Week 8 – Lecture 1
Web-Scale Link Analysis
- Link Spam

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- Midterm
  - We will go over the questions
  - Simple correction will be done in the class if needed
  - Please return your answer sheet
  - You can browse during the GTAs' office hours
  - Your final exam is not a comprehensive test
- Extra help hours will be available this week
- PA3 will be posted: Team assignment (work with your term project team)

1

1

Topics discussed in the previous class

- Web-Scale Link Analysis – Dead end and Spider Trap
- Overview of Midterm

Topics to be covered

- Topic 1: Link Spam
- Midterm review (Last 30 minutes)

Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

Link spam

How can I obtain the first search result place?

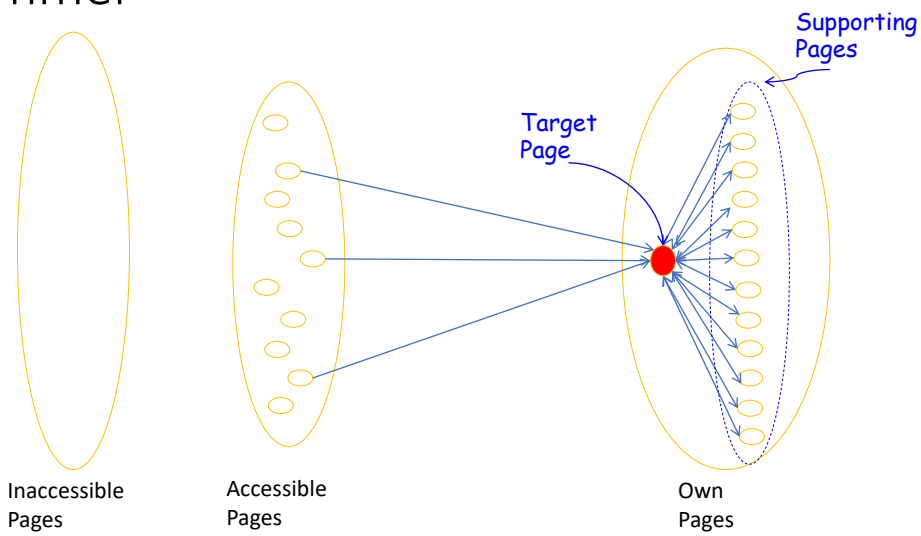
- The holiday season
- "Samsonite carry on luggage"
  - J.C. Penny was first on the list
    - Ahead of Samsonite.com
    - For months
- "black hat" optimization?
  - It is NOT illegal!
- "casual dresses", "evening dresses", "little black dress", or "cocktail dress"
  - 2,105 pages
  - Most of them were hosted by unrelated sites
  - Bounces back to J.C. Penny

https://www.nytimes.com/2011/02/13/business/13search.html

Architecture of a Spam Farm

- ***Spam Farm***
  - A collection of pages whose purpose is to increase the PageRank of a certain page or pages
- From the point of view of the **spammer**, the Web is divided into two parts
  - **Inaccessible pages**
    - The pages that the spammer cannot affect
    - Most of the Web
  - **Accessible pages**
    - Those pages that, while they are not controlled by the spammer, can be affected by the spammer

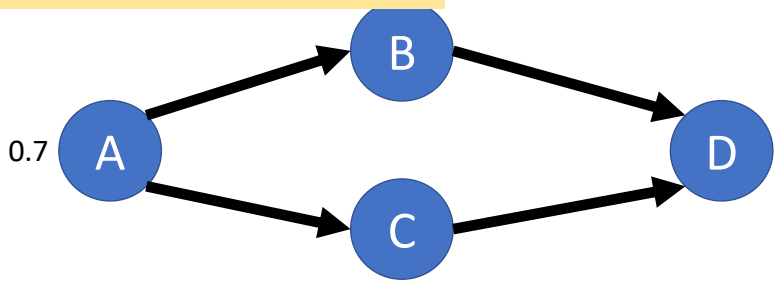
The Web from the point of view of the link spammer



Calculating the PageRank value

What is the Pagerank value of B after the current iteration?

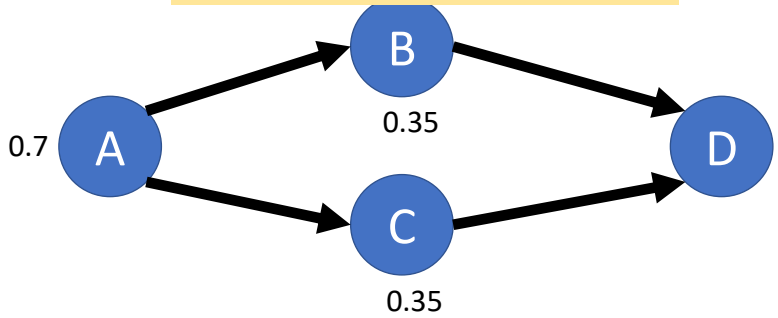
a. 0.7
b. 1
c. 0.35
d. 1.4



Calculating the PageRank value

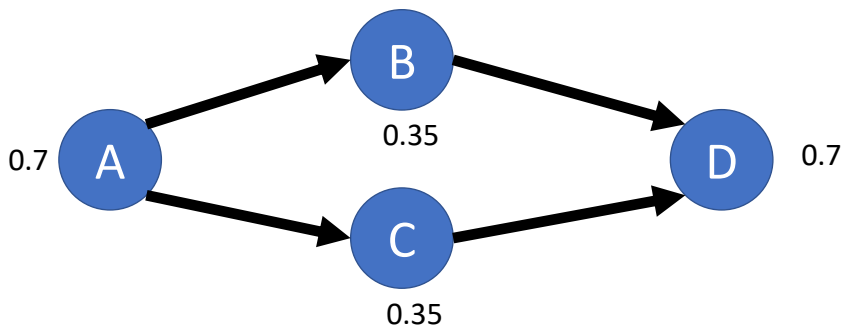
What is the pagerank value of D after the current iteration?

a. 0.7
b. 1
c. 0.35
d. 1.4



Understanding Spam Farm (1/2)

- Setting the links to the target page
  - Without link from outside, the spam farm is not useful
  - e.g. Blogs or news papers
    - Comments like "I agree. Please see my article at www.mySpamFarm.com"



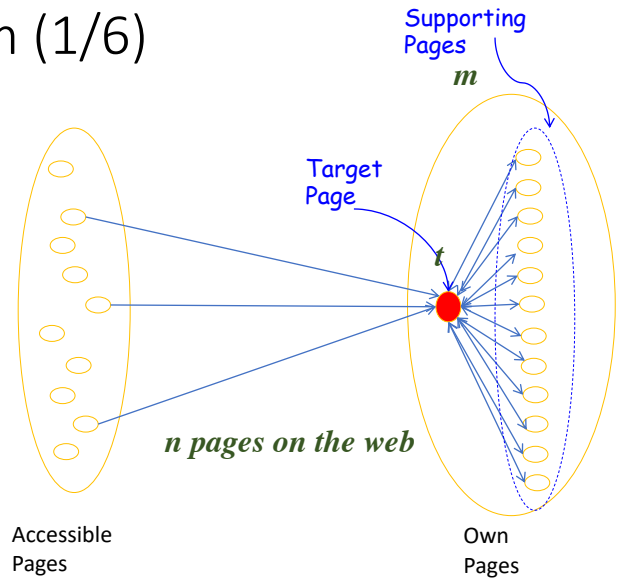
Understanding Spam Farm (2/2)

- There is **one page t**, the **target page**
  - Spammer attempts to place **as much PageRank as possible**

- There are a large number of **m supporting pages**
  - **Accumulate the portion of the PageRank** that is distributed equally to all pages
  - The fraction $1\text{-}\beta$ of the PageRank that represents surfers going to a random page
  - **Prevent the PageRank of t from being lost**
    - Note that all of the supporting pages links only to t

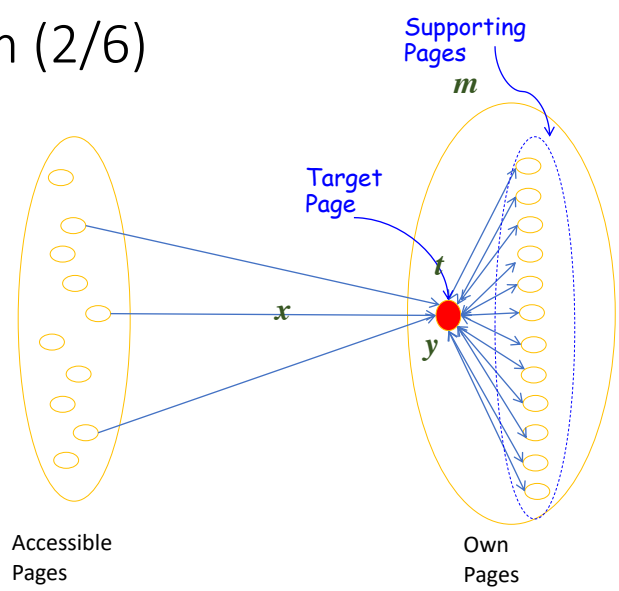
Analysis of a Spam Farm (1/6)

- A taxation parameter β
  - The fraction of a page's PageRank that gets distributed to its successors at the next round
- Let there be,
  - ***n* pages on the Web** in total
  - A target page ***t***
  - ***m* supporting pages**



Analysis of a Spam Farm (2/6)

- Let ***x*** be **the total amount of PageRank contributed by the <u>accessible pages</u>**
  - x is the sum over all accessible page p with a link to t, of the PageRank of p times β divided by the number of successors of p
- Finally, let ***y*** be the unknown PageRank of ***t***



Analysis of a Spam Farm (3/6)

- The PageRank of each supporting page
  - $\beta y/m+(1-\beta)/n$
- First term represents the **contribution from t**
  - βy is distributed to t's successors
- Second term is the supporting page's share of **the fraction $1-\beta$ of the PageRank that is divided equally among all pages on the Web**

Analysis of a Spam Farm (4/6)

- PageRank value of the target page t
  - y= β(*sum of the contributions from neighboring vertices*)+$(1-\beta)e/n$
- ***sum of the contributions from neighboring vertices (1+2)***

1. Contribution x from outside
   - x is the sum over all accessible page p with a link to t, of the PageRank of p times β divided by the number of successors of p
   - Assume that β is already multiplied
2. Contribution from the supporting pages
   - $m(\beta y/m+(1-\beta)e/n)$

- $(1-\beta)e/n$
  - The share of the fraction $1-\beta$ of the PageRank that belongs to t

  This amount is negligible

Analysis of a Spam Farm (5/6)

- From (1) and (2),

$$y = x + \beta m\left(\frac{\beta y}{m} + \frac{1-\beta}{n}\right) = x + \beta^2 y + \beta(1-\beta)\frac{m}{n}$$

$$y = \frac{x}{1-\beta^2} + c\frac{m}{n}$$

$$Where$$

$$c = \beta(1-\beta)/(1-\beta^2) = \beta/(1+\beta)$$

Analysis of a Spam Farm (6/6)

- If we choose $\beta=0.85$, then $1/(1-\beta^2)=3.6$
- $c= \beta/(1+\beta)=0.46$
- The structure has amplified the external PageRank contribution by 360%
- Also, it obtained an amount of PageRank that is 46% of the fraction of the Web, m/n, that is in the spam farm

Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

Combatting Link Spam

Combatting Link Spam

- **Detecting and eliminating link spam** have been critical for search engines
  - Just as it was critical to eliminate term spam in the previous decade
- Detecting particular structures
  - Spam farm
    - **One page links to a very large number of pages**
    - **Each of which links back to it**

Combatting Link Spam

- Modifying PageRank to lower the rank of link-spam pages automatically
  - **TrustRank**
  - **Measuring Spam mass**

TrustRank

- TrustRank is a **topic-sensitive PageRank**
  - **"topic" is a set of pages believed to be trustworthy (not spam)**
- Develop a suitable **teleport set of trustworthy pages**
  - Let humans examine a set of pages and decide which of them are trustworthy
- **Pick a domain** whose membership is controlled
  - University pages
  - .mil, or .gov

Calculating TrustRank (1/2)

- Then the topic-sensitive **PageRank for S** is the limit of the iteration,

$$v' = \beta M v + (1-\beta) e_S / |S|$$

- M is the transition matrix of the Web, and $|S|$ is the size of set S

Calculating TrustRank (2/2)

- Suppose we use $\beta=0.8$, and our trust rank is represented by the teleport set (trustworthy pages) ***S={B,D}***

$$\beta M = \begin{bmatrix} 0 & 2/5 & 5/4 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix}$$

$$v' = \begin{bmatrix} 0 & 2/5 & 5/4 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 1/10 \\ 0 \\ 1/10 \end{bmatrix}$$

$$\begin{bmatrix} 0/2 \\ 1/2 \\ 0/2 \\ 1/2 \end{bmatrix} \begin{bmatrix} 2/10 \\ 3/10 \\ 2/10 \\ 3/10 \end{bmatrix} \begin{bmatrix} 42/150 \\ 41/150 \\ 26/150 \\ 41/150 \end{bmatrix} \begin{bmatrix} 62/250 \\ 71/250 \\ 46/250 \\ 71/250 \end{bmatrix} \ldots \begin{bmatrix} 54/210 \\ \mathbf{59/210} \\ 38/210 \\ \mathbf{59/210} \end{bmatrix}$$

B and D get a higher PageRank than before

Spam Mass

- Measures the **fraction of its PageRank that comes from spam(non-trusted pages)** for each page
- For an arbitrary page p,
  - PageRank r
    - Computing ordinary PageRank
  - TrustRank t
    - Computing the TrustRank based on some teleport set of trustworthy pages
  - The spam mass
    - $(r - t)/r$

- A **negative or small positive spam mass**
  - p is probably not a spam page
- **Page with high spam mass score**
  - Should be eliminated

Example

- Suppose that both the **PageRank** and **TrustRank** were computed
- Teleport set was page **B** and **D**
- **Question 1**
  - Page B is not a spam page (True/False)
- **Question 2**
  - Page A has a higher chance to be a link spam compared to the page C. (True/False)

Web Page	PageRank	TrustRank	SpamMass
A	3/9	54/210	0.229
B	2/9	59/210	-0.264
C	2/9	38/210	0.186
D	2/9	59/210	-0.264

Example

- Suppose that both the **PageRank** and **TrustRank** were computed
- Teleport set was page **B** and **D**
  - Which nodes are not the link spams?
    - B and D
    - C has lower chance to be the link spam compared to A

Web Page	PageRank	TrustRank	SpamMass
A	3/9	54/210	0.229
B	2/9	59/210	-0.264
C	2/9	38/210	0.186
D	2/9	59/210	-0.264

Week 8 – Lecture 2
Scalable Clustering with MapReduce

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- PA2
  - Submission deadline extended
  - Extra help hours will be available

1

Topics discussed in the previous class

- Topic 1: Link Spam
- Midterm review (Last 30 minutes)

Topics to be covered

- Clustering and scalable solutions

Part 1. Large Scale Data Analytics

2. Clustering

Scalable *K*-Means Clustering

Clustering: Core concept

- Set of N-dimensional vectors
  - Can be in the order of millions
- **Group (or cluster) them** based on their **proximity** (or similarity) to each other in an N-dimensional space
  - Vectors or objects in a cluster (or group) are more similar to each other than in any other group

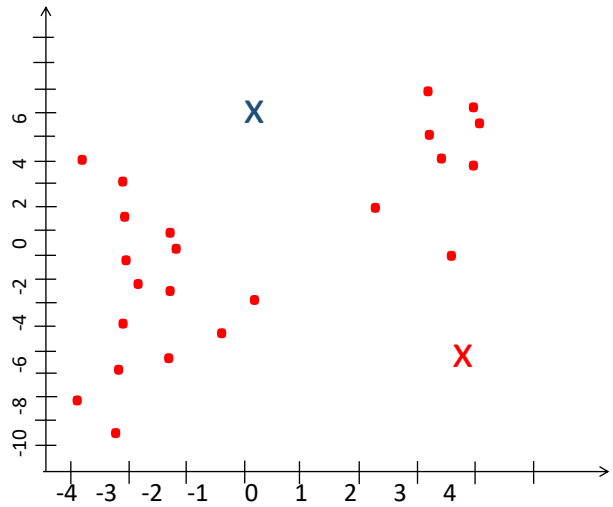
Clustering: Applications

- **Anomaly detection**
  - Fraud detection
- Recommendation systems
- **Medical imaging**
- Market research
- **Human genetic clustering**

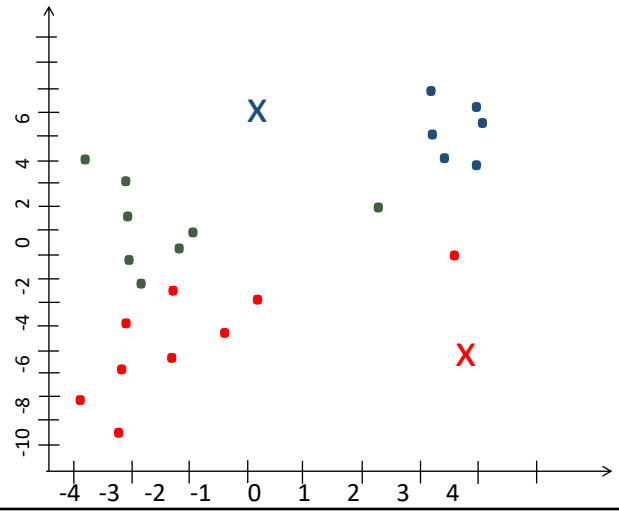
k-Means Clustering

- **Unlabeled** dataset
- Aims to partition m **observations** into k **clusters**
  - Each observation belongs to the cluster with the nearest mean

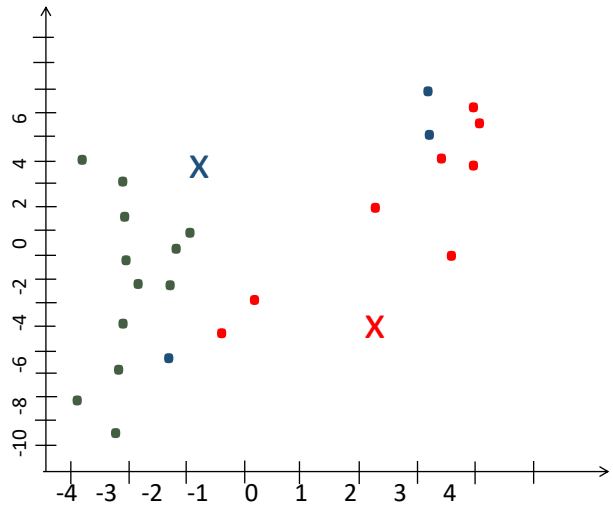
Concept: k-Means Clustering (1/4)



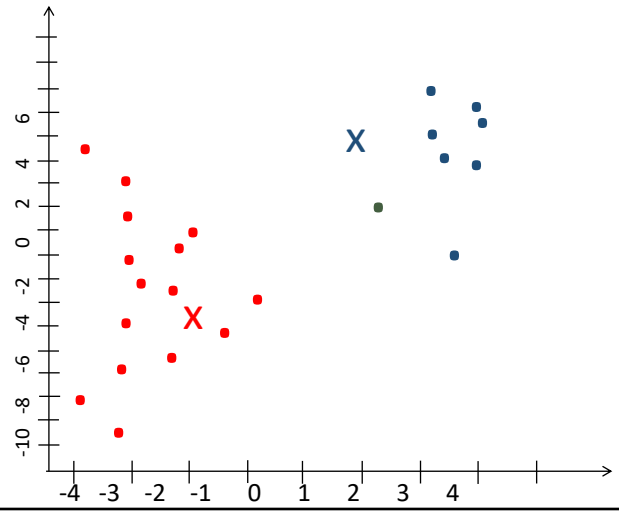
Concept: k-Means Clustering (2/4)



Concept: k-Means Clustering (3/4)



Concept: k-Means Clustering (4/4)



k-Means algorithm - Lloyd's Algorithm (1/2)

- Input
  - k (number of clusters)
  - Training set $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots x^{(m)}\}$

    $x^{(i)} \in R^n$

    (drop x_0= 1 convention)

k-Means algorithm - Lloyd's Algorithm (2/2)

- Randomly initialize K cluster centroids

$$\mu_1, \mu_1, \ldots \mu_k \in R^n$$

```
repeat{
  for i = 1 to m
    c(i):=index (from i to K) of cluster centroid
                                        closest to x(i)
  for k = 1 to K
     μk:= average (mean) of points assigned to cluster k

}
```

Cost function

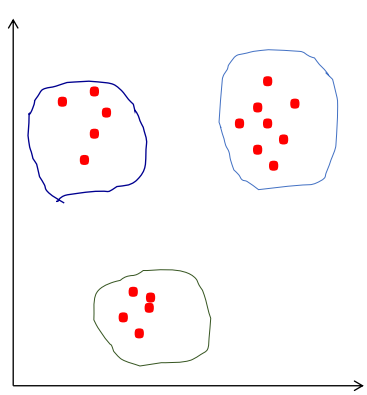
- The **objective is to find**:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} (\| x - \mu_i \|)^2$$

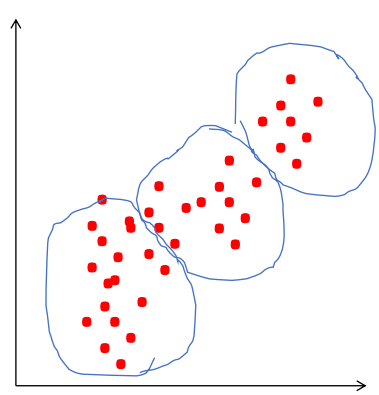
- Where μ_i is the mean of points in S_i

k-Means for non-separated clusters

Separated clusters

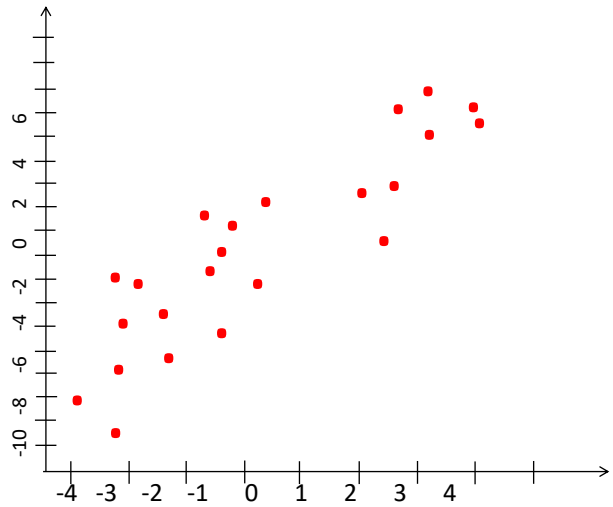


Non-Separated clusters

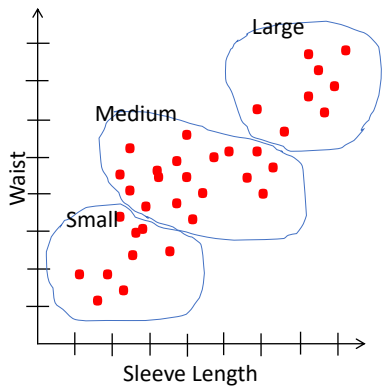


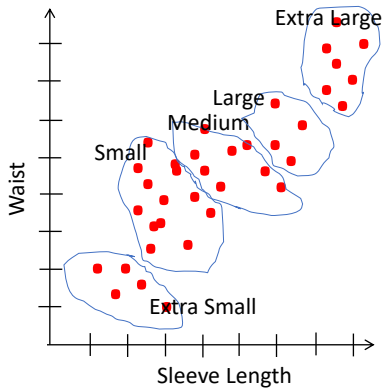
How to choose the number of clusters

- Value k in the algorithm



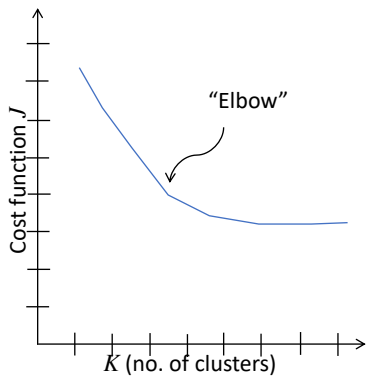
Choosing the value K (2/2)

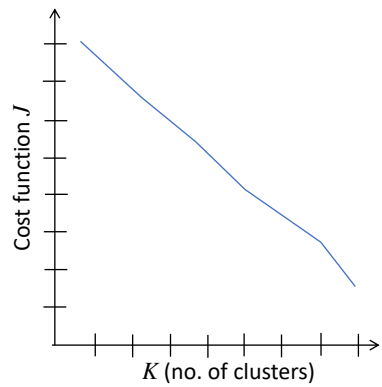




Choosing the value K (1/2)

Elbow Method





Evaluation of your model

- Are your clusters good?

