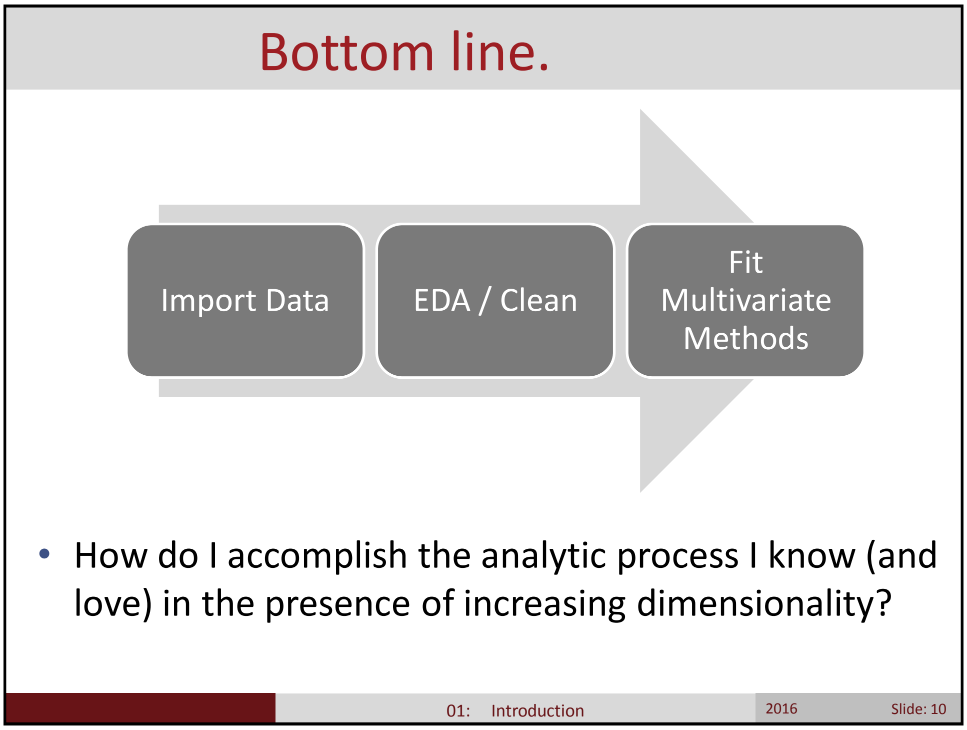
## Knowledge discovery in databases

- 1996 Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth publish "From Data Mining to Knowledge Discovery in Databases." They write:
- "Historically, the notion of finding useful patterns in data has been given a variety of names, including data mining, knowledge extraction, information discovery, information harvesting, data archeology, and data pattern processing... In our view, KDD [Knowledge Discovery in Databases] refers to the overall process of discovering useful knowledge from data, and data mining refers to a particular step in this process. Data mining is the application of specific algorithms for extracting patterns from data... the additional steps in the KDD process, such as data preparation, data selection, data cleaning, incorporation of appropriate prior knowledge, and proper interpretation of the results of mining, are essential to ensure that useful knowledge is derived from the data. Blind application of data-mining methods (rightly criticized as data dredging in the statistical literature) can be a dangerous activity, easily leading to the discovery of meaningless and invalid patterns."



## The analytic process

1. Break R trying to import data
2. Use the cloud and Hadoop to distribute something
3. ??
4. 
   - Great analysis
   - (profit)



## Bottom line.

- How do I accomplish the analytic process I know (and love) in the presence of increasing dimensionality?

## Reasonable questions

- What software environment can I use in large data?
  - What is Hadoop, or Spark?
- How can I store my data?
  - SQL versus NoSQL?
- What statistical methods should I use on my data?
- Is machine learning the answer?
  - What is machine learning…?

## Course Objectives

- Contextualize course with VM including Python, SQLite, Spark, and sample data.
- Applied understanding of data science in Python
- Applied understanding of data science in Apache Spark
- Fully define a modeling approach
- Evaluate the impact of big data on the analytic workflow
- Contrast machine learning versus “frequentist” approaches
- Focus on data management and a classification problem.

# Software Introduction

Health care analytics in the presence of big data: Case studies in Python and Apache Spark

colorado school of public health

## Virtual Machine!

- We have provided a virtual machine for this course:
  - Linux (CentOS7 Minimal install)
  - Anaconda Python Stack
  - Apache Spark
  - Sqlite
  - Some interesting datasets
- This can be run with Virtualbox
  - 8GB RAM needed.

## Data in the VM

- Group Medical Cost/Claims
  - I included the PDF description of this data source in your VM.
  - Cost data for 1.6 million patients in the late 90's.
  - I have cleaned/modified the data for ease of use.
- Simulated_Cluster_Person
  - I simulated this dataset of 10k patients
  - Some intentional data issues
  - Binary outcome of interest.

## Python for Analysis

- We will be using the Anaconda Python Distribution, Python 3.5.XX
  - This has everything we need, is open source, and is easy to install.
  - This includes…
  - The python kernel
  - Ipython (enhanced interpreter)
  - Spyder (IDE)
  - Scientific Stack of libraries (packages)
- We could build a Python stack from scratch, compiling all libraries we need…this is not trivial.

## Python Analytics Packages

- Ipython
  - Enhanced interpreter
- Numpy
  - Matlab type stuff
  - Ndarray object
- Matplotlib
  - Plotting in 2d originally (now some 3d)
- Pandas
  - Dataframe class, time series analysis
- Stat models
  - Frequentist multivariate approaches
- Scikit learn
  - Machine learning

## Basic Python

- The basic collection of objects in Python is a list
- Make a list with square brackets.

```
## This is a comment

## Create a collection (list) of numbers
x1 = [1,2,3,4]

## Create a list of characters
name = ['Evan','Debashis']
```

## Importing modules

- We bring in additional functionality by importing modules.

  Use the import statement.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## Python Analysis Overview

| | |
|---|---|
| Pandas | • Import and clean data, EDA |
| Patsy | • Create model matrices (numpy objects) |
| Numpy | • Matrices as inputs to models |
| Scikit-learn | • Pre-processing, Fit multivariate methods |

## Functional Examples

- Lets work through some python code together.
- All code is available on your virtual machine.
- You should be able to run all the code and get a feel for the environment.

# Big Data Analytics

Health care analytics in the presence of big data: Case studies in Python and Apache Spark

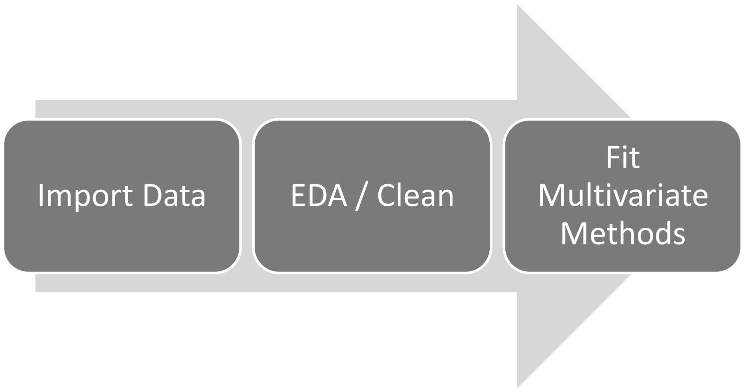
colorado school of public health

## Defining big data

- Big Data Definition:
  - Data exceeds memory
  - Data exceeds hard drive on typical workstation
  - Data exceeds hard drive space on a single server

  - I typically define “big data” functionally…whenever my environment or analytic approach must be changed due to the size of the data I am working with.

## Analytic process



- What are the issues posed by high dimensionality data in the above process?

## Possible issues

- Data will not fit into…
  - RAM
  - Hard drive
- Computations are too expensive
  - Algorithms take too long to converge
  - Algorithms fail to converge
  - Not enough memory to run algorithms!

## Vertical Scaling

- Not enough RAM
  - Buy more RAM
  - Spill to hard drive (Does R or Python do this?)
- Not enough hard drive
  - Buy bigger hard drive
- Won't fit on desktop
  - Buy server!
- Not enough server RAM
  - Upgrade server…?
- Computations take too long
  - Get faster processor!

## Horizontal Scaling

- Out of RAM/Disk space
  - Use more computers!
- Computations take too long
  - Distribute computations across multiple computers / threads
  - Parallel execution of tasks

## Distributed Computing

- Distributed Data
  - Data won't fit in one place.
  - Need to break data into pieces and store across multiple nodes.
- Distributed Computation
  - Computation is too expensive.
  - Gain overall efficiency by distributing computations

## Distributed Data

- What are some things we need in a distributed data model?
  - Fault Tolerance
  - Scalability
  - Storage (throughput)
  - Retrieval
  - Ease of analysis / merging data elements

## Data Partitions

- When we break data apart, we can intelligently partition the data according to future needs.
  - Partition by hospital ID
  - Partition by patient
  - Partition by interesting clusters…
- This will allow subsequent processing efficiency

## Approach #1: Store big, analyze small

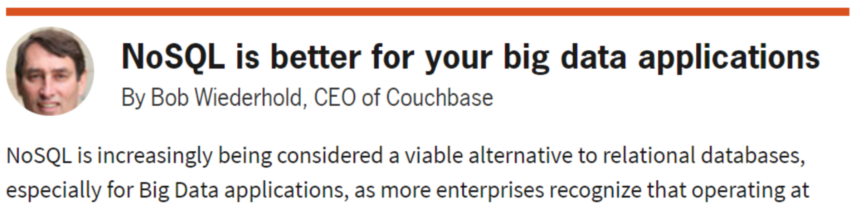
- Use a distributed data model to organize and keep large data
- Analyze subsets of the data
  - Take a 2% sample of the data
  - Bring into R/Python
  - Proceed…
- Analyze Summaries of the data
  - Reduce data with basic summary measures
  - Analyze summary measures

## Relational Database (RDBMS)

- Who knows what SQL/RDBMS is?
- Structured Query Language (SQL)
  - Came out in 1970's
- Microsoft SQL Server
- Oracle
- MySQL
- SQLite

## NoSQL

- Can someone define NoSQL?



