CS150: Database & Datamining
Final exam review

ShanghaiTech-SIST

Spring 2019

Acknowledgement: Slides are adopted from the Berkeley course CS186 by Joey Gonzalez and Joe Hellerstein, Stanford CS145 by Peter Bailis.

Transactions: Basic Definition

A **transaction ("TXN")** is a sequence of one or more ***operations*** (reads or writes) which reflects ***a single real-world transition***.

In the real world, a TXN either happened completely or not at all

```
START TRANSACTION
        UPDATE Product
        SET Price = Price – 1.99
        WHERE pname = 'Gizmo'
COMMIT
```

Transaction Properties: ACID

- Atomic
  - State shows either all the effects of txn, or none of them
- Consistent
  - Txn moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of txns is the same as txns running one after another (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

ACID continues to be a source of great debate!

ACID: Atomicity

- TXN's activities are atomic: **all or nothing**
  - Intuitively: in the real world, a transaction is something that would either occur *completely* or *not at all*
- Two possible outcomes for a TXN
  - It *commits*: all the changes are made
  - It *aborts*: no changes are made

A**C**ID: **C**onsistency

- The tables must always satisfy user-specified ***integrity constraints***
  - *Examples:*
    - Account number is unique
    - Stock amount can't be negative
    - Sum of *debits* and of *credits* is 0

- How consistency is achieved:
  - Programmer makes sure a txn takes a consistent state to a consistent state
  - *System* makes sure that the txn is **atomic**

ACID: Isolation

- A transaction executes concurrently with other transactions
- **Isolation**: the effect is as if each transaction executes in *isolation* of the others.
  - E.g. Should not be able to observe changes from other transactions during the run

ACI**D**: **D**urability

- The effect of a TXN must continue to exist (***"persist"***) after the TXN
  - And after the whole program has terminated
  - And even if there are power failures, crashes, etc.
  - And etc…
- Means: Write data to **disk**

Change on the horizon?
Non-Volatile Ram (NVRam).
Byte addressable.

Write-Ahead Logging (WAL)

- DB uses **Write-Ahead Logging (WAL)** Protocol:

Each update is logged! Why not reads?

1. Must *force log record* for an update *before* the corresponding data page goes to storage

→ Atomicity

2. Must *write all log records* for a TX *before commit*

→ Durability

I.e. transaction is not committed until all of its log records— including its "commit" record—are on the stable log.

Concurrency: Isolation & Consistency

- The DBMS must handle concurrency such that…

1. **Isolation** is maintained: Users must be able to execute each TXN **as if they were the only user**
   - DBMS handles the details of *interleaving* various TXNs

ACID

2. **Consistency** is maintained: TXNs must leave the DB in a **consistent state**
   - DBMS handles the details of enforcing integrity constraints

ACID

Why Interleave TXNs?

- Interleaving TXNs might lead to anomalous outcomes… why do it?

- Several important reasons:
  - Individual TXNs might be *slow*- don't want to block other users during!

  - Disk access may be *slow*- let some TXNs use CPUs while others accessing disk!

All concern large differences in ***performance***

Interleaving & Isolation

- The DBMS has freedom to interleave TXNs
- However, it must pick an interleaving or **schedule** such that isolation and consistency are maintained
  - Must be *as if* the TXNs had executed serially!

"With great power comes great responsibility"

ACID

DBMS must pick a schedule which maintains isolation & consistency

Scheduling Definitions

- A **serial schedule** is one that does not interleave the actions of different transactions

- A and B are **equivalent schedules** if, ***for any database state***, the effect on DB of executing A **is identical to** the effect of executing B

- *A* **serializable schedule** is a schedule that is equivalent to ***some*** serial execution of the transactions.

The word "**some**" makes this definition powerful & tricky!

Conflict Types

Two actions **conflict** if they are part of different TXNs, involve the same variable, and at least one of them is a write

- Thus, there are three types of conflicts:
  - Read-Write conflicts (RW)
  - Write-Read conflicts (WR)
  - Write-Write conflicts (WW)

Why no "RR Conflict"?

Interleaving anomalies occur with / because of these conflicts between TXNs *(but these conflicts can occur without causing anomalies!)*

See next section for more!

Conflict Serializability

- Two schedules are **conflict equivalent** if:
  - They involve *the same actions of the same TXNs*
  - Every *pair of conflicting actions* of two TXNs are *ordered in the same way*
- Schedule S is **conflict serializable** if S is *conflict equivalent* to some serial schedule

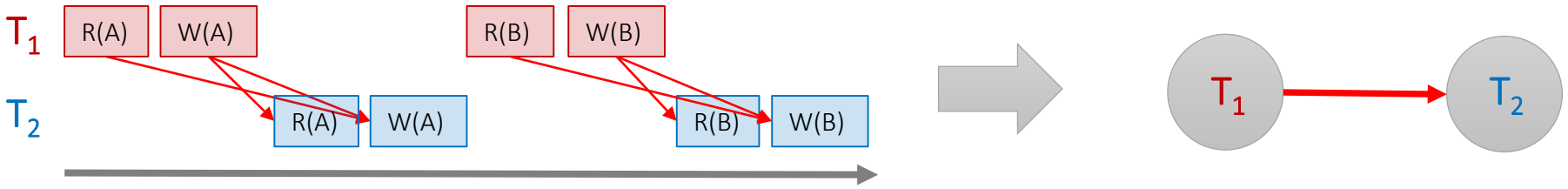
Conflict serializable ⇒ serializable

So if we have conflict serializable, we have consistency & isolation!

The Conflict Graph

- Let's now consider looking at conflicts **at the TXN level**

- Consider a graph where the **nodes are TXNs**, and there is an edge from $T_i \rightarrow T_j$ **if an action in T_i precede and conflict with an action in T_j**



Connection to conflict serializability

- In the conflict graph, a topological ordering of nodes corresponds to **a serial ordering of TXNs**

- Thus an **acyclic** conflict graph → conflict serializable!

Theorem: Schedule is **conflict serializable** if and only if its conflict graph is **acyclic**

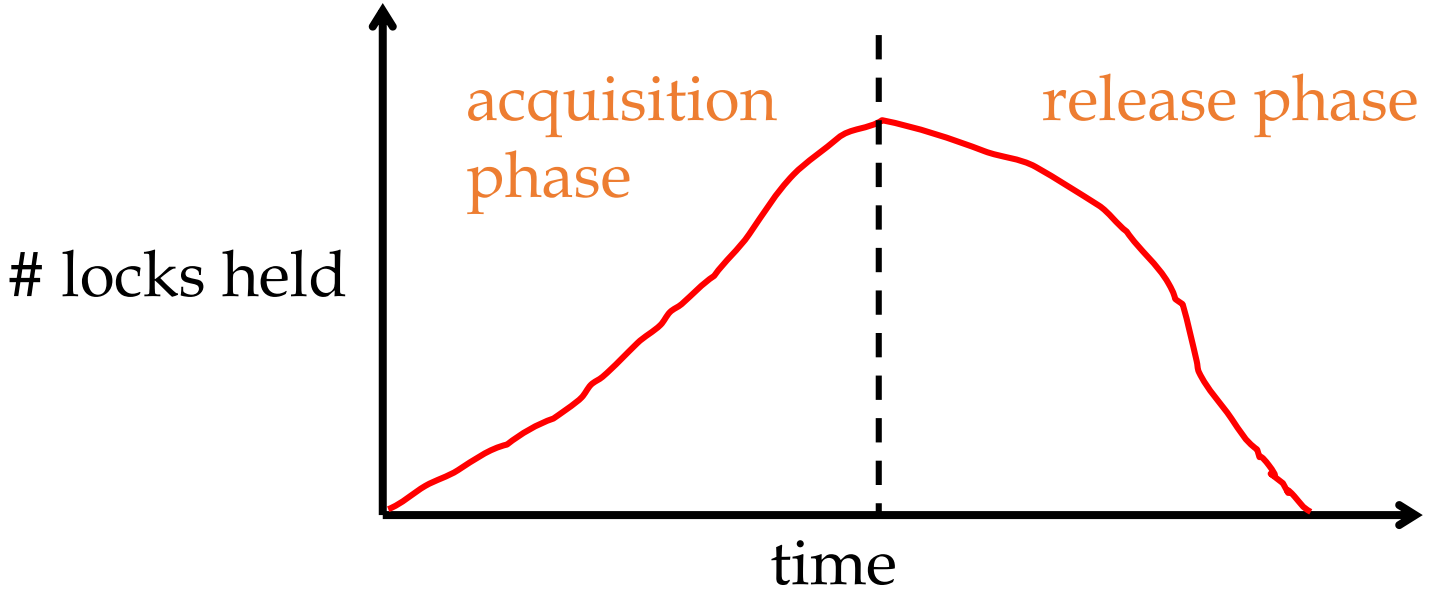
Two-phase Locking (2PL) Protocol:

TXNs obtain:

- An **X (*exclusive*) lock** on object before **writing**.
  - If a TXN holds, no other TXN can get a lock (S or X) on that object.
- An **S (*shared*) lock** on object before **reading**
  - If a TXN holds, no other TXN can get *an X lock* on that object
- TXNs cannot get new locks after releasing any locks.

Note: Terminology here- “exclusive”, “shared”- meant to be intuitive- no tricks!

Two-Phase Locking (2PL)



Lock Compatibility Matrix

	S	X
S	√	–
X	–	–

2PL guarantees conflict serializability ☺

But, does *not* prevent **Cascading Aborts**. ☹

Strict Two-phase Locking (Strict 2PL) Protocol:

TXNs obtain:

- Same as 2PL, except:
- All locks held by a TXN are released only when TXN completes. i.e., either:

  (a) transaction has committed (commit record on disk),

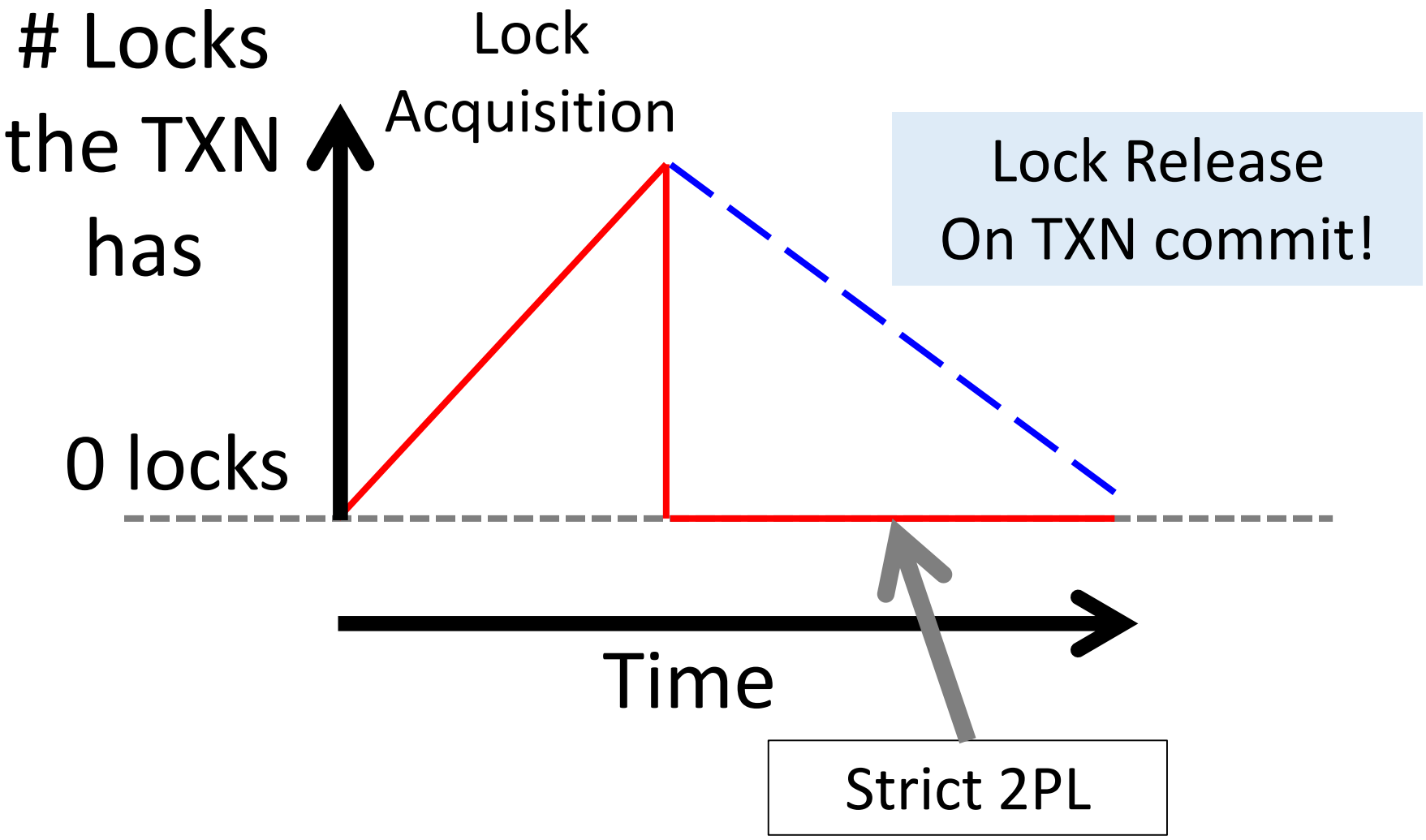
  or

  (b) transaction has aborted and rollback is complete.

Lock Compatibility Matrix

	S	X
S	√	–
X	–	–

Picture of 2-Phase Locking (2PL)



Strict 2PL

Theorem: Strict 2PL allows only schedules whose dependency graph is acyclic

Proof Intuition: In strict 2PL, if there is an edge $T_i \rightarrow T_j$ (i.e. T_i and T_j conflict) then T_j needs to wait until T_i is finished – so *cannot* have an edge $T_j \rightarrow T_i$

Therefore, Strict 2PL only allows conflict serializable ⇒ serializable schedules

Deadlocks

- **Deadlock**: Cycle of transactions waiting for locks to be released by each other.