- **Extrinsic Measures**
  - Require ground truth labels
  - Adjusted Rand index, Fowlkes-Mallows scores, Mutual information based scores, Homogeneity, Completeness and V-measure
- **Intrinsic Measures** *(more popular cases)*
  - Does not require ground truth labels
  - Silhouette Coefficient, Calinski-Harabasz Index, Davies-Bouldin Index

Distance Measures

- Euclidean Distance
- Manhattan Distance
- Cosine Distance
- Hamming Distance
- Jaccard Dissimilarity
- Edit Distance
- Smith Waterman Similarity
- Image Distance
- Etc.

Part 1. Large Scale Data Analytics

2. Clustering

Implementing Scalable k-Means using MapReduce (a) Canopy clustering algorithm

Scalable k-Means using MapReduce

- Computing the Euclidean distance between the sample vectors and the centroids can be **parallelized**
  - By splitting the data into individual subgroups and clustering samples in each subgroup separately
    - **By the mapper**
- Recalculating new centroid vectors
  - Divide the sample vectors into subgroups
  - Compute the sum of vectors in each subgroup in parallel
  - Reducer will add up the partial sums and compute the new centroids
- **Question**
  - **How much of data should be transferred for this steps?**
  - **How do we effectively parallelize this process?**

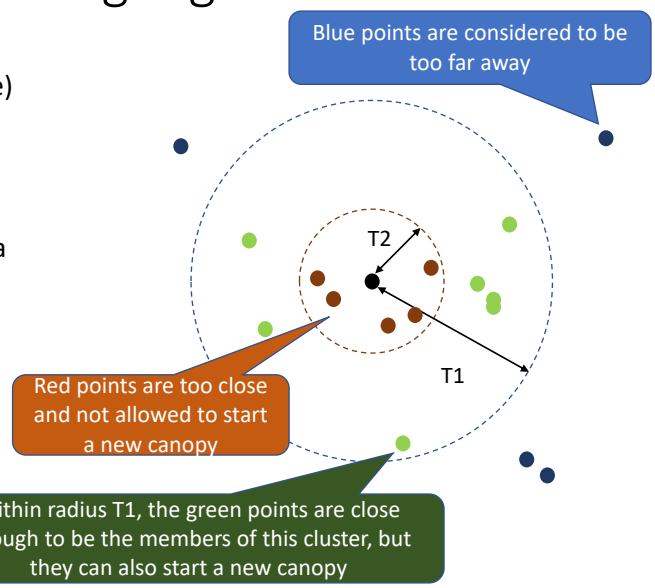
Canopy clustering algorithm

- **Unsupervised pre-clustering algorithm**
  - Defines **the proximity regions**
    - **Instead of starting with random points**
  - Often used as **preprocessing step** for k-Means
- Major goal of this algorithm is to **speed up** clustering operations on large datasets

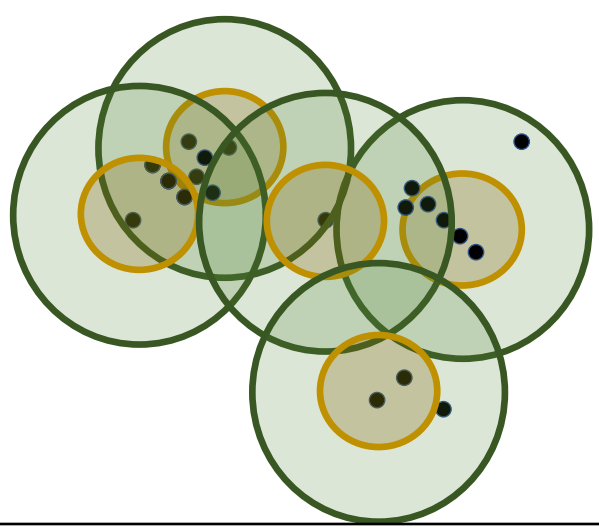
General Canopy Clustering Algorithm

- Using two thresholds T_1 (the loose distance) and T_2 (the tight distance), where $T_1 > T_2$

1. Begin with the set of data points to be clustered
2. **Remove** a point from the set, beginning a new "canopy"
3. For each point left in the set, assign it to the new canopy **if the distance is less than the loose distance T_1**
   - If the distance of the point is **less than** the tight distance T_2, **remove it** from the original set
4. Repeat steps 2-3-4, **until there are no more data points** in the set to cluster



Example of canopies



Using Canopy Clustering Algorithm to Perform k-means using MapReduce

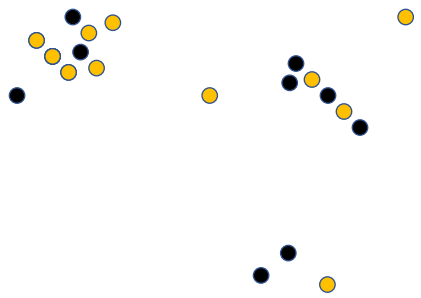
- Step 1. Canopy generation phase
  - **Mapper**
    - Each mapper processes **a subset of the total points** and applies the chosen distance measure and thresholds
    - Input: a subset of the total points
    - Functionality: run the canopy algorithm over a subset
    - Output: <"a_constant_key", canopy_centroid>
      - we will use only one reducer
  - **Reducer**
    - A single reducer
    - Input: <"a_constant_key", [a list of canopy_centroids]>
    - Functionality: run the canopy algorithm over the local canopy centroids
    - Output: Final set of canopy centroids

Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 2. Clustering phase (k-Means)
  - **Mapper**
    - Each mapper reads the **Canopies produced by the first phase (using a Distributed Cache)**
    - Input: a subset of the total points
    - Functionality: find the closest centroid to this point
    - Output: <canopy_centroid_ID, a_point_info>
  - **Reducer**
    - Input: <canopy_centroid_ID, [a list of points in this cluster]>
    - Functionality: calculate new centroid for this cluster
    - Output: a new centroid
- **Repeat Step 2 with the new set of centroids** (from reducer)

Orange points: Mapper A

Black points: Mapper B



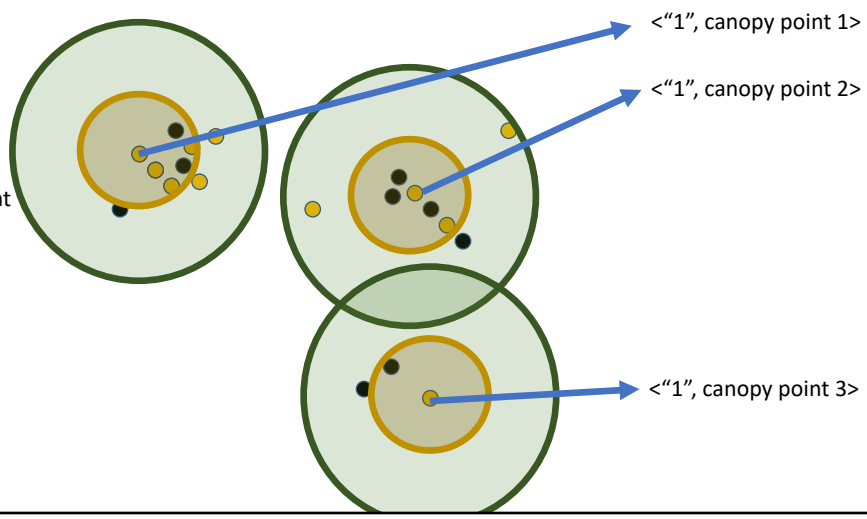
28

Phase 1: Mapper A

Orange points: Mapper A

Black points: Mapper B

Mapper A will select the canopy point 1, 2, and 3 and emits them with a constant key



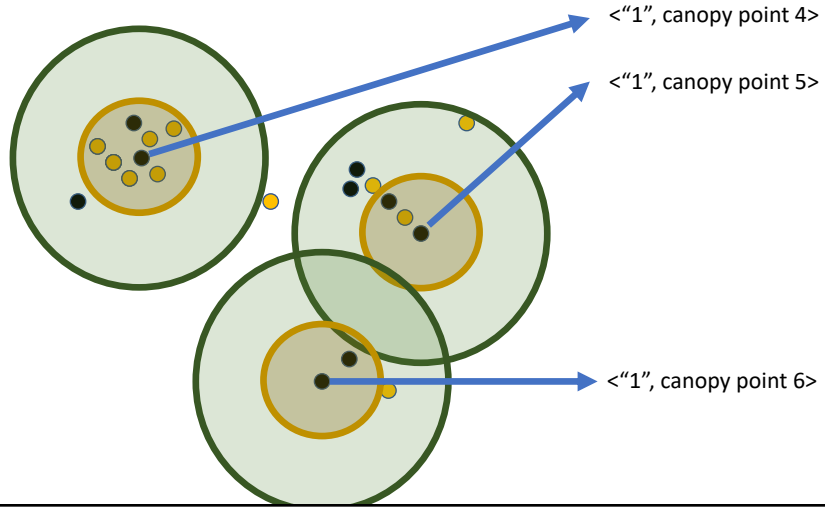
29

Phase 1: Mapper B

Orange points: Mapper A

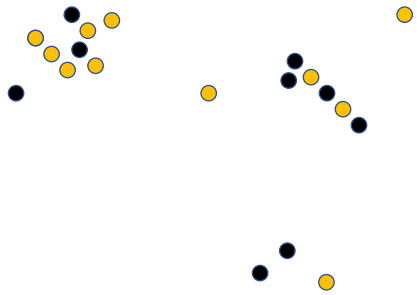
Black points: Mapper B

Mapper B will select the canopy point 1, 2, and 3 and emits them with a constant key



Phase 1: Single Reducer

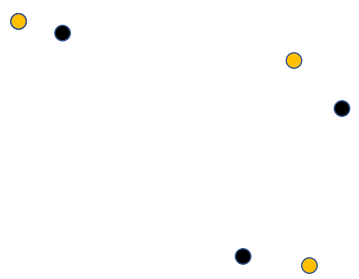
<“1”, [canopy point 1, canopy point 2, canopy point 3, canopy point 4, canopy point 5, canopy point 6] >



Phase 1: Single Reducer

Canopy points only

<"1", [canopy point 1, canopy point 2, canopy point 3, canopy point 4, canopy point 5, canopy point 6] >



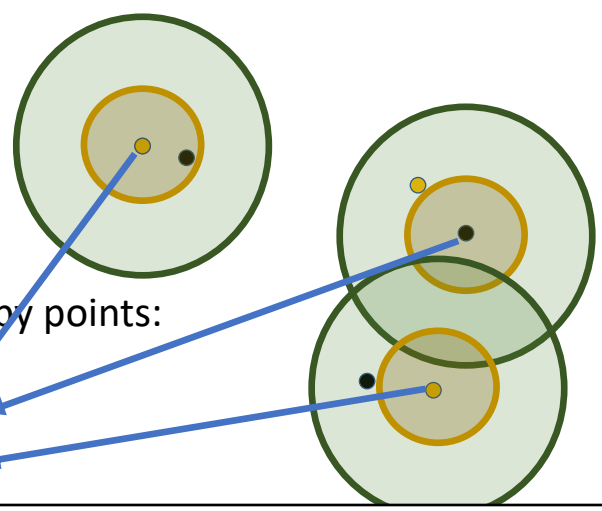
Now run the Canopy Algorithm here

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 8-B Fall 2021 Computer Science

Phase 1: Single Reducer

Canopy points only

<"1", [canopy point 1, canopy point 2, canopy point 3, canopy point 4, canopy point 5, canopy point 6] >



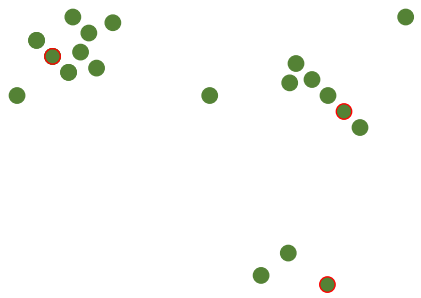
Final set of Canopy points:
Canopy point a
Canopy point b
Canopy point c

Phase 2: Mapper

Red points: canopy centroids

Green points: non-centroid points

Go over all of the points assigned to the current mapper
For the point k, calculate the distance between k to all of the canopy centroids and select the closest centroid
Emit the pair <closest centroid, k>

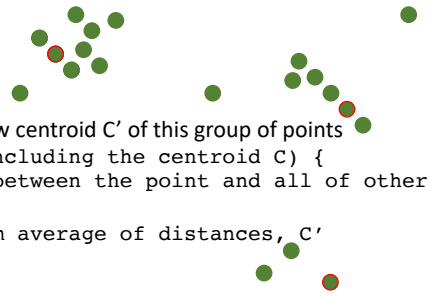


Phase 2: Reducer

Red points: canopy centroids

Green points: non-centroid points

All the points with the same closest centroid will be transferred to the same reducer
Input to a reducer will be: <centroid C, [a list of points sharing c as the closest centroid]>

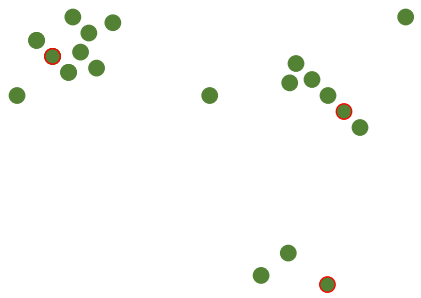


Functionality of this reducer: calculating a new centroid C' of this group of points

```
for all the points in this list (including the centroid C) {
   calculate average of distances between the point and all of other points in this list
}
select a point that has the minimum average of distances, C'
emit C'
```

Phase 2 will be iterated

Phase 2 will be started with the set of C' as new centroids
Repeat map and reduce until the set of centroids converges



REPEAT Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 1. Canopy generation phase
  - **Mapper**
    - Each mapper processes **a subset of the total points** and applies the chosen distance measure and thresholds
    - Input: a subset of the total points
    - Functionality: run the canopy algorithm over a subset
    - Output: <"a_constant_key", canopy_centroid>
      - we will use only one reducer
  - **Reducer**
    - A single reducer
    - Input: <"a_constant_key", [a list of canopy_centroids]>
    - Functionality: run the canopy algorithm over the local canopy centroids
    - Output: Final set of canopy centroids

REPEAT

Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 2. Clustering phase (k-Means)
  - **Mapper**
    - Each mapper reads the **Canopies produced by the first phase (using a Distributed Cache)**
    - Input: a subset of the total points
    - Functionality: find the closest centroid to this point
    - Output: <canopy_centroid_ID, a_point_info>
  - **Reducer**
    - Input: <canopy_centroid_ID, [a list of points in this cluster]>
    - Functionality: calculate new centroid for this cluster
    - Output: a new centroid
- **Repeat Step 2 with the new set of centroids** (from reducer)

Part 1. Large Scale Data Analytics

2. Clustering

Scalable k-Means algorithm
(b) k-Means++

This material is built based on,

- Arthur, D.; Vassilvitskii, S. (2007). "*k*-means++: the advantages of careful seeding" (PDF). *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035
- Bahmani, B., Moseley, B., Vattani, A., Kumar, R. and Vassilvitskii, S., 2012. Scalable k-means++. *arXiv preprint arXiv:1203.6402*.
- Apache Spark Mllib: Clustering
  - https://spark.apache.org/docs/latest/ml-clustering.html

k-Means algorithm- Lloyd's Algorithm: Strengths

- Embarrassingly parallel
  - little or no effort is needed to separate the problem into a number of parallel tasks
- Converges to a local minima
- $O(nkdi)$ runtime
  - n: number of data points
  - k: number of clusters
  - d: dimensionality of a data point
  - i: number of iterations

k-Means algorithm- Lloyd's Algorithm: Weaknesses

- $O(nkdi)$ runtime
- Large number of local minima
- Many local minima are poor
- k is unknown

The K-Means++ Algorithm

- Avoiding cold-start improves results
  - Reducing the number of total iterations
- Initialize cluster centers sequentially
- Only the first center is randomly selected
  - Each further center is selected probabilistically to be far from existing centers
- Result of this is an $O(\log k)$ approximation to the global optima

The K-Means++ Algorithm: Step-by-Step description

Step 1. **Choose one center uniformly at random** from among the data points

Step 2. For each data point *x*, **compute D(*x*)**, the shortest distance between *x* and the nearest center that has already been chosen

Step 3. Choose one new data point at random as a new center, using a weighted probability distribution where a point *x* is chosen with **probability proportional to D(*x*)^2.** ***(D^2 weighting)***

- ***D^2 weighting***: Take a new center, choosing x with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$

Step 4. Repeat Steps 2 and 3 until *k* centers have been chosen

Step 5. Now that the initial centers have been chosen, **proceed using standard k-Means**

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

The K-Means++ Algorithm: How can we improve this?

- GOAL 1: Can we reduce the number of iterations for initializing centroids,
  - Can we select multiple centroids at a time?
- GOAL 2: While considering un-uniformly distributed dataset,
  - Non-uniform selection?
- GOAL 3: With a reasonable approximation guarantees?

Scalable K-Means++(A.K.A. K-Means||) Algorithm

- Parallel version for initializing the centers
  - Oversampling factor $l = \Omega(k)$
- Select an initial center (uniformly at random)
- Computes initial cost of the clustering after this selection, Ψ
- Proceeds in log Ψ iterations
  - Samples each x with probability $ld^2(x, C)/\phi_X(C)$

Scalable K-Means++(A.K.A. K-Means||) Algorithm

- Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of points in the d-dimensional Euclidean space
- Let k be a positive integer specifying the number of clusters
- Let $\| a - b \|$ denote the Euclidean distance between a and b
- For a point x and a subset $Y \subseteq X$ of points, the distance is defined as

$$d(x, Y) = min_{y \in Y} \| x - y \|$$

- For a subset $Y \subseteq X$, its centroid be given by
  - $centroid(Y) = \frac{1}{|Y|} \sum_{y \in Y} y$
- Let C=$\{c_1, c_2, \dots c_k\}$ be a set of points and let $Y \subseteq X$
- The cost of Y with respect to C as
- $\phi_Y(C) = \sum_{y \in Y} d^2(y, C) = \sum_{y \in Y} \min_{i=1,\dots,k} \| y - c_i \|^2$

Scalable K-Means++(A.K.A. K-Means||) Algorithm

1: C ← sample a point uniformly at random from X
2: ψ←ϕ_X(C)
3: for O(log ψ) times do
4: C← sample each point x ∈ X independently with probability of $ld^2(x, C)/\phi_X(C)$
5: C ← C ∪ $C_{\phi X(C)}$
6: end for
7: For x∈C, set w to be the number of points in X closer to x than any other point in C
8: Recluster the weighted points in C into k clusters

- l: Number of points chosen in each iteration
- Total number of points in C is $llog\psi$ $(> \mathrm{k})$
- To reduce the number of centers
  - Step 7 assigns weights to the points in C
  - Step 8 reclusters these weighted points to obtain k centers

K-Means|| Algorithm: A Parallel Implementation

1: C ← sample a point uniformly at random from X
2: ψ←ϕ_X(C)
3: for O(log ψ) times do
4: C← sample each point x ∈ X independently with probability of $ld^2(x, C)/\phi_X(C)$
5: C ← C ∪ $C_{\phi X(C)}$
6: end for
7: For x∈C, set w to be the number of points in X closer to x than any other point in C
8: Recluster the weighted points in C into k clusters

- Line 2: calculate the cost function
- Line 4: each mapper can sample independently
- Line 7: identical to the line 2
  - Each mapper calculate the local

K-Means|| Algorithm: A Parallel Implementation

1: C ← sample a point uniformly at random from X
2: ψ←ϕ_X(C)
3: for O(log ψ) times do
4: C ← sample each point x ∈ X independently with probability of $ld^2(x, C)/\phi_X(C)$
5: C ← C ∪ $C'_{\phi X(C)}$
6: end for
7: For x∈C, set w to be the number of points in X closer to x than any other point in C
8: Recluster the weighted points in C into k clusters

- Line 2: calculate the cost function
  - Mapper: Go over the inputsplit and find the closest centroid and aggregate the shortest distances
  - Reducer aggregate the global cost
- Line 4: each mapper (RDD) can sample independently
  - Random number generation per points, r. if r>=probability, sample it otherwise drop it
- Line 7: identical to the line 2

K-Means|| Algorithm: Properties

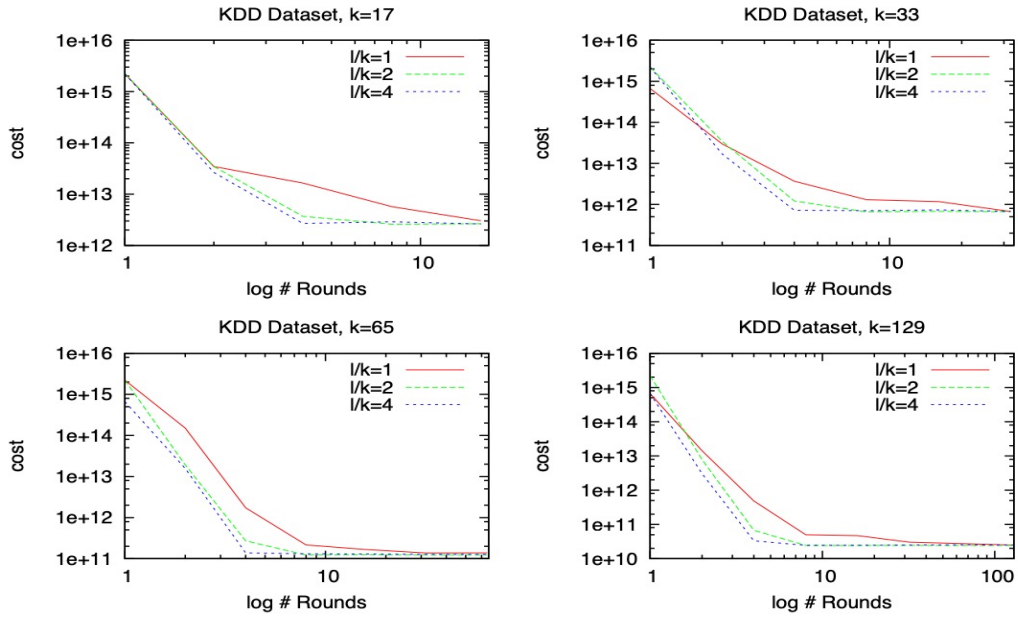
- Requires $O(log\psi)$iterations
- A constant number of iterations is usually enough
- Creates an $O(\log k)$ approximation to the global optima
- Uses results from probability and sequence theory
- Each iteration reduces error by a constant amount and a term proportional to the global error
- Expected value of the cost function approaches a multiple of the global optima
- Performance (with KDDCup 1999 data)
- Paper: Bahmani, B., Moseley, B., Vattani, A., Kumar, R. and Vassilvitskii, S., 2012.