**NoSQL is better for your big data applications**

By Bob Wiederhold, CEO of Couchbase

NoSQL is increasingly being considered a viable alternative to relational databases, especially for Big Data applications, as more enterprises recognize that operating at

http://www.networkworld.com/article/2226514/tech-debates/what-s-better-for-your-big-data-application--sql-or-nosql-.html

## Not Only SQL (NoSQL)

- Focus on Horizontal Scalability
- Don't need to fully define data elements before it is stored
- No Complex schema requirements
- Can collect many different types of data
- High throughput
- Simplest Example:
  - Key – Value pairs
  - JSON Databased

## DBMS (SQL) Versus NoSQL

| DBMS | NoSQL |
|---|---|
| • Fast Read Access<br>• Long Load Time<br>• Transaction level-fault tolerance<br>• Vertical Scalability | • Slower Random Read Access<br>• High Throughput<br>• Horizontal Scalability |

## Distributed Computations

- Work on one piece of data at a time
  - Process chunk
  - Store result
  - Work on next chunk
- What common data processing steps might not work well in this environment?
- What are the newly incurred costs associated with this approach?

## Serial Versus Parallel Computation

Result 1 → Result 2 → Result 3 → Result 4…

## Apache Hadoop

- What do you know about Hadoop?
- How popular is it?
- Advantages?
- Disadvantages?

## Apache Hadoop

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

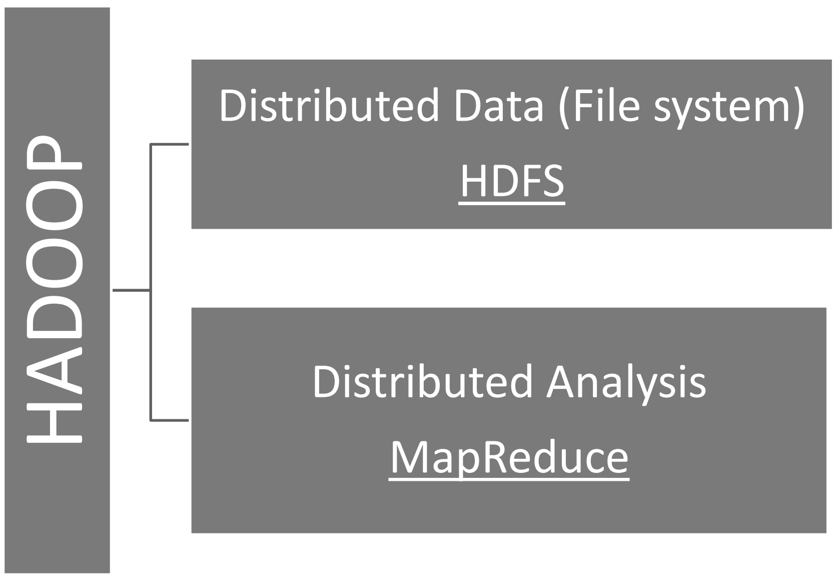
The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

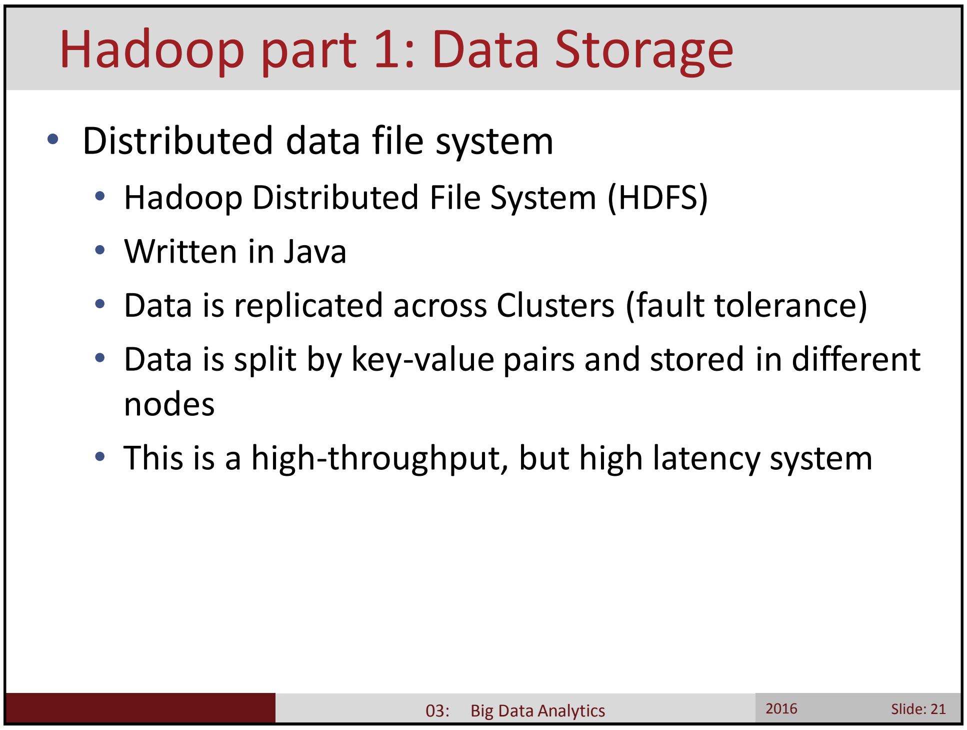
-From hadoop.apache.org

## Hadoop

- History:
  - Apache Project
  - 2004 – Google publishes Map-Reduce paper
  - Doug Cutting created Hadoop ~ 2006
  - Hadoop 0.1.0 released April 2006
  - 2009 – HDFS and Map-reduce designated separate sub-projects
  - 2010 – Apache Hive, Pig
  - 2012 – Hadoop 1.0 released
  - 2013 – October – Hadoop 2.2 released
  - 2015 – Hadoop 2.7 released

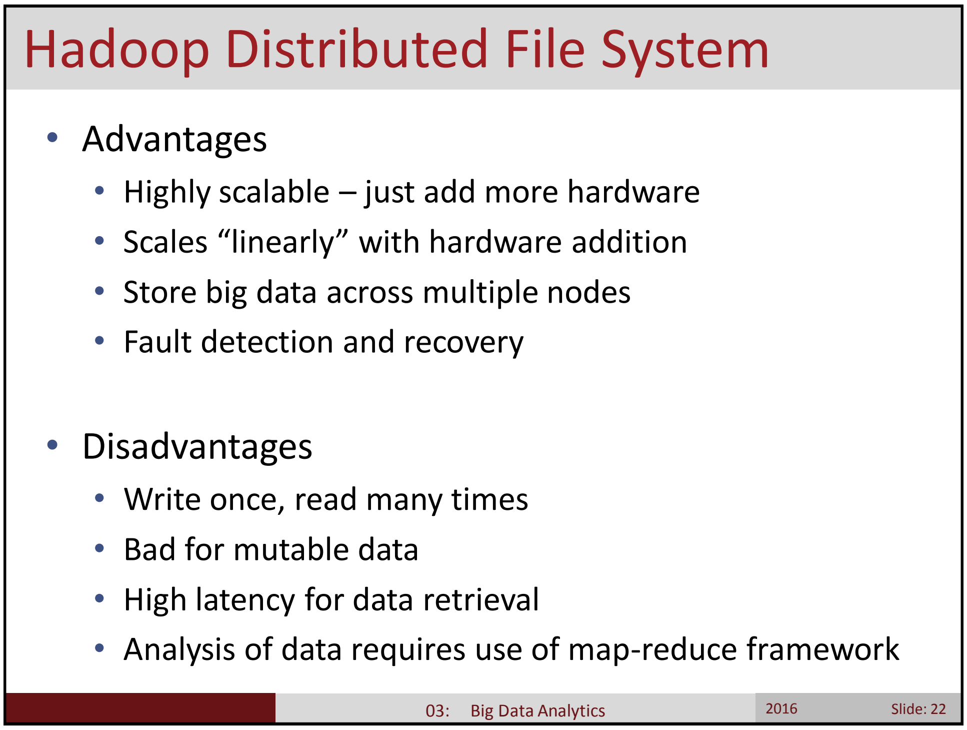
## Hadoop





## Hadoop part 1: Data Storage

- Distributed data file system
  - Hadoop Distributed File System (HDFS)
  - Written in Java
  - Data is replicated across Clusters (fault tolerance)
  - Data is split by key-value pairs and stored in different nodes
  - This is a high-throughput, but high latency system