- Two ways of dealing with deadlocks:

  1. Deadlock prevention

  2. Deadlock detection

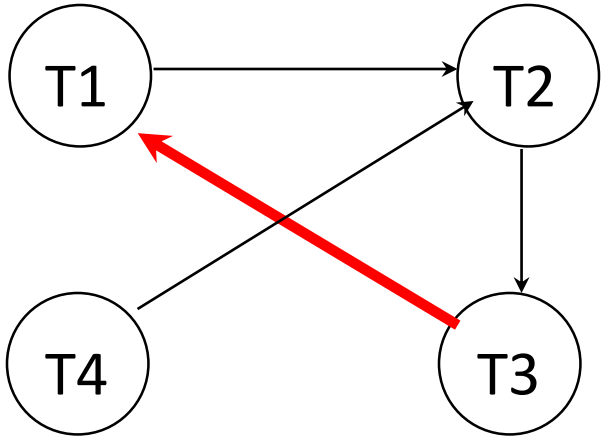
Deadlock Detection

- Create the **waits-for graph**:
  - Nodes are transactions
  - There is an edge from $T_i \rightarrow T_j$ if T_i is *waiting for T_j to release a lock*
- Periodically check for (***and break***) cycles in the waits-for graph

Deadlock Detection Example

Example:

T1: S(A), S(D), S(B)
T2: X(B) X(C)
T3: S(D), S(C), X(A)
T4: X(B)



Deadlock Avoidance

- Assign priorities based on timestamps.
- Say Ti wants a lock that Tj holds

Two possible policies:

Read the names like a “ternary predicate” on priorities, with the form:

(Ti > Tj) ? X : Y;

Wait-Die: if Ti has higher priority, Ti waits for Tj;
else Ti aborts

Wound-wait: if Ti has higher priority, Tj aborts;
else Ti waits

Data Warehouse

Collects and organizes historical data from multiple sources

Data is *periodically* **ETL**ed into the data warehouse:

- **Extracted** from remote sources
- **Transformed** to standard schemas
- **Loaded** into the (typically) relational system

Extracting Data from Sources

- Need to collect data from multiples sources
  - Various RDBMS vendors
  - Structured files JSON, XML
- Often done using SQL interfaces
- Validate extracted data
  - Flag corrupted records …

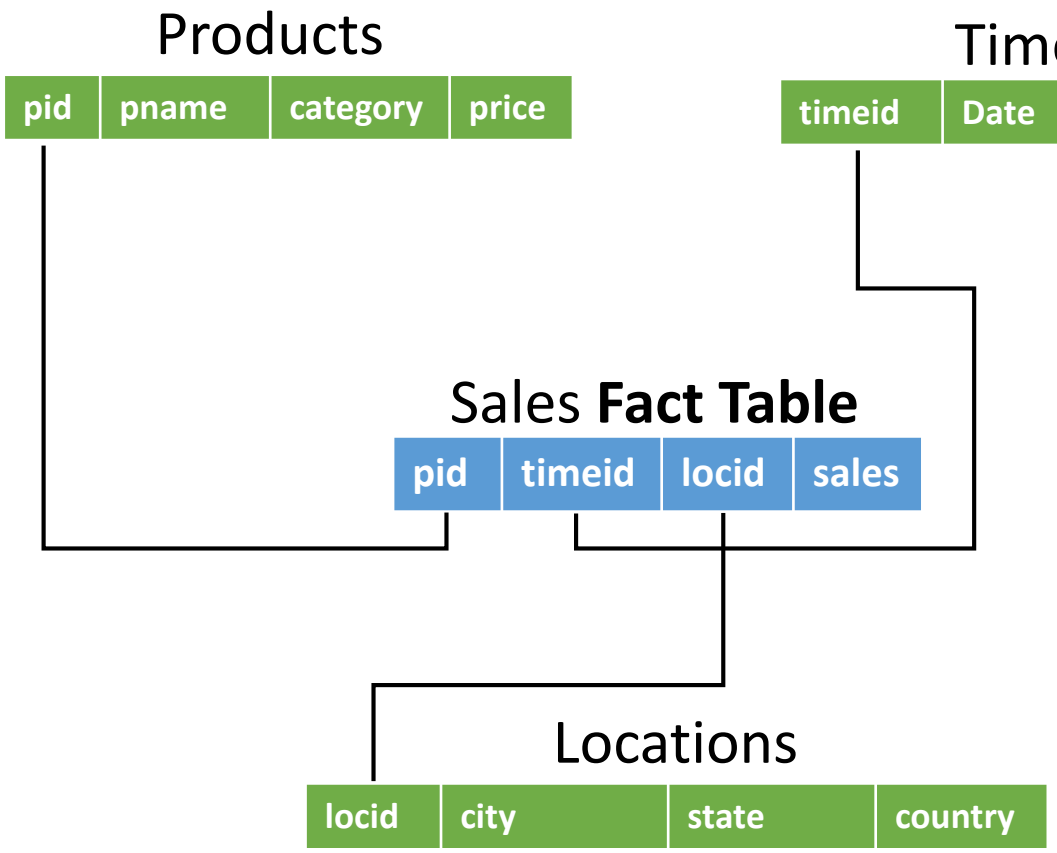
Transforming “Cleaning” Data

- Additional data validation and filtering
- Schema manipulation
  - Extract key fields
  - Encoding text
  - Verifying and enforcing constraints
- Data normalization (time zones, currency)

Loading Data

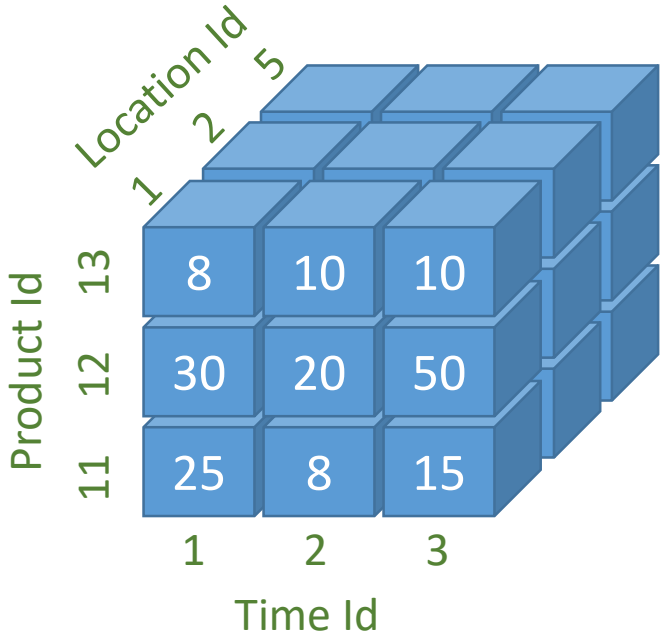
- Data is bulk loaded into large relations
  - Fact tables … (more on this later)
- Update:
  - Indexes
  - Metadata tables: Data about the data
    - When and how was it collected
    - Meaning of fields
  - Updating materialized views … (more on this later)
- Occasionally move older data to archival storage
  - Data aging

Multidimensional Data: **Star Schema**

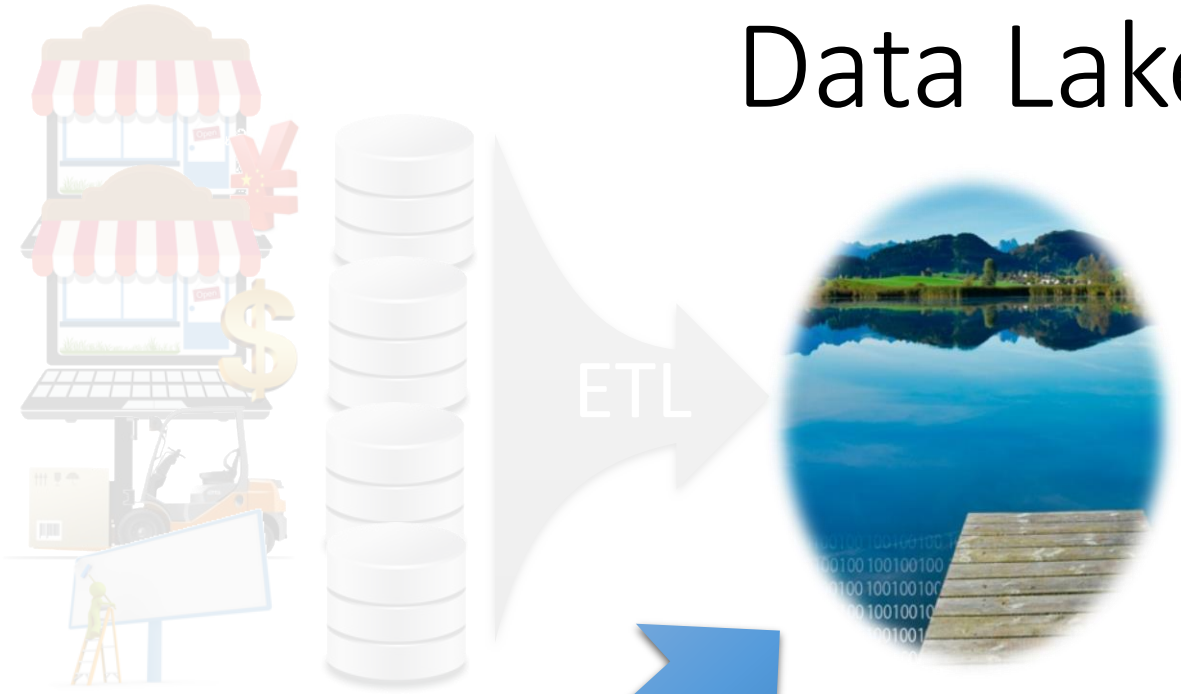


Dimension Tables

← This looks like a star …



Data Lake*



*Still being defined…

[Buzzword Disclaimer]

Text/Log Data

Save

Photos & Videos

Big Idea:

Maintain a copy of all the data in one place and *free** data consumers to choose how to transform and use it.

*free to solve all the problems themselves

Data Lake

- Store unstructured data in **raw form**
  - Schema-on-**Read**: *determine the best organization **when data is used***
  - **Contrast:** Data Warehouses are Schema-on-Load (ET***L***)
    - Plan ahead (Fact tables and Dimensions)
- Often much **larger** than data warehouses
- Technologies
  - **Storage:** Large distributed file systems (e.g., HDFS)
    - Semi-structured formats (JSON, Parquet)
  - **Computation:** Map-Reduce
    - Recent trend to add SQL (or SQL like) functionality
- More Agile (?):
  - Don’t worry about schema & verification when loading
  - Disaggregated compute and storage → BYOF
    - bring your own compute frameworks …

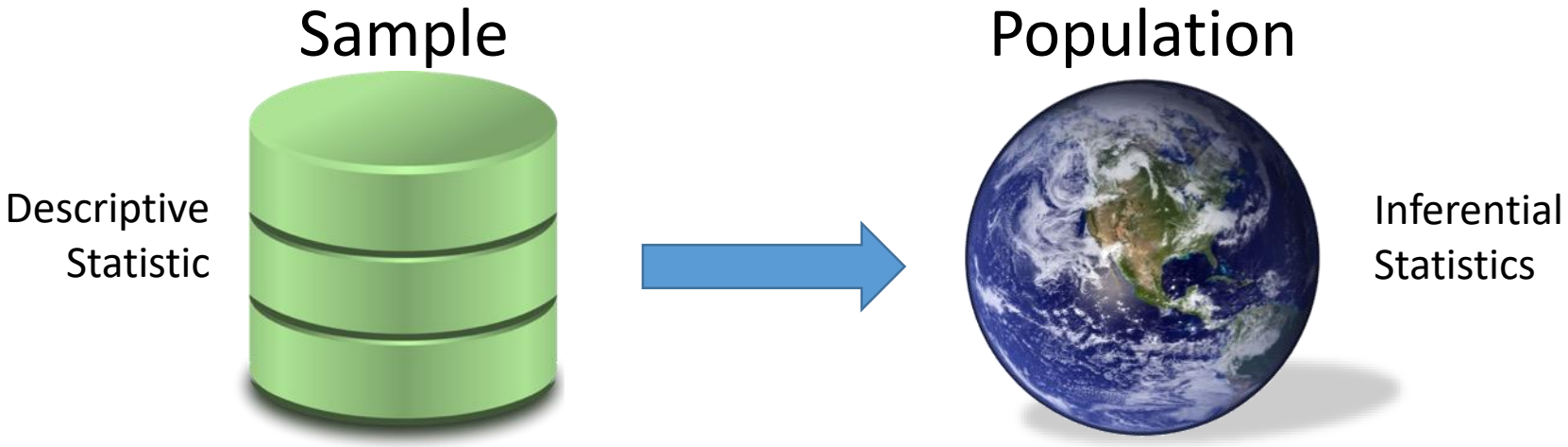
- **What could go wrong?**

Data Lake → Data Swamp

- Cultural shift: *Curate* → Save **Everything**!
  - Signal to Noise ratio drops …
- Limited data governance → more agile →

  hdfs://important/joey_big_file3.csv_with_json
  - **What** does it contain? **What** are all the **"fields"**
  - **When** and **how** and **from where** was it created
- Without cleaning and verification we begin to collect a rich history of **dirty data**
- Limited compatible with traditional tools

Descriptive vs. **Inferential** Statistics



- **Descriptive Statistics:** *describe* the sample data
  - Example: *Average* sales last quarter
  - Can be **measured directly** from the database
- **Inferential Statistics:** *estimate* the population
  - Example: *Expected* sales next quarter
  - May be **estimated** using descriptive statistics

The Basic KDD Process

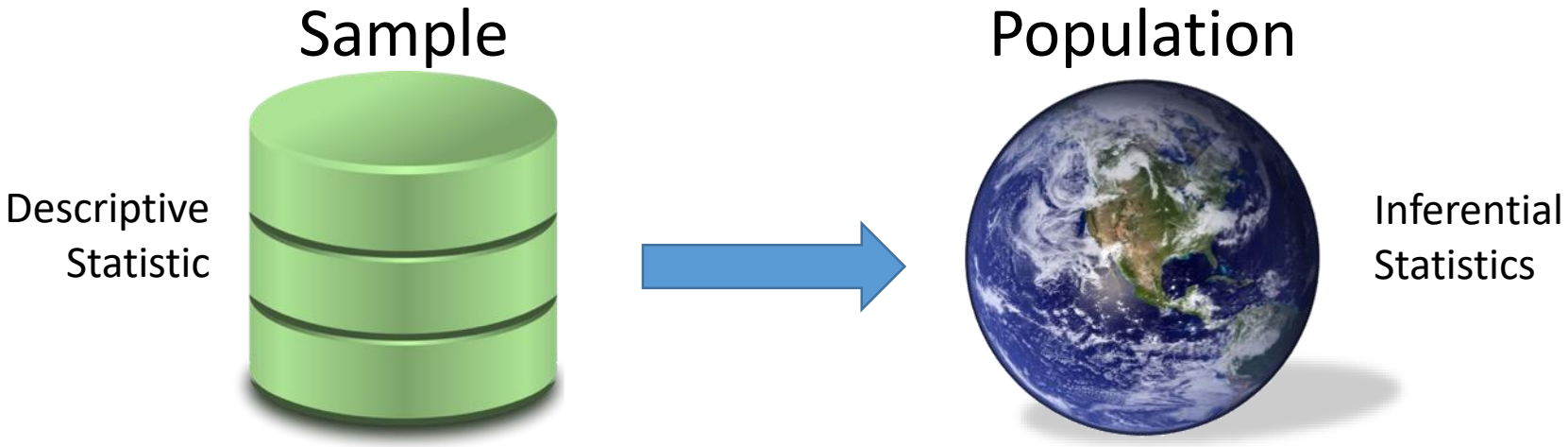
- **Data Selection:** *What data do I need for a given task?*
  - If data was already collected, how was the data collected?

- **Data Cleaning:** *Preparing the data for a given task*
  - Typically most challenging (time consuming) part.
  - Why might ETL not be enough?

- **Data Mining & ML:** *Running algorithms to infer patterns*
  - The fun part! Many tools, many options, complex tradeoffs.

- **Evaluation:** *Verifying that patterns are significant*
  - Algorithms will typically find patterns especially when none exist.

Descriptive vs. **Inferential** Statistics



- **Descriptive Statistics:** *describe* the sample data
  - Example: *Average* sales last quarter
  - Can be **measured directly** from the database
- **Inferential Statistics:** *estimate* the population
  - Example: *Expected* sales next quarter
  - May be **estimated** using descriptive statistics

The Basic KDD Process

- **Data Selection:** *What data do I need for a given task?*
  - If data was already collected, how was the data collected?

- **Data Cleaning:** *Preparing the data for a given task*
  - Typically most challenging (time consuming) part.
  - Why might ETL not be enough?

- **Data Mining & ML:** *Running algorithms to infer patterns*
  - The fun part! Many tools, many options, complex tradeoffs.

- **Evaluation:** *Verifying that patterns are significant*
  - Algorithms will typically find patterns especially when none exist.

What is Machine Learning?