Scalable k-means++. *arXiv preprint arXiv:1203.6402*

- Data: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

	$k = 20$	$k = 50$	$k = 100$
Random	176.4	166.8	60.4
k-means++	38.3	42.2	36.6
k-means‖ $\ell = 0.5k, r = 5$	36.9	30.8	30.2
k-means‖ $\ell = 2k, r = 5$	23.3	28.1	29.7

Note: all values scaled down by 10^{10}


KDD Dataset, k=17
KDD Dataset, k=33
KDD Dataset, k=65
KDD Dataset, k=129
l/k=1
l/k=2
l/k=4
cost
log # Rounds
1e+16
1e+15
1e+14
1e+13
1e+12
1e+11
1e+10
1
10
100

Questions?

Week 9 – Lecture 1
Recommendation Systems

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- PA2
  - Submission deadline extended
  - Extra help hours will be available
- Term project proposal
  - Check if your team members are still available
  - If your team has less than 3 members, please let me know ASAP!

1

1

Topics discussed in the previous class

- Large scale clustering algorithms
  - Canopy algorithm
  - K-means++ and K-means||
  - Implementing with MapReduce

Topics to be covered

- Term project proposal
- Introduction to Recommendation Systems

Term project: Proposal

- Submit via Canvas
  - Team submission
  - 1,000 ~ 1,200 words
  - Document only
- Components of proposal
  - Title
  - Problem formulation
  - Your strategy to solve the problem
  - Your dataset
  - Timeline
  - Bibliography

Dataset example - Web data

- Web data commons
  - http://webdatacommons.org
  - Hyperlink Graph
    - 3.5 billion web pages and 128 billion hyperlinks
  - Product data corpus
    - 5.6 million product records
  - What are the example project?
  - Web analysis
    - Where is your web page?
    - Using In-degree, out-degree, harmonic centrality, PageRank…
  - https://www.wim.uni-mannheim.de/fileadmin/lehrstuehle/ki/pub/Meusel-etal-GraphStructureOfTheWeb.pdf

Dataset example: Enron Email data

- Enron Email dataset
  - 1,227,255 emails with 493,384 attachments
  - https://www.cs.cmu.edu/~./enron/

  - Possible project
    - Spam filter

Dataset example: Network data

- SNAP
  - Stanford Large Network Dataset Collection
  - https://snap.stanford.edu/data/index.html

  - Amazon network
  - Twitter messages
  - Memetracker messages
  - Online community (Reddit, and Flicker)
  - And more

Dataset example: Kaggle

- https://www.kaggle.com/
- Read Kaggle's questions
- Pros: Many sets of interesting data
- Cons: Size of the most of datasets is compact.
- Please check the data size and make sure it is a "Big Data" problem

Dataset: etc.

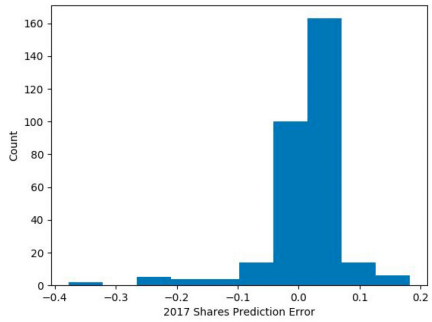
- AWS public data collection
- Reddit comments
  - https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/
  - ~1.7 billion comments (250GB compressed)
- Wikipedia dump
  - https://meta.wikimedia.org/wiki/Data_dumps

Example project

- Predicting Weekly Domestic Box Office Share with Convolutional Neural Networks
  - CS435 2018: Alper Karslioglu, Ben Goodwin, Calvin Le, David Im
- Problem formulation
  - Why is "the movie recommendation" important?
- Methodology
  - Predicting Box Office share using Neural Networks
  - IMDB and Google Trends API

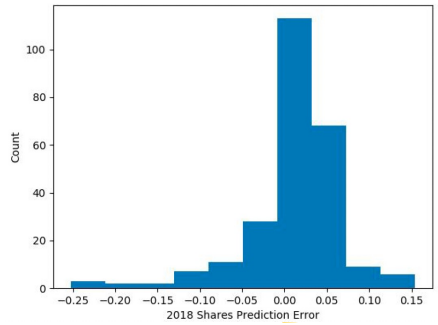
10

Example project



312 movies in 2017

246 movies in 2018



Overall Mean Absolute Error: 0.0459

11

Example project from Fall 2020

- The 2020 Election According to Twitter: Sentiment Analysis vs. the Polls and Election Results
- Predicting the Stock Market using Spark and Machine Learning
- Creating and Testing Stock Trading Algorithms Using Spark
- News and Elon Musk Tweets Effect on Tesla Stock Price
- Exploration of Movie Recommender System Implementations
- Basic Airline Delay Prediction in the United States
- Collaborative Filtering Recommendation Algorithm on Yelp Dataset
- Data Analysis of Wikipedia Network of Top Categories
- Application of Generative Adversrial Networks to Colorize Greyscale Images

Example projects from Fall 2019

- Guess Who's Coming to Dinner
- Incident Analysis and Prediction in San Francisco
- Bitcoin Transaction Patterns
- Finding Helpful Yelp Reviews Using Textual Analysis
- Identifying, Predicting, & Blocking Clickbait Headlines
- Optimal Time to reach YouTube Trending list
- Does initial article tone affect publicity for world events?
- Composition ratio drift during a painters life

Example projects from Fall 2018

- The Six Degrees of Separation in the Movie Industry
- Weather or Not, There is Crime - Relationships between crime and weather patterns in Chicago
- Recognizing ASL Hand Signs Using Spark Deep Learning
- Using Apache Spark to Predict Ozone, Nitrogen Dioxide, and Sulfur Dioxide Effect on Temperature
- Analyzing Song Lyrics to Predict Song Popularity Across 53 Countries
- Shrimp and Whales- An in-depth look into the purchasing traits and group dynamics of Steam accounts
- Top “Invisible” Reasons Businesses Get Their Reviews

Example projects from Spring 2017

- Product rating and classification using NLP
- Amazon DVD Sales Analysis Based on Reviews and Genre
- Aiming for business success using Yelp and Big Data
- Finding Global Health Outcome Correlations
- Analysis of Bike Sharing Service in the Bay Area
- NBA Team Success and Relation to Hometown
- Using Big Data Analytics to Track the Cost/Benefit Ration of Higher Education
- Too Many Games, Not Enough Time
- Recommendation System using Amazon Reviews
- Analysis Prediction and Visualization of Chicago Crime Rate by Area in Real-time
- Hashtag/tweet location correlation analysis
- Effectiveness of intrusion detection using old datasets

Example projects from Spring 2016

- Combined Fuel Economy of Medium Duty Hybrid Trucks
- Where to Live
- Analysis of Reddit Comments for Word Popularity and Trends Using Hadoop
- Music Recommendation Based on Genre-using Million song data
- Trend Analysis to Improve Donations
- Movie recommendation system based on movie attributes and human-movie interaction information
- Song Genre generation
- Predicting Wind Speeds Based on Atmospheric Data
- Darknet Market Survival and Similarity Analysis
- Trends in Baby Names and Predicting Popular Names
- Predicting PM2.5
- Predicting Stock Similarity Using K-Means
- Analyzing Trends on Twitter

Example projects from Spring 2015

- Similarity and Clustering on The Million Song DatasetWhere to Live
- Analysis on Methods Used to Analyze Eclipse and Mozilla Bug Data
- Movie Recommendation using Collaborative Filtering
- A Retrospective Study Of Change In The English Language Via Textual Analysis
- 1929 - 2009 Climate Visualization and Predictive Analysis
- Wikipedia Link analysis
- Google book N gram analysis
- Analysis of stock market trends
- Hazardous Materials Source Mapping
- Wikipedia Page Traffic Statistics Analysis
- Million Song Dataset Geographical Analysis
- Trending Topics and Page Count Prediction on Wikipedia Traffic Log Data

What is a "good" proposal?

- Your proposal must demonstrate
  - **Importance**
  - **Challenging**
  - **Feasibility**
  - Also, your topic should be appropriate to the academic environment
- Final grading (12/20) will be based on
  - **Relevance**
  - **Completeness**
  - **Challenging**
- Your approach
  - Well-structured
  - Your approach should lead you to "good results"
  - Then, what is the "good results"?
    - Interesting knowledge
    - Demonstrating validity of your approach
    - Providing estimable accuracy/performance

Part 1. Large Scale Data Analytics
3. Recommendation Systems
Introduction

This material is built based on

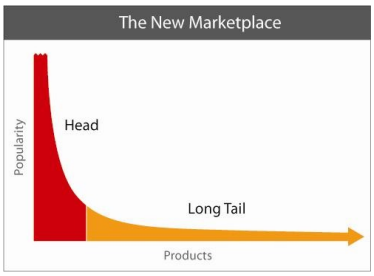
- Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining* (ICDM '08). IEEE Computer Society, Washington, DC, USA, 263-272. DOI=http://dx.doi.org/10.1109/ICDM.2008.22
- Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills, Advanced Analytics with Spark, O'Reilly, 2015

"What percentage of the top 10,000 titles in any online media store (Netflix, iTunes, Amazon, or any other) will rent or sell at least once a month?"

a) 25%
b) 50%
c) 75%
d) More than any of above answers

The long tail phenomenon [1/2]

- Distribution of numbers with a portion that has a large number of occurrences far from the "head" or central part of the distribution
  - The vertical axis represents popularity
  - The items are ordered on the horizontal axis according to their popularity
  - The long-tail phenomenon forces online institutions to recommend items to individual users



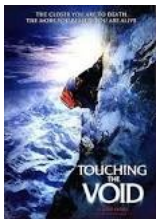
Erik Brynjolfsson, Yu (Jeffrey) Hu, and Duncan Simester. 2011. Goodbye Pareto Principle, Hello Long Tail: The Effect of Search Costs on the Concentration of Product Sales. *Manage. Sci.* 57, 8 (August 2011), 1373-1386. DOI=http://dx.doi.org/10.1287/mnsc.1110.1371

22

22

The long tail phenomenon [2/2]

- "Touching the Void", Joe Simpson, 1988



- "Into Thin Air: A Personal Account of the Mt. Everest Disaster" , Jon Krakauer, 1997

23

23

Recommendation systems

- Seek to predict the "rating" or "preference" that a user would give to an item

Applications of Recommendation Systems

- **Product recommendations**
  - Amazon or similar online vendors
- **Movie recommendations**
  - Netflix offers its customers recommendations of movies they might like
- **News articles**
  - News services have attempted to identify articles of interest to readers based on the articles that they have read in the past
  - Blogs, YouTube

Netflix Prize

- The Netflix Prize challenge concerned recommender systems for movies (October, 2006)
- Netflix released a training set consisting of data from almost 500,000 customers and their ratings on 18,000 movies.
  - More than 100 million ratings
- The task was to use these data to build a model to predict ratings for a hold-out set of 3 million ratings

Large Scale Data Analytics

3. Recommendation Systems

Background: Data Similarity

Jaccard Coefficient (a. without description)

- Compare two sets P and Q with the following formula:

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|}$$

- **Measures the fraction of the data that is shared between P and Q**
- Compared to all data available in the union of these two sets.
- What are P and Q?
  - Set of tokens from Strings
  - Complete description about data (candidates)

Example

- ***StringJaccard***

$S(c_1)$ = {Thomas, Sean, Connery}
$S(c_2)$ = {Sir, Sean, Connery}
What is the string Jaccard coefficient between c1 and c2?

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|} = 2/4 = 0.5$$

Jaccard Coefficient (b. with description)

- Given two candidates, the Jaccard coefficient of two candidates *c1* and *c2* is given by,

$$DescriptionJaccard(c1,c2) = \frac{|OD(c1) \bigcap OD(c2)|}{|OD(c1) \bigcup OD(c2)|}$$

Example

- Now, specify the parts of a person's name as
  - **title, firstname, middlename, and lastname**

$OD(c_1) = \{(\textbf{firstname}, Thomas), (\textbf{middlename}, Sean), (\textbf{lastname}, Connery)\}$
$OD(c_2) = \{(\textbf{title}, Sir), (\textbf{middlename}, Sean), (\textbf{lastname}, Connery)\}$

$DescriptionJaccard(c_1, c_2) = 2/4$

Question

- *DescriptionJaccard* and *StringJaccard* have the same value. Is this always true?

a) Yes

b) No

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 9-A Fall 2021 Computer Science

Example

- ***What if "Sean" would have been put in the firstname/middlename attribute?***

$OD(c_1) = \{(middlename, Thomas), (\mathbf{firstname, Sean}), (lastname, Connery)\}$
$OD(c_2) = \{(title, Sir), (\mathbf{middlename, Sean}), (lastname, Connery)\}$

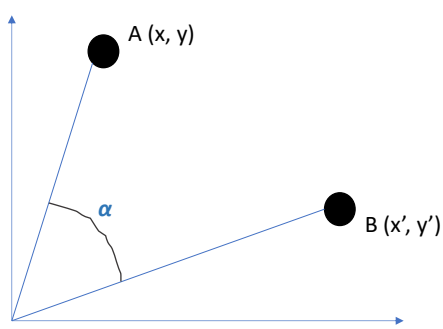
$DescriptionJaccard(c_1, c_2) = 1/5$

Deficiencies of the Jaccard Similarity

- Some attribute is more descriptive
  - **`Title` is less descriptive** than `firstname` and `lastname`
- Very sensitive to **typographical errors** in single tokens
  - ***Shean Conery*** and ***Sean Connery*** have a similarity of zero.

Cosine similarity

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors***



Cosine similarity - Continued

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors*** as

$$CosineSimilarity(V,W) = \cos(\alpha) = \frac{V \bullet W}{\|V\| \times \|W\|}$$

Where $\|V\|$ is the length of the vector $V = [a,b,c..]$ computed as

$$\sqrt{a^2 + b^2 + c^2 + \ldots}$$

Cosine similarity - Continued

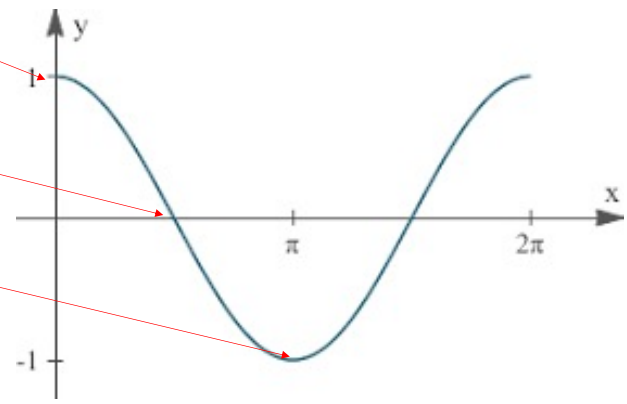
- The vectors V and W
  - Tokens in a string
  - Descriptions of a candidate
- The ***d* dimensions** of these vectors correspond to all ***d* distinct tokens** in a set of strings.
  - Denoted as D
- For a large database, d may be large
  - V and W have high dimensionality d

Cosine similarity - Continued

If the degree between the vectors is 0
→Cosine similarity: 1

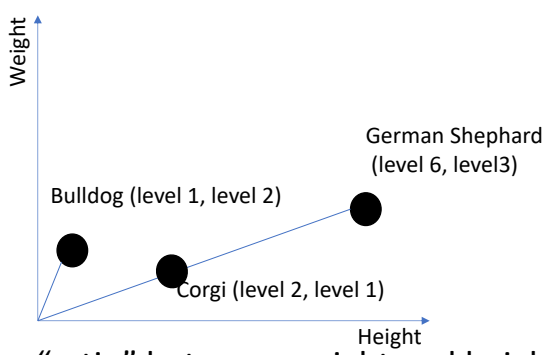
If the vectors are orthogonal
→ Cosine similarity: 0

If the vectors are opposite
→ Cosine similarity : -1



Cosine similarity - Continued

- Which pair of breeds has the highest cosine similarity?



Corgi

Bulldog

German Shephard

- Preserving the "ratio" between weight and height

Wait! They don't look similar enough..

Weight of Token

- Vector contains a **weight** for each of the d distinct tokens
- How to measure the weight?
  - **Measuring frequency**
  - **Term frequency – inverse document frequency** (tf-idf)

Term frequency

- **Number of times** that term t occurs **in the document d**
  - Raw term frequency: $f_{t,d}$
  - Boolean "frequencies" $tf(t,d)$ is 1 if t occurs in d and 0 otherwise
  - e.g. Logarithmically scaled frequency:

$$tf(t,d) = 1 + \log f_{t,d}$$

    - 0 if $f_{t,d}$ is zero

Example: What is the raw $tf_{"American", C3}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

Example: What is the raw $tf_{"American", C3}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

$tf_{"American", C3} = 1$

Inverse document frequency

- **Assigns higher weights** to tokens that **occurs less frequently in the scope of all candidate descriptions**

$$idf(t,D) = \log_{10} \frac{N}{|\{d \in D : t \in d\}|}$$

- Where, N is the total number of documents in the corpus
- Number of documents where the term t appears (i.e. $tf(t,d) \neq 0$)

Example: What is the $idf_{"American", D}$? (D: this corpus)

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

Example: What is the $idf_{"American", D}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

$idf_{"American", D} = log(10/3)$

tf-idf weighting

- The product of its *tf* weight and its *idf* weight

$$W_{t,d} = (1 + \log_{10} tf_{t,d}) \times \log_{10}(N / df_t)$$

For the total number of documents, N

- If the numbers of terms are different in the documents, you should normalize $tf_{t,d}$ to $(tf_{t,d} / (\text{number of words in } d))$
- Best known weighting scheme in information retrieval
  - Note: the "-" in *tf-idf* is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Example: What is the $tf\text{-}idf_{\text{"American"}, C3,D}$ and $tf\text{-}idf_{\text{"American"}, C8,D}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

Example: What is the $tf\text{-}idf_{"American", C3,D}$ and $tf\text{-}idf_{"American", C8,D}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John

$tf\text{-}idf_{"American", C3,D} = (1+\log(1/4)) \times \log(10/3)$

$tf\text{-}idf_{"American", C8,D} = (1+\log(1/6)) \times \log(10/3)$

Example continued

- Compute the similarity between the two strings s1="Farmers Insurance", s2 = "Liberty Insurance"
- Term vector = (Allstate, American, Automobile, Association, National, Insurance, Farmers, Liberty, ….)
- *s1="Farmers Insurance" = (0, 0, 0, 0, 0, 1, 1, 0 …)*
- *s2="Liberty Insurance" = (0, 0, 0, 0, 0, 1, 0, 1, …)*
- Next step: find the weights

Example continued

CID	Name
c1	Allstate
c2	American Automobile Association
c3	American National Insurance Company
c4	Farmers Insurance
c5	GEICO
c6	John Hancock Insurance
c7	Liberty Insurance
c8	Mutual Insurance of America Life Insurance
c9	Safeway Insurance Group
c10	Westfield

Q	V	W
Allstate	0	0
America	0	0
Automobile	0	0
Association	0	0
National	0	0
Insurance		
Company	0	0
Farmers		0
GEICO	0	0
John	0	0
Hancock	0	0
Liberty	0	
…		

Example continued

- Compute the similarity between the two strings *s1="Farmers Insurance", s2 =" Liberty Insurance"*

- Six of the candidates contain the token "***Insurance***".
- $idf_{\text{"Insurance"}, D} = \log_{10}(10/6) = 0.78$
- $idf_{\text{"Farmers"}, D} = \log_{10}(10/1) = 1$
- $idf_{\text{"Liberty"}, D} = \log_{10}(10/1) = 1$

- $\text{tf-}idf_{\text{"Farmers"}, c4} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/1) = 0.7$
- $\text{tf-}idf_{\text{"Insurance"}, c4} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/6) \approx 0.55$
- $\text{tf-}idf_{\text{"Liberty"}, c7} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/1) = 0.7$
- $\text{tf-}idf_{\text{"Insurance"}, c7} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/6) \approx 0.55$

Example continued

CID	Name
c1	Allstate
c2	American Automobile Association
c3	American National Insurance Company
c4	Farmers Insurance
c5	GEICO
c6	John Hancock Insurance
c7	Liberty Insurance
c8	Mutual Insurance of America Life Insurance
c9	Safeway Insurance Group
c10	Westfield

Q	V	W
Allstate	0	0
America	0	0
Automobil e	0	0
Association	0	0
National	0	0
Insurance	0.55	0.55
Company	0	0
Farmers	0.7	0
GEICO	0	0
John	0	0
Hancock	0	0
Liberty	0	0.7
…		

Example continued

- Compute the similarity between the two strings **s1=Farmers Insurance**, **s2 = Liberty Insurance**

$$CosSimilarity(V,W) = \cos(\alpha) = \frac{V \bullet W}{\| V \| \times \| W \|}$$

$$= \frac{0.55 \times 0.55}{\sqrt{0.55^2 + 0.7^2}^2} \cong 0.38$$

- What is the ***Jaccard similarity*** for the same case?

Questions?

Week 9 – Lecture 2
Recommendation Systems

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- Term project proposal
  - Check if your team members are still available
  - If your team has less than 3 members, please let me know ASAP!

1

Topics discussed in the previous class

- Term project proposal
- Introduction to Recommendation Systems

Topics to be covered

- Recommendation Systems: Collaborative Filtering
- Evaluation Methods

Large Scale Data Analytics

3. Recommendation Systems

Collaborative Filtering

REVIEW

Jaccard Coefficient (a. without description)

- Compare two sets P and Q with the following formula:

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|}$$

- **Measures the fraction of the data that is shared between P and Q**
- Compared to all data available in the union of these two sets.
- What are P and Q?
  - Set of tokens from Strings
  - Complete description about data (candidates)

REVIEW

Example

- ***StringJaccard***

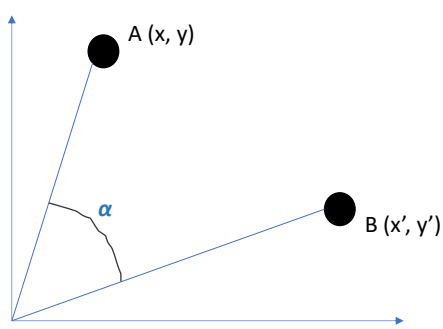
$S(c_1) = \{Thomas, Sean, Connery\}$
$S(c_2) = \{Sir, Sean, Connery\}$
What is the string Jaccard coefficient between c1 and c2?

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|} = 2/4 = 0.5$$

REVIEW

Cosine similarity

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors***



REVIEW

Cosine similarity - Continued

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors*** as

$$CosineSimilarity(V,W) = \cos(\alpha) = \frac{V \bullet W}{\|V\| \times \|W\|}$$

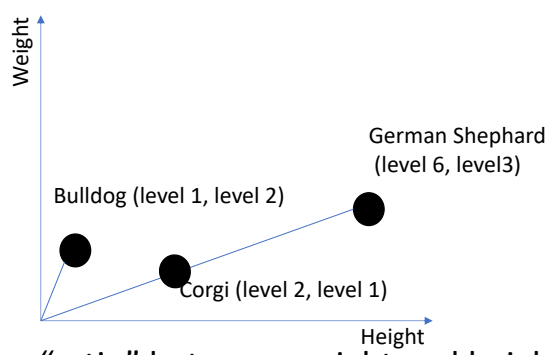
Where $\|V\|$ is the length of the vector $V = [a,b,c..]$ computed as

$$\sqrt{a^2 + b^2 + c^2 + \ldots}$$

REVIEW

Cosine similarity - Continued

- Which pair of breeds has the highest cosine similarity?



Corgi

Bulldog

German Shephard

- Preserving the "ratio" between weight and height

REVIEW

Weight of Token

- Vector contains a **weight** for each of the d distinct tokens
- How to measure the weight?
  - **Measuring frequency**
  - **Term frequency – inverse document frequency** (tf-idf)

Methodologies for the Recommendation Systems

- Content-based filtering
  - Based on a description of the item and a profile of the user's preferences
  - Learning a classifier for the user's preference based on an item's features
  - User's preferences (e.g. previous ratings)
  - Example: Since Alice liked an oversized tennis racket before, she will like "an oversized badminton racket"
- Collaborative filtering
  - "*People who agreed in the past will agree in the future*"
  - The system generates recommendations using only information about rating profiles for different users or items
  - Example: "Since Alice and Bob showed a similar ranking patterns for movies, Alice will like the movie that Bob liked yesterday."