## Hadoop Distributed File System

- Advantages
  - Highly scalable – just add more hardware
  - Scales “linearly” with hardware addition
  - Store big data across multiple nodes
  - Fault detection and recovery
- Disadvantages
  - Write once, read many times
  - Bad for mutable data
  - High latency for data retrieval
  - Analysis of data requires use of map-reduce framework

## Hadoop part 2: Data Processing

- Distributed Data Processing: MapReduce
- Map-reduce is the programming approach that allows the scalability and parallel nature of Hadoop
- Map
  - This step selects and subsets values, then returns them as a key and value pair. This is the intermediate key-value pair object
- Reduce
  - This step receives the key-value object from Map, using the keys to collate all values with identical keys. We define some set of functions to each key sub-group and return the result as a key-value pair.
- Original Google Publication:
  - http://research.google.com/archive/mapreduce.html

## MapReduce Analytics

- Advantages of Hadoop:
  - Scalable to thousands of nodes
  - We can do analysis on datasets too large for memory
  - Once programming in map-reduce is written, it is automatically scaled with more nodes
- Disadvantages of Hadoop:
  - Map-reduce has overhead costs that exist no matter how small the data is.
  - Interactive analysis is difficult
  - Map-reduce programming is difficult, and requires us to explicitly make algorithms parallel

## Related Hadoop Projects

- Apache Pig
  - High-level platform to interface with HDFS
  - Uses “pig-latin”
  - Requests are translated to map-reduce jobs
- Apache Hive
  - Data warehouse
  - SQL esque interface, Hive Query Language (HQL)
  - HQL is translated to map-reduce jobs
  - JDBC Drivers are available

## Related Hadoop Projects

- Apache Hbase
  - NoSQL database modeled after Google’s BigTable
  - Overcomes some HDFS limitations of random read
  - This is a key-value store that sites on top of HDFS
  - Requires a custom language to learn

## Dream Software

- Distributed data model with enough structure to make interactive use easy.
- Dataframe like object native to the software
- Fast.
- Designed with data analysis and statistical frameworks in mind.
  - I don't want to hard code a correlation algorithm…I want to call the cor() function!
- Don't want to learn yet another language to use it.
  - Should have a few API options analysts already know.
- Free! Umm, I mean Open source!

## Apache Spark

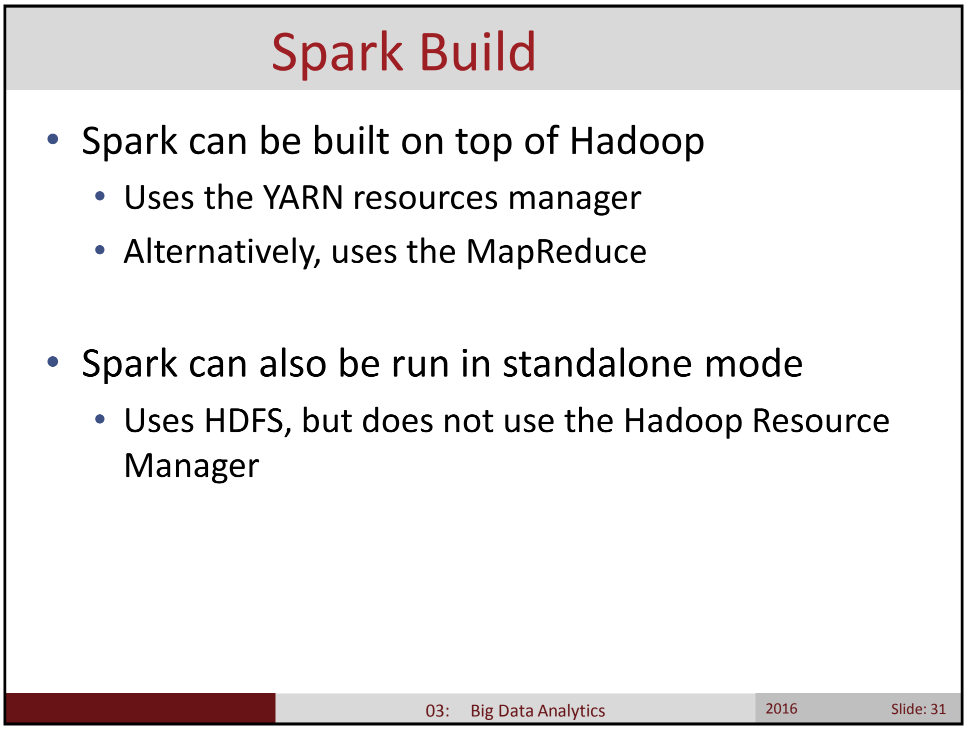
- "Apache Spark is a fast and general-purpose cluster computing system." – Spark Homepage
- Developed at UC Berkely AMPLab
- Donated to Apache Foundation, now part of the Hadoop ecosystem
- Often associated with Hadoop, but <u>truly distinct</u>

## Apache Spark History

- 2009
  - Started by Matei Zaharia at UC Berkley AMPLab
- 2012 – Version 0.5.1
- 2013
  - Version 0.8.1
  - Donated to Apache Software Foundation
- 2014
  - Version 1.0
- 2016
  - Version 2.0

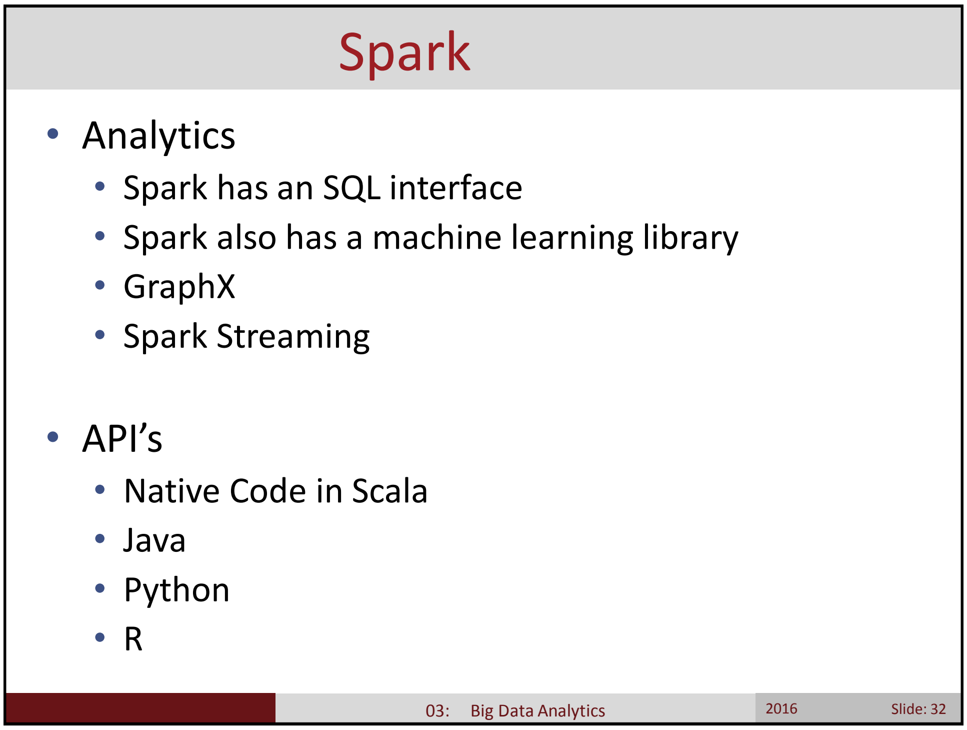
## Spark

- Distributed Data Structures
  - Resilient Distributed Dataset(RDD)
    - Not so much structure
  - Dataset/Dataframe
    - Newly added functionality
    - More structure to make data optimizations
- Speed
  - Spark is significantly faster than Hadoop
  - Spark operates in memory when possible
  - Spark reduces the number of read/write cycles compared to MapReduce
  - Spark =/= MapReduce



## Spark Build

- Spark can be built on top of Hadoop
  - Uses the YARN resources manager
  - Alternatively, uses the MapReduce
- Spark can also be run in standalone mode
  - Uses HDFS, but does not use the Hadoop Resource Manager



## Spark

- Analytics
  - Spark has an SQL interface
  - Spark also has a machine learning library
  - GraphX
  - Spark Streaming
- API's
  - Native Code in Scala
  - Java
  - Python
  - R

# Big-data Machine Learning using Python and Spark

Evan Carey and Debashis Ghosh

Departments of Epidemiology and Biostatistics and Informatics, Colorado School of Public Health

2016 Seattle Symposium on Health Care Analytics

## Basic Example: Logistic regression

- Let $Y \in \{0, 1\}$ denote the response
- Let $\mathbf{X}$ denote the covariates
- Model:
$$\text{logit}\{P(Y = 1|\mathbf{X})\} = \alpha + \mathbf{X}'\beta,$$
where $\text{logit}(u) = \log\{u/(1-u)\}$ and $(\alpha, \beta)$ are the regression coefficients

## Logistic regression (cont'd.)

- Estimation approaches
  1. Maximum likelihood
  2. Method of moments
  3. Robust estimation
  4. Penalized/Bayes procedures
- These will define the loss function to be used

## Logistic regression (cont'd.)

- We also need an algorithm for estimation
  1. Newton-Raphson (Fisher scoring)
  2. Simulation-based
  3. Markov Chain Monte Carlo

## Logistic regression (cont'd.)

- This example, while simple, illustrates key points about modeling:
  1. Give a model **specification**
  2. Propose a means for **estimation**
  3. Use an **algorithm** for estimation

## Big Data Paradigm

- Focus on the **algorithm**
- Many of the computations we are "used to making" become infeasible for a variety of reasons
  1. Data can't be stored on one server
  2. Matrices become too big to work with
- Big data requires a revisit and reexploration of our "standard" algorithms in which we determine techniques to make them scalable.
- However, that does not mean forget about **specification** and **estimation**!