Study of algorithms that:

- That improve their **performance**
  - Ability to understand what you are saying
- at some **task**
  - Voice recognition
- through **experience**
  - Transcribed speech data

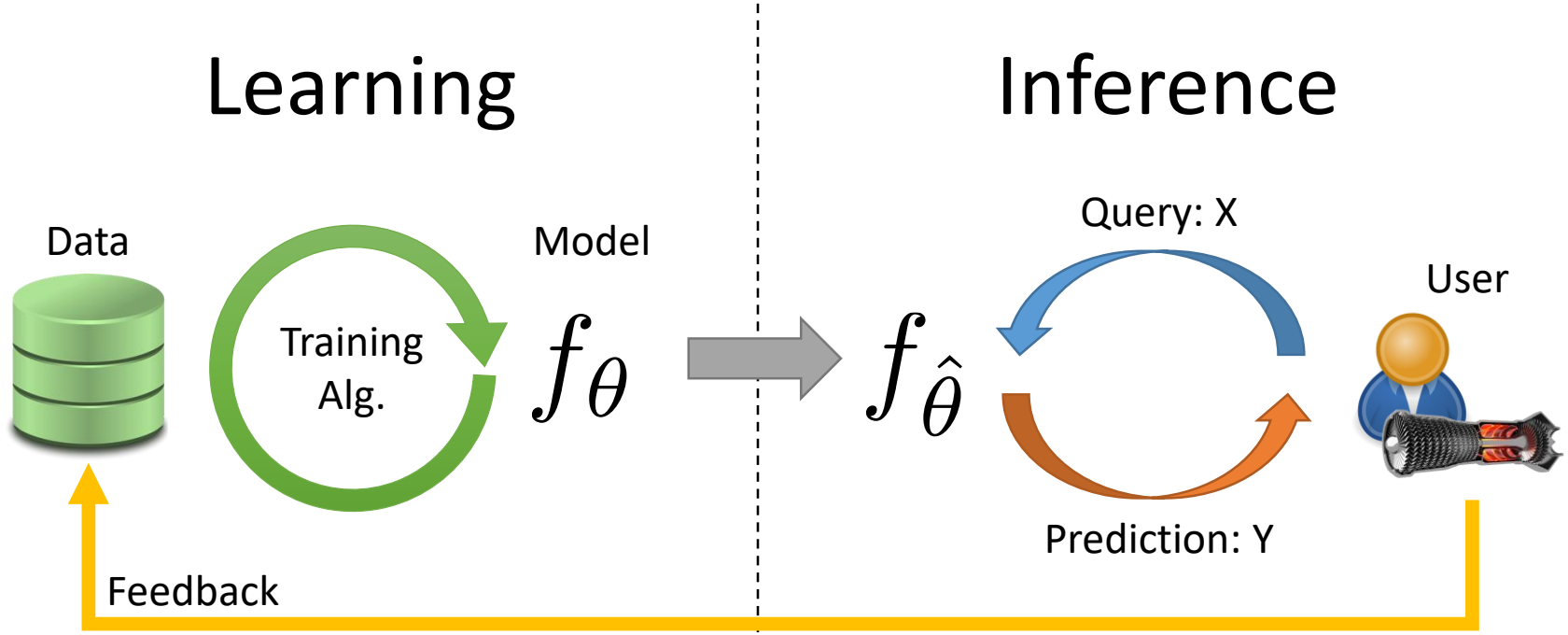
-- Prof. Tom Mitchell, *CMU*

*"Machine Learning is the **second best** solution to any problem. The **first best** is of course to **solve the problem** directly."*

-- Prof. Yaser S. Abu-Mostafa, *Caltech*

How would you write a program to recognize human speech?

Machine Learning Lifecycle

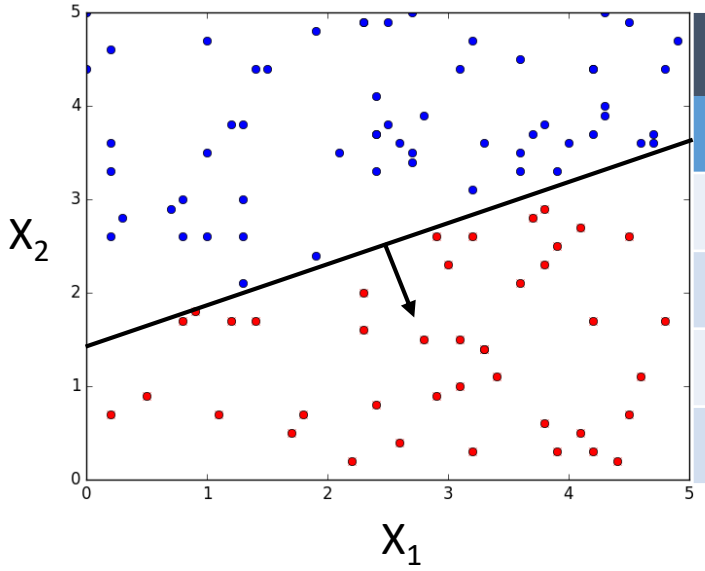


- Typically a time consuming iterative batch process
  - Feature engineering
  - Validation

➢Focus is on making fast robust predictions
  - Monitoring and tracking feedback
  - Materialization + fast model inference

Learning: *Fitting the Model*

- **Training Data**
  - **X**: Features
  - **Y**: Label/Obs.
- Learn a function that **generalizes** the relationship between *X* and *Y*



X		Y
x_1	x_2	
3.9	3.1	T
0.2	0.1	F
4.2	0.9	F
1.1	2.9	T

Features

Function class / Model Family

$$f_\theta(X) \rightarrow Y$$

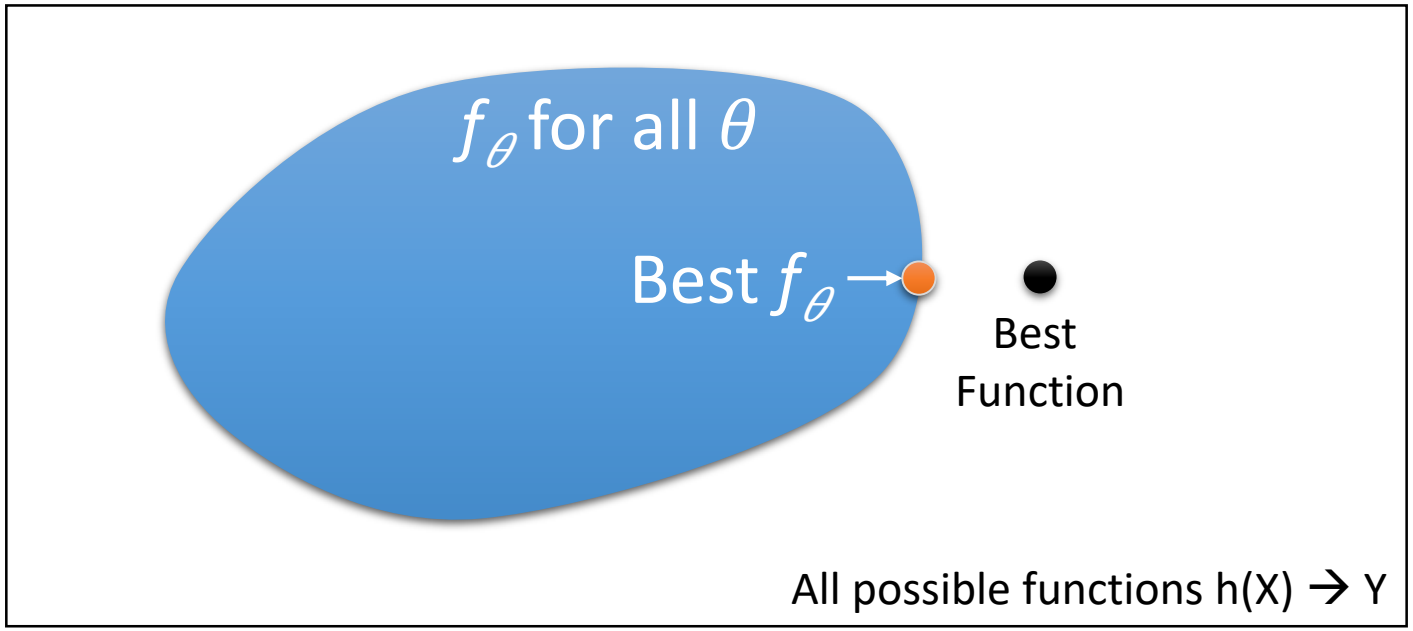
Labels / Observations

Model Parameters

Finding the Best Parameters

$$f_\theta(X) \rightarrow Y$$

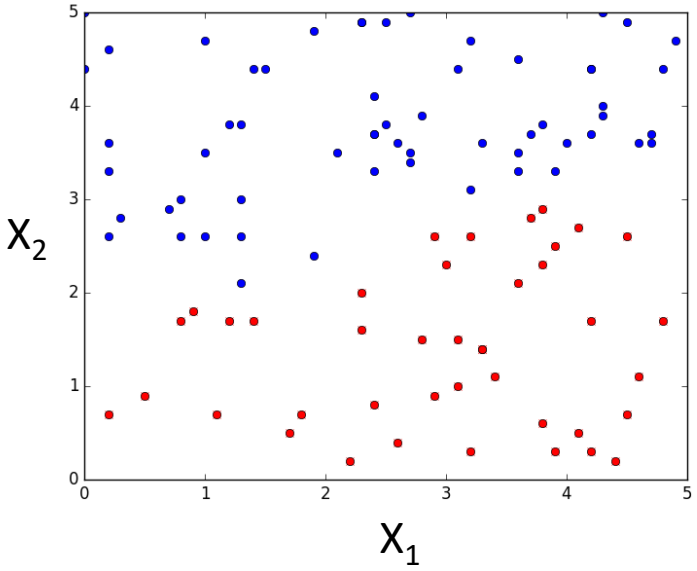
- Define some **objective** (e.g., prediction error)
- Search for best θ with respect to the objective

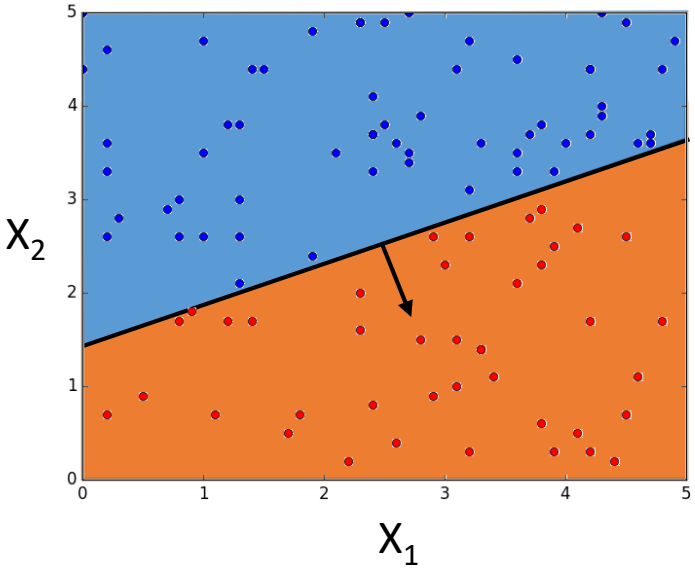


Generalization …

Sample

Population





Inference: *Rendering Predictions*

- Evaluating the model on input queries:

$$f_{\hat{\theta}}(X) \rightarrow Y$$

- Online vs Offline:
  - Pre-computed **offline**: *movie rankings*
  - Computed **online** with each query: *speech recognition*
- May want to track confidence in prediction
- May require additional pre and post-processing
  - Feature lookup, content ranking, etc…

Feedback: *Incorporating New Data*

- After rendering a prediction we may get feedback on the results of the prediction:
  - **Explicit**: the *correct value* was “cat”
  - **Implicit**: the predicted animal was *incorrect*
  - Can be **noisy** …

- Watch out for **sample bias:**
  - Model affects the data is uses for training in the future
  - **Example**: only play top40 songs ...

Taxonomy
of Machine Learning

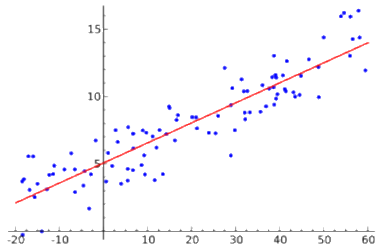
Labeled Data

Indirect (reward)

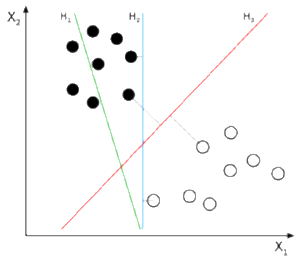
Unlabeled Data

Supervised Learning

Regression



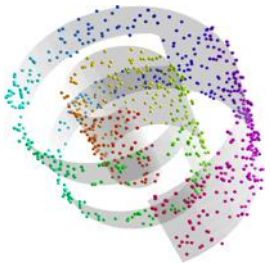
Classification



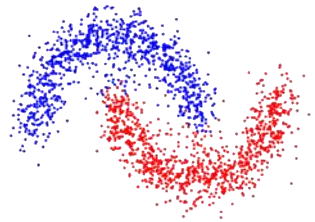
Reinforcement & Bandit Learning

Unsupervised Learning

Dimensionality Reduction



Clustering



K-Means Algorithm: **Details**

```
centers ← pick k initial Centers

while (centers are changing) {
    // Compute the assignments (E-Step)
    asg ← [(x, nearest(centers, x)) for x in data]
```

What do we mean by "nearest":

A: Euclidean Distance

$$\arg \min_{c \in \text{centers}} ||c - x||_2^2 = \sum_{i=1}^{d} (c_i - x_i)^2$$

K-Means Algorithm: **Details**

```
centers ← pick k initial Centers

while (centers are changing) {
   // Compute the assignments (E-Step)
   asg ← [(x, nearest(centers, x)) for x in data]

   // Compute the new centers (M-Step)
   for i in range(k):
      centers[i] =
         mean([x for (x, c) in asg if c == i])
}
```

Compute the "Expected" Assignment

Find centers that maximize the data "likelihood"

K-Means Algorithm: **Details**

```
centers ← pick k initial Centers

while (centers are changing) {
   // Compute the assignments (E-Step)
   asg ← [(x, nearest(centers, x)) for x in data]

   // Compute the new centers (M-Step)
   for i in range(k):
      centers[i] =
         mean([x for (x, c) in asg if c == i])
}
```

Guaranteed to converge!

… to what?

To a local optimum. ☹

Depends on Initial Centers

K-Means Algorithm: **Details**

```
centers ← pick k initial Centers
```

How do we pick initial centers?

```
while (centers are changing) {
   // Compute the assignments (E-Step)
   asg ← [(x, nearest(centers, x)) for x in data]

   // Compute the new centers (M-Step)
   for i in range(k):
      centers[i] =
         mean([x for (x, c) in asg if c == i])
}
```

Guaranteed to converge!