Collaborative filtering

- Focus on the **similarity** of the **different customers' user ratings** for items
- Users are **similar if their rating vectors are close** according to some distance measure
  - E.g. Jaccard or cosine distance
- Collaborative filtering
  - The process of identifying similar users and recommending what similar users like

Measuring similarity between users

- How to measure similarity of users or items from their rows or columns in the utility matrix?
  - Jaccard Similarity for A and B: 1/5 (without considering the weight)
  - Jaccard Similarity for A and C: 2/4 (without considering the weight)
- For user A, user C **might have similar opinion** than user B
- Can user C provide a prediction for A?

	AV	EG	ET	TW	FF1	FF2	FF3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Cosine similarity (1/2)

- We can treat blanks as a 0 values
  - A = (4,0,0,5,1,0,0), B = (5,5,4,0,0,0,0)
- The **cosine of the angle** between A and B is

$$\frac{4\times 5}{\sqrt{4^2+5^2+1^2}\sqrt{5^2+5^2+4^2}}=0.380$$

	AV	EG	ET	TW	FF1	FF2	FF3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Cosine similarity (2/2)

- We can treat blanks as 0 values
  - A = (4,0,0,5,1,0,0), C = (0,0,0,2,4,5,0)
- The **cosine of the angle** between A and C is

$$\frac{5\times 2+1\times 4}{\sqrt{4^2+5^2+1^2}\sqrt{2^2+4^2+5^2}}=0.322$$

- A is slightly closer to B (0.380) than to C (0.322)

Then, what will you recommend to the user A?

	AV	EG	ET	TW	FF1	FF2	FF3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Normalizing ratings (1/3)

- Generous vs. Harsh Ratings
- What if we **normalize** ratings by subtracting from each rating the average rating of that user?
- Some rating (very low) will turn into negative numbers
- If we take the cosine distance, the opposite views of the movies will have vectors in **almost opposite directions**
  - It can be as far apart as possible

Normalizing ratings (2/3)

- What if we **normalize** ratings by subtracting from each rating the average rating of that user?

	AV	EG	ET	TW	FF1	FF2	FF3	Average
A	4 (12/10-10/3 = 2/3)			5	1			10/3
B	5 (15/3 – 14/3 = 1/3)	5 (15/3 – 14/3 = 1/3)	4					14/3
C				2	4	5		11/3
D		3 (6/2 – 6/2 = 0)					3	6/2

Normalizing ratings (3/3)

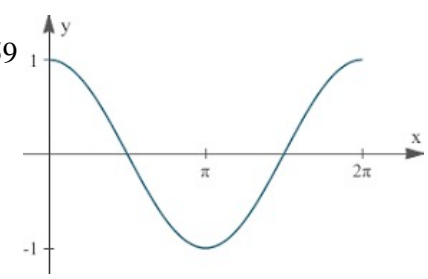
- The cosine of the angle between A and B

$$\frac{(2/3)\times(1/3)}{\sqrt{(2/3)^2+(5/3)^2+(-7/3)^2}\sqrt{(1/3)^2+(1/3)^2+(-2/3)^2}}=0.092$$

	AV	EG	ET	TW	FF1	FF2	FF3
A	2/3			5/3	-7/3		
B	1/3	1/3	-2/3				
C				-5/3	1/3	4/3	
D		0					0

- The cosine of the angle between A and C

$$\frac{(5/3)\times(-5/3)+(-7/3)\times(1/3)}{\sqrt{(2/3)^2+(5/3)^2+(-7/3)^2}\sqrt{(-5/3)^2+(1/3)^2+(4/3)^2}}=-0.559$$



- A and C are much further apart than A and B.
- Neither pair is very close
- **A and C disagree on the two movies** they rated in common, while **A and B give similar scores** to the one movie they rated in common

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3			5/3	-7/3		
B	1/3	1/3	-2/3				
C				-5/3	1/3	4/3	
D		0					0

Computational complexity

- Worst case
  - ***O(MN)*** where M is the number of customers and N is the number of product catalog items
  - It examines M **customers** and up to N **items** for each customer

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 9-B Fall 2021 Computer Science

Computational complexity (1/3)

- The average customer vector is **extremely sparse!**
  - The algorithm's performance tends to be closer to ***O(M+N)***
  - Scanning every customer
    - ***O(M)*** **not** ***O(MN)***
    - Almost every customer has very small N
  - Few customers who have purchased or rated a significant percentage of items
    - Requires $O(N)$
  - **10 million customers and 1 million items?**

Computational complexity (2/3)

- We can reduce M by:
  - **Randomly sampling** the customers
  - Discarding customers with few purchases
- We can reduce N by:
  - Discarding **very popular or unpopular items**
  - **Partitioning the item space** based on the product category or subject classification

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 9-B Fall 2021 Computer Science

Computational complexity (3/3)

- Dimensionality reduction techniques can reduce M or N by a large factor
  - **Clustering**
  - Principal component analysis

Disadvantage of space reduction

- Reduced recommendation quality
  - Sampled customer
    - More similar customers will be dropped
  - Item-space partitioning
    - It will restrict recommendations to a specific product or subject area
  - Discarding most popular or unpopular items
    - They will never appear as recommendations

Large Scale Data Analytics

3. Recommendation Systems

Amazon.com : Item-to-item collaborative filtering

This material is built based on,

- Greg Linden, Brent Smith, and Jeremy York, "Amazon.com Recommendations, Item-to-Item Collaborative Filtering" IEEE Internet Computing, 2003

- **Amazon.com** uses recommendations as a targeted marketing tool
  - Email campaigns
  - Most of their web pages

Item-to-item collaborative filtering

- It does **NOT** match the **user to similar customers**
- Item-to-item collaborative filtering
  - **Matches each of the user's purchased and rated items to similar items**
  - Combines those similar items into a recommendation list

Determining the most-similar match

- The algorithm builds a **similar-items table**
  - By finding items that **customers tend to purchase** **together** in the customer's history
- How about building a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair?
- Many product pairs have no common customer
  - If you already bought a TV today, will you buy another TV again today?

Co-occurrence in a product-to-product matrix

- Suppose that there are 4 users A, B, C, D, and E with 5 products P1, P2, P3, P4, and P5
- A purchased P1, P2, P3
- B purchased P2, P3, P5
- C purchased P3, P4
- D purchased P2, P4

P1

P2

P3

P4

P5

Co-occurrence in a product-to-product matrix

- Suppose that there are 4 users A, B, C, D, and E with 5 products p1, p2, p3, p4, and p5
  - A purchased p1, p2, p3
    - (P1,P2) (P2,P1): recorded as 1
    - (P2,P3)(P3,P2): recorded as 1
    - (P1,P3)(P3,P1): recorded as 1
  - B purchased p2, p3, p5
  - C purchased p3, p4
  - D purchased p2, p4

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Computing similarity

- Using **cosine measure**
  - Measuring the similarity between two items
  - M dimensions correspond to customers who have purchased that item

Building a Similarity product-to-product matrix

- Similarity between products
  - P1: T-shirt
  - P2: video cable
  - P3: projector
  - P4: tv
  - P5: smart tv
- P1 = (0,1,1,0,0)
- P2 = (1,0,1,1,1)
- P3 = (1,1,0,1,0)
- P4 = (0,1,1,0,0)
- P5 = (0,1,0,0,0)

P1 P2 P3 P4 P5

	p1	**p2**	**p3**	**p4**	**p5**
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Co-occurrence matrix

Building a Similarity product-to-product matrix

- Similarity between products
- P1 = (0,1,1,0,0)
- P2 = (1,0,1,1,1)
  - CosSim(P1, P2) $=\frac{1}{\sqrt{2}\sqrt{4}}=\frac{1}{2\sqrt{2}}$ = 0.35
- P1 = (0,1,1,0,0)
- P3 = (1,1,0,1,0)
  - CosSim(P1, P3) = $\frac{1}{\sqrt{2}\sqrt{3}}=\frac{1}{\sqrt{6}}$=0.41

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Building a Similarity product-to-product matrix

$\sqrt{2} = 1.41$
$\sqrt{3} = 1.73$
$\sqrt{6} = 2.45$

- Similarity between products
- P2 = ?
- P3 = ?
  - CosSim(P2, P3) =?
- P2 = ?
- P4 = ?
  - CosSim(P2, P4) = ?
- P2 = ?
- P5 = ?
  - CosSim(P1, P3) =?

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Building a Similarity product-to-product matrix

- Similarity between products
- P2 = (1,0,1,1,1)
- P3 = (1,1,0,1,0)
  - CosSim(P2, P3) $= \frac{2}{\sqrt{4}\sqrt{3}} = \frac{2}{2\sqrt{3}} = 0.58$
- P2 = (1,0,1,1,1)
- P4 = (0,1,1,0,0)
  - CosSim(P2, P4) $= \frac{1}{\sqrt{4}\sqrt{2}} = \frac{1}{2\sqrt{2}} = 0.35$
- P2 = (1,0,1,1,1)
- P5 = (0,1,0,0,0)
  - CosSim(P2, P5) = 0

P1 P2 P3 P4 P5

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

36

36

Building a Similarity product-to-product matrix

- Similarity between products
- P3 = (1,1,0,1,0)
- P4 = (0,1,1,0,0)
  - CosSim(P3, P4) $= \frac{1}{\sqrt{3}\sqrt{2}} = \frac{1}{\sqrt{6}} = 0.41$
- Now generate the similarity matrix

P1 P2 P3 P4 P5

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

37

37

Building a Similarity product-to-product matrix

- Similarity between products
  - CosSim(P1, P2) = 0.35
  - CosSim(P1, P3) = 0.41
  - CosSim(P2, P3) = 0.58
  - CosSim(P2, P4) = 0.35
  - CosSim(P2, P5) = 0

  - CosSim(P3, P4) = 0.41

	p1	p2	p3	p4	p5
p1	0	1 0.35	1 0.41	0	0
p2	1 0.35	0	1 0.58	1 0.35	1 0
p3	1 0.41	1 0.58	0	1 0.41	0
p4	0	1 0.35	1 0.41	0	0
p5	0	1 0	0	0	0

Scalability (1/2)

- Amazon.com has around **300 million** customers (200 million Amazon Prime members) and more than **12 million** cataloged items (including market place: more than 350 million items)
- Traditional collaborative filtering does little or no offline computation
- Online computation scales with the number of customers and catalog items.

Scalability (2/2)

- Cluster models can perform much of the computation offline
  - Recommendation **quality is relatively poor**

- REVISIT: Content-based model?
  - It cannot provide recommendations with interesting, targeted titles
  - Not scalable for customers with numerous purchases and ratings

Algorithm

- Calculating the similarity between a single product and all related products (without going through all (#_of_users x #_of_products) combinations)

```
For each item in product catalog, P-a
     For each customer C who purchased P-a
       For each item P-b purchased by customer C
         Record that a customer purchased P-a and P-b

     For each item P-b
         Compute the similarity between P-a and P-b
```

Creating a similarity product-to-product table

- Similar-items table is extremely computing intensive
  - **Offline computation**
  - $O(N^2M)$ in the worst case
    - Where N is the number of items and M is the number of users
  - Average case is closer to $O(NM)$
    - Most customers have very few purchases
  - Sampling customers who purchase best-selling titles reduces runtime even more
    - With little reduction in quality

Key scalability strategy for amazon recommendations

- Creating the **expensive similar-items table offline**
- Online component
  - Looking up similar items for the user's purchases and ratings
  - Scales independently of the catalog size or the total number of customers
- It is dependent only on how many titles the user has purchased or rated

Computing with MapReduce

Part 1

```
For each item in product catalog, P-a
   For each customer C who purchased P-a
    For each item P-b purchased by customer C
      Record that a customer purchased P-a and P-b

   For each item P-b
      Compute the similarity between P-a and P-b
```

- A Mapper reads a customer's purchase history
  - Extracts product id(s) and generates co-purchased pairs
  - Emits the combination of the product IDs (sorted) → ((P1, P2), "1")
- A Reducer adds the occurrences of the co-purchased products
  - Emits the (a-pair-of-co-purchased-products, total count)

Computing with MapReduce: Two MapReduce jobs

```
For each item in product catalog, P-a
  For each customer C who purchased P-a
    For each item P-b purchased by customer C
      Record that a customer purchased P-a and P-b

For each item P-b
      Compute the similarity between P-a and P-b
```

Part 2

- **Originally, part 1 and part 2 were combined in a single for-loop. We will separate part 2.**
- Cartesian product design pattern (see the join pattern)
- A pair of product vectors will be provided to a map function
  - E.g. P1 (0, 1, 0, 0, 0, 1, 0), P2(0, 1, 1, 0, 0, 1, 1)
- Mapper will calculate the cosine similarity
  - Emits Cosine_Similarity ((0, 1, 0, 0, 0, 1, 0), (0, 1, 1, 0, 0, 1, 1))
- No reducer required

Recommendation quality

- The algorithm recommends highly correlated similar items
  - Recommendation quality is excellent
  - Algorithm performs well with limited user data

Large Scale Data Analytics

4. Evaluation and Validation

Why evaluation/validation? (1/2)

- Process for model selection and performance estimation
- Model selection (fitting the model)
  - Most of the models have one or more **free parameters**

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

You should find these values to generate (fit) your model.

- How do we select the **"optimal" parameter(s)** or model for a given classification problem/predictive analytics?

Why evaluation/validation? (2/2)

- Performance estimation
  - Once we have chosen a model, how do we **estimate its performance (accuracy)**?
- Performance is typically measured by the **true error rate**
  - e.g. the classifier's error rate on the entire population

Challenges (1/2)

- If we had access to **an unlimited number** of examples these questions have a straightforward answer
  - Choose the model that provides the **lowest error rate** on the entire population
    - Of course, that error rate is the true error rate
- In real applications we only have access to **a subset of examples**, usually smaller than we wanted
  - What if we use the **entire available data to fit our model** and estimate the error rate?
    - The final model will normally overfit the training data
    - We already used the test dataset to train the data

Challenges (2/2)

- This problem is more pronounced with models that have a **large number of parameters**
  - The error rate estimate will be overly optimistic (lower than the true error rate)
    - In fact, it is not uncommon to have 100% correct classification on training data
- A much better approach is to **split the training data** into disjoint subsets: the holdout method

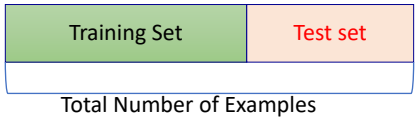
Large Scale Data Analytics

4. Evaluation and Validation

1. The Holdout method

The Holdout Method

- Split dataset into two groups
  - Training set
    - Used to train the model
  - Test set
    - Used to estimate the error rate of the trained model



  - Simplest cross validation method

Drawbacks of the holdout method

- Drawbacks
  - For **a sparse dataset**, we may **not be able to set aside a portion** of the dataset for testing
  - Based on the **where "split" happens**, the **estimate of error can be misleading**
    - Sample might not be representative
- The limitations of the holdout can be overcome with a family of resampling methods
  - More computational expense
  - Stratified sampling
  - Cross Validation
    - Random subsampling
    - K-Fold cross validation
    - Leave-one-out Cross-Validation

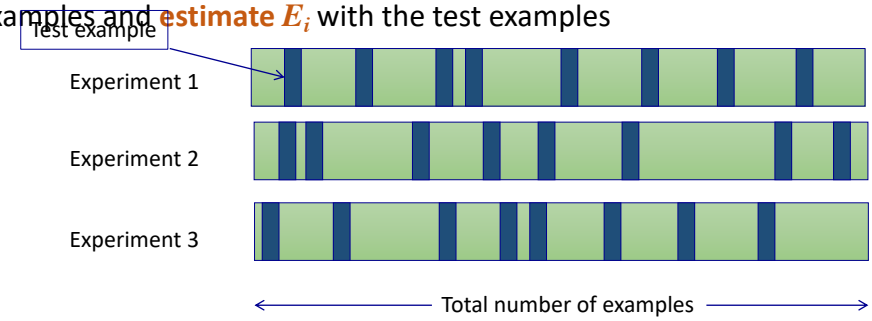
Large Scale Data Analytics

4. Evaluation and Validation

2. Random Subsampling

Random Subsampling

- K data splits of the dataset
  - Each split **randomly selects a (fixed) number of examples** without replacement
  - For each data split, retain the classifier from scratch with the training examples and **estimate E_i** with the test examples



True Error Estimate

- The true error estimate is obtained as the **average of the separate estimates E_i**

- This estimate **is significantly better than the holdout estimate**

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

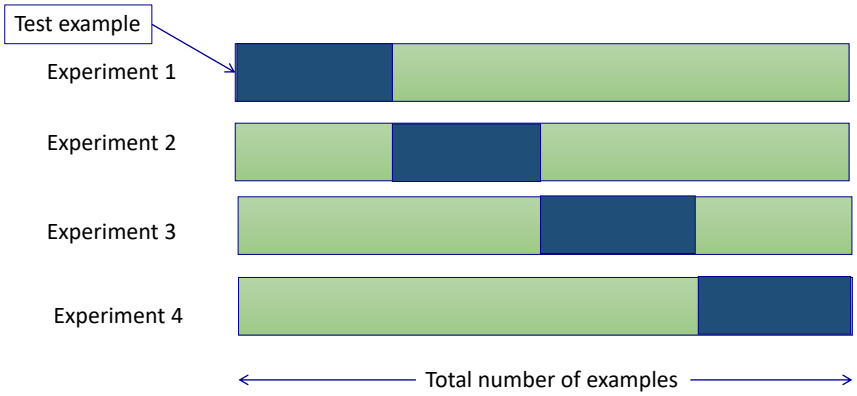
Large Scale Data Analytics

4. Evaluation and Validation

3. k-Fold Cross-validation

k-Fold Cross-validation

- Create a ***k*-fold partition** of the dataset
  - For each of the k experiments use ***K-1* folds for training**
    - The remaining one for testing



True error estimate

- k-fold cross validation is similar to random subsampling
- The advantage of k-Fold Cross validation
  - All the examples in the dataset are **eventually used for both training and testing**
- The true error is estimated as the average error rate

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

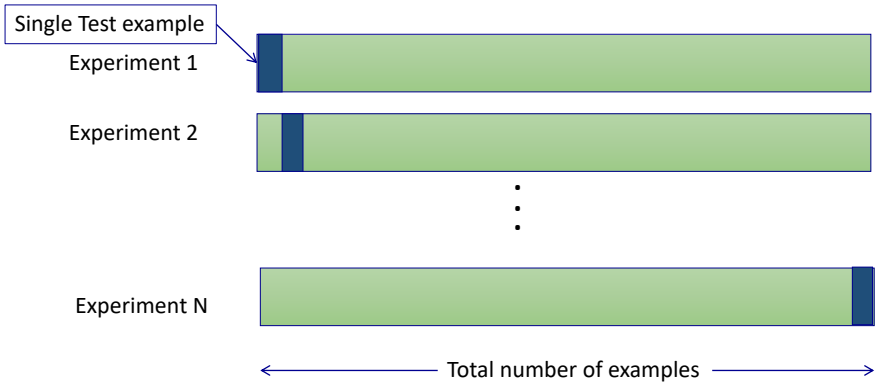
Large Scale Data Analytics

4. Evaluation and Validation

4. Leave-one-out Cross-validation

Leave-one-out Cross Validation

- Leave-one-out is the degenerate case of k-Fold Cross validation
  - k is chosen as the total number of examples
- For a dataset with N examples, perform N experiments
  - Use N-1 examples for training, the remaining example for testing



How many folds are needed?

- The choice of the number of folds depends on the size of the dataset
  - For large datasets, even 3-Fold Cross Validation will be quite accurate
  - For very sparse datasets, you may have to consider leave-one-out
    - To get maximum number of experiments
- A common choice for **k-fold Cross Validation is k=10**

Three-way data splits

- If model selection and true error estimates are computed **simultaneously**
  - The data needs to be divided into three disjoint sets
  - Training set
    - E.g. to find the optimal weights
  - Validation set
    - A set of examples used to tune the parameters of a model
    - To find the "optimal" number of hidden units or determine a stopping point for the back propagation algorithm
  - Test set
    - Used only to assess the performance of a fully-trained model
- **After assessing the final model with the test set, you must not further tune the model**

Large Scale Data Analytics

4. Evaluation and Validation

Classifiers

Classification

Mammals

Fishes

Birds

Insects

Plain Accuracy

- Classifier accuracy
  - General measure of classifier performance

Accuracy = (Number of correct decisions made) / (Total number of decision made)

- 1 – error rate

- Pros
  - **Very easy to measure**
- Cons
  - **Cannot consider realistic cases**

The Confusion Matrix

- A type of **contingency table**
- n classes
  - n x n matrix
    - The columns labeled with actual classes
    - The rows with predicted classes
- Separates out the decisions made by the classifier
  - How one class is being confused for another
  - Different sorts of errors may be dealt with separately

	p (positive)	**n (negative)**
Y (predicted)	True positive	False positive
N (predicted)	False negative	True negative

Problems with Unbalanced Classes

- Consider a classification problem where one class is **rare**
  - Sifting through a large population of normal entities to find a relatively small number of unusual ones
  - Looking for **defrauded customers**, or **defective parts**
  - The class distribution is **unbalanced or skewed**

Confusion Matrix of A: evaluated with 1,000 datapoints

	positive	**negative**
True	250	450
False	250	50

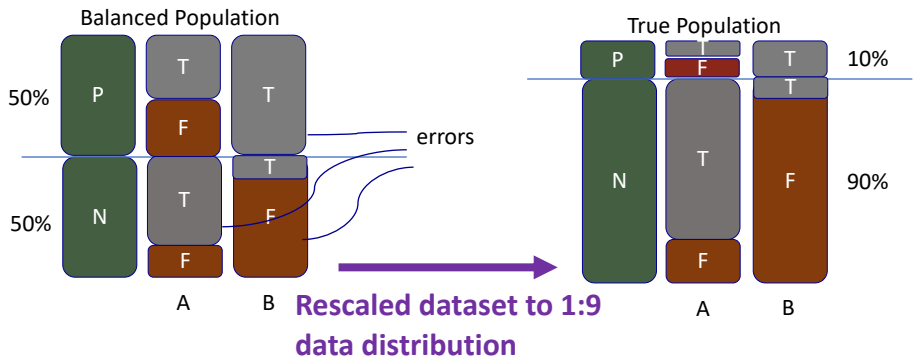
Confusion Matrix of B: evaluated with 1,000 datapoints

	positive	**negative**
True	500	50
False	0	450

- **Which model is better?**

Accuracy can mislead our understanding

Which model is better?



	positive	negative
True	250	450
False	250	50

Confusion Matrix of A

	positive	negative
True	500	50
False	0	450

Confusion Matrix of B

F-measure ($F1$ score)

- Summarizes confusion matrix
- True positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN)

	p	n
Y	TP	FN
N	FP	TN

- True positive rate (sensitivity) = $TP/(TP+FN)$
- False negative rate (miss rate) = $FN/(TP+FN)$
- F-measure = $2(precision \times recall)/(precision + recall)$
  - precision = $TP / (TP+FP)$
  - recall = $TP / (TP+FN)$
- Accuracy = $(TP + TN) / (P + N)$

Questions?

Questions?