## Logistic regression revisited

- "Standard" estimation approach: maximum likelihood with Newton-Raphson
- This is equivalent to an iteratively reweighted least squares algorithm
- This requires matrix inversion and matrix multiplication, which can be computationally infeasible

## Logistic regression

- One approach to make this scalable: use **gradient descent** algorithm
- Such an algorithm corresponds to adopting an $L_1$ penalty for $\beta$
- This yields the following penalized log-likelihood:

$$\sum_{i=1}^{n} Y_i \log p_i(\alpha,\beta) + (1-Y_i)\log\{1-p_i(\alpha,\beta)\} + \lambda(|\alpha| + \sum_{j=1}^{p} |\beta_j|),$$

where $\lambda \geq 0$ is a smoothing parameter and

$$p_i(\alpha,\beta) = \frac{\exp(\alpha + \mathbf{X}'\beta)}{1+\exp(\alpha+\mathbf{X}'\beta)},$$

$i = 1, \ldots, n$.

## Model specification

- Several types of categorizations:
  1. $Y|\mathbf{X}$ (prospective) versus $\mathbf{X}|Y$ (retrospective)
  2. $Y|\mathbf{X}$ or $\mathbf{X}|Y$ (discriminative) versus $(\mathbf{X}, Y)$ (generative)
- Associated properties
  1. Distributional family for $Y$ (e.g., binomial distribution for logistic regression)
  2. Link function

## Formulation as optimization problem

- Typically, statisticians have worked with maximum likelihood estimation
- Attractive properties:
  1. Assume exponential family with "proper" parameter space: equivalent to maximizing a convex function over a convex set $\Rightarrow$ maximizer (MLE) will be a global optimizer

## Formulation as optimization problem (cont'd.)

- Machine learning: Kuhn-Karush-Tucker (KKT) theory
- Formulate as an optimization problem subject to equality and inequality constraints
- Generic formulation: Maximize

$$f(x)$$

subject to $g_i(x) \leq 0$ $(i = 1, \ldots, I)$ and $h_j(x) = 0$, $j = 1, \ldots, J$

- Equivalently expressed as minimize

$$f(x) + \sum_{i=1}^{I} \lambda_i g_i(x) + \sum_{j=1}^{J} \lambda_j^* h_j(x)$$

## Formulation as optimization problem (cont'd.)

- KKT necessary conditions for finding a global optimum:
  1. Stationary conditions (first derivative equals zero)
  2. Feasibility: at optimum, inequality and equality constraints are satisfied
  3. Complementary slackness: at optimum, $\lambda_i g_i(x^*) = 0$

## Formulation as optimization problem (cont'd.)

- KKT sufficient conditions for finding a global optimum: $f$ is an **invex** function (generalization of convex): there exists a vector-valued function $g(x, u)$ such that

$$f(x) - f(u) \geq g(x, u) \cdot \nabla f(u)$$

for all $x, u$, where $\nabla f$ denotes the gradient of $f$

## Example of KKT (Liu et al., 2007)

- Formulate a semiparametric linear model for a continuous $y$:

$$y_i = \beta_0 + {x_i}^T\beta + h(z_i) + e_i$$

where $i = 1, \ldots, n$,
$x_i = (x_{i1}, \ldots, x_{iq})^T$,
$z_i = (z_{i1}, \ldots, z_{ip})^T$
$\beta = (\beta_1, \ldots, \beta_q)^T$
$h(z_i)$ = unknown smooth nonparametric function, where
$h \in \mathcal{H}$=some functional space,
$e \sim N(0, \sigma^2 I)$.

## Liu et al., 2007, continued

- Estimate $\beta$ and $h(\cdot)$ by minimizing the penalized sum of squares:

$$\sum_{i=1}^{n}\{y_i - (\beta_0 + x_i{}^T\boldsymbol{\beta} + h(z_i))\}^2 + \lambda\|h\|_{\mathcal{H}}^2$$

- This is a standard penalized likelihood problem
- Standard approach: take $\mathcal{H}$ to be a reproducing kernel Hilbert space (Wahba, 1990)

## Liu et al., 2007, continued

- Primal formulation:

$$\begin{array}{rl} \min & \frac{1}{2}\sum_{i=1}^{n} e_i^2 + \frac{1}{2}\lambda\|\boldsymbol{\omega}\|^2 \\ s.t. & e_i = y_i - \{\beta_0 + x_i{}^T\boldsymbol{\beta} + \phi(z_i)^T\boldsymbol{\omega}\} \end{array}$$

for $i = 1, \ldots, n$

## Liu et al., 2007, continued

- Introduce the Lagrangian multiplier (dual parameters) $\boldsymbol{\gamma}$ and form the Lagrangian function

$$\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\beta}, \rceil; \boldsymbol{\gamma}) \quad =$$

$$\frac{1}{2}\sum_{i=1}^{n} e_i^2 + \frac{1}{2}\lambda\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{n}\gamma_i\{\beta_0 + x_i{}^T\boldsymbol{\beta} + \boldsymbol{\phi}(z_i)^T\boldsymbol{\omega} + e_i - y_i\}$$

- The dimension of $\boldsymbol{\gamma} = n$ (low dimension).
- The dual formulation is obtained by removing the high dimensional parameters $\boldsymbol{\omega}$ and writing $\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\beta}, \rceil; \boldsymbol{\gamma})$ as a function of dual parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ alone.

## Liu et al., 2007, continued

- Optimality conditions

$$\begin{cases} \nabla_{\boldsymbol{\omega}}\mathcal{L} = / & \rightarrow & \boldsymbol{\omega} = \frac{1}{\lambda}\sum_{i=1}^{n}\gamma_i\boldsymbol{\phi}(z_i) \\ \frac{\partial\mathcal{L}}{\partial e_i} = 0 & \rightarrow & e_i = \gamma_i \\ \nabla_{\boldsymbol{\beta}}\mathcal{L} = / & \rightarrow & \sum_{i=1}^{n}\gamma_i x_i = 0 \\ \frac{\partial\mathcal{L}}{\partial\gamma_i} = 0 & \rightarrow & x_i{}^T\boldsymbol{\beta} + \boldsymbol{\phi}(z_i)^T\boldsymbol{\omega} + e_i - y_i = 0 \end{cases}$$

## Liu et al., 2007, continued

- The dual formulation is obtained by substituting $\widehat{\omega}$ and $\widehat{e}$ into the last equation:

$$\begin{cases} y_i - x_i{}^T\boldsymbol{\beta} - \frac{1}{\lambda}\sum_{i'=1}^n \gamma_{i'}\phi(z_i)^T\phi(z_{i'}) - \gamma_i = 0 \\ \sum_{i=1}^n \gamma_i x_i = 0 \end{cases}$$

- Estimation in the dual formulation is low dimensional.
- The estimator $\widehat{h}(z) = \lambda^{-1}\sum_{i=1}^n \widehat{\gamma}_i\phi(z)^T\phi(z_i)$.
- Computation of $\widehat{\gamma}$ and $\hat{h}(z)$ hence only requires evaluating the kernel function

$$k(z,z') =< \phi(z), \phi(z') >= \phi(z)^T\phi(z').$$

- This is referred to as a kernel method (visit later)

## Optimization

- Most machine learners phrase modelling as an optimization problem with constraints as in prior example
- For most problems, this will correspond to a loss function

$$L(\beta) = \sum_{i=1}^n l(Y_i, \mathbf{X}_i, \beta)$$

## Optimization (cont'd.)

- Examples
  1. $l(Y_i, \mathbf{X}_i, \beta) = (Y_i - \mathbf{X}_i\beta)^2$ (quadratic loss)
  2. $l(Y_i, \mathbf{X}_i, \beta) = |Y_i - \mathbf{X}_i\beta|_\epsilon$, where

     $$|u|_\epsilon = \begin{cases} u^2/2, & |u| < \epsilon \\ \epsilon(|u| - \epsilon/2), & \text{else} \end{cases}$$

     This is Huber loss
  3. If $Y$ is binary,

     $$l(Y_i, \mathbf{X}_i, \beta) \equiv Y_i \log P(Y_i = 1|\mathbf{X}_i) + (1 - Y_i) \log P(Y_i = 0|\mathbf{X}_i)$$

     is the binomial likelihood
  4. Other losses are possible (e.g., hinge loss for support vector machines, boosting, etc.)