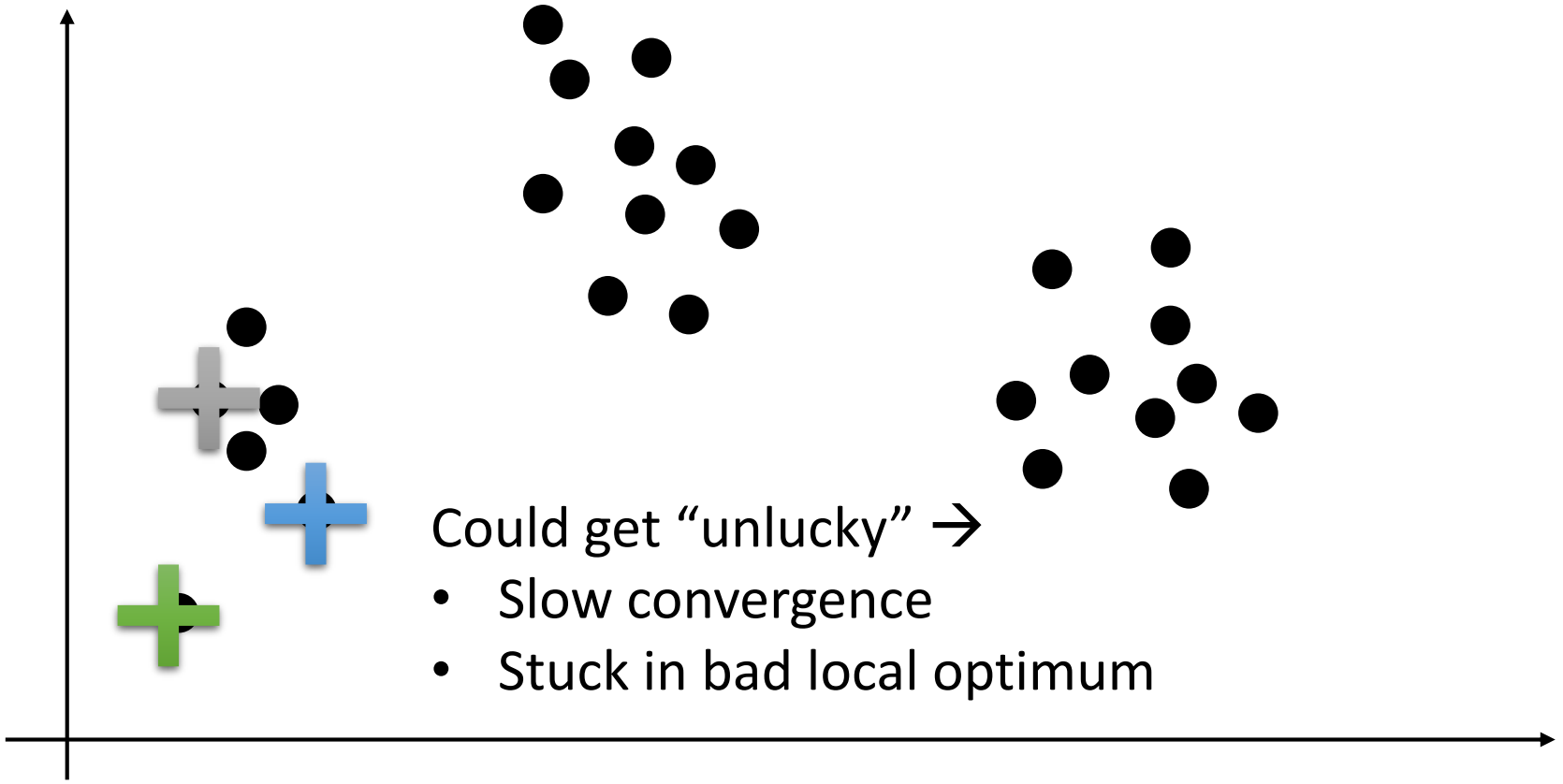
… to what?

To a local optimum. ☹

Depends on Initial Centers

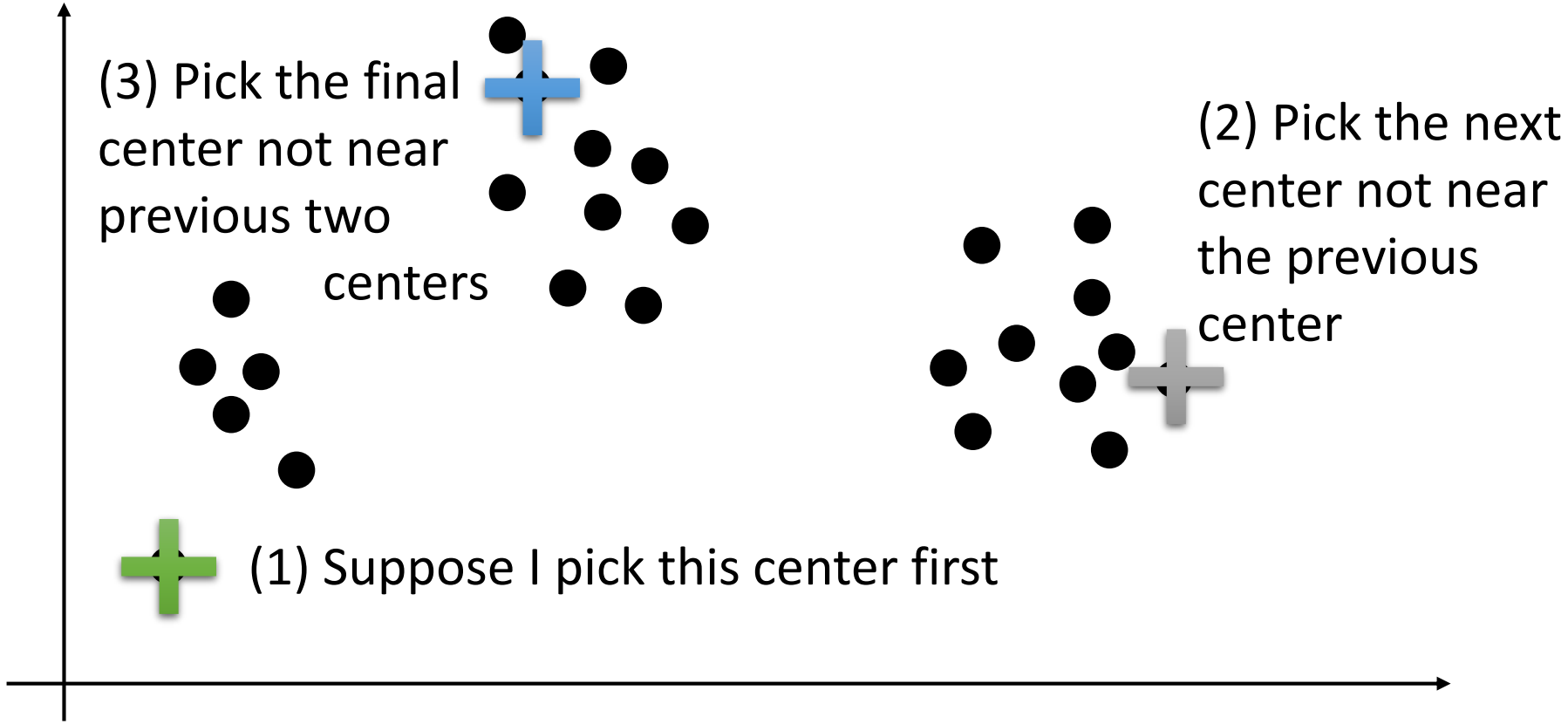
Picking the Initial Centers

- **Simple Strategy:** select *k* points at random
  - What could go wrong?



Picking the Initial Centers

- **Better Strategy:** kmeans++
  - Randomized approx. algorithm
  - Intuition select points that are not near existing centers

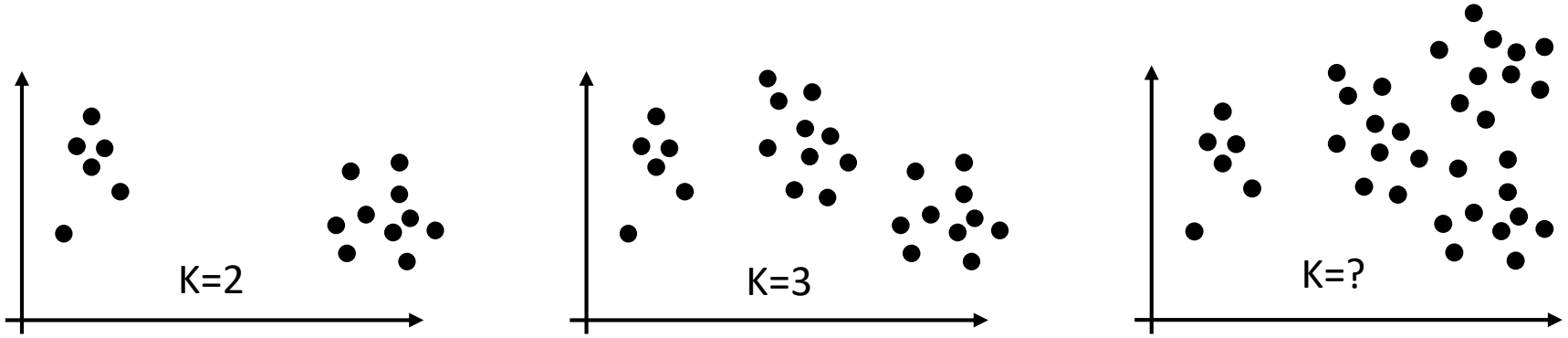


K-Means++ Algorithm

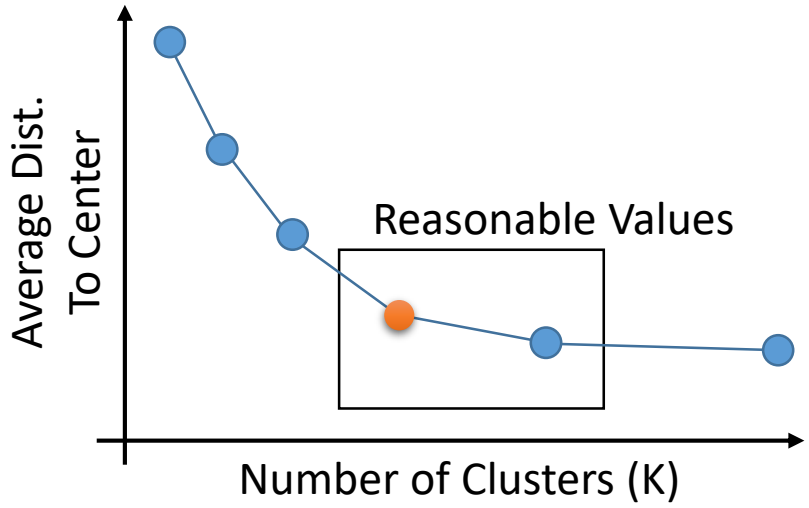
```
centers ← set(randomly select a single point)


while len(centers) < k:
  # Compute the distance of each point
  # to its nearest center dSq = d^2
  dSq ← [(x, dist_to_nearest(centers, x)^2) for x in data]
  # Sample a new point with probability
  # proportional to dSq
  c ← sample_one(data, prob = dSq / sum(dSq))
  # Update the clusters
  centers.add(c)
```

http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

How do we choose K?



- Basic Elbow Method (Easy and what you do in HW)
  - Try range of K-values and plot average distance to centers
- Cross-Validation (Better)
  - Repeatedly split the data into training and validation datasets
  - Cluster the training dataset
  - Measure Avg. Dist. To Centers on validation data

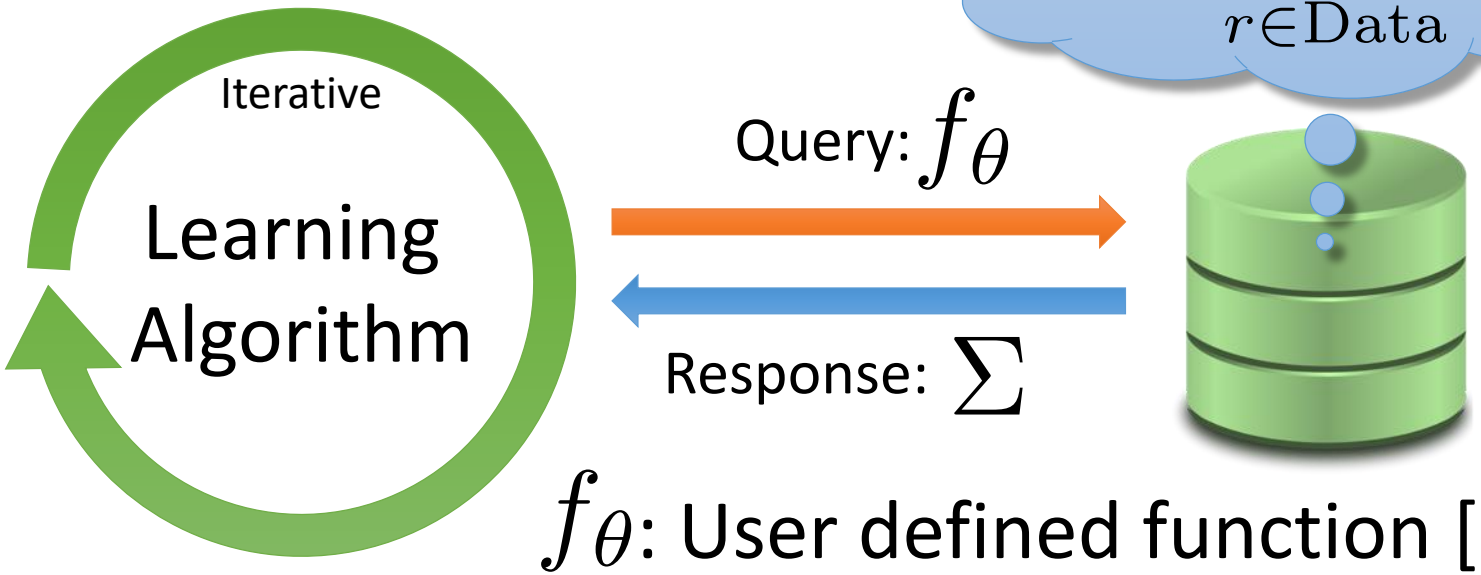


Statistical Query Pattern

Common Machine Learning Pattern

- Computing aggregates of user defined functions
- Data-Parallel computation
  - Scalable!

$$\Sigma = \bigoplus_{r \in \text{Data}} f_\theta(r)$$

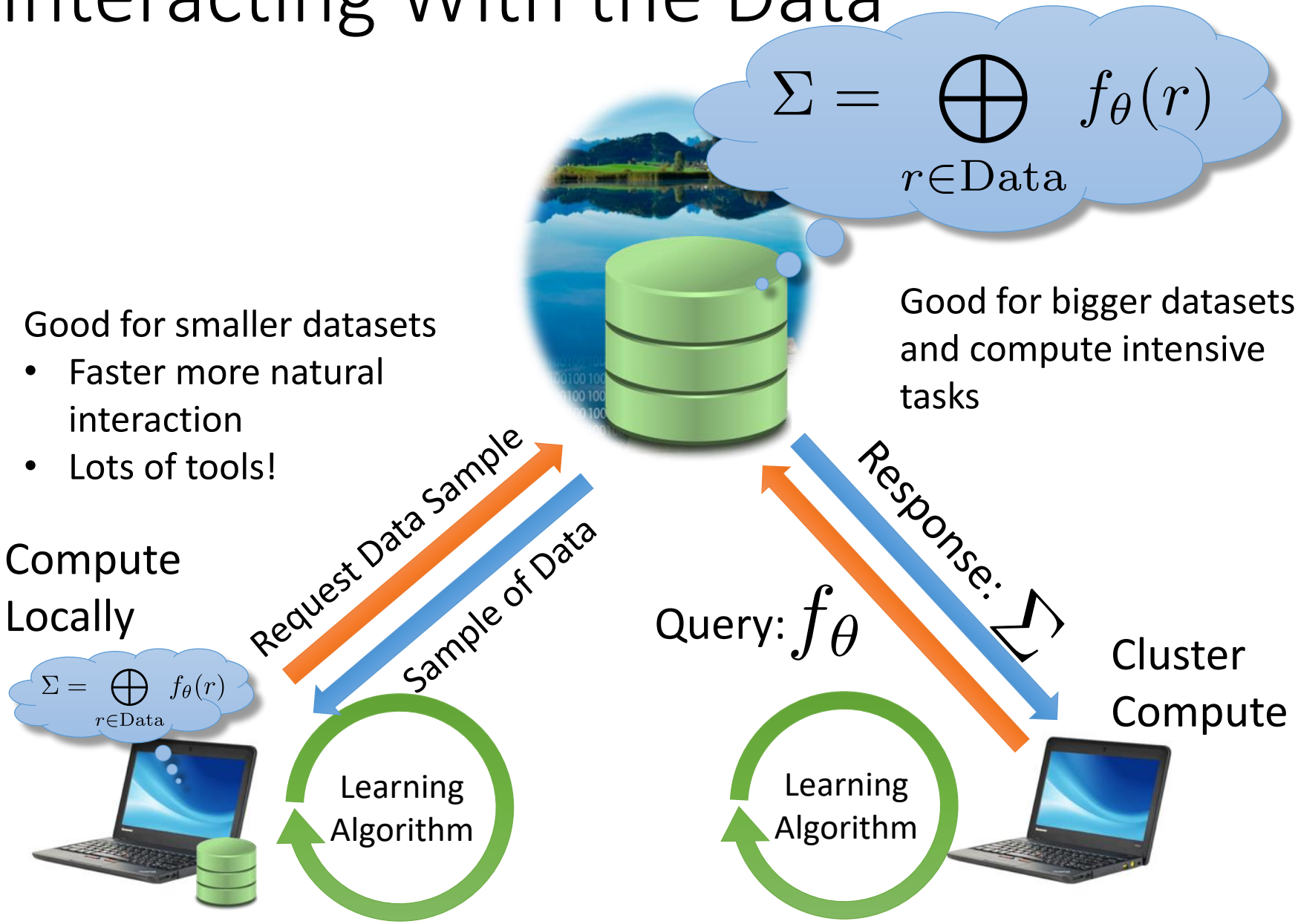


f_θ: User defined function [**UDF**]

$\bigoplus$: User defined aggregate [**UDA**]

D. Caragea et al., *A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees*. Int. J. Hyb

Interacting With the Data



Can we express K-Means in the **Statistical Query Pattern?**

```
centers ← pick k initial Centers
while (centers are changing):
    for i in range(k):
        new_centers[i] =
            mean([x for x in data if nearest(centers, x) == i])
    centers = new_centers
```

Group by query:

```
SELECT nearest_UDF(centers, x) AS cid, mean_UDA(x)
FROM data GROUPBY cid
```

Can we express K-Means++ in the **Statistical Query Pattern**?