CS435 Introduction to Big Data

Week 10 – Lecture 1 In-Memory Cluster Computing -Apache Spark

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- PA 2
  - Please schedule your demonstration
- PA 3
  - Group assignment (Work with your project team)
  - Using RDD should be enough
  - Read the description carefully
- Term project proposal

1

1

Topics discussed in the previous class

- Recommendation Systems: Collaborative Filtering
- Evaluation Methods

Topics to be covered

- PA 3
- Evaluation Methods – continued
- In-memory cluster computing
  - Apache Spark

PA 3: Estimating PageRank Values of Wikipedia Articles using Apache Spark

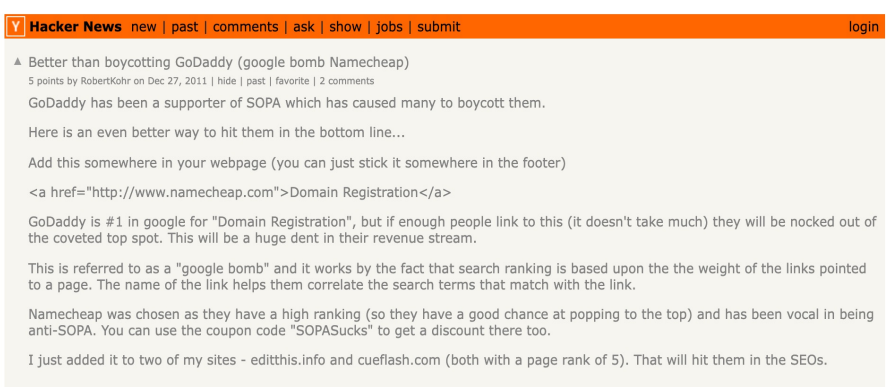
- Use the Taxation method

$$v' = \beta M v + (1-\beta) e / n$$

- Use the random walk style implementation
  - You do NOT need to use matrix-vector multiplication
  - There are many examples available
- Make your Wiki Bomb
  - Google Bomb (A.K.A. Google washing)
    - Making web site ranking highly in web search engine results for irrelevant, unrelated or off-topic search terms by linking heavily
  - Make a Wiki Bomb for the ”Rocky Mountain National Park” page
  - Modify the “link” file
  - Which pages should be modified?
  - This should be done “AFTER” you calculate the page rank values of all the pages

Google Bomb used to boycott GoDaddy.com

- SOPA (Stop Online Piracy Act), October 2011



Hacker News new | past | comments | ask | show | jobs | submit login

▲ Better than boycotting GoDaddy (google bomb Namecheap)

5 points by RobertKohr on Dec 27, 2011 | hide | past | favorite | 2 comments

GoDaddy has been a supporter of SOPA which has caused many to boycott them.

Here is an even better way to hit them in the bottom line...

Add this somewhere in your webpage (you can just stick it somewhere in the footer)

<a href="http://www.namecheap.com">Domain Registration</a>

GoDaddy is #1 in google for "Domain Registration", but if enough people link to this (it doesn't take much) they will be nocked out of the coveted top spot. This will be a huge dent in their revenue stream.

This is referred to as a "google bomb" and it works by the fact that search ranking is based upon the the weight of the links pointed to a page. The name of the link helps them correlate the search terms that match with the link.

Namecheap was chosen as they have a high ranking (so they have a good chance at popping to the top) and has been vocal in being anti-SOPA. You can use the coupon code "SOPASucks" to get a discount there too.

I just added it to two of my sites - editthis.info and cueflash.com (both with a page rank of 5). That will hit them in the SEOs.

Workflow for PA 3

- Calculate PageRank values for Wikipedia pages
- Create your Wiki Bomb for the "Rocky Mountain National Park"
- Simple searching mechanism for the search key "surfing"
  - You may create a command line environment
    - E.g. find "surfing"
    - Returns list of pages sorted on their pagerank values
  - Or, any search capability from text editor will also work

REVIEW

Large Scale Data Analytics

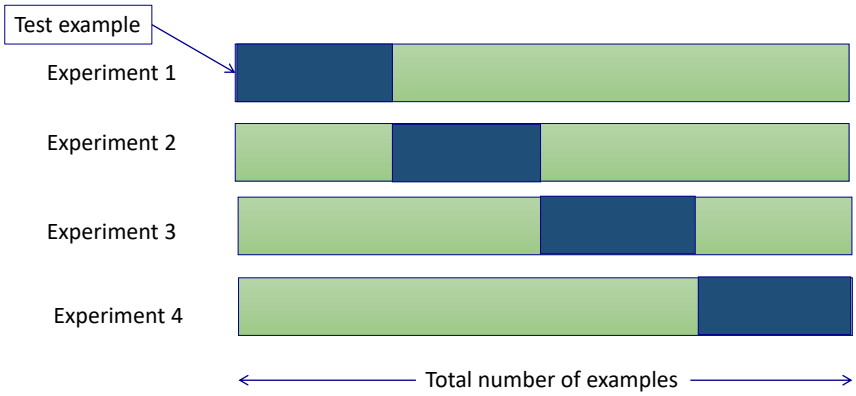
4. Evaluation and Validation

3. k-Fold Cross-validation

REVIEW

k-Fold Cross-validation

- Create a ***k*-fold partition** of the dataset
  - For each of the k experiments use ***K-1* folds for training**
    - The remaining one for testing



REVIEW

True error estimate

- k-fold cross validation is similar to random subsampling
- The advantage of k-Fold Cross validation
  - All the examples in the dataset are **eventually used for both training and testing**
- The true error is estimated as the average error rate

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

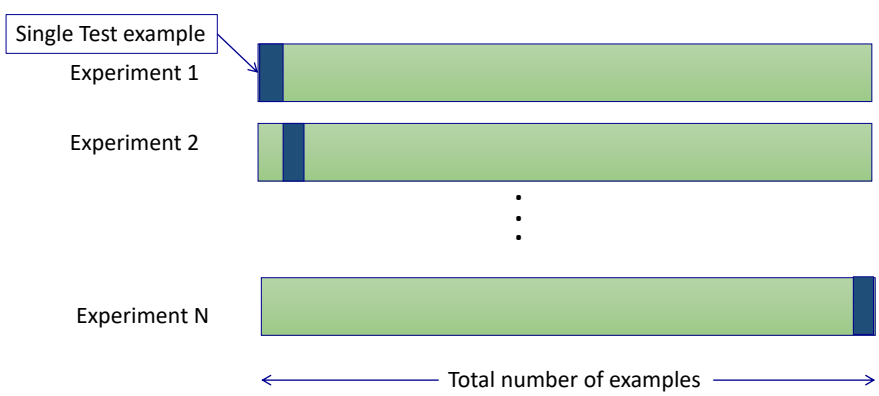
Large Scale Data Analytics

4. Evaluation and Validation

4. Leave-one-out Cross-validation

Leave-one-out Cross Validation

- Leave-one-out is the degenerate case of k-Fold Cross validation
  - k is chosen as the total number of examples
- For a dataset with N examples, perform N experiments
  - Use N-1 examples for training, the remaining example for testing



How many folds are needed?

- The choice of the number of folds depends on the size of the dataset
  - For large datasets, even 3-Fold Cross Validation will be quite accurate
  - For very sparse datasets, you may have to consider leave-one-out
    - To get maximum number of experiments

- A common choice for **k-fold Cross Validation is k=10**

Three-way data splits

- If model selection and true error estimates are computed **simultaneously**
  - The data needs to be divided into three disjoint sets
  - Training set
    - E.g. to find the optimal weights
  - Validation set
    - A set of examples used to tune the parameters of a model
    - To find the "optimal" number of hidden units or determine a stopping point for the back propagation algorithm
  - Test set
    - Used only to assess the performance of a fully-trained model

- **After assessing the final model with the test set, you must not further tune the model**

Accuracy measurements

- Mean Absolute Error (MAE)
  - Avg(Abs(Actual-Forecast))
  - $\frac{1}{n}\sum_{n}^{t=1}|A_t - F_t|$
- Root Mean Square Error (RMSE)
  - *Sqrt(Avg(Power(Actual -Forecast)))*
  - $\sqrt{\frac{1}{n}\sum_{n}^{t=1}|A_t - F_t|^2}$
  - Punishing larger errors (compared to MAE)

Large Scale Data Analytics

4. Evaluation and Validation

Classifiers

Classification

Mammals

Fishes

Birds

Insects

Plain Accuracy

- Classifier accuracy
  - General measure of classifier performance

Accuracy = (Number of correct decisions made) / (Total number of decision made)

- 1 – error rate

- Pros
  - **Very easy to measure**
- Cons
  - **Cannot consider realistic cases**

The Confusion Matrix

- A type of **contingency table**
- n classes
  - n x n matrix
    - The columns labeled with actual classes
    - The rows with predicted classes
- Separates out the decisions made by the classifier
  - How one class is being confused for another
  - Different sorts of errors may be dealt with separately

	p (positive)	**n (negative)**
Correct	True positive	True negative
Incorrect	False negative	False negative

Problems with Unbalanced Classes

- Consider a classification problem where one class is **rare**
  - Sifting through a large population of normal entities to find a relatively small number of unusual ones
  - Looking for **defrauded customers**, or **defective parts**
  - The class distribution is **unbalanced or skewed**

Confusion Matrix of A: evaluated with 1,000 datapoints

	positive	**negative**
Correct	250	450
Incorrect	250	50

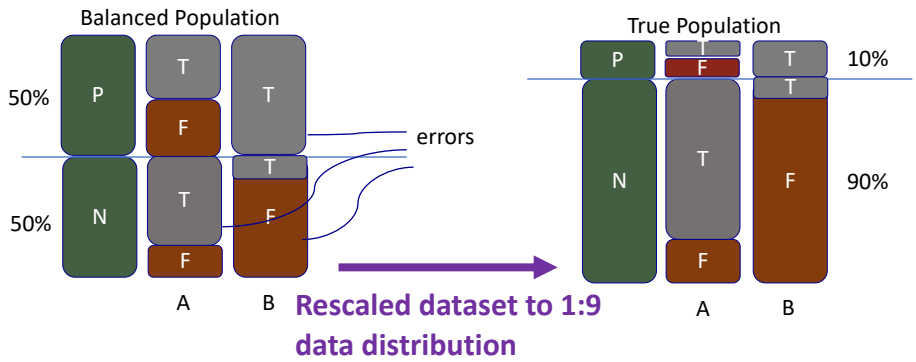
Confusion Matrix of B: evaluated with 1,000 datapoints

	positive	**negative**
Correct	500	50
Incorrect	0	450

- **Which model is better?**

Accuracy can mislead our understanding

Which model is better?



	positive	negative
True	250	450
False	250	50

Confusion Matrix of A

	positive	negative
True	500	50
False	0	450

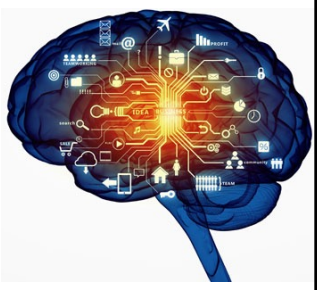
Confusion Matrix of B

F-measure ($F1$ score)

- Summarizes confusion matrix
- True positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN)

	p	n
Y	TP	FN
N	FP	TN

- True positive rate (sensitivity) = $TP/(TP+FN)$
- False negative rate (miss rate) = $FN/(TP+FN)$
- F-measure = $2(precision \times recall)/(precision + recall)$
  - precision = $TP / (TP+FP)$
  - recall = $TP / (TP+FN)$
- Accuracy = $(TP + TN) / (P + N)$



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Large Scale Data Analytics

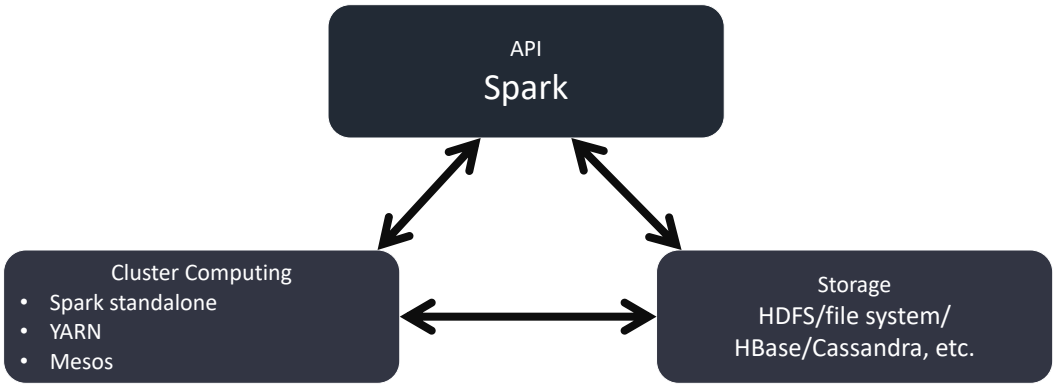
In-Memory Cluster Computing: Apache Spark

Introduction

This material is built based on

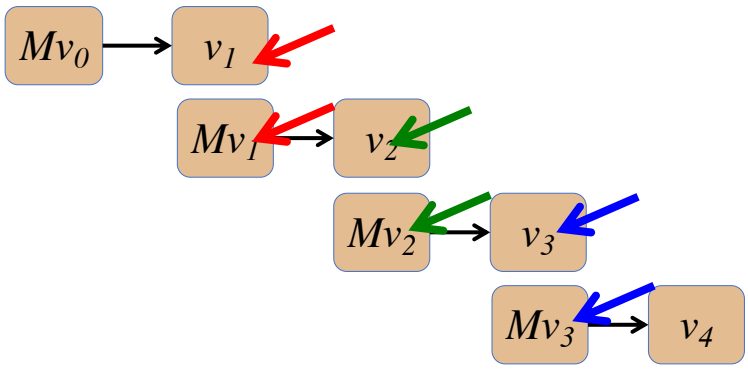
- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)
- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015
- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

Distributed processing with the Spark framework



Inefficiencies for emerging applications: (1) Data reuse

- Data reuse is common in many iterative machine learning and graph algorithms
  - e.g. PageRank, and K-means clustering



Inefficiencies for emerging applications: (2) Interactive data analytics

- User runs multiple ad-hoc queries on the **same subset** of the data

Existing/Previous approaches

- Hadoop
  - Writing output to an external stable storage system
    - e.g. HDFS
    - Substantial overheads due to data replication, disk I/O, and serialization
- Pregel
  - Iterative graph computations
- HaLoop
  - Iterative MapReduce interface
- Pregel/HaLoop support specific computation patterns
  - e.g. looping a series of MapReduce steps

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

RDD (Resilient Distributed Dataset)

RDD (Resilient Distributed Dataset)

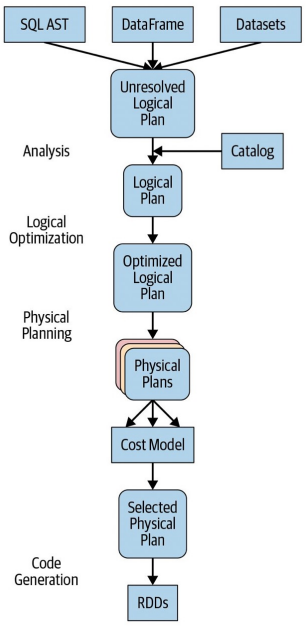
- **Read-only, memory resident partitioned collection of records**
  - A fault-tolerant collection of elements that can be operated on in parallel
- RDDs are the core unit of data in Spark
  - Most Spark programming involves performing operations on RDDs
- DataFrame API
  - Adding concepts with named columns and schemas
    - Table
    - Column has a specific data types: String, integer, array, map, real, date, timestamp..
- SparkSQL API
  - High-level structured functionality
  - Interoperable with other storage systems (e.g. Hbase)

Spark SQL

- Query support
  - Expressive query support
  - Easy to integrate with analytics

<Example>

```
SELECT State, Color, Count, sum(Count) AS
Total
FROM MNM_TABLE_NAME
GROUP BY State, Color, Count
ORDER BY Total DESC
```



RDD?

- Very fundamental data structure of Apache Spark
- Distribution, Partition, Fault tolerance, and scheduling computation
  - All automated by Apache Spark
  - With some user's configuration

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Simple Example with Spark RDD

- *Map vs. Filter vs. FlatMap*

Loading data from a file to an RDD

First RDD with data loaded from a text file

Your first RDD has been used here

Location of the raw data file

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts
        = textFile.flatMap(s -> Arrays.asList(s.split(" "))
                                .iterator())
          Add more operations here…
counts.saveAsTextFile("hdfs://...");
```


Example: Word Count

https://raw.githubusercontent.com/apache/spark/master/examples/src/main/java/org/apache/spark/examples/JavaWordCount.java

```
package org.apache.spark.examples;
import scala.Tuple2;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.sql.SparkSession;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;
```

A compiled representation of a regular expression

```
public final class JavaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");
  public static void main(String[] args) throws Exception {
    if (args.length < 1) {
      System.err.println("Usage: JavaWordCount <file>");
      System.exit(1);
    }
    SparkSession spark = SparkSession
        .builder()
        .appName("JavaWordCount")
        .getOrCreate();
```

Provide the app name

FlatMap: Each item can be mapped to one or more output items

Generating an RDD from the file

```
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

JavaRDD<String> words = lines.flatMap(s ->
    Arrays.asList(SPACE.split(s)).iterator());
    JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s,
1));
    JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
    List<Tuple2<String, Integer>> output = counts.collect();
    for (Tuple2<?,?> tuple : output) {
      System.out.println(tuple._1() + ": " + tuple._2());
    }
    spark.stop();
  }
}
```

Tokenizing a string

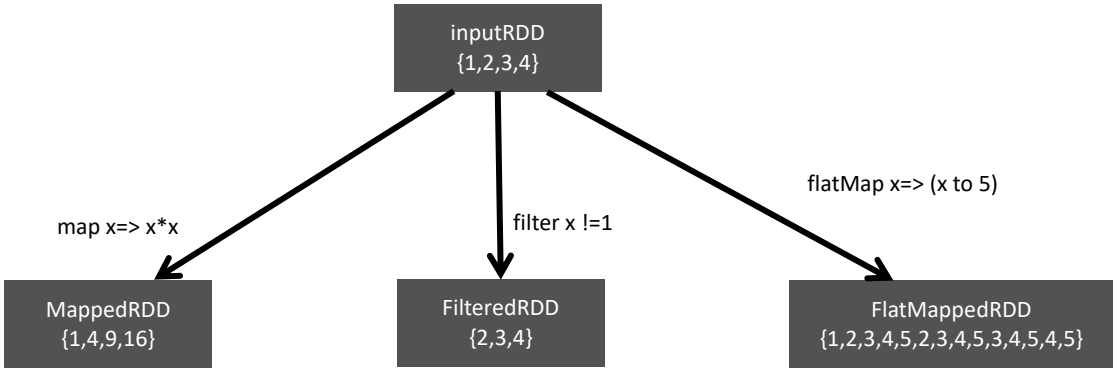
Bring them back to the driver program

Driver?
Program that creates the `SparkContext`, connecting to a given Spark Master.

Simple RDD operations
`map()` vs. `filter()` vs. `flatMap()` [1/3]

- The **`map()`** operation takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD
- The **`filter()`** operation takes in a function and returns an RDD that only has elements that pass the filter() function
- The **`flatMap()`** is similar to map, but each input item can be mapped to 0 or more output items (so *func*should return a Seq rather than a single item).

`map()` vs. `filter()` vs. `flatMap()` [2/3]



`map()` vs. `filter()` vs. `flatMap()` [3/3]

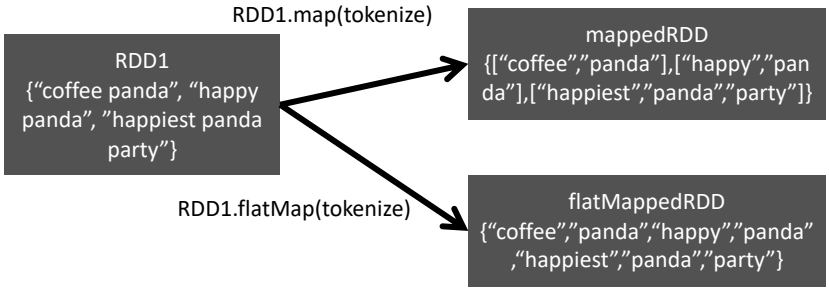
- `map()` that squares all of the numbers in an RDD

```
JavaRDD <Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4));
JavaRDD <Integer> result = rdd.map(new Function < Integer, Integer >() {
   public Integer call(Integer x) {
        return x*x;
   }
});


System.out.println(StringUtils.join(result.collect(),","));
```


`map()` vs. `flatMap()` with String [1/2]

- As results of **`flatMap()`**, we have an RDD of the elements
  - Instead of RDD of lists of elements



`map()` vs. `flatMap()` with String [2/2]

- Using `flatMap()` that splits lines to multiple words

```
JavaRDD < String > lines = sc.parallelize( Arrays.asList("hello world",
"hi"));

JavaRDD < String > words =
  lines.flatMap(new FlatMapFunction < String, String >() {
     public Iterable < String > call( String line) {
       return Arrays.asList(line.split(" "));
     }
  });

words.first();
// returns "hello"
```


Delivering output to the driver node `collect()` vs. `take(n)`

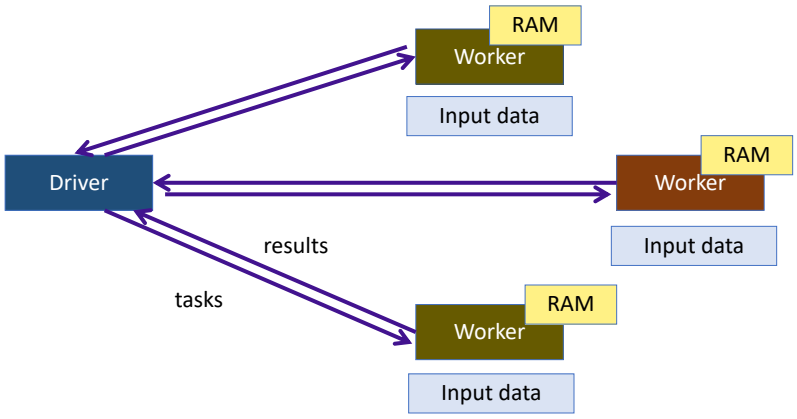
- collect()
  - Retrieves all the elements of the RDD/DataFrame/Dataset (from all nodes) to the driver node
- take()
  - returns `n` elements from the RDD and attempts to minimize the number of partitions it accesses
  - It may represent a biased collection
  - It does not return the elements in the order you might expect
  - Useful for unit testing
- What if a spark program does not use collect or take or an equivalent operation?
  - Local outputs will stay in the local nodes

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Understanding Spark RDD

RDDs in Spark: The Runtime



User's driver program launches multiple workers,
which read data blocks from a distributed file system and can persist computed RDD partitions in memory

Representing RDDs

- A set of partitions
  - Atomic pieces of the dataset
- **A set of dependencies on parent RDDs**
- **A function for computing** the dataset based on its parents
- Metadata about its partitioning scheme
- **Data placement**

Overview of RDD

- **Partitioning**
  - Users can specify the partitioning method across machines based on a key in each record
- **Lineage**
  - How it was derived from other dataset to compute its partitions from data in stable storage?
    - RDDs do not need to be materialized at all times
- **Persistence**
  - Users can indicate which RDDs they will reuse and the storage strategy

Spark Programming Interface to RDD: Type 1. Transformation [1/3]

- **"transformations"**
  - Operations that **create** RDDs
    - Return pointers to new RDDs
    - e.g. `map, filter, and join`
  - RDDs can only be created through **deterministic operations** on either
    - **Data in stable storage**
    - **Other RDDs**

Spark Programming Interface to RDD: Type 2: Action [2/3]

- **"actions"**
  - Operations that **return a value to the application** or **export data to a storage system**
    - e.g. `count`: returns the number of elements in the dataset
    - e.g. `collect`: returns the elements themselves
    - e.g. `save`: outputs the dataset to a storage system

Spark Programming Interface to RDD: Type 3: Persist [3/3]

- **"persist"**
  - Indicates which RDDs they want to **reuse in future operations**
  - Spark keeps persistent RDDs **in memory** by default
  - If there is not enough RAM
    - It can spill them to disk
  - Users are allowed to,
    - store the RDD only on disk
    - replicate the RDD across machines
    - specify a persistence priority on each RDD

Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an **action**
  - Spark internally records metadata to indicate that this operation has been requested
- **Loading data from files into an RDD** is lazily evaluated
- Reduces the number of passes it has to take over our data by grouping operations together

Example: Console Log Mining [1/3]

- Suppose that a web service is experiencing errors and an operator wants to search terabytes of logs in the Hadoop file system (HDFS) to find the cause
- The user load **the error messages** from the logs into the RAM across a set of nodes and query them interactively

```
lines = spark.textFile("hdfs://…")
errors=lines.filter(_.startsWith("ERROR"))
errors.persist()
```

No work has been performed
User can use the RDD in actions

52

52

Example: Console Log Mining [2/3]

- Users can perform further transformations and actions on the RDD

```
//To count number of error messages
errors.count()

//Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

//Return the time fields of errors mentioning
//HDFS as an array (assuming time is field
//number 3 in a tab-separated format
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

After the first **action** involving `errors` runs, Spark will store the partitions of errors in memory.