## Optimization (cont'd.)

- For several reasons, machine learners **penalize** the loss function
- Reasons
  1. Numerical stability
  2. Allow for nonlinearities
  3. Expand set of available big-data scalable algorithms
  4. Leads to better **generalizability**
- Amounts to minimizing

  $$L(\beta) + \lambda P(\beta)$$

  where $\lambda > 0$ is a smoothing parameter and $P(\beta)$ is the penalty term

## Choices for $P(\beta)$

- Take $P(\beta)$ to be an $L_1$ constraint on the regression coefficients; this is commonly referred to as LASSO (least absolute selection and shrinkage operator)
- Perform LASSO of $Y$ on $\mathbf{X}$ by solving:

$$\sum_{i=1}^{n} l(Y_i, \mathbf{X}_i, \beta) \text{ s.t. } \sum_{j=1}^{p} |\beta_j| \leq t, \qquad (1)$$

  where $\beta$ are unknown regression coefficients, and $t \geq 0$ is a smoothing parameter.
- Has the property of **exactly estimating** certain coefficients to be zero $\Rightarrow$ automated feature selection

## LASSO: Some theory

- Statisticians have explored when this type of procedure leads to consistent model selection
- Sufficient conditions
  1. The irrelevant variables cannot be too correlated with the true variables
  2. Subgaussian tails for the random variables
- Key is that this is a penalty and lead to improved prediction on new datasets.

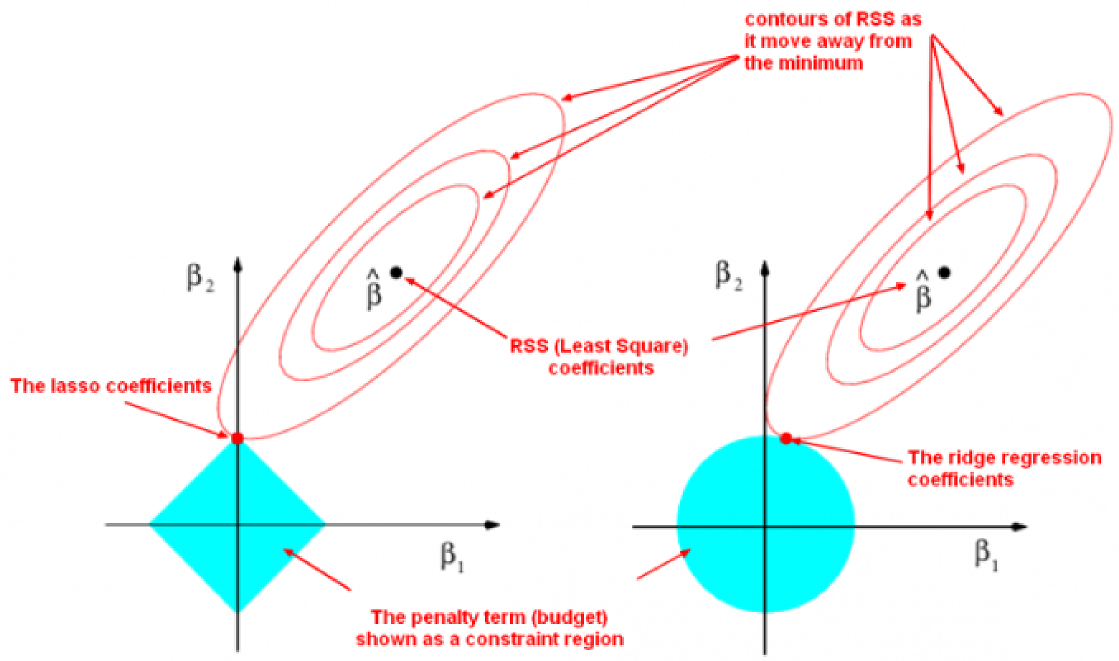
## Ridge regression

- Take $P(\beta)$ to be an $L_2$ constraint on the regression coefficients; this is commonly referred to as ridge regression(least absolute selection and shrinkage operator)
- Perform LASSO of $Y$ on $\mathbf{X}$ by solving:

$$\sum_{i=1}^{n} l(Y_i, \mathbf{X}_i, \beta) \text{ s.t. } \sum_{j=1}^{p} \beta_j^2 \leq t, \qquad (2)$$

  where $\beta$ are unknown regression coefficients, and $t \geq 0$ is a smoothing parameter.
- Has the property of shrinking **correlated features** towards zero but not estimating any coefficients to be exactly zero

## Comparison of LASSO and ridge regression http://bit.ly/2dXxw7y


contours of RSS as it move away from the minimum
$\beta_2$
$\hat{\beta}$
$\beta_1$
The lasso coefficients
RSS (Least Square) coefficients
The ridge regression coefficients
The penalty term (budget) shown as a constraint region

## Penalized Regression

- Compromise of Ridge and LASSO: Elastic Net (Zou et al., 2005), which minimizes

$$\sum_{i=1}^{n} l(Y_i, \mathbf{X}_i, \beta) + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \qquad (3)$$

  where $\lambda_1, \lambda_2 \geq 0$ are smoothing parameters
- Has the effect of performing (a) automatic grouping of correlated predictors and (b) variable selection.

## Optimization Algorithms

- Many standard algorithms in statistical procedures are sequential
- Generically, this means that computation at iteration $t+1$ requires saving output from iteration $t$ as well as data
- For big data, this might not be efficient from either a computational or storage point of view
- Sequential algorithms in general do not scale

## Big Data Optimization Techniques

- Active area of research
- Some proposed solutions:
  1. Randomization
  2. Distributed algorithms
  3. Asynchronous algorithms

## Randomization

- Classical example: gradient descent
- **Goal:** minimize $f(\mathbf{w})$, where $\mathbf{w} \in R^d$
- find a local minimum of a function by iteratively taking steps in the direction of steepest descent
- Finding the direction of steepest descent can be computationally unscalable for big datasets
- One solution: pick a descent direction at random and compute the gradient
- Iterate this to convergence

## Stochastic Gradient Descent

- Works well with penalization problems of the form

$$\lambda P(\beta) + \sum_{i=1}^{n} l(Y_i, \mathbf{X}_i, \beta)$$

- Pick one observation from random
- Compute **subgradient** with respect to data point (inexact gradient)
- Move objective function in the direction of the negative subgradient with respect ot data point
- Cheap computations

## Distributed Algorithms

- Make use of a parallel computing environment
- Take task and have several nodes run the task in parallel
- Two issues that complicate these algorithms:
  1. Communication: nodes talking to each other
  2. Synchronization: nodes must be coordinated in performing parallel tasks

## Distributed Algorithms (cont'd.)

- Separable functions like

$$\sum_{i=1}^{n} l(Y_i, \mathbf{X}_i, \beta)$$

  are amenable to distributed algorithms (embarassingly parallel)
- Have a node dedicated to some parameters and only compute the gradient for those
- Alternative: asynchronous computations

## Spark: an aside

- Main abstraction in Spark is that of a resilient distributed dataset (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost (lineage)
- written in Scala, a typed high-level programming language for the Java VM
- Can "parallelize" a Scala collection (e.g., an array) in the driver program, which means dividing it into a number of slices that will be sent to multiple nodes
- Can perform distributed versions of SGD

## Support vector machines

- Suppose that we have two groups of observations
- Intuition behind SVMs: find the separating hyperplane that maximizes the **margin** between two groups and perfectly classifies observations
- **margin**: distance between the hyperplane and points
- Sometimes to achieve perfect classification, slack variables are introduced into the problem, along with a regularization parameter $C$

## SVM: 2-D representation (from Jaggi, 2013)

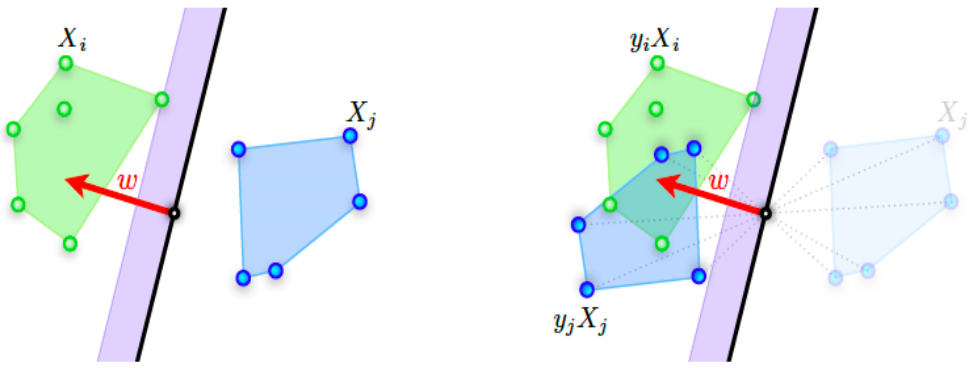


**Figure 1:** A linear classifier: illustration of the separation of two point classes, for a normal vector $w$. Here we highlight two points $i, j$ of different labels, with $y_i = +1$ and $y_j = -1$.

## SVM

- SVM primal optimization:

$$\min_{\mathbf{w},b} \|\mathbf{w}\|^2$$

subject to $Y_i(\mathbf{w} \cdot \mathbf{X}_i - b) \geq 1,\ i = 1, \ldots, n$.

- $1/\|\mathbf{w}\|$ is proportional to the margin, so can reexpress as maximizing the margin
- SVM dual formulation:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j Y_i Y_j < \mathbf{X}_i, \mathbf{X}_j >$$

subject to $\alpha_i \geq 0$ $(i = 1, \ldots, n)$ and $\sum_{i=1}^{n} \alpha_i Y_i = 0$ where $< \cdot, \cdot >$ denotes inner product

## SVM: remarks

- Typically, the dual SVM problem has been simpler to solve, as it is a quadratic programming problem
- Not necessarily the case with big data
- SVM can be cast in the loss function framework as

$$\sum_{i=1}^{n} |1 - Y_i f(\mathbf{X}_i)|_+ + \lambda P(f)$$

where $|u|_+ = \max(u, 0)$ and $P(f)$ is a penalty on $f$.