- *Yes*, however there is a better version: **K-Means||**
  - More complex but much faster
- Or you can parallelize **K-Means++** directly
  - Requires more passes
- Challenging Step?
  - Parallel weighted sampling:

```
sample_one(data, prob = dSq / sum(dSq))
```

  - How do you select one point uniformly at random?

Res-A: weighted reservoir sampling

- **Goal:** *Sample **k** records from a stream where record **i** is included in the sample with probability proportional to w_i*
- **Algorithm:**
  - For each record ***i*** draw a uniform random number:

$$u_i \sim \mathbf{Unif}(0, 1)$$

  - Select the top-k records ordered by: u_i^{1/w_i}

- **Common ML Pattern?**
  - **Query Function:** `[pow(rand(), 1 / record.w), record]`
  - **Agg. Function:** *top-k heap*

Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *Inf. Process. Lett.* 97, 5 (March 2006), 181-185

The Linear Model

Data:

x_1	x_2	y
1.1	2.7	3.6
4.2	3.2	7.5
9.8	9.2	17
…	…	…

Vector of Parameters

Vector of Features

Real Valued Observations

$$y = \theta^T x + \epsilon$$

Real Value Noise

Linear Combination of Covariates

$$\sum_{i=1}^{p} \theta_i x_i$$

$$\theta, x \in \mathbb{R}^p$$

Feature Engineering

- A key part of most machine learning applications

- Common tasks:
  - Transforming raw features into **vector representations**
  - Encoding **prior knowledge** (e.g., translating currencies)
  - Transformations that **increase the expressivity** of the model … (more on this soon)

- Critical to model performance:
  - **engineers compete** to get the best features

- A few standard techniques (that we will cover):
  - one-hot encoding
  - bag-of-words

Finding the Best Parameters

Model: $f_\theta(x) := \theta^T x$

Step 1: define a **Loss Function:** *Average Prediction Error*

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_\theta(x_i)\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \theta^T x_i\right)^2$$

- Difference between **true (y_i)** and **predicted $f_\theta(x_i)$** values

Finding the Best Parameters

Model: $f_\theta(x) := \theta^T x$

Step 1: define a **Loss Function:**

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_\theta(x_i)\right)^2$$

Step 2: Search for best model parameters θ

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_\theta(x_i)\right)^2$$

Writing the data in Matrix form

- Represent data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ as:

Covariate (Design) Matrix

$$X = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ & \cdots & \\ — & x_n & — \end{bmatrix} \in \mathbb{R}^{np}$$

n

p

Response Vector

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^{n}$$

n

1

Minimizing the Squared Error

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$

- Setting equal to zero and solving for θ:

$$\sum_{i=1}^{n} (\theta^T x_i) x_i = \sum_{i=1}^{n} y_i x_i \Rightarrow X^T X \theta = X^T y$$

- Normal Equation:

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

- Solved using any standard linear algebra library

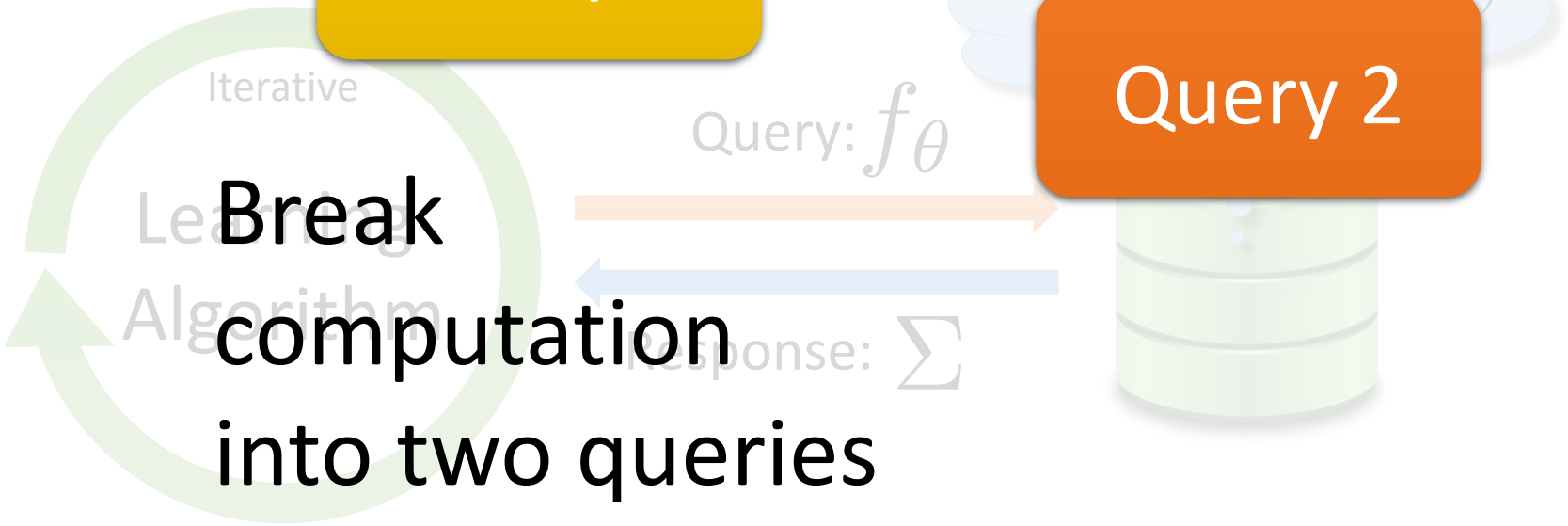
Can we compute

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

using the statistical query pattern?

Query 1

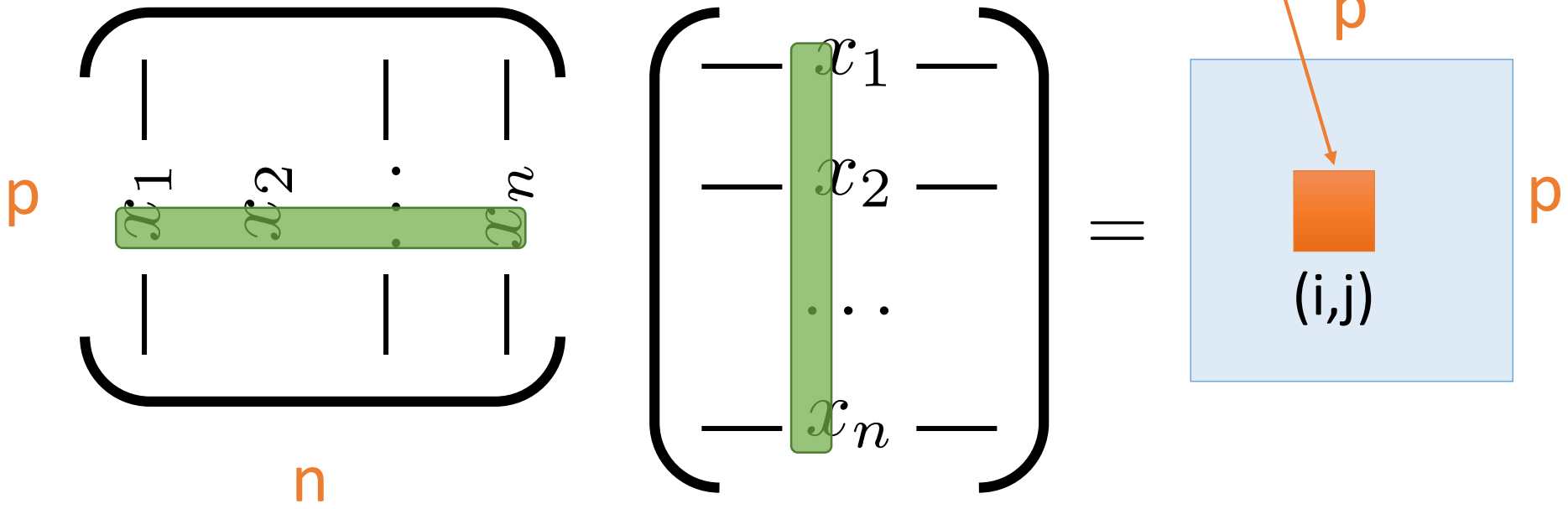
Query 2



Break computation into two queries

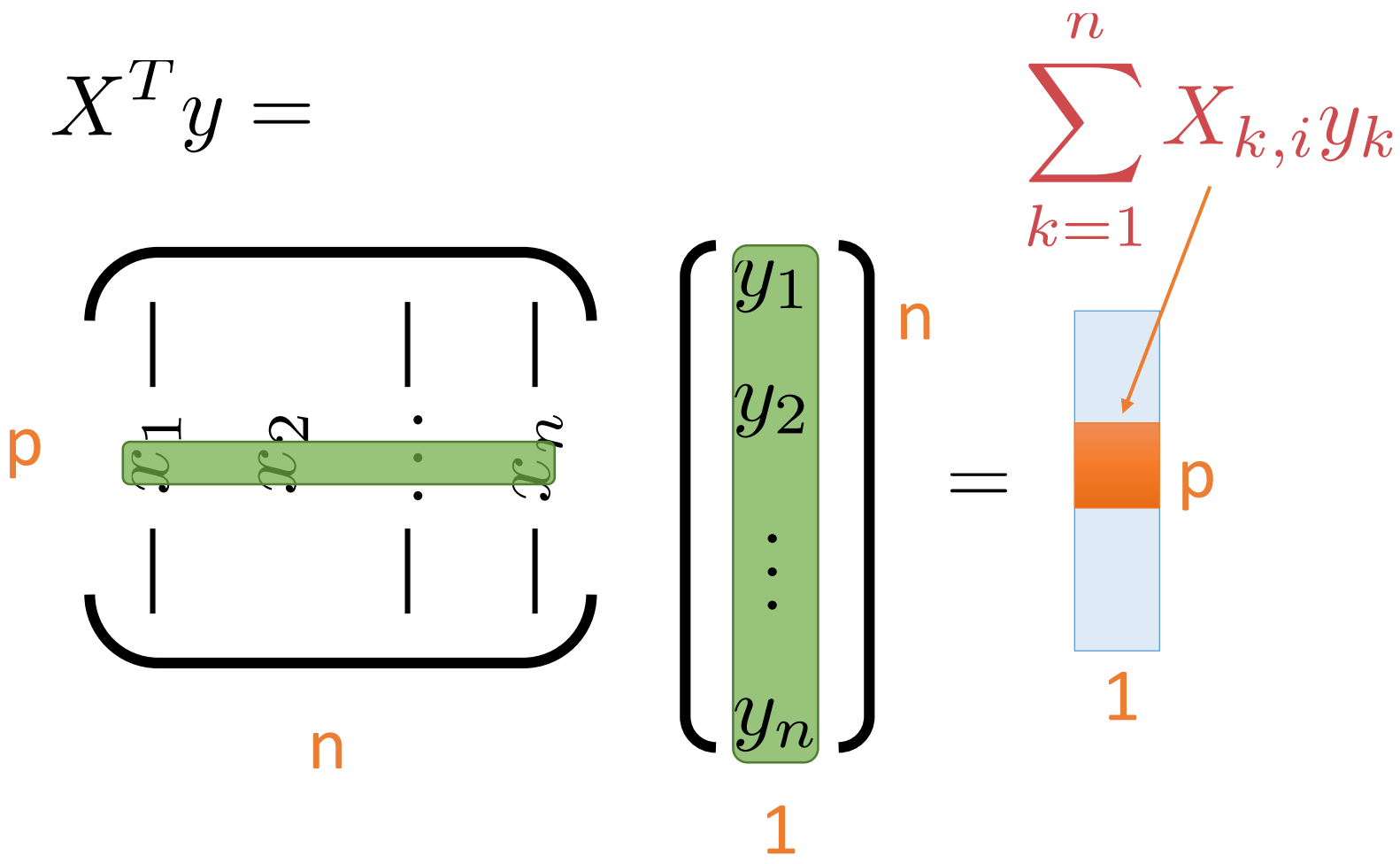
Computing Query 1

$$X^T X =$$



Computing Query 2

$$X^T y =$$

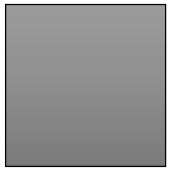


Least Squares Regression using the **Statistical Query Pattern**

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

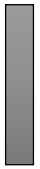
- In database compute sums:

p

p 

$$C = X^T X = \sum_{i=1}^{n} x_i x_i^T \qquad O(np^2)$$

1

p 

$$d = X^T y = \sum_{i=1}^{n} x_i y_i \qquad O(np)$$

- On client compute:

$$\hat{\theta} = C^{-1} d \qquad O(p^3)$$

- Rather than directly solving:

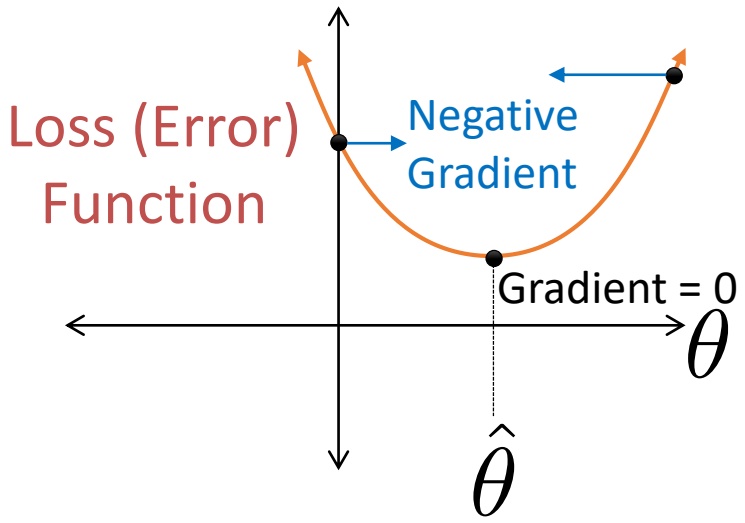
$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \theta^T x_i)$$

- Instead we compute the gradient of the loss:

$$G(\theta; X, y) = \nabla_\theta \frac{1}{n} \sum_{i=1}^{n} L(y_i, \theta^T x_i) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta L(y_i, \theta^T x_i)$$