53

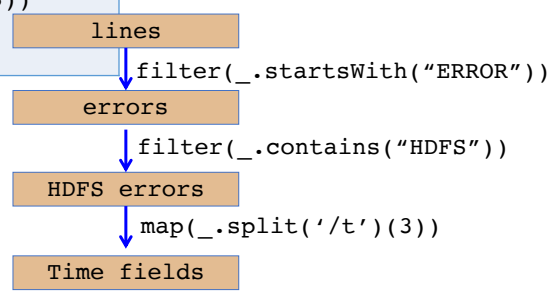
53

Example: Console Log Mining [3/3]

Spark code

```
lines = spark.textFile("hdfs://…")
errors=lines.filter(_.startsWith("
ERROR"))
errors.persist()
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

Lineage graph



If a partition of `errors` is lost Spark rebuilds it by applying a filter on only the corresponding partition of `lines`

Benefits of RDDs as a distributed memory abstraction [1/7]

- RDD vs. Distributed Shared Memory (DSM)?
  - How does RDD work differently compared to DSM?
  - Write/Consistency/Fault-Recovery mechanism/Straggler mitigation
- RDDs can only be created ("written") through **coarse-grained transformations**
  - Coarse-grained transformations are applied over an entire dataset
  - Reads on RDDs can still be fine-grained
    - A large read-only lookup table
  - Applications perform bulk writes
- **More efficient fault tolerance**
  - Lineage based bulk recovery

Benefits of RDDs as a distributed memory abstraction [2/7]

RDD vs. DSM- Read operation

- RDD
  - Either coarse grained or fine grained
- DSM
  - The read operation in Distributed shared memory is fine-grained

Benefits of RDDs as a distributed memory abstraction [3/7]

RDD vs. DSM- Write operation

- RDD
  - The write operation in RDD is coarse grained
- DSM
  - The write operation in Distributed shared memory is fine-grained

Benefits of RDDs as a distributed memory abstraction [4/7]

RDD vs. DSM- Consistency

- RDD
  - RDD is immutable in nature
    - Any changes on RDD is permanent
  - The level of consistency is high
- DSM
  - If the programmer follows the rules, the memory will be consistent and the results of memory operations will be predictable

Benefits of RDDs as a distributed memory abstraction [5/7]

RDD vs. DSM- Fault-Recovery Mechanism

- RDD
  - The lost data can be easily recovered in Spark RDD using lineage graph at any moment
  - For each transformation, new RDD is formed
  - RDDs are immutable
- DSM
  - Fault tolerance is achieved by a checkpointing technique which allows applications to roll back to a recent checkpoint rather than restarting

Benefits of RDDs as a distributed memory abstraction [6/7]

RDD vs. DSM- Straggler Mitigation

- Stragglers
  - Nodes taking more time to complete than their peers
  - Due to load imbalance, I/O blocks, garbage collections, etc.
- RDD
  - **Creates backup copies of slow tasks**
    - without accessing the same memory
- DSM
  - It is quite difficult to achieve straggler mitigation

Benefits of RDDs as a distributed memory abstraction [7/7]

- Runtime can schedule tasks **based on data locality**
  - To improve performance
- RDDs degrade gracefully when there is **insufficient memory**
  - **Partitions that do not fit in the RAM are stored on disk**

Applications not suitable for RDDs

- RDDs are best suited for **batch applications** that apply the same operations to **all elements of a dataset**
  - Steps are managed by **lineage graph** efficiently
  - Recovery is managed effectively
- RDDs would not be suitable for applications
  - Making asynchronous fine-grained updates to shared state
  - e.g. a **storage system for a web application** or an incremental web crawler

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

RDD in Spark

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

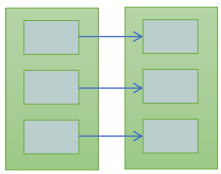
RDD Dependency in Spark

Dependency between RDDs [1/4]

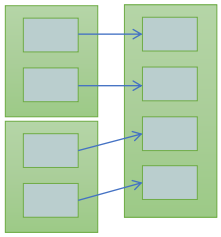
- **Narrow** dependency
- **Wide** dependency

Dependency between RDDs [2/4]

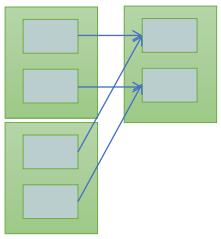
- **Narrow** dependency
  - Each partition of the parent RDD is used by **at most one partition** of the child RDD



map, filter



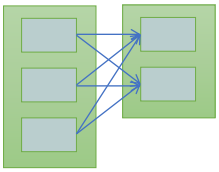
union



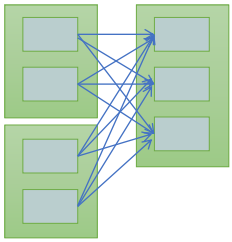
Join with inputs co-partitioned

Dependency between RDDs [3/4]

- **Wide** dependency
  - **Multiple child partitions** may depend on a single partition of parent RDD



groupByKey



Join with inputs not co-partitioned

Dependency between RDDs [4/4]

- Narrow dependency
  - Pipelined execution on one cluster node
  - e.g. a map followed by a filter
  - Failure recovery is more straightforward
- Wide dependency
  - Requires data from all parent partitions to be available and to be shuffled across the nodes
  - Failure recovery could involve a large number of RDDs
    - Complete re-execution may be required

Questions?

CS435 Introduction to Big Data

Week 10 – Lecture 2 In-Memory Cluster Computing -Apache Spark

Sangmi Lee Pallickara

Computer Science, Colorado State University

FAQs

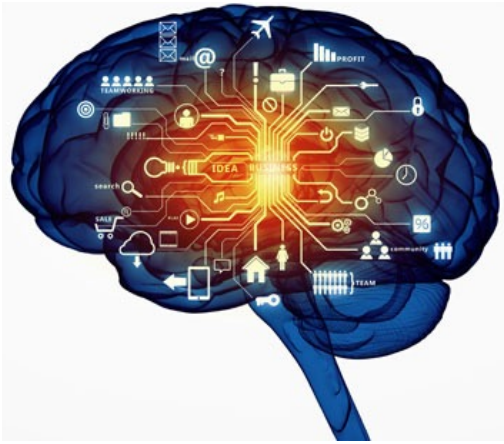
- PA 3
  - Group assignment (Work with your project team)
  - Using RDD should be enough
  - Read the description carefully

Topics discussed in the previous class

- PA 3
- Evaluation Methods – continued
- In-memory cluster computing
  - Apache Spark

Topics to be covered

- In-memory cluster computing - Continued
  - Apache Spark



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

This material is built based on

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)

- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015

- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

REVIEW

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Simple Example with Spark RDD

- *Map vs. Filter vs. FlatMap*

REVIEW

Loading data from a file to an RDD

First RDD with data loaded from a text file

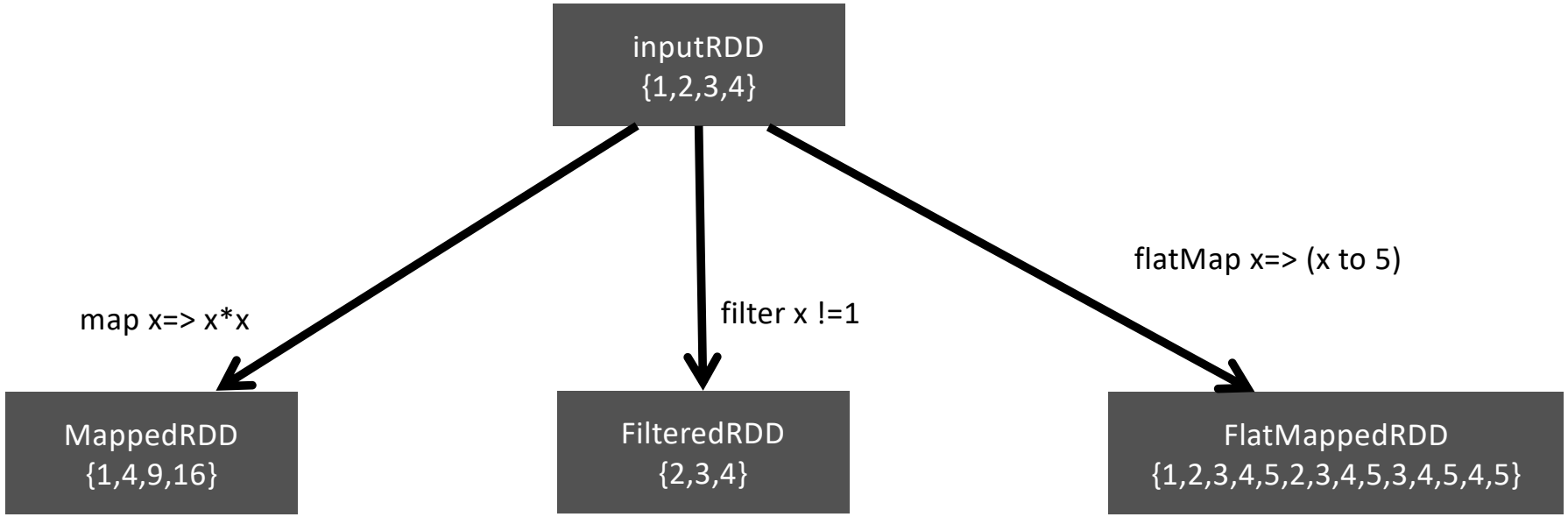
Your first RDD has been used here

Location of the raw data file

```
JavaRDD<Str g> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts
        = textFile.flatMap(s -> Arrays.asList(s.split(" "))
                                .iterator())
          Add more operations here…
counts.saveAsTextFile("hdfs://...");
```

`map()` vs. `filter()` vs. `flatMap()` [2/3]

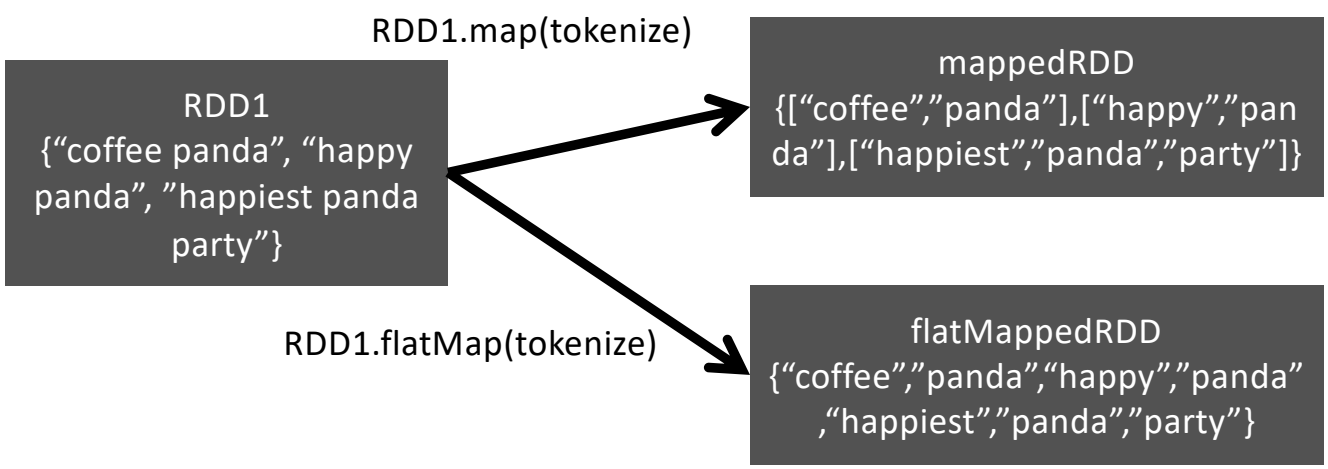
REVIEW



REVIEW

`map()` vs. `flatMap()` with String [1/2]

- As results of **`flatMap()`**, we have an RDD of the elements
  - Instead of RDD of lists of elements



REVIEW

Delivering output to the driver node `collect() vs. take(n)`

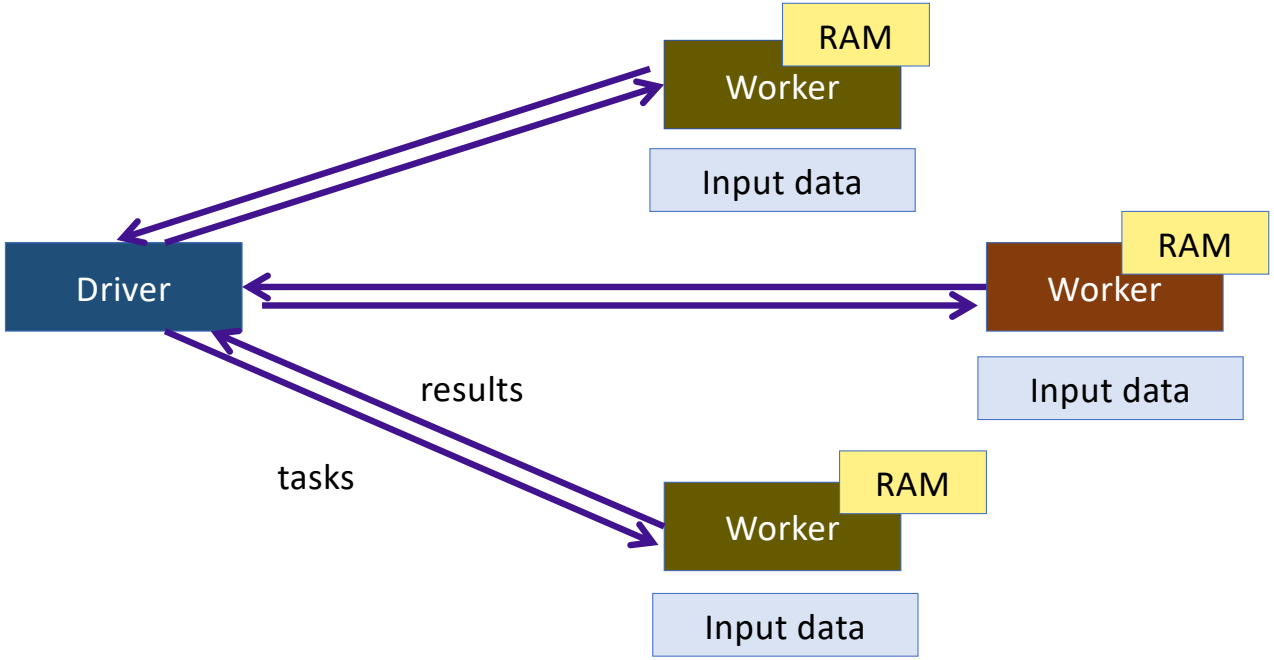
- collect()
  - Retrieves all the elements of the RDD/DataFrame/Dataset (from all nodes) to the driver node
- take()
  - returns `n` elements from the RDD and attempts to minimize the number of partitions it accesses
  - It may represent a biased collection
  - It does not return the elements in the order you might expect
  - Useful for unit testing
- What if a spark program does not use collect or take or an equivalent operation?
  - Local outputs will stay in the local nodes

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Understanding Spark RDD

RDDs in Spark: The Runtime



User's driver program launches multiple workers,
which read data blocks from a distributed file system and can persist computed RDD partitions in memory

Representing RDDs

- A set of partitions
  - Atomic pieces of the dataset
- **A set of dependencies on parent RDDs**
- **A function for computing** the dataset based on its parents
- Metadata about its partitioning scheme
- **Data placement**

Overview of RDD

- **Types**
  - Standard RDD
    - Single column table (list)
  - Pair RDD
    - Key-value pairs
- **Partitioning**
  - Users can specify the partitioning method across machines based on a key in each record
- **Persistence**
  - Users can indicate which RDDs they will reuse and the storage strategy
- **Lineage**
  - How was this RDD derived from other dataset to compute its partitions from data in stable storage?
    - RDDs do not need to be materialized at all times

Spark Programming Interface to RDD

- **Spark does NOT allow any program to modify already existing RDDs**

- **Spark Operations (Programming interface to RDD)**
  - **Transformation**
  - **Action**
  - **Persistence**

Spark Programming Interface to RDD: Type 1. Transformation [1/3]

- **"transformations"**
  - Operations that **create** RDDs
    - Return pointers to new RDDs
    - e.g. `map, filter, and join`
  - RDDs can only be created through **deterministic operations** on either
    - **Data in stable storage**
    - **Other RDDs**

Spark Programming Interface to RDD: Type 2: Action [2/3]

- **"actions"**
  - Operations that **return a value to the application** or **export data to a storage system**
    - e.g. `count`: returns the number of elements in the dataset
    - e.g. `collect`: returns the elements themselves
    - e.g. `save`: outputs the dataset to a storage system

Spark Programming Interface to RDD:
Type 3: Persist [3/3]

- **"persist"**
  - Indicates which RDDs they want to **reuse in future operations**
  - Spark keeps persistent RDDs **in memory** by default
  - If there is not enough RAM
    - It can spill them to disk
  - Users are allowed to,
    - store the RDD only on disk
    - replicate the RDD across machines
    - specify a persistence priority on each RDD

Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an **action**
  - Spark internally records metadata to indicate that this operation has been requested

- **Loading data from files into an RDD** is lazily evaluated

- Reduces the number of passes it has to take over our data by grouping operations together

Example: Console Log Mining [1/3]

- Suppose that a web service is experiencing errors and an operator wants to search terabytes of logs in the Hadoop file system (HDFS) to find the cause

- The user load **the error messages** from the logs into the RAM across a set of nodes and query them interactively

Standard RDD

Standard RDD

This RDD will stored in the memory until the end of the current job

```
lines = spark.textFile("hdfs://...")
errors=lines.filter(_.startsWith("ERROR"))
errors.persist()
//To count number of error messages
errors.count()
```

No work has been performed until this point: No "ACTION" so far

The first "Action" operation
Evaluation of all the above lines will be performed now

Example: Console Log Mining [2/3]

- Users can perform further transformations and actions on the RDD

```
//Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

//Return the time fields of errors mentioning
//HDFS as an array (assuming time is field
//number 3 in a tab-separated format
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

This is another transformation. No immediate evaluation

This is another transformation. No immediate evaluation

This is another transformation. No immediate evaluation

The second "Action" operation
Evaluation of all the above lines will be performed now

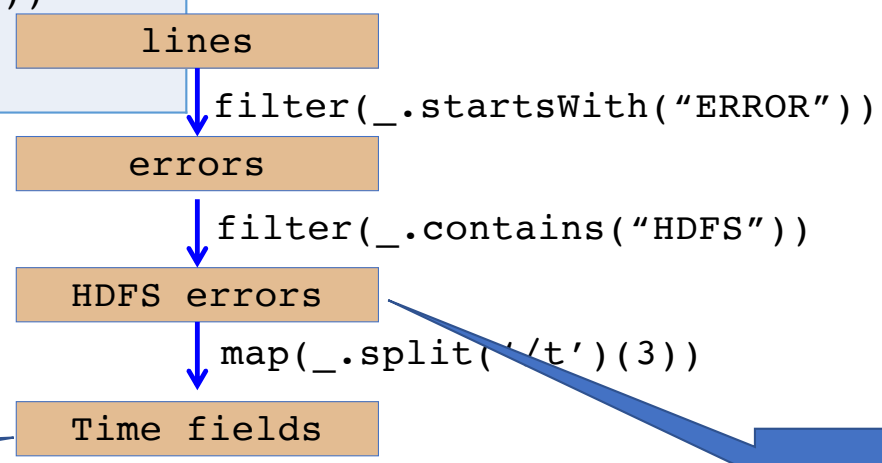
After the first **action** involving `errors` runs, Spark will store the partitions of errors in memory.