## SVM: optmization

- Typically done using a sequential minimization optimization (SMO) algorithm
- With big data, SMO is not usable because of storage issues
- An alternative is to use stochastic gradient descent, which is what is implemented in Spark for linear kernel

## Decision trees

- Decision tree is a greedy algorithm that performs a recursive binary partitioning of the feature space.
- Also referred to as classification and regression trees in the literature
- MLlib 1.2 adds several features for scaling up to larger (deeper) trees and tree ensembles
- Big concern: overfitting
- Inherently, decision trees are *unstable* classifiers

## Random Forests

- Ensemble of decision trees
- By averaging unstable classifiers, we have a more stable prediction algorithm
- In Spark, these are fit using randomized approach

## Dimensionality reduction methods

- Principal components analysis (PCA)
- Singular value decomposition (SVD)
- These are unsupervised methods

## PCA: background

- Goals of PCA
  1. **Geometry:** Find a new coordinate system obtained by rotating the original system with $\mathbf{X}_1, \dots, \mathbf{X}_p$ as the coordinate axes, where the new axes represent the directions with maximum variability and provide a simpler and more parsimonious description of the covariance structure.
  2. **Optimization:** Find the set of **uncorrelated** linear combinations of variables that explain the greatest amount of variability of variables

## Variance-Covariance matrix: review

- Let $\mathbf{X} \equiv (X_1 \quad X_2 \quad \cdots \quad X_p)'$ denote a $p \times 1$ vector
- Then the (population) variance-covariance matrix is

$$\Sigma = [\sigma_{ij}]_{i,j=1,\dots,p},$$

  where $\sigma_{ij} = E[(X_i - \mu_i)(X_j - \mu_j)]$, and $\mu_i$ is the mean of $X_i$.
- Now variance is a $p \times p$ matrix, and what summarizes the variability are $p$ points, each of which is a $p-$dimensional vector.
- Note that $\Sigma$ will be symmetric
- If the distribution for $\mathbf{Y}$ is nondegenerate, then $\Sigma$ will also be positive definite (i.e., all eigenvalues of $\Sigma$ are positive)

## PCA

- principal components (PCs): the linear combinations of $(X_1, ..., X_p)$ that are uncorrelated and for which
$$\mathrm{Var}(\mathbf{a}_i'\mathbf{X}) = \mathbf{a_i}'\Sigma\mathbf{a}_i, \quad i = 1, \ldots, p$$
is as large as possible.
- Statistical goal: Determine $\mathbf{a}_1, \ldots, \mathbf{a}_p$.
- By definition, $\mathbf{a}_i'\mathbf{a_j} = 0$ for $i \neq j$
- Also add in the constraint that $\mathbf{a}_i'\mathbf{a}_i = 1$ for $i = 1, \ldots, p$.

## PC Definition

- First PC: linear combination $\mathbf{a}_1'\mathbf{X}$ that maximizes $\mathrm{Var}(\mathbf{a}_1'\mathbf{X})$ subject to $\mathbf{a}_1'\mathbf{a}_1 = 1$.
- Second PC: linear combination $\mathbf{a}_2'\mathbf{Y}$ that maximizes $\mathrm{Var}(\mathbf{a}_2'\mathbf{X})$ subject to $\mathbf{a}_2'\mathbf{a}_2 = 1$ and $\mathrm{Cov}(\mathbf{a}_1'\mathbf{X}, \mathbf{a}_2'\mathbf{X}) = 0$.
- In general, $i$th PC: linear combination $\mathbf{a}_i'\mathbf{X}$ that maximizes $\mathrm{Var}(\mathbf{a}_i'\mathbf{X})$ subject to $\mathbf{a}_i'\mathbf{a}_i = 1$ and $\mathrm{Cov}(\mathbf{a}_i'\mathbf{Y}, \mathbf{a}_j'\mathbf{X}) = 0$ for $j = 1, 2, \ldots, i-1$.
- The solution to this optimization problem is given by the eigenvalue/eigenvectors of $\Sigma$.

## Sample estimator of a covariance matrix

- Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be a random sample of $p-$dimensional vectors
- Then the sample covariance matrix estimator is given by

$$\mathbf{S} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})',$$

  where $\bar{\mathbf{x}} = n^{-1}\sum_{i=1}^{n}\mathbf{x}_i$.
- Alternatively, $\mathbf{S} = [s_{ij}]$, $i, j = 1, \ldots, p$, where

$$s_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}(x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j),$$

  where $\bar{x}_i$ and $\bar{x}_j$ are the $i$th and $j$th component of $\bar{\mathbf{x}}$
- The diagonals of $\mathbf{S}$: sample variances
- The off-diagonals of $\mathbf{S}$: sample covariances

## Sample PCA

- Algorithm:
- Estimates of population PCs obtained by calculating the eigenvectors of $\mathbf{S}$, sorted from largest to smallest eigenvalue
- Estimate of $\mathbf{a}_i$ ($i$th PC): eigenvector of $\mathbf{S}$ corresponding to to $i$th largest eigenvalue, $i = 1, \ldots, p$. Note: the estimated eigenvalues $\hat{\lambda}_1 \geq \hat{\lambda}_2 \geq \cdots \geq \hat{\lambda}_p \geq 0$.
- Proportion of total population variance of $\mathbf{X}$ explained by the $i$th PC is estimated by

$$\frac{\hat{\lambda}_i}{\sum_{j=1}^{p}\hat{\lambda}_j}.$$

PCA: number of components

- Major question: how many PCs to use in analysis
- Statistically, this is a **hard** problem
- Some old solutions: do a series of hypothesis tests on eigenvalues, testing $H_0 : \lambda_i = \lambda_{i+1}$ versus $H_A : \lambda_i > \lambda i + 1$
- Old theory based on asymptotics and multivariate normality of original data
- A more robust solution: bootstrap and get CIs for $\lambda$
- A more practical solution: choose number of PCs so that variability explained is $80 - 90\%$

## Scree plot

- A plot of eigenvalues of **S** in decreasing order.
- By looking for an elbow (bend) in the scree plot, we can determine the number of PCs.
- A visual test for number of PCs

## More about Principal Components Analysis (PCA)

- Recall:
  1. PCA results are different based on whether or not sample covariance or sample correlation matrix is used
  2. More generally, results are not invariant to changes in scale of variables (e.g., it does matter whether weight data in matrix are measured in pounds or kilograms)
  3. PCA is a dimension reduction procedure
  4. The PCs are linear combinations of the variables that comprise the columns of the data matrix

## Utility of PCA

- Implicitly, PCA assumes that the variables in the data are continuous and can take any real value
- In many situations, we do not have that
- What are the situations in which PCA gives meaningful results?

## Gower (1966, *Biometrika*)

- Work with the idea of a "distance" between objects:
- We assume that we have the matrix $\mathbf{X}$, which is $n \times p$
- View each row as a $p-$dimensional vector. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ denote the rows of $\mathbf{X}$; these are the objects
- Assume that the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^{p} (x_{il} - x_{jl})^2;$$

this is referred to as Euclidean distance

## Properties of distance

- $d$: distance function/metric: takes as input two vectors and returns a nonnegative scalar
- For a proper distance metric $d$, we need the following properties to hold:
  1. $d(\mathbf{x}, \mathbf{x}) = 0.$
  2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x});$
  3. $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}).$

## A new interpretation of PCA

- Consider a distance matrix $\mathbf{Q} = [q_{ij}]$, where

$$q_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$$

- Create an association matrix $\mathbf{A} = [a_{ij}]$ that is $n \times n$ and whose $i, j$th entry is computed by defining

$$a_{ii} = 0; \quad a_{ij} = -q_{ij}^2/2.$$

- Construct a matrix $\alpha = [\alpha_{ij}]$ by

$$\alpha_{ij} = a_{ij} - \bar{a}_i - \bar{a}_j + \bar{a}.$$

- Perform PCA on $\alpha$; this will give a set of points that provide the closest "reconstruction" of the points in the original $p-$dimensional space.

## PCA: remarks

- The key feature that is needed is that $\alpha$ be a positive semi-definite matrix.
- This implies conditions on $\mathbf{A}$ and on $\mathbf{Q}$.
- Euclidean distance works fine with continuous variables that take values $(a, b)$ for any values of $a$ and $b$, but it does NOT work with discrete variables
- For discrete variables, one should use a different metric. One example is the Hamming distance. For two $p-$dimensional vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, this is given by

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^{p} I(x_{il} \neq x_{jl}).$$

## PCA and latent factor models

- So far, we have primarily motivated PCA as an algorithm with a geometric and algorithmic interpretation
- Is there a probabilistic model in which PCA is some type of estimator?
- A special case of a so-called *latent factor* model

## Latent factor models

- The formula for a latent factor is given by

$$\mathbf{x} = \mu + \mathbf{A}\mathbf{u} + \mathbf{e},$$

  where
  - $\mu$ is an unknown constant $p \times 1$ vector (intercept term)
  - $\mathbf{x}$ is a $p \times 1$ vector
  - $\mathbf{u}$ is a $d \times 1$ vector of random variables
  - $\mathbf{A}$ is a $p \times d$ vector of unknown constants that relates $\mathbf{x}$ and $\mathbf{u}$
  - $\mathbf{e}$ is the residual error term, also a vector of random variables
- Interpretation: $\mathbf{u}$ provides a more parsimonious representation of the data (i.e. fewer parameters/variables) when $d < p$.