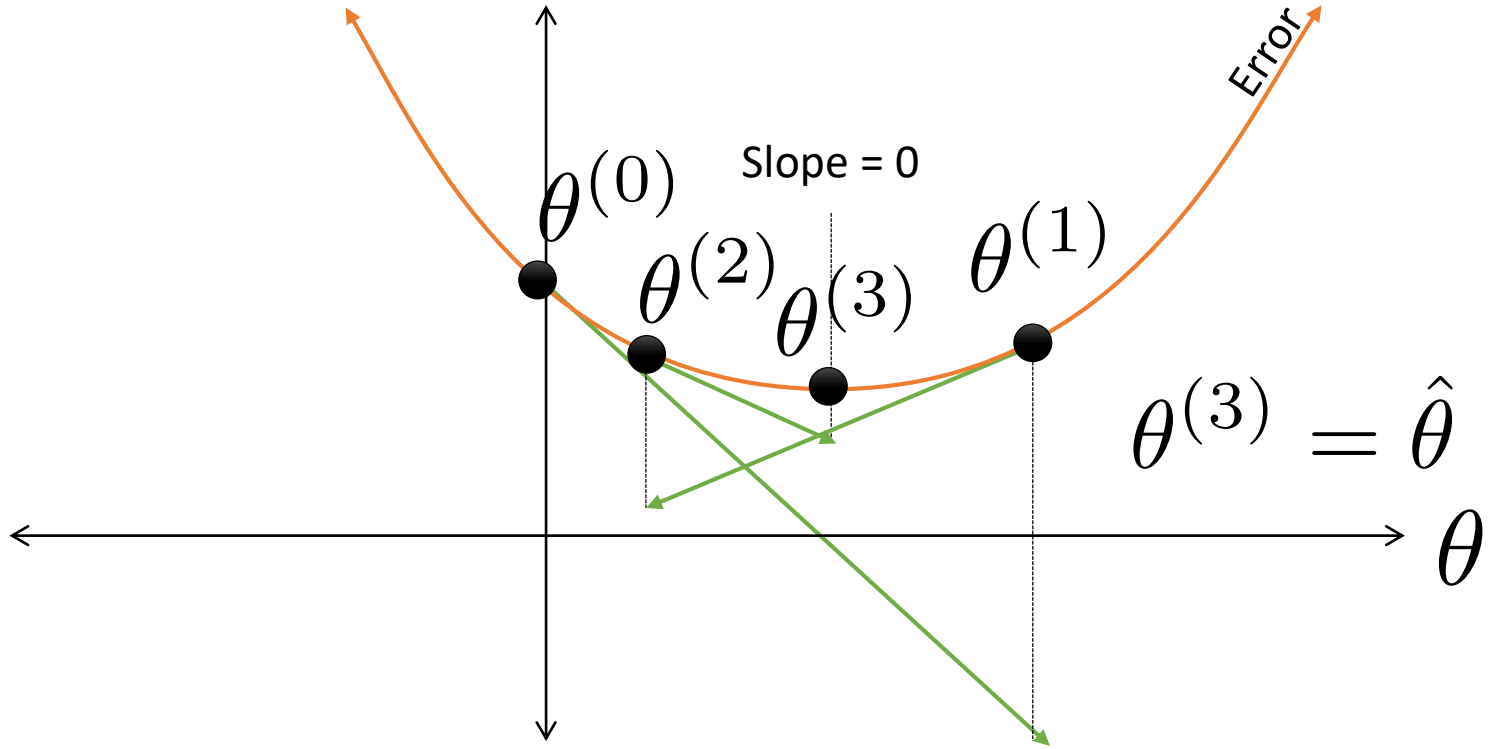
$$= \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T x_i) x_i$$

- **Big Idea:** Negative gradient points in the direction of steepest descent



Gradient Descent Algorithm

```
t ← 0
θ(0) ← Vec(0)
while (not converged):
    θ(t+1) ← θ(t) – Stepsize(t) * G(θ; X,Y)
    t ← t + 1
```



Gradient Descent Algorithm

```
t ← 0
θ(0) ← Vec(0)
while (not converged):
    θ(t+1) ← θ(t) – Stepsize(t) * G(θ; X,y)
    t ← t + 1
```

- Does this fit the statistical query pattern
  - Yes! Only dependence on data is:

Size? 

$$G(\theta; X, y) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T x_i) x_i$$

Complexity? $O(np)$

  - Can we go even faster?
    - **Stochastic Gradient Descent (SGD)**: Approximate the gradient by sampling data (typically several hundred records per query).

Stochastic Gradient Descent

- Update the parameters for each training case in turn, according to its own gradients

```
t ← 0
θ(0) ← Vec(0)
while (not converged):
    θ(t+1) ← θ(t) – Stepsize(t) * G(θ; X,y)
    t ← t + 1
```

$$G(\theta; X, y) = (\boldsymbol{y}_i - \theta^T x_i) x_i$$

$$\text{Stepsize}^{(t)} = \frac{1}{t+1}$$

Complexity?

$O(p)$

Bias-Variance Tradeoff

- So far we have minimized the **training error**: the error on the training data.
  - low training error does not guarantee good expected performance (due to over-fitting)
- We would like to reason about the **test error**

Theorem:

$$\text{Test Error} = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

Noisy data has inherent error (measurement error)

Error due to models **inability to fit** the data. **(Under Fitting)**

Error due to inability to estimate model parameters. **(Over-fitting)**

Bias Variance Plot

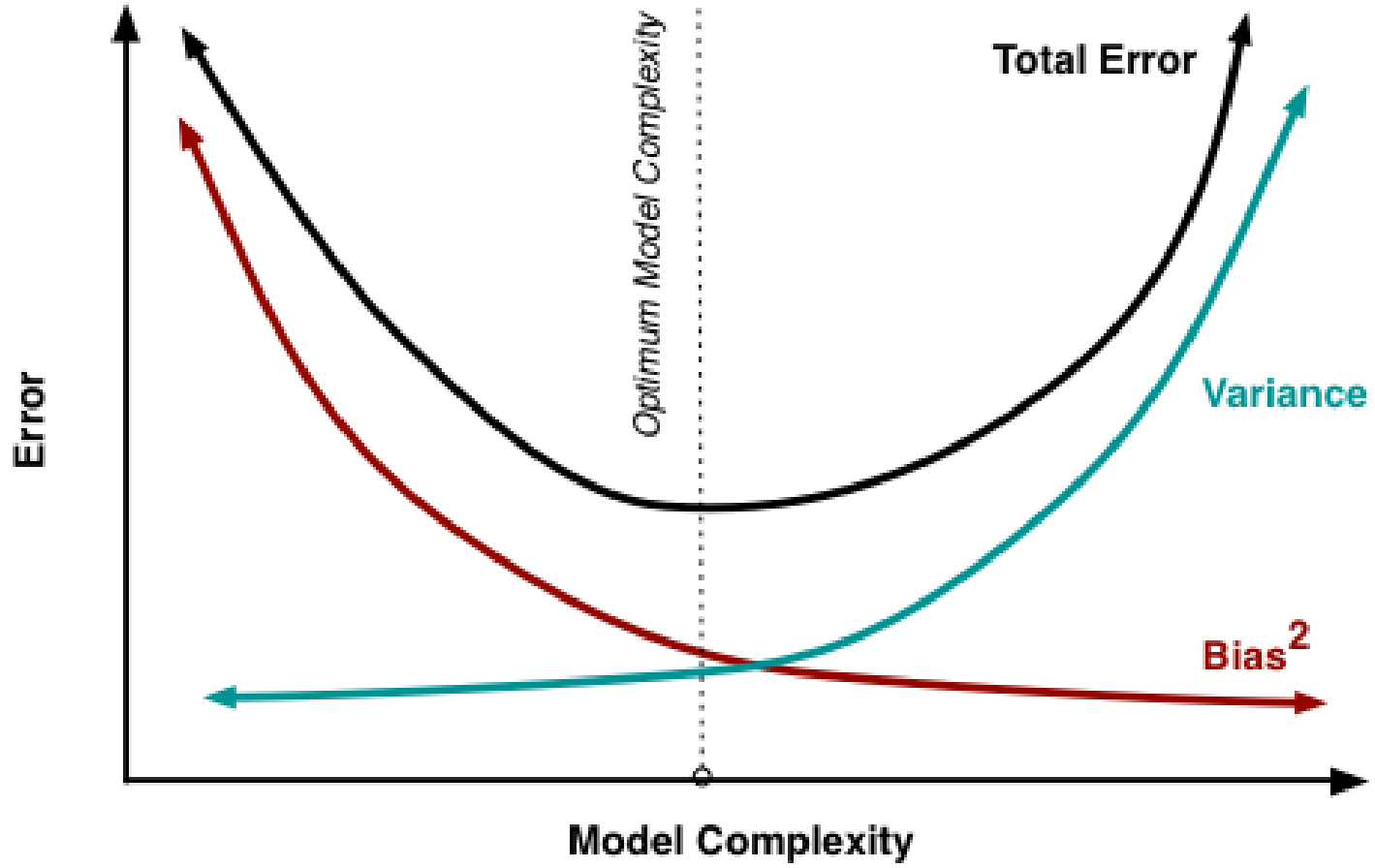


Image from http://scott.fortmann-roe.com/docs/BiasVariance.html

Regularization to Reduce Over-fitting

- We can add a regularization term:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 + \lambda R(\theta)$$

Squared Loss

Regularization Function

Regularization Parameter

- Common of Regularization Functions:

Ridge (L2-Reg) Regression $R_{\text{Ridge}}(\theta) = \sum_{i=1}^{d} \theta_i^2$

Lasso (L1-Reg) $R_{\text{Lasso}}(\theta) = \sum_{i=1}^{d} |\theta_i|$

  - Encourage small parameter values
- The parameter λ determines amount of reg.
  - Larger → more reg. → lower variance → higher bias

Regularization to Reduce Over-fitting

- We can add a regularization term:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 + \lambda R(\theta)$$

Squared Loss

Regularization Function

Regularization Parameter

- Solving the regularized problem:
  - Closed form solution for Ridge regression (L2):

$$\hat{\theta}_{\text{Ridge}} = (X^T X + \lambda I)^{-1} X^T Y$$

  - Iterative methods for Lasso (L1):
    - Stochastic gradient …
- How do we choose λ?

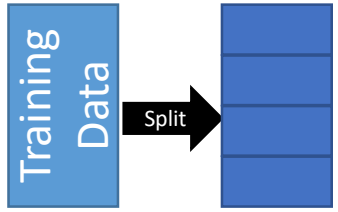
Picking The Regularization Parameter λ

- **Proposal:** Minimize **training** error

$$\arg \min_{\theta \in \mathbb{R}^p, \lambda \geq 0} \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 + \lambda R(\theta)$$

  - Trivial solution ➔ λ = 0
- Intuition we want to minimize **test** error
  - **Test error**: error on unseen data
- **2nd Proposal:** Split training data into training and evaluation sets
  - For a range of λ values compute optimal θ_λ using only the reduced training set
  - Evaluate θ_λ on the separate evaluation set and select the λ with the lowest error

K-Fold Cross Validation



- Split training data into K-equally sized parts
  - In practice K is relatively small (e.g., 5)
- For each part train on the other k-1 parts and compute the error on that part:

K = 4

Error is averaged over all 4 steps

- Compute the average test error over held out parts
- Select reg. param. that minimizes average test error

Logistic Regression

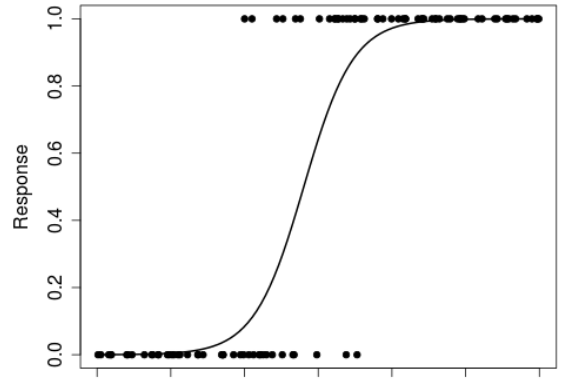
- Basic Model:

$$\mathbf{P}\left(y|x,\theta\right) = \sigma\left(y(\theta^T x)\right) = \frac{1}{1+\exp\left(-y(\theta^T x)\right)}$$

Note that y is either +1 or -1

- Logistic Function:

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$



- Symmetric

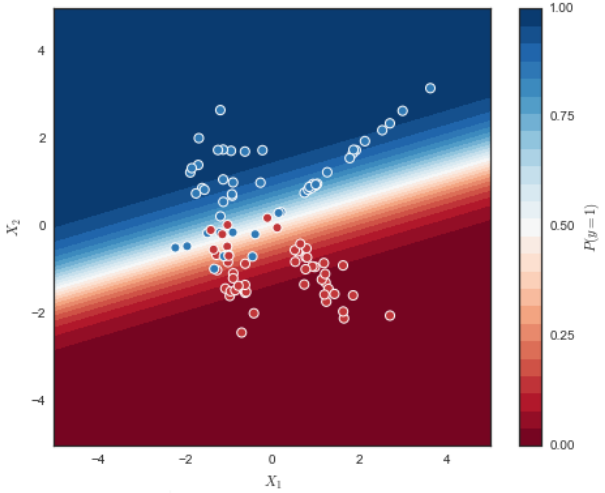
$$1-\sigma(a) = \frac{\exp(-a)}{1+\exp(-a)} = \frac{1}{\exp(a)+1} = \sigma(-a)$$

- Gradient

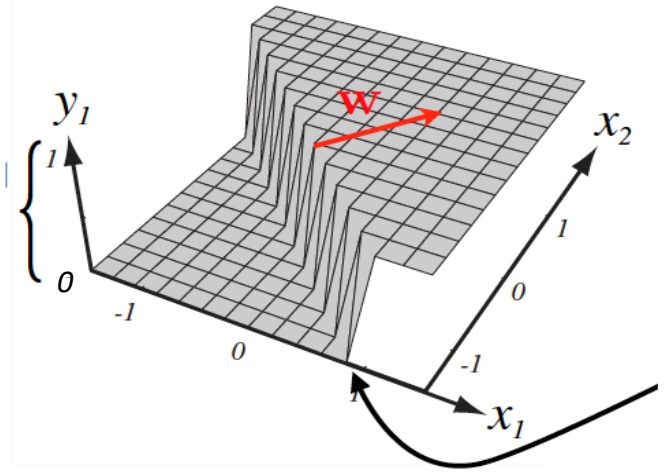
$$\sigma'(a) = \frac{\exp(-a)}{(1+\exp(-a))^2} = \sigma(a)(1-\sigma(a))$$

Logistic Regression

- 2D example:



X		Y
x_1	x_2	
3.9	3.1	+1
0.2	0.1	-1
4.2	0.9	-1
1.1	2.9	+1



Class decision boundary

Learning the Logistic Regression Model

- How do we fit the Logistic Regression model?
  - method of maximum likelihood
- Select the best θ by maximizing prob. of data
  - Solve the following **convex** optimization problem

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \quad \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i(\theta^T x_i)\right)\right) + \lambda R(\theta)$$

  - Regularized using same techniques as regression

- Optimized using numerical methods
  - **SGD**: Stochastic Gradient Descent

Nearest Neighbors

- Training example in Euclidean space: $\mathbf{x} \in \Re^d$
- Idea: The value of the target function for a new query is estimated from the known value(s) of the nearest training example(s)
- Distance typically defined to be Euclidean:

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

Algorithm:

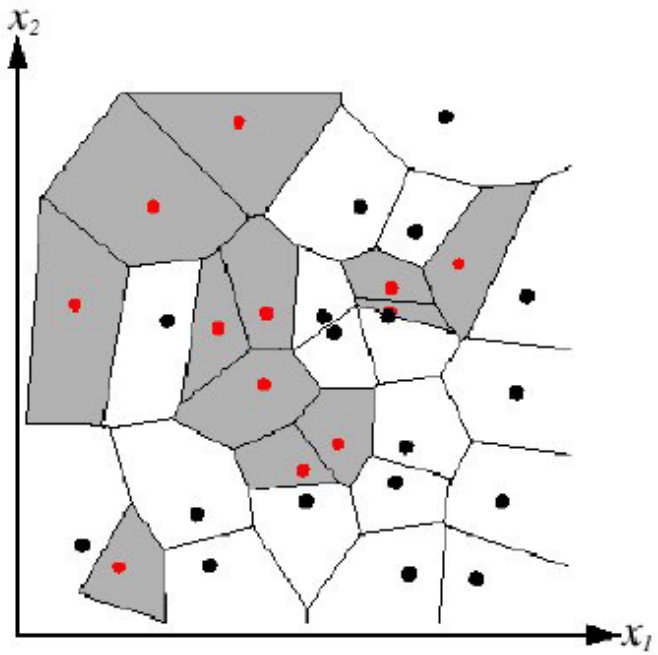
1. Find example $(\mathbf{x}^*, t^*)$ (from the stored training set) closest to the test instance $\mathbf{x}$. That is:

$$\mathbf{x}^* = \underset{\mathbf{x}^{(i)} \in \text{train. set}}{\text{argmin}} \text{ distance}(\mathbf{x}^{(i)}, \mathbf{x})$$