Example: Console Log Mining [3/3]

Key operations from the previous Spark code

```
lines = spark.textFile("hdfs://…")
errors=lines.filter(_.startsWith("
ERROR"))
errors.persist()
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

Lineage graph



If a partition of `errors` is lost Spark rebuilds it by applying a filter on only the corresponding partition of `lines`

Internal RDD

Internal RDD

RDD Lineage

- A graph of all the parent RDDs of an RDD
  - A graph of what transformations need to be executed after an action has been called
- A.K.A. Logical execution plan
  - starts with the earliest RDDs (those with no dependencies on other RDDs or reference cached data) and ends with the RDD that produces the result of the action that has been called to execute
- **Fault tolerance with lineage**
  - If any loss occurs, Spark can recover the failure using the lineage graph
    - Finds an available RDD and repeat the computations

Why Key/Value Pairs?

- Pair RDDs
  - Spark provides special operations on RDDs containing key/value pairs
  - Pair RDDS allow you to **act on each key in parallel** or **regroup data across the network**

- e.g.
  - `reduceByKey()`
    - **Aggregates data separately for each key**

  - `join()`
    - Merge two RDDs by grouping elements with the **same key**

Creating Pair RDDs

-- using the first word as the key

- Running `map()` function
  - A transformation that passes each dataset element through a function and returns a new RDD representing the results
  - Returns **key/value pairs**

```
PairFunction < String, String, String > keyData =
   new PairFunction < String, String, String >() {
     public Tuple2 < String, String > call(String x) {
       return new Tuple2(x.split(" ")[0], x);
   }
};

JavaPairRDD <String,String> pairs = lines.mapToPair(keyData);
```

PairFunction: A function that returns key-value pairs (Tuple2<K, V>), and can be used to construct PairRDDs
Tuple2: A function that creates a tuple

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Transformations on Standard RDDs and Pair RDDs

Transformations on an RDD (e.g. rdd={ 1, 3, 6}, other={2,3,4})

Function	purpose	Example	Result
map(*func*)	Return a new RDD formed by passing each element through a function func	`rdd.map(x*x)`	{1, 9, 36}
filter(*func*)	Select element(s) on which the func returns true	`rdd.filter(x>1)`	{3, 6}
flatMap(*func*)	Similar to map but returns 0 or more outputs	`rdd.filter(2..x)`	{2,3,2,3,4,5,6}
union(*otherDataset*)	Create a union set of two RDDs	`rdd.union(other)`	{1,2,3,4,6}
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD	`Heavy initialization such as DB connection`	Single DB connection per partition
intersection(*otherDataset*)	Intersecting two RDDs	`rdd.intersection (other)`	{3}

Pair RDDs are still RDDs

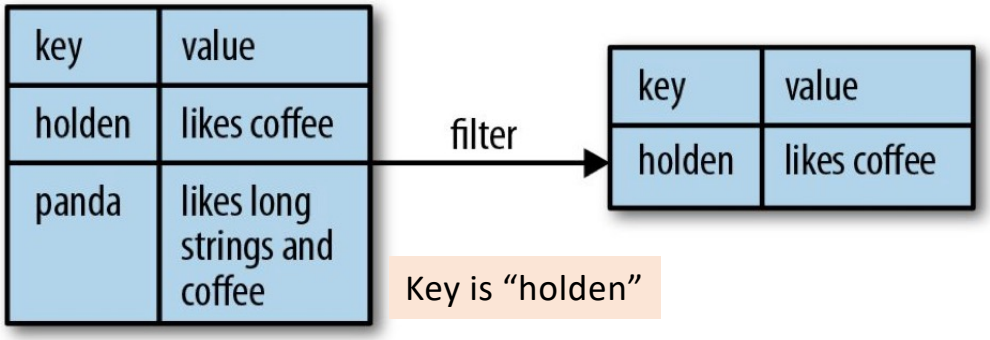
- Supports the **same functions as RDDs**

e.g. filtering out the long words

```
Function < Tuple2 < String, String >, Boolean > longWordFilter =
   new Function < Tuple2 < String, String >, Boolean >() {
     public Boolean call( Tuple2 < String, String > keyValue) {
       return (keyValue._2().length() < 20);
     }
};

JavaPairRDD < String, String > result =
                                  pairs.filter(longWordFilter);
```

`Filter` on Pair RDDs



Transformations on a pair RDD (e.g. rdd={(1, 2), (3, 4), (3, 6)})

Function	purpose	Example	Result
`mapValues(func)`	Apply a function to each value of a pair RDD **without changing the key**	`rdd.mapValues(x=>x+1)`	`{(1,3),(3,5),(3,7)}`
`flatMapValues(func)`	Apply a function that returns an **iterator**	`rdd.flatMapValues(x=>(x to 5)`	`{(1,2),(1,3),(1,4),(1,5),(3,4),(3,5)}`
`keys()`	Return an RDD of just the **keys**	`rdd.keys()`	`{1,3,3}`
`values()`	Return an RDD of just the **values**	`rdd.values()`	`{2,4,6}`
`sortByKey()`	Return an RDD **sorted by the key**	`rdd.sortByKey()`	`{(1,2),(3,4),(3,5)}`

Transformations on a pair RDD (e.g. rdd={(1, 2), (3, 4), (3, 6)})

Function	purpose	Example	Result
`reduceByKey()`	**Combine** values with the **same key**	`rdd.reduceByKey( (x, y) => x + y)`	{(1,2),(3,10)}
`groupByKey()`	**Group** values with the same key	`rdd.groupByKey()`	{(1,[2]), (3,[4,6])}
`combineByKey( createCombiner, mergeValue, mergeCombiners, partitioner)`	Combine values with the same key using a different result type	Slides: 44	

Transformations on two pair RDDs (e.g. rdd={(1, 2), (3, 4), (3, 6)}) other={(3,9)}

Function	purpose	Example	Result
`subtractByKey`	**Remove elements with a key present in the other RDD**	`rdd.subtractByKey(other)`	`{(1,2)}`
`join`	**Inner join**	`rdd.join(other)`	`{(3,(4,9)),(3,(6,9))}`
`rightOuterJoin`	Perform a join where the **key must be present in the other RDD**	`rdd.rightOuterJoin(other)`	`(3,(Some(4),9)),(3,(Some(6),9))}`
`leftOuterJoin()`	Perform a join where the key must be present in the first RDD	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3,(4,Some(9))), (3,(6,Some(9)))}`
`coGroup`	Group data from both RDDs sharing the same key	`rdd.cogroup(other)`	`{(1,([2],[])),(3,([4,6],[9]))}`

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Aggregations with PairRDDs

Aggregations with PairRDDs

- **Aggregate** statistics across all elements **with the same key**
- Example:
- **`reduceByKey()`**
  - Similar to `reduce()`
  - Takes a function and use it to combine values
  - Runs several parallel reduce operations
    - One for each key in the dataset
    - Each operations combines values that have the same keys
  - `reduceByKey()` is **not implemented as an action that returns a value to the user program**
    - **There can be a large number of keys**
    - It **returns a new RDD** consisting of each key and the reduced value for that key
  - **Therefore, `reduceByKey()` is a transformation**

Example

- Key-value pairs are represented using the `Tuple2` class

```
JavaRDD<String> lines = sc.textFile("data.txt");

JavaPairRDD<String, Integer> pairs =
                    lines.mapToPair(s -> new Tuple2(s, 1));

JavaPairRDD<String, Integer> counts =
                    pairs.reduceByKey((a, b) -> a + b);
```

Revisit the word count example

```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
```

`combineByKey()`

- The most **general of the per-key aggregation functions**
  - Most of the other per-key combiners are implemented using it
- Similar to reduceByKey()
  - More flexible
  - Allows the user to return values that are not the same type as the input data
- `createCombiner()`
  - If `combineByKey()` finds a new key
  - This happens the first time a key is found in each **partition**, rather than only the first time the key is found in the RDD
- `mergeValue()`
  - If it is not a new value in that partition
- `mergeCombiners()`
  - Merging the results from each partition

Per-key average using `combineByKey()` [1/2]

```
public static class AvgCount implements Serializable {
   public AvgCount(int total, int num) {
        total_ = total;
        num_ = num; }
   public int total_;
   public int num_;
   public float avg() {
        return total_ / (float) num_;
   }
}

Function <Integer, AvgCount > createAcc = new
   Function < Integer, AvgCount >() {
     public AvgCount call(Integer x) {
       return new AvgCount(x, 1); }
   };

Function2 < AvgCount, Integer, AvgCount > addAndCount = new
   Function2 < AvgCount, Integer, AvgCount >() {
      public AvgCount call(AvgCount a, Integer x) {
          a.total_ + = x;
          a.num_ + = 1;
```

Per-key average using `combineByKey()` [2/2]

```
      return a; }
};
Function2 < AvgCount, AvgCount, AvgCount > combine =
   new Function2 < AvgCount, AvgCount, AvgCount >() {
   public AvgCount call( AvgCount a, AvgCount b) {
      a.total_ + = b.total_;
      a.num_ + = b.num_;
      return a; }
};

AvgCount initial = new AvgCount(0,0);
JavaPairRDD < String, AvgCount > avgCounts =
nums.combineByKey(createAcc, addAndCount, combine);
Map < String, AvgCount > countMap = avgCounts.collectAsMap();
for (Entry < String, AvgCount > entry : countMap.entrySet()) {
        System.out.println( entry.getKey() + ":"
                            + entry.getValue().avg());
}
```

Tuning the level of parallelism

- When performing aggregations or grouping operations, we can ask Spark to use a specific number of partitions
  - `reduceByKey((x, y) = > x + y, 10)`

- **`repartition()`**
  - Shuffles the data across the network to create a new set of partitions
  - Expensive operation
  - Optimized version: **`coalesce()`**
    - Reduces data movement
    - A subset of current nodes will be kept

`groupByKey()`

- Group our data using the key in our RDD
- On an RDD consisting of keys of type `K` and values of type `V`
  - Results will be RDD of type `[K, Iterable[V]]`

- `cogroup()`
  - Grouping data from **multiple RDDs**
  - Over two RDDs sharing the same key type, `K`, with the respective value types `V` and `W` gives us back `RDD[(K, (Iterable[V], Iterable[W]))]`

joins

- **Join operations generates a new RDD**
  - **Join is a transformation operation**

- Inner join
  - Only keys that are present in both pair RDDs are output

- `leftOuterJoin(other)` and `rightOuterJoin(other)`
  - One of the pair RDDs can be missing the key

- `leftOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the source RDD

- `rightOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the other RDD

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Key-Value pairs : Actions available on RDDs

Actions on standard RDDs(example: rdd={1,3,4})

Function	Description	Example	Result
`reduce(func)`	Aggregate the elements of the dataset using a function *func*	`rdd.reduce(x+y)`	`{8}`
`collect()`	**Return all the elements**	`rdd.collect()`	`{1,3,4}`
`count()`	**Return the number of elements**	`rdd.count()`	`3`
`first()`	**Return the first elements**	`rdd.first()`	`1`
`take(n)`	**Return first n elements**	`rdd.take(1)`	`1`
`saveAsTextFile(path)`	Write the elements of the dataset as a text file	`rdd.saveAsTextFile (file_path)`	`Text file`

Actions on pair RDDs(example({(1,2),(3,4),(3,6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key	`rdd.countByKey()`	`{(1,1),(3,2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup at the driver	`rdd.collectAsMap()`	`Map{(1,2),(3,4),(3,6)}`
`lookup(key)`	Return all values associated with the provided key	`rdd.lookup(3)`	`[4,6]`

Revisit: Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an action
  - Spark internally records metadata to indicate that this operation has been requested
- **Loading data from files into an RDD** is lazily evaluated
- Reduces the number of passes it has to take over our data by grouping operations together

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

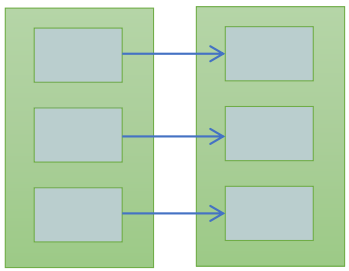
RDD Dependency in Spark

Dependency between RDDs [1/4]

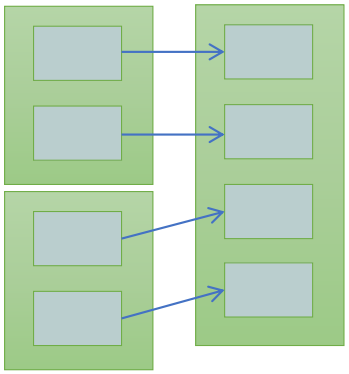
- **Narrow** dependency
- **Wide** dependency

Dependency between RDDs [2/4]

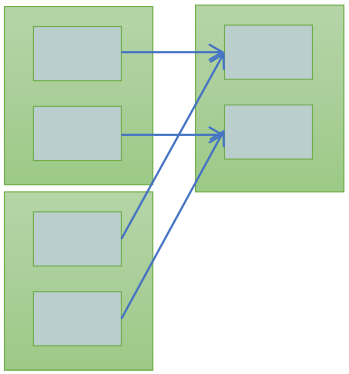
- **Narrow** dependency
  - Each partition of the parent RDD is used by **at most one partition** of the child RDD



map, filter



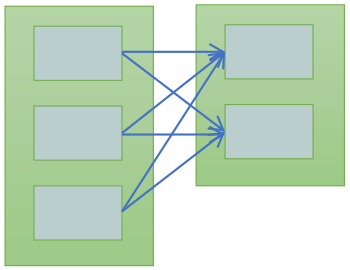
union



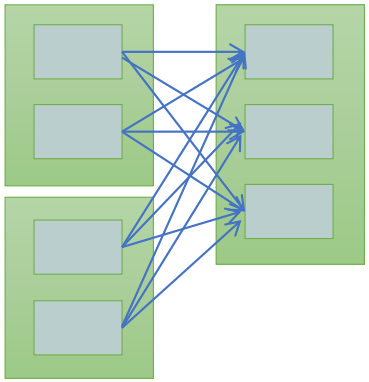
Join with inputs co-partitioned

Dependency between RDDs [3/4]

- **Wide** dependency
  - **Multiple child partitions** may depend on a single partition of parent RDD



groupByKey



Join with inputs not co-partitioned

Dependency between RDDs [4/4]

- Narrow dependency
  - Pipelined execution on one cluster node
  - e.g. a map followed by a filter
  - Failure recovery is more straightforward
- Wide dependency
  - Requires data from all parent partitions to be available and to be shuffled across the nodes
  - Failure recovery could involve a large number of RDDs
    - Complete re-execution may be required

Interface used to represent RDDs in Spark

- `partitions()`
  - Returns a list of partition objects
- `preferredLocations(p)`
  - List nodes where partition p can be accessed faster due to data locality
- `dependencies()`
  - Return a list of dependencies
- `iterator(p, parentIters)`
  - Compute the elements of partition p given iterators for its parent partitions
- `partitioner()`
  - Return metadata specifying whether the RDD is hash/range partitioned

CS435 Introduction to Big Data

Week 11 – Lecture 1 In-Memory Cluster Computing -Apache Spark

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- Resubmission
  - PA0 ,1, 2, and 3(will be applicable after the submission deadline)
  - Deadline: 5PM 11/30/2021
  - Submission: via email (to cs435@cs.colostate.edu)
  - Score deduction: 25% of the score you have received
  - What if I already submitted and my homework has been graded?
    - Any higher score will be selected
- Term project proposal
  - Comments/feedbacks are sent via email
    - If your team did not get any email from me, let me know
  - Unless I have said that your team should change the topic, please continue working on the current topic
  - If your team wants to change the topic, let me know!

1

1

FAQs - continued

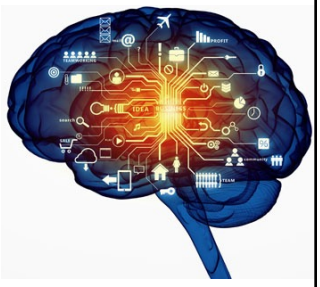
- Final Term Project software submission and demonstration
  - Visit your canvas page
  - Sign up for the software demonstration
    - Link included in the canvas page
- Final Term Project presentation sign up
  - Visit your canvas page
  - Sign up for the software demonstration
    - Link included in the canvas page

Topics discussed in the previous class

- In-memory cluster computing - Continued
  - Apache Spark
  - Introduction to the Apache Spark's Transformations and Actions
  - Lazy evaluation
  - RDD Lineage
  - Transformations (for standard RDD and pair RDD)

Topics to be covered

- In-memory cluster computing - Continued
  - Apache Spark
  - Aggregation
  - Actions



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

This material is built based on

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)
- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015
- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Aggregations with PairRDDs

Aggregations with PairRDDs

- **Aggregate** statistics across all elements **with the same key**
- Example:
- **`reduceByKey()`**
  - Similar to `reduce()`
  - Takes a function and use it to combine values
  - Runs several parallel reduce operations
    - One for each key in the dataset
    - Each operations combines values that have the same keys
  - `reduceByKey()` is **not implemented as an action that returns a value to the user program**
    - **There can be a large number of keys**
    - It **returns a new RDD** consisting of each key and the reduced value for that key
  - **Therefore, `reduceByKey()` is a transformation**

Example

- Key-value pairs are represented using the `Tuple2` class

```
JavaRDD<String> lines = sc.textFile("data.txt");

JavaPairRDD<String, Integer> pairs =
                    lines.mapToPair(s -> new Tuple2(s, 1));

JavaPairRDD<String, Integer> counts =
                    pairs.reduceByKey((a, b) -> a + b);
```

Revisit the word count example

```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
```


`combineByKey()`

- The most **general of the per-key aggregation functions**
  - Most of the other per-key combiners are implemented using it
- Similar to reduceByKey()
  - More flexible
  - Allows the user to return values that are not the same type as the input data
- `createCombiner()`
  - If `combineByKey()` finds a new key
  - This happens the first time a key is found in each **partition**, rather than only the first time the key is found in the RDD
- `mergeValue()`
  - If it is not a new value in that partition
- `mergeCombiners()`
  - Merging the results from each partition

Per-key average using `combineByKey()` [1/2]

```
public static class AvgCount implements Serializable {
   public AvgCount(int total, int num) {
        total_ = total;
        num_ = num; }
   public int total_;
   public int num_;
   public float avg() {
        return total_ / (float) num_;
   }
}

Function <Integer, AvgCount > createAcc = new
   Function < Integer, AvgCount >() {
     public AvgCount call(Integer x) {
       return new AvgCount(x, 1); }
   };

Function2 < AvgCount, Integer, AvgCount > addAndCount = new
   Function2 < AvgCount, Integer, AvgCount >() {
      public AvgCount call(AvgCount a, Integer x) {
          a.total_ + = x;
          a.num_ + = 1;
```


Per-key average using `combineByKey()` [2/2]

```
      return a; }
};
Function2 < AvgCount, AvgCount, AvgCount > combine =
   new Function2 < AvgCount, AvgCount, AvgCount >() {
   public AvgCount call( AvgCount a, AvgCount b) {
      a.total_ + = b.total_;
      a.num_ + = b.num_;
      return a; }
};

AvgCount initial = new AvgCount(0,0);
JavaPairRDD < String, AvgCount > avgCounts =
nums.combineByKey(createAcc, addAndCount, combine);
Map < String, AvgCount > countMap = avgCounts.collectAsMap();
for (Entry < String, AvgCount > entry : countMap.entrySet()) {
        System.out.println( entry.getKey() + ":"
                            + entry.getValue().avg());
}
```

Tuning the level of parallelism

- When performing aggregations or grouping operations, we can ask Spark to use a specific number of partitions
  - `reduceByKey((x, y) = > x + y, 10)`

- **`repartition()`**
  - Shuffles the data across the network to create a new set of partitions
  - Expensive operation
  - Optimized version: **`coalesce()`**
    - Reduces data movement
    - A subset of current nodes will be kept

`groupByKey()`

- Group our data using the key in our RDD
- On an RDD consisting of keys of type `K` and values of type `V`
  - Results will be RDD of type **`[K, Iterable[V]]`**

- `cogroup()`
  - Grouping data from **multiple RDDs**
  - Over two RDDs sharing the same key type, `K`, with the respective value types `V` and `W` gives us back `RDD[(K, (Iterable[V], Iterable[W]))]`

joins

- **Join operations generates a new RDD**
  - **Join is a transformation operation**
- Inner join
  - Only keys that are present in both pair RDDs are output
- `leftOuterJoin(other)` and `rightOuterJoin(other)`
  - One of the pair RDDs can be missing the key
- `leftOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the source RDD
- `rightOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the other RDD

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Key-Value pairs : Actions available on RDDs

Actions on standard RDDs(example: rdd={1,3,4})

Function	Description	Example	Result
`reduce(func)`	Aggregate the elements of the dataset using a function *func*	`rdd.reduce(x+y)`	`{8}` → a collection Note: it is not an RDD
`collect()`	Return all the elements	`rdd.collect()`	`{1,3,4}` → a collection
`count()`	Return the number of elements	`rdd.count()`	3
`first()`	Return the first elements	`rdd.first()`	1
`take(n)`	Return first n elements	`rdd.take(1)`	1
`saveAsTextFile(path)`	Write the elements of the dataset as a text file	`rdd.saveAsTextFile(file_path)`	Text file

reduce(func) vs. reducyByKey()?
reduce() operates on RDD of objects (standard RDD), whereas reduceByKey operates on RDD of key value pairs. The function reduce() is a member of RDD[T] class, while reduceByKey() is member of PairRDDFunctions[K,V] class.
reduce() function returns a **collection** which will be included in DAG for the next level, so it is implemented as action. While reduceByKey() returns another RDD which might a part of sequence of operations in a DAG, so it is implemented as transformation.

Actions on pair RDDs(example({(1,2),(3,4),(3,6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key	`rdd.countByKey()`	`{(1,1),(3,2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup at the driver	`rdd.collectAsMap()`	`Map{(1,2),(3,4),(3,6)}`
`lookup(key)`	Return all values associated with the provided key	`rdd.lookup(3)`	`[4,6]`

countByKey

```
public scala.collection.Map<K,Object> countByKey()
```

Count the number of elements for each key, collecting the results to a local Map.

Returns:
(undocumented)

Note:
This method should only be used if the resulting map is expected

Revisit: Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an action
  - Spark internally records metadata to indicate that this operation has been requested
- **Loading data from files into an RDD** is lazily evaluated
- Reduces the number of passes it has to take over our data by grouping operations together

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

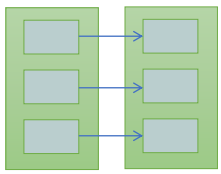
RDD Dependency in Spark

Dependency between RDDs [1/4]

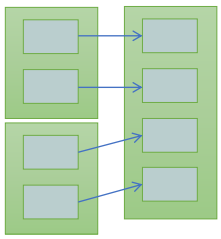
- **Narrow** dependency
- **Wide** dependency

Dependency between RDDs [2/4]

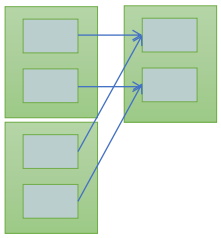
- **Narrow** dependency
  - Each partition of the parent RDD is used by **at most one partition** of the child RDD



map, filter



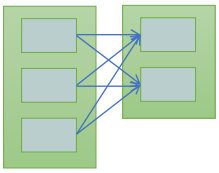
union



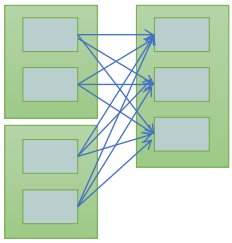
Join with inputs co-partitioned

Dependency between RDDs [3/4]

- **Wide** dependency
  - **Multiple child partitions** may depend on a single partition of parent RDD



groupByKey



Join with inputs not co-partitioned

Dependency between RDDs [4/4]

- Narrow dependency
  - Pipelined execution on one cluster node
    - e.g. a map followed by a filter
  - Failure recovery is more straightforward
- Wide dependency
  - Requires data from all parent partitions to be available and to be shuffled across the nodes
  - Failure recovery could involve a large number of RDDs
    - Complete re-execution may be required

Interface used to represent RDDs in Spark

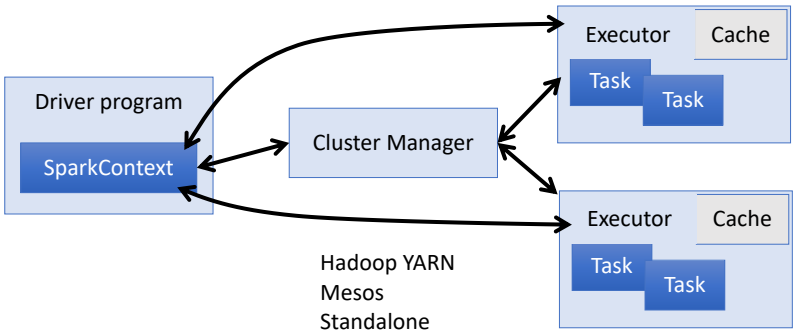
- `partitions()`
  - Returns a list of partition objects
- `preferredLocations(p)`
  - List nodes where partition p can be accessed faster due to data locality
- `dependencies()`
  - Return a list of dependencies
- `iterator(p, parentIters)`
  - Compute the elements of partition p given iterators for its parent partitions
- `partitioner()`
  - Return metadata specifying whether the RDD is hash/range partitioned

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Spark Cluster

Spark cluster and resources



Spark cluster [1/3]

- Each application gets its **own executor processes**
  - Must be up and running for the duration of the entire application
  - Run tasks in multiple threads
  - Isolate applications from each other
    - Scheduling side (each driver schedules its own tasks)
    - Executor side (tasks from different applications run in different JVMs)
  - **Data cannot be shared across different Spark applications** (instances of SparkContext) without writing it to an external storage system

Spark cluster [2/3]

- Spark is agnostic to the underlying cluster manager
  - As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN)

Spark cluster [3/3]

- **Driver program** must **listen for and accept incoming connections** from its executors throughout its lifetime
  - Driver program must be network addressable from the worker nodes
- Driver program should run close to the worker nodes
  - **On the same local area network**

Cluster Manager Types

- **Standalone**
  - Simple cluster manager included with Spark
- **Mesos**
  - Fine-grained sharing option
    - Frequently shared objects for Interactive applications
  - **Mesos master determines the machines that handle the tasks**
- **Hadoop YARN**
  - Resource manager in Hadoop 2

Dynamic Resource Allocation

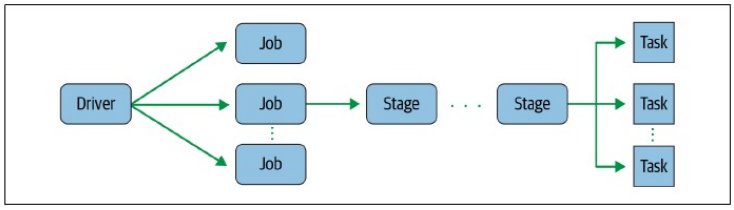
- Dynamically adjust the resources that the applications occupy
  - Based on the workload
  - Your application **may give resources back to the cluster** if they are no longer used
- Only available on **coarse-grained cluster managers**
  - Standalone mode, YARN mode, Mesos coarse grained mode

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Scheduling

Behind your Spark program



- Application
  - A spark submit (A user program)
- Job
  - A group of tasks that need to run to evaluate an action
- Stage
  - Jobs are divided into “stages” based on the shuffle boundary
- Task
  - A unit operation
    - Each stage is further divided into tasks based on the number of partitions in the RDD. So tasks are the smallest units of work for Spark

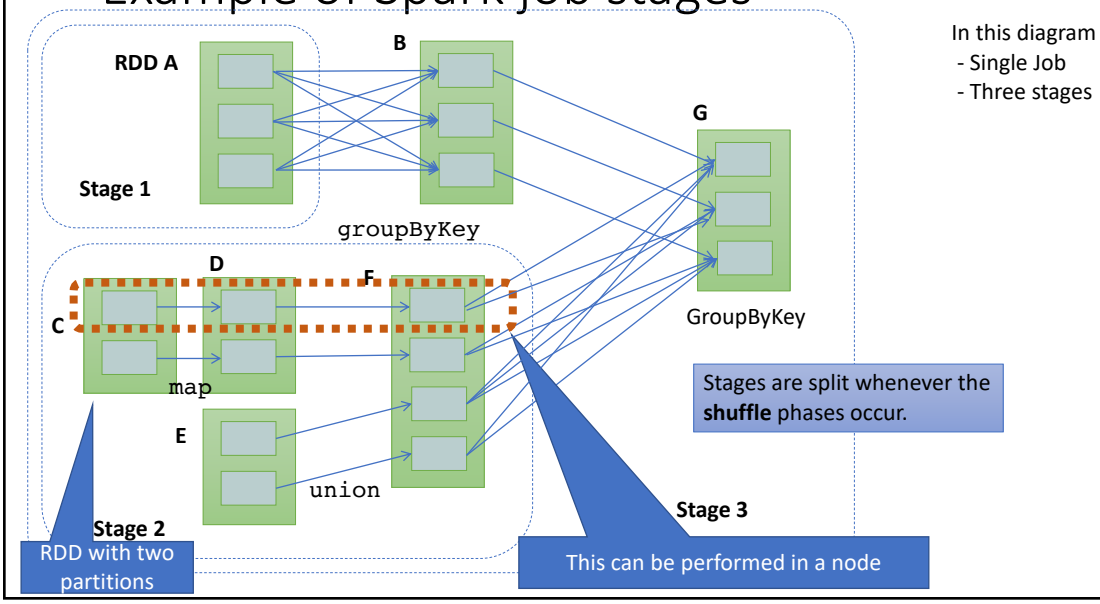
Jobs in Spark application

- **"Job"**
  - **A Spark action (e.g. save, collect) and any tasks that need to run to evaluate that action**
- Within a given Spark application, multiple parallel tasks can run simultaneously
  - If they were submitted from separate threads

Job scheduling

- User runs an **action** (e.g. count or save) on an RDD
  - A job will be created
  - Scheduler examines that **RDD's lineage graph to build a DAG of stages** to execute
- Each stage contains as many pipelined transformations as possible
  - With **narrow** dependencies
- The boundaries of the stages are the **shuffle operations**
  - For **wide** dependencies
  - For any already computed partitions that can short circuit the computation of a parent RDD

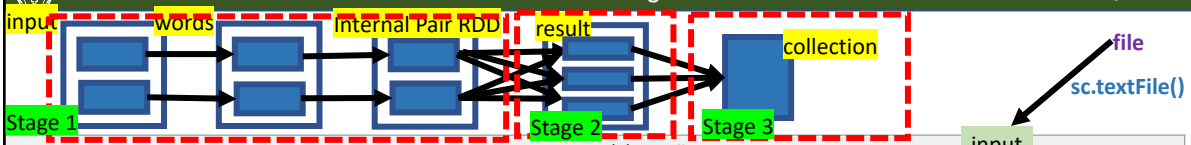
Example of Spark job stages



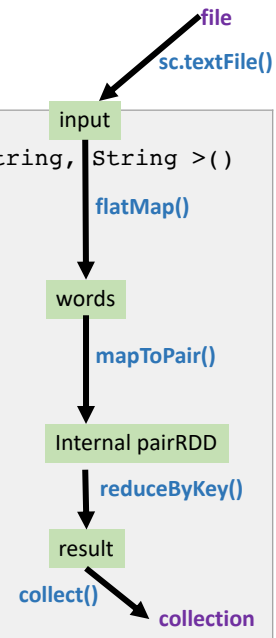
In this diagram
- Single Job
- Three stages

Revisit the word count example