## Latent factor models (cont'd.)

- Further assumptions:
  1. $\mathbf{u} \sim N(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ is $d \times d$ matrix
  2. $\mathbf{e} \sim N(\mathbf{0}, \Sigma)$
  3. Implication of these distributional assumptions $\Rightarrow$

$$\mathbf{x} \sim N(\mu, \mathbf{A}\mathbf{A}' + \Sigma)$$

- Note: $\mathbf{A}$ is known up to an orthogonal matrix
- Parameters of interest: $\mu$, $\mathbf{A}$, and $\Sigma$

## Latent factor models and PCA

- Assume further that $\Sigma = \sigma^2\mathbf{I}$, where $\mathbf{I}$ is $p \times p$
- Tipping and Bishop (1999) show that the maximum likelihood estimate (MLE) of $\mathbf{A}$ for a fixed $\sigma^2$ is given by

$$\hat{\mathbf{A}} = \mathbf{U}_p(\boldsymbol{\Lambda}_p - \sigma^2\mathbf{I})^{1/2}\mathbf{R},$$

  where
  1. $\mathbf{U}_p$ is a $p \times d$ matrix whose columns are the eigenvectors of $\mathbf{S}$, the empirical variance-covariance matrix of $\mathbf{x}$
  2. $\Lambda_p$ is a $p \times p$ diagonal matrix of the eigenvalues of $\mathbf{S}$
  3. $\mathbf{R}$ is an arbitrary orthogonal matrix (take to equal $\mathbf{I}$)
- The MLE of $\sigma^2$ is given by

$$\hat{\sigma}^2 = \frac{1}{p-d}\sum_{j=d+1}^{p} \lambda_j,$$

  where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq \lambda_{d+1} \geq \cdots \lambda_p$ are the eigenvalues corresponding to $\mathbf{S}$.

## Latent factor models and PCA: remarks

- PCA corresponds to situation where $\sigma^2 \to 0$; this is not really a proper statistical model (imagine a linear regression model where there is no stochastic component)
- The major issue: choice of $d$; this is analogous to selecting the number of principal components
- The goal of factor analysis is the same as PCA: dimension reduction of multivariate data into a data-driven summary score that can be used for further analyses

## PCA and SVD

- Note that SVD is a special case of PCA
- SVD generates a three matrix decomposition of a $n \times p$ matrix $\mathbf{M}$

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

  where $\mathbf{U}$ is an $n \times n$ matrix with $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}_n$
  $\mathbf{D}$ is an $n \times p$ matrix with nonnegative numbers on the diagonal, and
  $\mathbf{V}$ is a $p \times p$ with $\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_p$
- PCA: SVD of a square symmetric matrix

## PCA and SVD (cont'd.)

- For big data, computing eigenvalues and eigenvectors becomes a complicated problem
- Spark uses a distributive algorithm (links to ARPACK, `http://www.caam.rice.edu/software/ARPACK/`
- Other options:
  1. Create randomized matrices and perform computation on those
  2. Idea: approximate original data matrix with a low-rank approximation in a randomized manner; the randomized manner will speed up computation

## Clustering: Intuition

- A basic tool in data mining/pattern recognition:
- Divide a set of data into groups.
- Samples in one cluster are close and clusters are far apart.
- Motivation:
  - Discover classes of data in an unsupervised way (unsupervised learning).
  - Efficient representation of data: fast retrieval, data complexity reduction.
  - Various engineering purposes: tightly linked with pattern recognition.

## Approaches to Clustering

- Represent samples by feature vectors.
- Define a distance measure to assess the closeness between data.
- "Closeness" can be measured in many ways.
- Define distance based on various norms/metrics.
- Multivariate distance – depends on the setting $\Rightarrow$ The variables have incompatible units and no prior known relationship. The result of clustering will depends on the arbitrary choice of variable scaling.
- Clustering: grouping of similar objects (unsupervised learning)

## Approaches to clustering (cont'd.)

- Approaches
  1. Prototype methods: K-means (for vectors), K-center (for vectors), D2-clustering (for bags of weighted vectors)
  2. Statistical modeling: mixture modeling by the EM algorithm, Modal clustering
  3. Pairwise distance based partition: Spectral graph partitioning, Dendrogram clustering (agglomerative): single linkage (friends of friends algorithm), complete linkage, etc.

## Philosophies of Clustering

- Parametric versus nonparametric
- Probabilistic versus algorithmic
- Pros and cons of each approach
- With any method, must realize that **a notion of distance** is used.

## K-means

- Assume there are $M$ prototypes (observations) denoted by
$$\mathcal{Z} = \{z_1, z_2, ..., z_M\} .$$
- Each training sample is assigned to one of the prototype. Denote the assignment function by $A(\cdot)$. Then $A(x_i) = j$ means the $i$th training sample is assigned to the $j$th prototype.
- Goal: minimize the total mean squared error between the training samples and their representative prototypes, that is, the trace of the pooled within cluster covariance matrix.
$$\arg\min_{\mathcal{Z},A} \sum_{i=1}^{N} \parallel x_i - z_{A(x_i)} \parallel^2$$
- Denote the objective function by
$$L(\mathcal{Z}, A) = \sum_{i=1}^{N} \parallel x_i - z_{A(x_i)} \parallel^2 \quad .$$

## Necessary Conditions

- If $\mathcal{Z}$ is fixed, the optimal assignment function $A(\cdot)$ should follow the nearest neighbor rule, that is,

$$A(x_i) = \arg\min_{j \in \{1,2,\ldots,M\}} \parallel x_i - z_j \parallel \ .$$

- If $A(\cdot)$ is fixed, the prototype $z_j$ should be the average (centroid) of all the samples assigned to the $j$th prototype:

$$z_j = \frac{\sum_{i:A(x_i)=j} x_i}{N_j} \ ,$$

where $N_j$ is the number of samples assigned to prototype $j$.

## The Algorithm

- Based on the necessary conditions, the k-means algorithm alternates between the two steps:
  - For a fixed set of centroids (prototypes), optimize $A(\cdot)$ by assigning each sample to its closest centroid using Euclidean distance.
  - Update the centroids by computing the average of all the samples assigned to it.
- The algorithm converges since after each iteration, the objective function decreases (non-increasing).
- Usually converges fast.
- Stopping criterion: the ratio between the decrease and the objective function is below a threshold.

## Example

- Training set: $\{1.2, 5.6, 3.7, 0.6, 0.1, 2.6\}$.
- Apply k-means algorithm with 2 centroids, $\{z_1, z_2\}$.
- Initialization: randomly pick $z_1 = 2$, $z_2 = 5$.

| fixed | update |
|---|---|
| 2 | {1.2, 0.6, 0.1, 2.6} |
| 5 | {5.6, 3.7} |
| {1.2, 0.6, 0.1, 2.6} | 1.125 |
| {5.6, 3.7} | 4.65 |
| 1.125 | {1.2, 0.6, 0.1, 2.6} |
| 4.65 | {5.6, 3.7} |

The two prototypes are: $z_1 = 1.125$, $z_2 = 4.65$. The objective function is $L(\mathcal{Z}, A) = 5.3125$.

## Example (cont'd.)

- Initialization: randomly pick $z_1 = 0.8$, $z_2 = 3.8$.

| fixed | update |
|---|---|
| 0.8 | {1.2, 0.6, 0.1} |
| 3.8 | {5.6, 3.7, 2.6} |
| {1.2, 0.6, 0.1 } | 0.633 |
| {5.6, 3.7, 2.6 } | 3.967 |
| 0.633 | {1.2, 0.6, 0.1} |
| 3.967 | {5.6, 3.7, 2.6} |

The two prototypes are: $z_1 = 0.633$, $z_2 = 3.967$. The objective function is $L(\mathcal{Z}, A) = 5.2133$.

- Starting from different initial values, the k-means algorithm converges to different local optimum.
- It can be shown that $\{z_1 = 0.633, z_2 = 3.967\}$ is the global optimal solution.

## Initialization

- Randomly pick up the prototypes to start the k-means iteration.
- Different initial prototypes may lead to different local optimal solutions given by k-means.
- Try different sets of initial prototypes, compare the objective function at the end to choose the best solution.
- When randomly select initial prototypes, better make sure no prototype is out of the range of the entire data set.
- Initialization in the above simulation:
  - Generated $M$ random vectors with independent dimensions. For each dimension, the feature is uniformly distributed in $[-1, 1]$.
  - Linearly transform the $j$th feature, $Z_j$, $j = 1, 2, ..., p$ in each prototype (a vector) by: $Z_j s_j + m_j$, where $s_j$ is the sample standard deviation of dimension $j$ and $m_j$ is the sample mean of dimension $j$, both computed using the training data.

## Mixture Model-based Clustering

- Each cluster is mathematically represented by a parametric distribution. Examples: Gaussian (continuous), Poisson (discrete).
- The entire data set is modeled by a mixture of these distributions.
- An individual distribution used to model a specific cluster is often referred to as a component distribution.
- Suppose there are $K$ components (clusters). Each component is a Gaussian distribution parameterized by $\mu_k$, $\Sigma_k$. Denote the data by $\mathbf{X}$, $\mathbf{X} \in \mathcal{R}^d$.