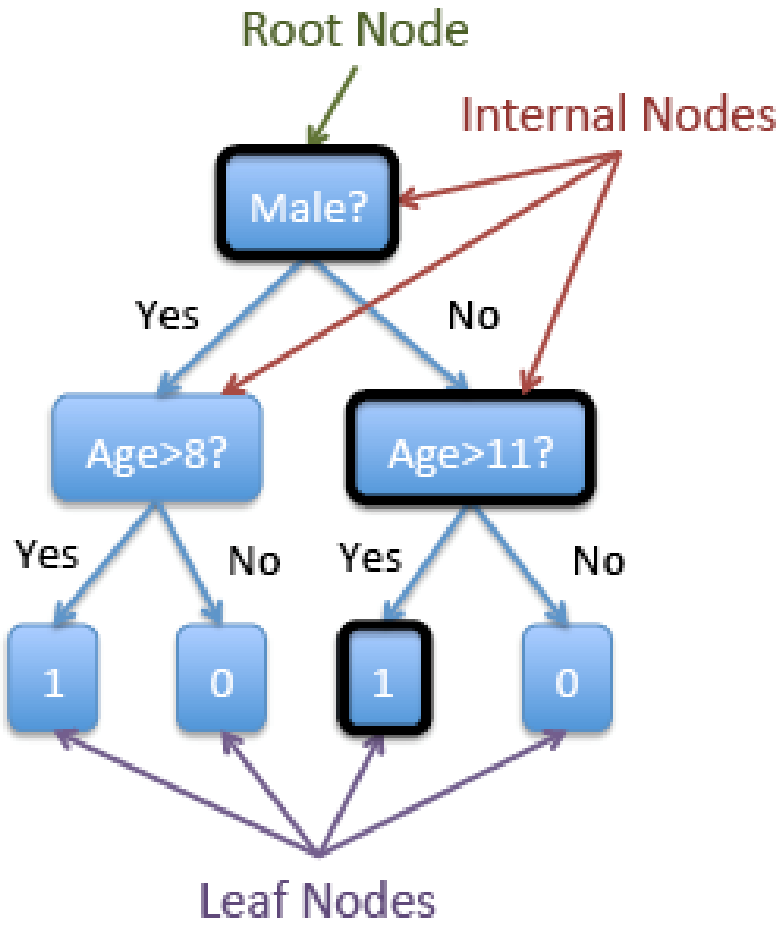
2. Output $y = t^*$

Nearest Neighbors

- Nearest neighbor algorithm does not explicitly compute decision boundaries, but these can be inferred
- Decision boundaries: Voronoi diagram visualization
  - show how input space divided into classes
  - each line segment is equidistant between two points of opposite classes



Decision tree



Input:  **Alice**
Gender: Female
Age: 14

Prediction: Height > 55”

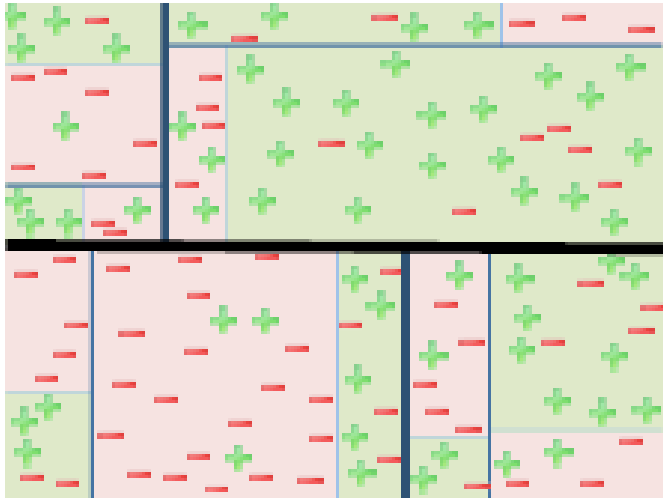
Every **internal node** has a **binary** query function q(x).

Every **leaf node** has a prediction, e.g., 0 or 1.

Prediction starts at **root node**.
Recursively calls query function.
Positive response → Left Child.
Negative response → Right Child.
Repeat until Leaf Node.

Decision tree function class

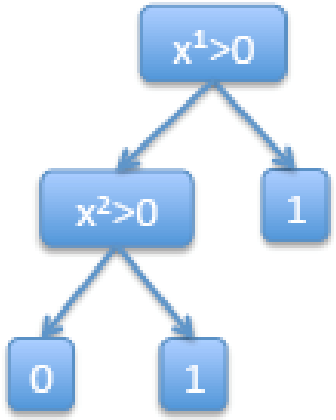
- "Piece-wise Static" Function Class
  - All possible partitionings over feature space.
  - Each partition has a static prediction.
- Partitions axis-aligned
  - E.g., No Diagonals
- (Extensions next week)



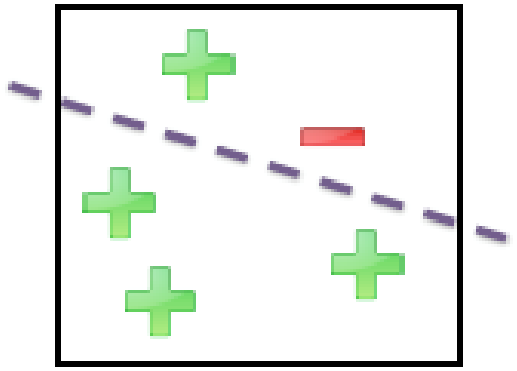
Decision tree vs Linear model

- Decision Trees are NON-LINEAR Models!

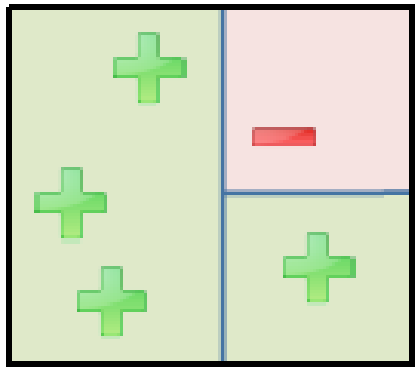
- Example:



No Linear Model
Can Achieve 0 Error



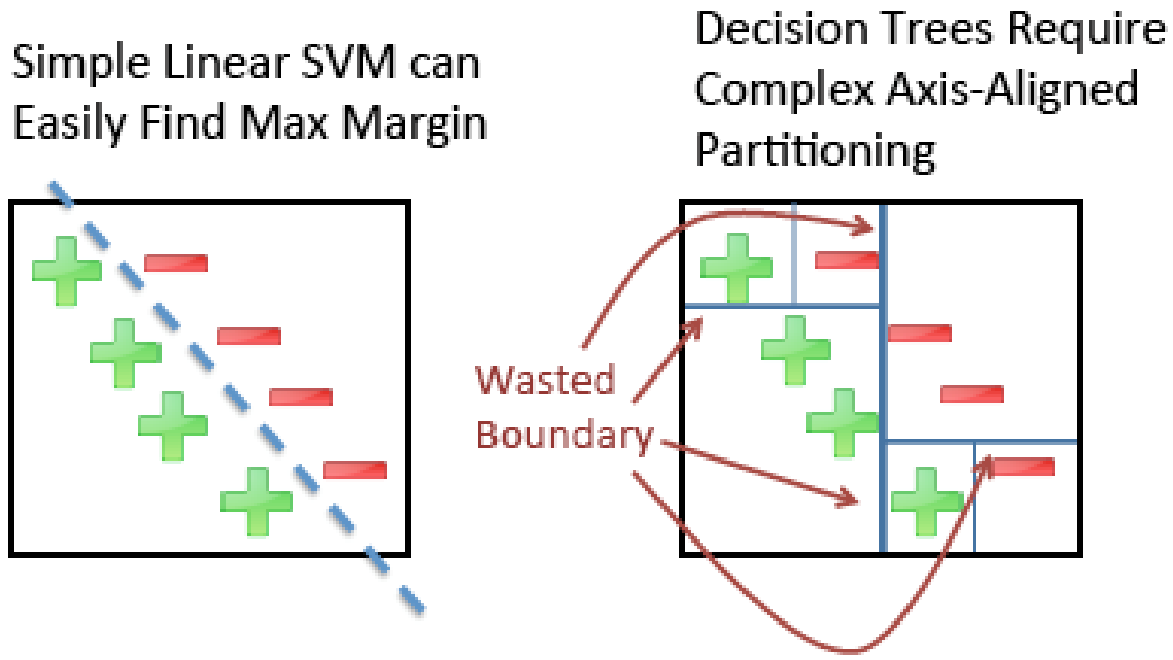
Simple Decision Tree
Can Achieve 0 Error



Decision tree vs Linear model

- Decision Trees are AXIS-ALIGNED!
  - Cannot easily model diagonal boundaries

- Example:



Decision tree: inference

- Classification and Regression

  - Each path from root to a leaf defines a region R_m of input space
  - Let $\{(x^{(m_1)}, t^{(m_1)}), \ldots, (x^{(m_k)}, t^{(m_k)})\}$ be the training examples that fall into R_m
  - **Classification tree**:
    - discrete output
    - leaf value y^m typically set to the most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$
  - **Regression tree**:
    - continuous output
    - leaf value y^m typically set to the mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

Note: We will only talk about classification

Decision tree: Impurity = Loss

- **Training Goal:**
  - Find decision tree with low impurity.

- **Impurity Over Leaf Nodes = Training Loss**

$$L(S) = \sum_{S'} L(S')$$

S' iterates over leaf nodes
Union of S' = S
(Leaf Nodes = partitioning of S)

$$L(S') = \min_{\hat{y} \in \{0,1\}} \sum_{(x,y) \in S'} 1_{[\hat{y} \neq y]}$$

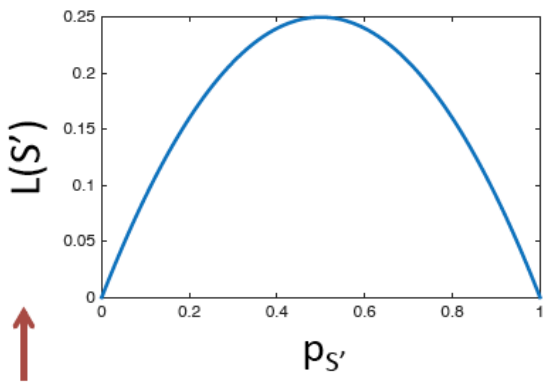
Classification Error on S'

Decision tree: Surrogate impurity

- Want more continuous impurity measure
- First try: Bernoulli Variance:

$$L(S') = |S'| p_{S'}(1 - p_{S'}) = \frac{\#pos * \#neg}{|S'|}$$

$p_{S'}$ = fraction of S' that are positive examples



Assuming |S'|=1

Worst Purity

P = 1/2, L(S') = |S'|*1/4

P = 1, L(S') = |S'|*0

P = 0, L(S') = |S'|*0

Perfect Purity

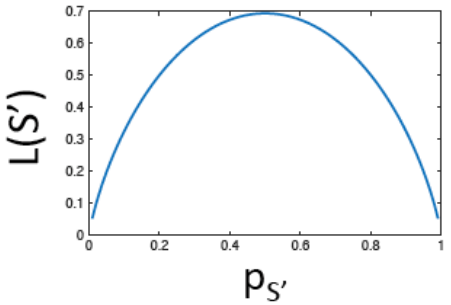
Decision tree: Surrogate impurity

Define: 0*log(0) = 0

- Entropy: $L(S') = -|S'|\left(p_{S'}\log p_{S'} + (1-p_{S'})\log(1-p_{S'})\right)$
  - **aka: Information Gain:**

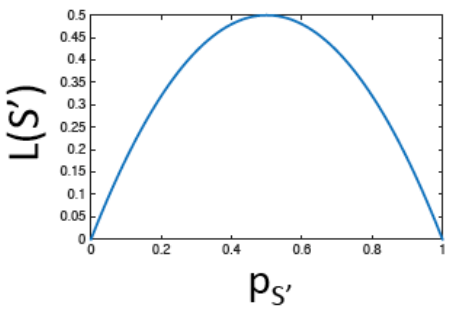
$$IG(A,B \mid S') = L(S') - L(A) - L(B)$$

  - (aka: Entropy Impurity Reduction)
  - Most popular.



- Gini Index:

$$L(S') = |S'|\left(1 - p_{S'}^2 - (1-p_{S'})^2\right)$$



Decision tree: Learning

- How do we construct a useful decision tree?

  - Define impurity measure L(S')
    - E.g., L(S') = Bernoulli Variance

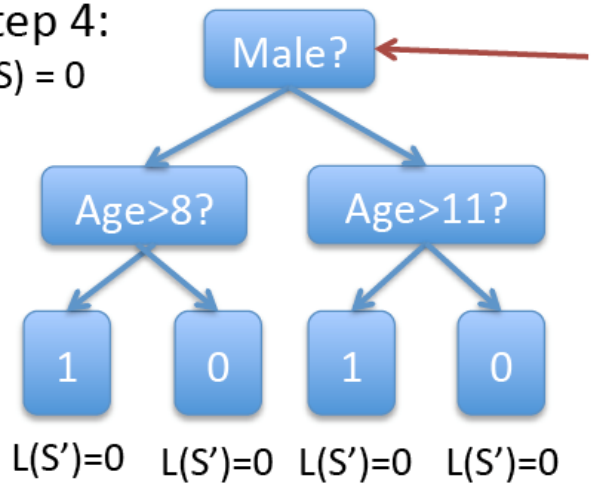
Loop: Choose split with greatest impurity reduction (over all leaf nodes).
Repeat: until stopping condition.

Step 1:
L(S) = 12/7

Step 2:
L(S) = 5/3

Step 3:
L(S) = 2/3

Step 4:
L(S) = 0



S

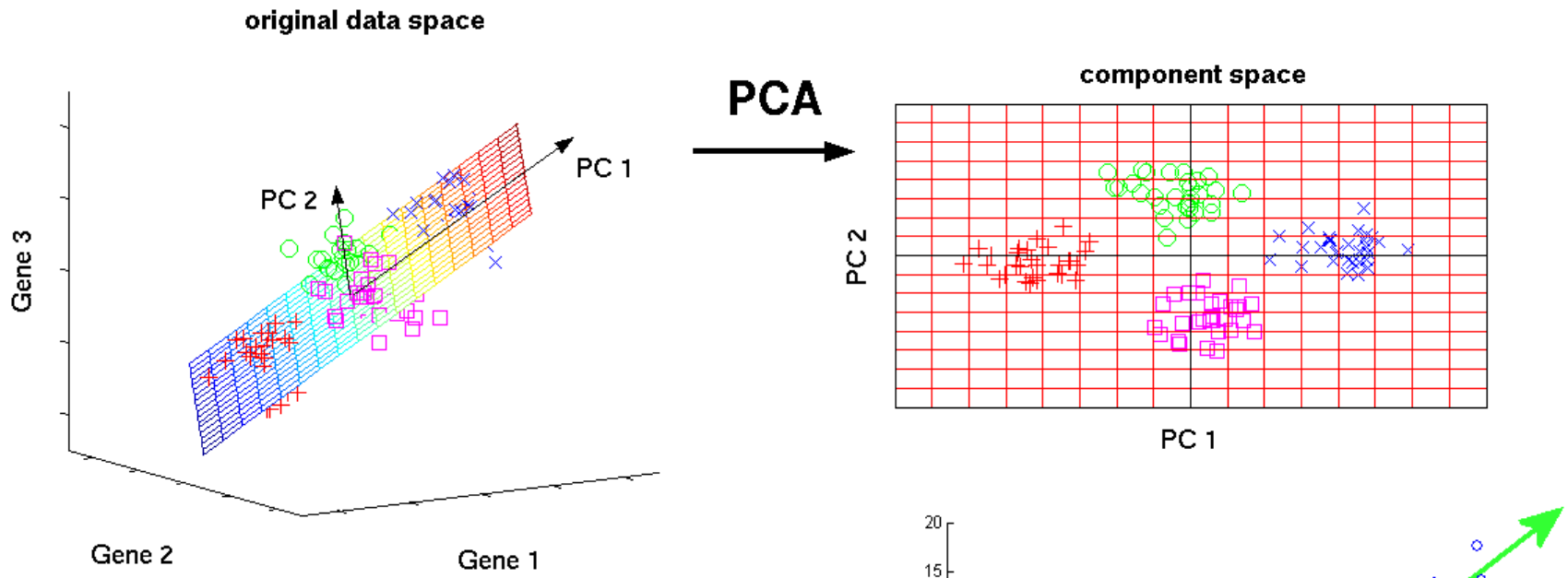
Name	Age	Male?	Height > 55"
Alice	14	0	1
Bob	10	1	1
Carol	13	0	1
Dave	8	1	0
Erin	11	0	0
Frank	9	1	1
Gena	10	0	0

x y

Decision tree: Learning

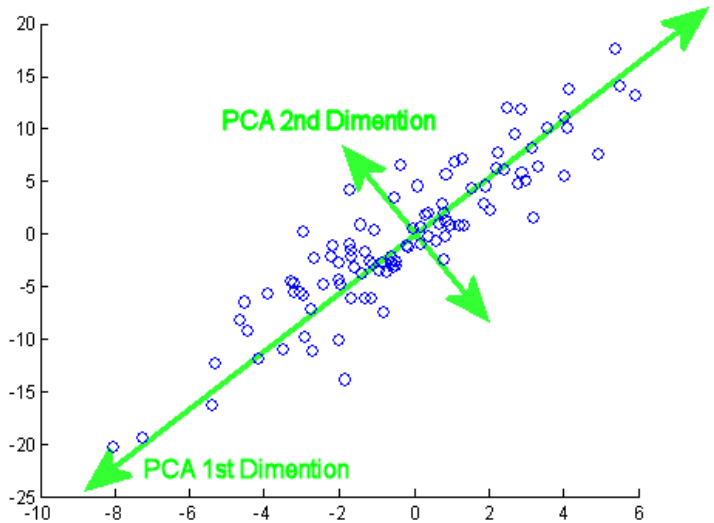
- When to stop splitting?
  - **Minimum Size:** do not split if resulting children are smaller than a minimum size.
    - **Most common stopping condition.**
  - **Maximum Depth:** do not split if the resulting children are beyond some maximum depth of tree.
  - **Maximum #Nodes:** do not split if tree already has maximum number of allowable nodes.
  - **Minimum Reduction in Impurity:** do not split if resulting children do not reduce impurity by at least $\delta\%$.

Principal Component Analysis



Big Ideas

- Identify dimensions of maximum variance
- Project data onto those dimensions



Finding Principal Components

- To find the principal component directions, we center the data (subtract the sample mean from each variable)
- Calculate the empirical covariance matrix:

$$C = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}^{(n)} - \bar{\mathbf{x}})(\mathbf{x}^{(n)} - \bar{\mathbf{x}})^T$$

  with $\bar{\mathbf{x}}$ the mean
- What's the dimensionality of C?
- Find the M eigenvectors with largest eigenvalues of C: these are the principal components
- Assemble these eigenvectors into a $D \times M$ matrix U
- We can now express D-dimensional vectors $\mathbf{x}$ by projecting them to M-dimensional $\mathbf{z}$

$$\mathbf{z} = U^T \mathbf{x}$$

Scaling Principal Component Analysis

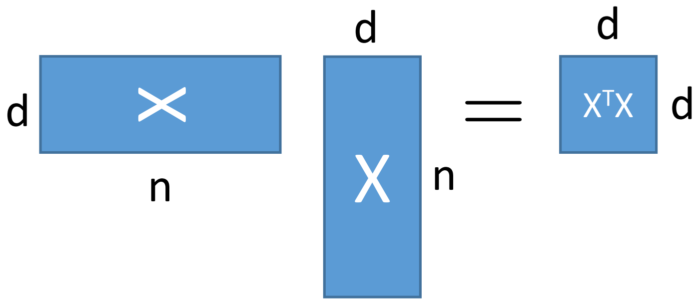
- **PCA Algorithm**
  - Computes eigenvectors of of covariance matrix

$$\mathbf{Cov}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T = \frac{1}{n}X^T X - \bar{x}\bar{x}^T$$

  - The covariance matrix *d×d* is generally smaller than X (*n×d)*

- We therefore only need to compute:

$$X^T X =$$

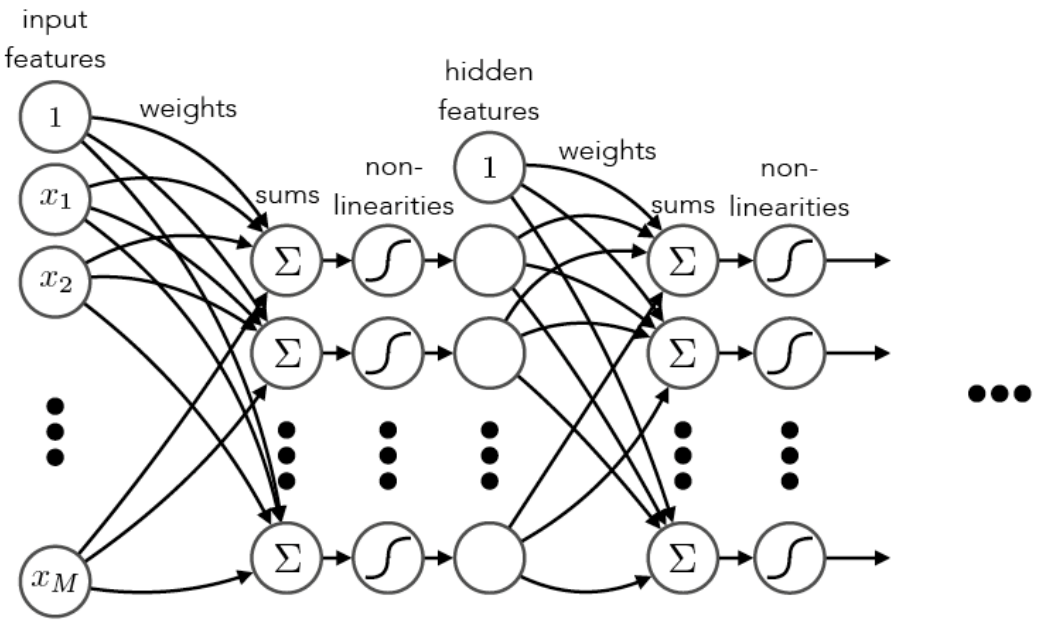


$$= \sum_{i=1}^{n} x_i x_i^T$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i =$$

- In summation form
- Only one pass required!

Multilayer networks



network: *sequence of parallelized weighted sums and non-linearities*

DEFINE $\mathbf{x}^{(0)} \equiv \mathbf{x}$, $\mathbf{x}^{(1)} \equiv \mathbf{h}$, ETC.

1st layer

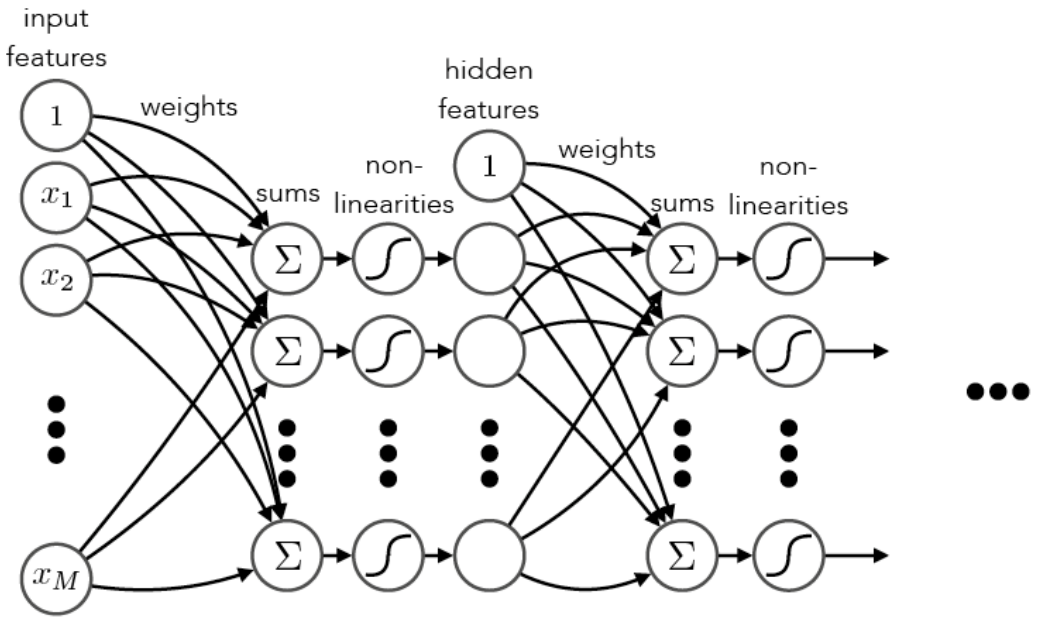
$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)\intercal}\mathbf{x}^{(0)}$$
$$\mathbf{x}^{(1)} = \sigma(\mathbf{s}^{(1)})$$

2nd layer

$$\mathbf{s}^{(2)} = \mathbf{W}^{(2)\intercal}\mathbf{x}^{(1)}$$
$$\mathbf{x}^{(2)} = \sigma(\mathbf{s}^{(2)})$$

•••

Multilayer networks



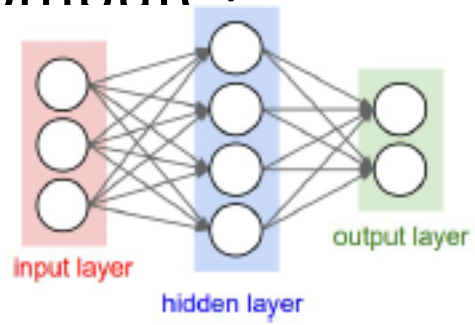
network: *sequence of parallelized weighted sums and non-linearities*

$$= \sigma(\ \cdots \ \sigma(\quad \sigma(\quad) \ ) \cdots)$$

output 2nd weights 1st weights input

Forward pass

- What does the network compute?
  - Foll



- Output of the network can be written as:

$$h_j(\mathbf{x}) = f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji})$$

$$o_k(\mathbf{x}) = g(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj})$$

(j indexing hidden units, k indexing the output units, D number of inputs)

Backward pass

- Trainin
  - Find weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

    where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network

  - Define a loss function, eg:
    - Squared loss: $\sum_k \frac{1}{2}(o_k^{(n)} - t_k^{(n)})^2$
    - Cross-entropy loss: $-\sum_k t_k^{(n)} \log o_k^{(n)}$
  - Gradient descent:

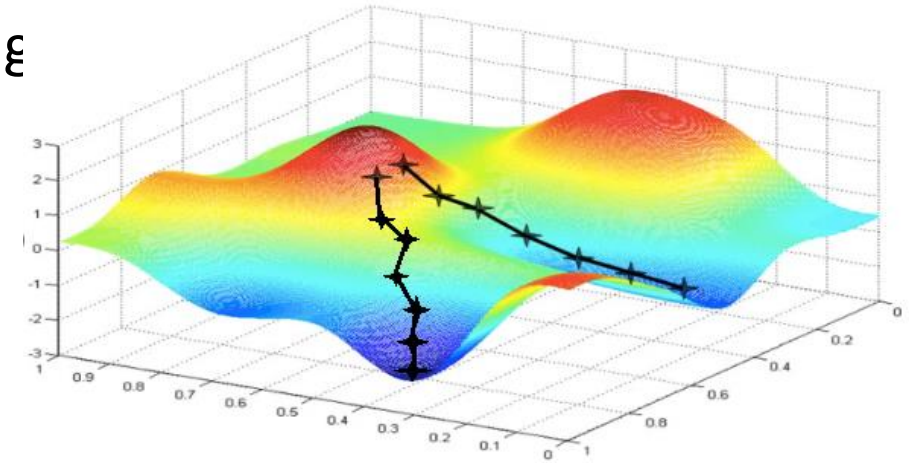
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial E}{\partial \mathbf{w}^t}$$

    where η is the learning rate (and E is error/loss)

Learning as optimization

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

- Update weights by g



- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Backward pass

- Backpropagation
  - An efficient method for computing gradients in NNs
  - A neural network as a function of composed operations

$$f_L(\mathbf{w}_L, f_{L-1}(\mathbf{w}_{L-1}, \dots f_1(\mathbf{w}_1, \mathbf{x}) \dots))$$

and the loss $\mathcal{L}$ is a function of the network output

→ use chain rule to calculate gradients

chain rule example

$$y = w_2 e^{w_1 x}$$

input x output y parameters w_1, w_2

evaluate parameter derivatives: $\frac{\partial y}{\partial w_1}, \frac{\partial y}{\partial w_2}$

define

$$v \equiv e^{w_1 x} \longrightarrow y = w_2 v$$

$$u \equiv w_1 x \longrightarrow v = e^u$$

then $\frac{\partial y}{\partial w_2} = v = e^{w_1 x}$

chain rule

$$\frac{\partial y}{\partial w_1} = \boxed{\frac{\partial y}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial w_1}} = w_2 \cdot e^{w_1 x} \cdot x$$

Association Rules

- **<u>Goal:</u>** Identify **predictive rules** that relate a subset of items to the *likely* presence of other items in the basket
- **Association Rule:** *{LHS}* ➔ *{RHS}*
  - Example: {pasta} ➔ {sauce}
- **Support of an Association Rule**
  - Support of the set LHS ∪ RHS
- **Confidence of an Association Rule**
  - Support (LHS ∪ RHS) / Support (LHS)

- **Probabilistic Interpretation:** *Conditional Probability*

$$\frac{\mathbf{P}\left(\mathrm{LHS} \in \mathrm{Basket}, \mathrm{RHS} \in \mathrm{Basket}\right)}{\mathbf{P}\left(\mathrm{LHS} \in \mathrm{Basket}\right)} = \mathbf{P}\left(\mathrm{RHS} \in \mathrm{Basket} \,|\, \mathrm{LHS} \in \mathrm{Basket}\right)$$

Interesting association rules

- **Not all high-confidence rules are interesting**
  - The rule $X \rightarrow \textit{milk}$ may have high confidence for many itemsets X, because milk is just purchased very often (independent of X) and the confidence will be high
- **Interest** of an association rule $I \rightarrow j$:
  abs. difference between its confidence and the fraction of baskets that contain j

$$\text{Interest}(I \rightarrow j) = |\,\text{conf}(I \rightarrow j) - \Pr[j]\,|$$

  - Interesting rules are those with high positive or negative interest values (usually above 0.5)

Association Rule Mining Algorithm

- Input Parameters:
  - **minsup** – the minimum support of discovered rules
  - **minconf** – the minimum confidence or discovered rules
- Algorithm:
  - Compute frequent itemsets with **minsup**
  - For each frequent itemset consider all possible subsets as LHS and compute confidence
    - Recall confidence = Sup(LHS $\cup$ RHS) / Sup(LHS)
  - Return all rules with confidence > **minconf**
- Resulting association rules can be used for:
  - marketing decisions: beer and diapers near each other
  - Triggers in an online targeting systems:
    - Given items in a basket what are most likely items to be added
- **Important**: *Rule does not imply causality*