## Mixture Model-based Clustering (cont'd.)

- The density of component $k$ is

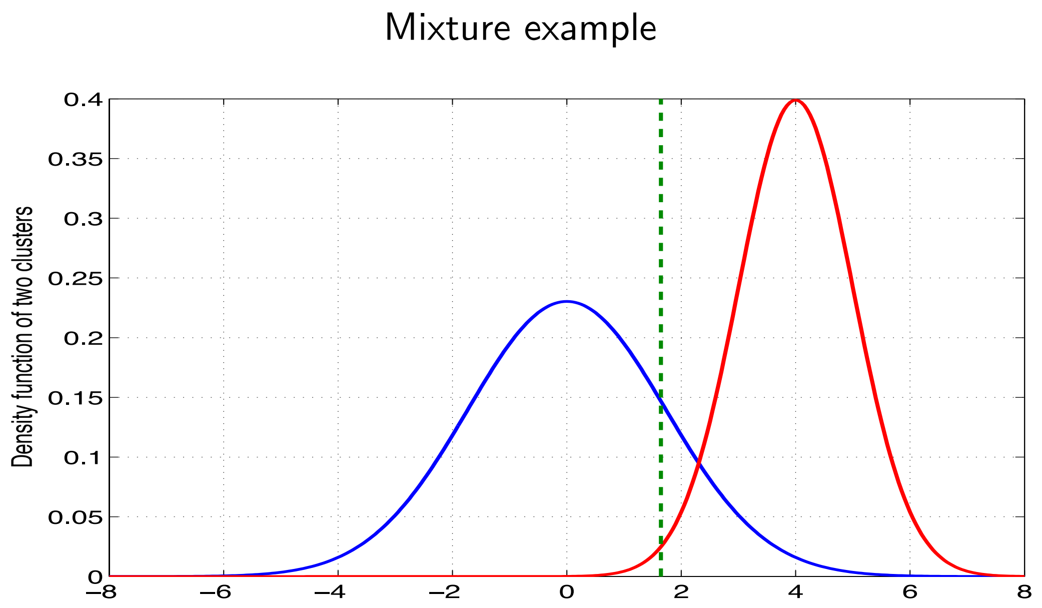
$$\begin{aligned} f_k(\mathbf{x}) &= \phi(\mathbf{x} \mid \mu_k, \Sigma_k) \\ &= \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} \exp\left(\frac{-(\mathbf{x}-\mu_k)^t \Sigma_k^{-1} (\mathbf{x}-\mu_k)}{2}\right) . \end{aligned}$$

- The prior probability (weight) of component $k$ is $a_k$. The mixture density is:

$$f(x) = \sum_{k=1}^{K} a_k f_k(x) = \sum_{k=1}^{K} a_k \phi(x \mid \mu_k, \Sigma_k) .$$

## Remarks

- A mixture model with high likelihood tends to have the following traits:
  - Component distributions have high "peaks" (data in one cluster are tight)
  - The mixture model "covers" the data well (dominant patterns in the data are captured by component distributions).
- Advantages
  - Well-studied statistical inference techniques available.
  - Flexibility in choosing the component distributions.
  - Obtain a density estimation for each cluster.
  - A "soft" classification is available.



## Estimation and EM algorithm

- The parameters are estimated by the maximum likelihood (ML) criterion using the expectation-maximization (EM) algorithm.
- The EM algorithm provides an iterative computation of maximum likelihood estimation when the observed data are incomplete.
- Incompleteness can be conceptual.
  - We need to estimate the distribution of $X$, in sample space $\mathcal{X}$, but we can only observe $X$ indirectly through $Y$, in sample space $\mathcal{Y}$.
  - In many cases, there is a mapping $x \rightarrow y(x)$ from $\mathcal{X}$ to $\mathcal{Y}$, and $x$ is only known to lie in a subset of $\mathcal{X}$, denoted by $\mathcal{X}(\dagger)$, which is determined by the equation $y = y(x)$.

## Estimation and EM algorithm

- The distribution of $X$ is parameterized by a family of distributions $f(x \mid \theta)$, with parameters $\theta \in \Omega$, on $x$. The distribution of $y$, $g(y \mid \theta)$ is

$$g(y \mid \theta) = \int_{\mathcal{X}(\dagger)} f(\mathbf{x} \mid \theta) dx \ .$$

- The EM algorithm aims at finding a $\theta$ that maximizes $g(y \mid \theta)$ given an observed $y$.
- Introduce the function

$$Q(\theta' \mid \theta) = E(\log f(x \mid \theta') \mid y, \theta) \ ,$$

that is, the expected value of $\log f(x \mid \theta')$ according to the conditional distribution of $x$ given $y$ and parameter $\theta$. The expectation is assumed to exist for all pairs $(\theta', \theta)$. In particular, it is assumed that $f(x \mid \theta) > 0$ for $\theta \in \Omega$.

## EM algorithm iterations

- E-step: Compute $Q(\theta \mid \theta^{(p)})$.
- M-step: Choose $\theta^{(p+1)}$ to be a value of $\theta \in \Omega$ that maximizes $Q(\theta \mid \theta^{(p)})$.
- M-step is typically easy
- This guarantees that the algorithm will converge to a local maximum
- Same issue as K-means
- Different initial values, see which one maximizes the likelihood

## EM for the Mixture of Normals

- Observed data (incomplete): $\{x_1, x_2, ..., x_n\}$, where $n$ is the sample size. Denote all the samples collectively by $\mathbf{x}$.
- Complete data: $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where $y_i$ is the cluster (component) identity of sample $x_i$.
- The collection of parameters, $\theta$, includes: $a_k$, $\mu_k$, $\Sigma_k$, $k = 1, 2, ..., K$.
- The likelihood function is:

$$L(\mathbf{x}|\theta) = \sum_{i=1}^{n} \log \left( \sum_{k=1}^{K} a_k \phi(x_i|\mu_k, \Sigma_k) \right) .$$

- $L(\mathbf{x}|\theta)$ is the objective function of the EM algorithm (maximize). Numerical difficulty comes from the sum inside the log.

## EM for the Mixture of Normals (cont'd.)

- The $Q$ function is:

$$\begin{aligned} Q(\theta^*|\theta) &= E\left[\log \prod_{i=1}^{n} a'_{y_i} \phi(x_i \mid \mu'_{y_i}, \Sigma'_{y_i}) \mid \mathbf{x}, \theta\right] \\ &= E\left[\sum_{i=1}^{n} \left(\log(a'_{y_i}) + \log \phi(x_i \mid \mu'_{y_i}, \Sigma'_{y_i}\right) \mid \mathbf{x}, \theta\right] \\ &= \sum_{i=1}^{n} E\left[\log(a'_{y_i}) + \log \phi(x_i \mid \mu'_{y_i}, \Sigma'_{y_i}) \mid x_i, \theta\right] . \end{aligned}$$

  The last equality comes from the fact the samples are independent.
- Note that when $x_i$ is given, only $y_i$ is random in the complete data $(x_i, y_i)$. Also $y_i$ only takes a finite number of values, i.e, cluster identities 1 to $K$. The distribution of $Y$ given $X = x_i$ is the posterior probability of $Y$ given $X$.

## EM for the Mixture of Normals (cont'd.)

- Once posterior probabilities (E-step) are estimated, maximizers (M-step) are easy to compute
  1. Initialize parameters
  2. E-step: Compute the posterior probabilities for all $i = 1, ..., n$, $k = 1, ..., K$.
$$p_{i,k} = \frac{a_k^{(p)}\phi(x_i \mid \mu_k^{(p)}, \Sigma_k^{(p)})}{\sum_{k=1}^K a_k^{(p)}\phi(x_i \mid \mu_k^{(p)}, \Sigma_k^{(p)})}\ .$$
  3. M-step:
$$a_k^{(p+1)} = n^{-1}\sum_{i=1}^n p_{i,k} \quad \mu_k^{(p+1)} = \sum_{i=1}^n p_{i,k}x_i / \sum_{i=1}^n p_{i,k}$$
$$\Sigma_k^{(p+1)} = \frac{\sum_{i=1}^n p_{i,k}(x_i - \mu_k^{(p+1)})(x_i - \mu_k^{(p+1)})^t}{\sum_{i=1}^n p_{i,k}}$$
  4. Repeat step 2 and 3 to convergence.

## Remarks

- One can use a model selection criteria like BIC to choose $K$; it works well empirically
- In practice, we may want to reduce model complexity by putting constraints on the parameters. For instance, assume equal priors, identical covariance matrices for all the components.
- If a different $\Sigma_k$ is allowed for each component, the likelihood function is not bounded. Global optimum is meaningless. (Don't overdo it!)
- Initializing group assignments: one approach is with k-means
  1. Apply k-means first.
  2. Initialize $\mu_k$ and $\Sigma_k$ using all the samples classified to cluster $k$.
  3. Initialize $a_k$ by the proportion of data assigned to cluster $k$ by k-means.

## Clustering and Big Data

- Many of the ideas can be adopted here
- Distributed computing across nodes for E and M-step of the EM algorithm or for k-means
- Randomized directions of descent