```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
result.collect();
```




```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
result.collect();
```



Default FIFO scheduler

- By default, Spark's scheduler runs jobs in **FIFO** fashion
- First job gets the first priority on all available resources
  - Then the second job gets the priority, etc.
  - As long as the resource is available, jobs in the queue will start right away

Fair Scheduler

- Assigns tasks between jobs in a "**round robin**" fashion
  - All jobs get a roughly equal share of cluster resources
- Short jobs that were submitted when a long job is running can start receiving resources right away
  - Good response times, without waiting for the long job to finish
- Best for multi-user settings

Fair Scheduler Pools

- Supports grouping jobs into pools
  - With different options (e.g. weights)
  - "high-priority" pool for more important jobs
- This approach is modeled after the **Hadoop Fair Scheduler**
- **Default behavior of pools**
  - Each pool gets **an equal share of the cluster**
  - Inside each pool, jobs run in **FIFO** order
  - If the Spark cluster creates one pool per user
    - Each user will get an equal share of the cluster
    - Each user's queries will run in order

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Closures

Understanding closures

- To execute jobs, Spark breaks up the processing of RDD operations into tasks to be executed by **an executor**
- Prior to execution, Spark computes the task's **closure**
  - The closure is those **variables and methods** that **must** be **visible for the executor** to perform its computations on the RDD
- This closure is serialized and sent to each executor

Understanding closures

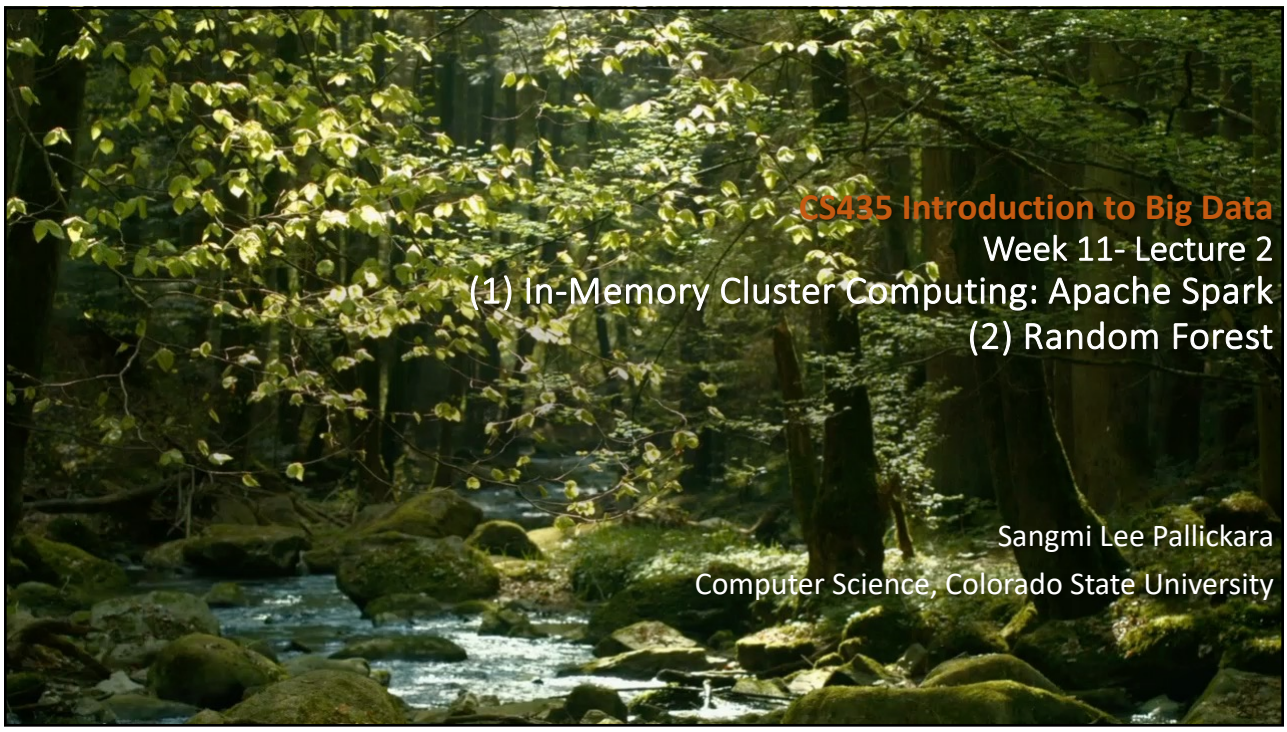
```
1: int counter = 0;
2: JavaRDD<Integer> rdd = sc.parallelize(data);
3:
4: rdd.foreach(x -> counter += x);
5:
6: println("Counter value: " + counter);
```

- **counter**(in line 4) is referenced within the `foreach` function, it's no longer the **counter** (in line 1) on the driver node
- **counter**(in line 1) will still be zero
- In local mode, in some circumstances the `foreach` function will actually execute within the same JVM as the driver
  - **counter** may be actually updated

Solutions?

- Closures (e.g. loops or locally defined methods) **should not** be used to mutate some global state
  - Spark does not define or guarantee the behavior of mutations to objects referenced from outside the closures
- **Accumulator** provides a mechanism for safely updating a variable when execution is split up across worker nodes in a cluster

Questions?



0

FAQs

- ReduceByKey() vs. Reduce()?
  - ReduceByKey() is a transformation
  - Reduce() is an action
  - Why?

1

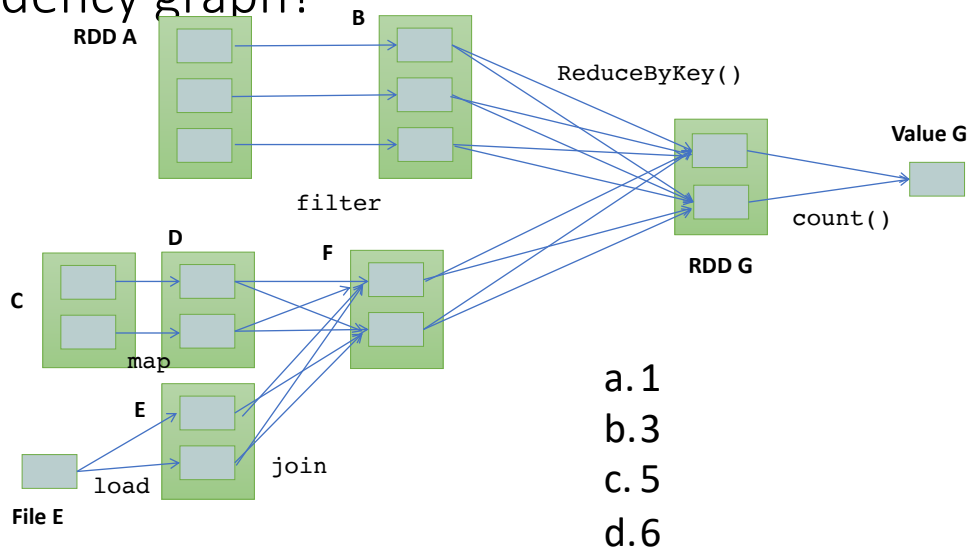
Topics discussed in the previous class

- In-memory cluster computing - Continued
  - Aggregation with PairRDD
  - Actions
  - RDD Dependency
    - Wide dependency vs. Narrow dependency
  - Application/Job/Stage/Task

Topics to be covered

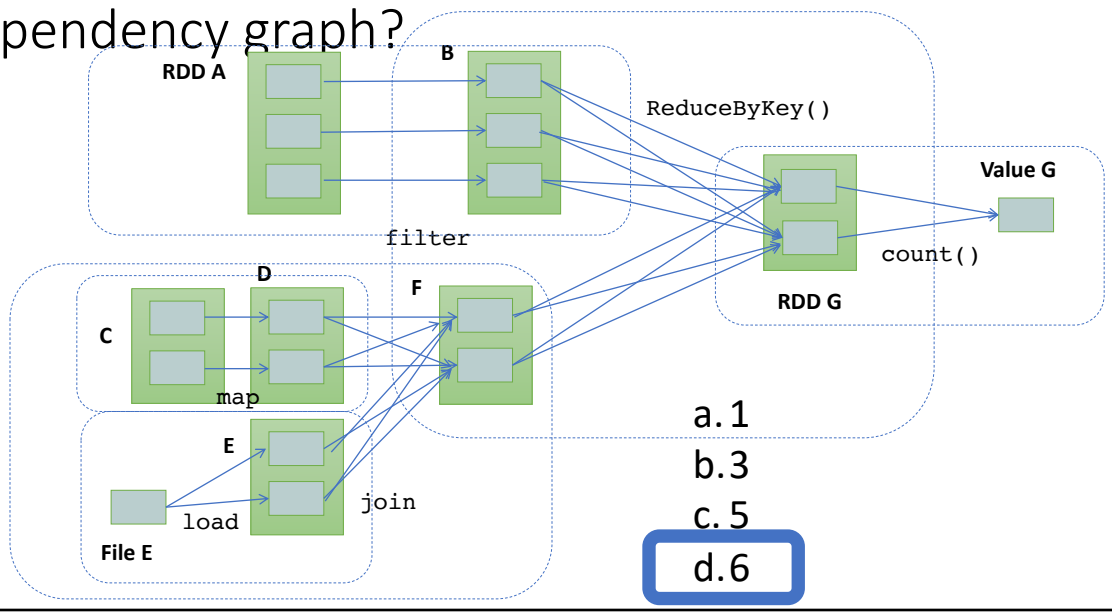
- In-memory cluster computing - Continued
  - Partitions
  - Why RDD? Contrasting RDD to the traditional distributed memory approach

Review Question: How many Stages are in this RDD dependency graph?

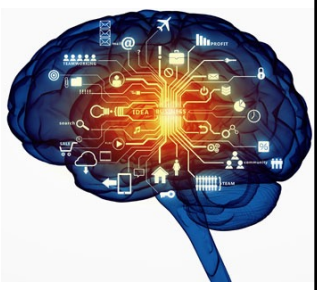


a. 1
b. 3
c. 5
d. 6

Review Question: How many Stages are in this RDD dependency graph?



a. 1
b. 3
c. 5
d. 6



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

This material is built based on

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)
- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015
- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

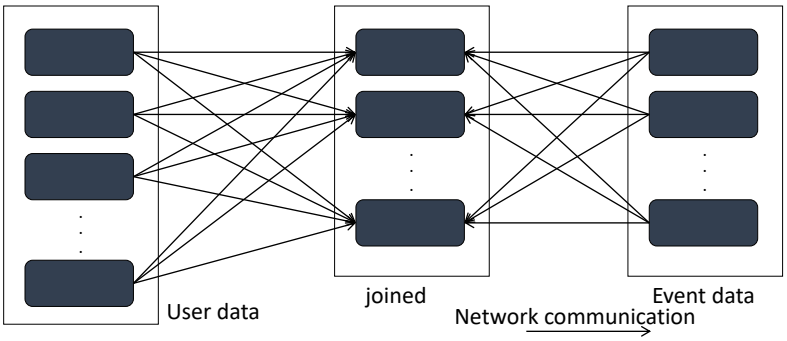
Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Data Partitioning

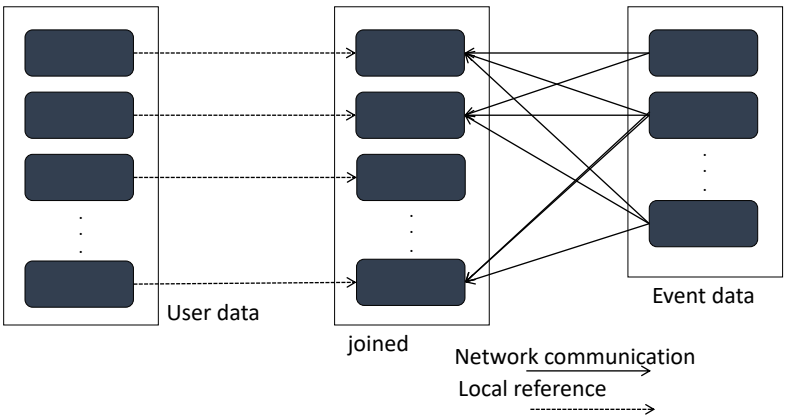
Why partitioning?

- Consider an application that keeps a large table of user information in memory
  - An RDD of `(UserID, UserInfo)` pairs
  - The application periodically combines this table with a smaller file representing events that happened in the last five minutes



Using `partitionBy()`

- Transforms `userData` to hash-partitioned RDD



Spark RDD Persistence

- Spark is persisting (or caching) a dataset in memory across operations
- Reuses them in other actions on that dataset
- Future actions will be much faster
- Key tool for iterative algorithms (>10x faster)
- Mark your RDD with persist() or cache()

Spark RDD persist() with a storage level

Storage Level	Description
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.

More options:
https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-persistence

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Distributed Memory Abstraction: Why RDD?

Benefits of RDDs as a distributed memory abstraction [1/7]

- RDD vs. Distributed Shared Memory (DSM)?
  - How does RDD work differently compared to DSM?
  - Write/Consistency/Fault-Recovery mechanism/Straggler mitigation
- RDDs can only be created ("written") through **coarse-grained transformations**
  - Coarse-grained transformations are applied over an entire dataset
  - Reads on RDDs can still be fine-grained
    - A large read-only lookup table
  - Applications perform bulk writes
- **More efficient fault tolerance**
  - Lineage based bulk recovery

Benefits of RDDs as a distributed memory abstraction [2/7]

RDD vs. DSM- Read operation

- RDD
  - Either coarse grained or fine grained
- DSM
  - The read operation in Distributed shared memory is fine-grained

Benefits of RDDs as a distributed memory abstraction [3/7]

RDD vs. DSM- Write operation

- RDD
  - The write operation in RDD is coarse grained
- DSM
  - The write operation in Distributed shared memory is fine-grained
  - Read and Write happens into the same location of memory

Benefits of RDDs as a distributed memory abstraction [4/7]

RDD vs. DSM- Consistency

- RDD
  - RDD is immutable in nature
    - Any changes on RDD is permanent
  - The level of consistency is high
- DSM
  - If the programmer follows the rules, the memory will be consistent, and the results of memory operations will be predictable

Benefits of RDDs as a distributed memory abstraction [5/7]

RDD vs. DSM- Fault-Recovery Mechanism

- RDD
  - The lost data can be easily recovered in Spark RDD using lineage graph at any moment
  - For each transformation, new RDD is formed
  - RDDs are immutable
- DSM
  - Fault tolerance is achieved by a checkpointing technique which allows applications to roll back to a recent checkpoint rather than restarting

Benefits of RDDs as a distributed memory abstraction [6/7]

RDD vs. DSM- Straggler Mitigation

- Stragglers
  - Nodes taking more time to complete than their peers
  - Due to load imbalance, I/O blocks, garbage collections, etc.
- RDD
  - **Creates backup copies of slow tasks**
    - without accessing the same memory
- DSM
  - It is quite difficult to achieve straggler mitigation

Benefits of RDDs as a distributed memory abstraction [7/7]

- Runtime can schedule tasks **based on data locality**
  - To improve performance
- RDDs degrade gracefully when there is **insufficient memory**
  - **Partitions that do not fit in the RAM are stored on disk**

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 11-B Fall 2021 Computer Science

Applications NOT suitable for RDDs

- RDDs are best suited for **batch applications** that apply the same operations to **all elements of a dataset**
  - Steps are managed by **lineage graph** efficiently
  - Recovery is managed effectively
- RDDs would not be suitable for applications
  - Making asynchronous fine-grained updates to shared state
  - e.g. a **storage system for a web application** or an incremental web crawler

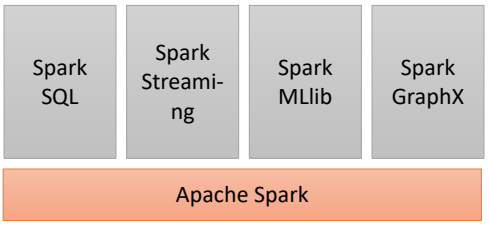
Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Software stack

Spark stack of libraries

- SQL and DataFrames
- Machine learning
- GraphX
- Spark Streaming
- 3rd party libraries
  - https://spark.apache.org/third-party-projects.html



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

Regressions and Classifications

- **Regression analysis**
  - A statistical process for estimating the relationships among variables
  - Dependent variable vs. independent variable ("predictors")
  - How the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed
  - Predicting numerical values
    - Income, temperature, size
- **Classifications**
  - Identifying which of a set of categories a new observation belongs
  - Predicting a label or category
    - Spam, picture of a cat

Features and Feature Vectors

- `Input` and `output` to regression and classification
- Example
  - **Predicting tomorrows temperature given today's weather**
- Features (dimensions, predictors, or variables) of today's weather
  - Today's **high temperature**
  - Today's **low temperature**
  - Today's **average humidity**
  - **Whether it's cloudy, rainy, or clear today**
  - **The number of weather forecasters predicting a cold snap tomorrow**
- Each of these features can be quantified
- `(13.1, 19.0, 0.73, cloudy, 1)`
  - Feature vector

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 11-B Fall 2021 Computer Science

Features and Feature Vectors

- Feature vector
  - These features together, in order
  - `(13.1, 19.0, 0.73, cloudy, 1)`
  - Features do NOT need to be the same type
- **Categorical features**
  - Features that have no ordering
    - Cloudy, clear
- **Numeric features**
  - Features that can be quantified by a number and have a meaningful ordering
    - 23F, 56F (23F < 56F)

Training Examples

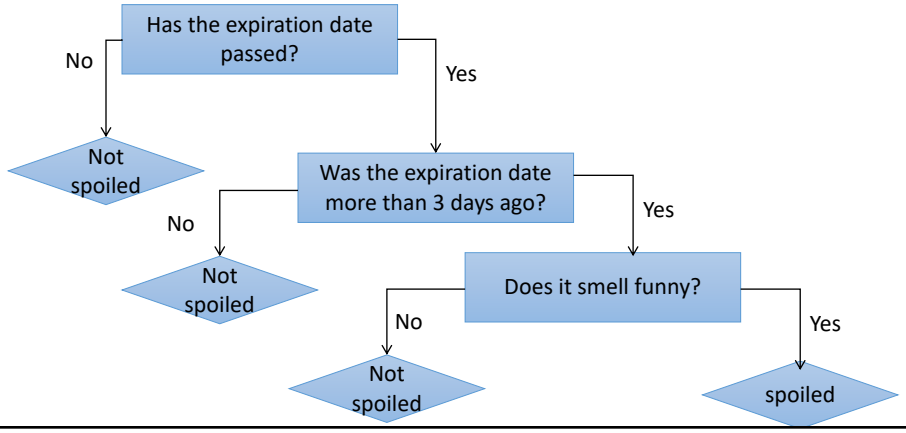
- A learning algorithm needs to train on data in order to make predictions
- Inputs
- Correct outputs (from historical data)
  - "One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was **17.2 degrees**"
  - **Target (output)**
    - 17.2 degree
- Feature vector often includes the target value
  - `(13.1, 19.0, 0.73, cloudy, 1, 17.2)`

Decision Trees and Forests

- Decision trees can naturally handle both categorical and numeric features
  - Easy to parallelize
  - Robust to **outliers**
    - A few extreme and possibly erroneous data points may not affect predictions at all
- Random Decision Forests
  - Extended decision tree algorithm

Decision tree

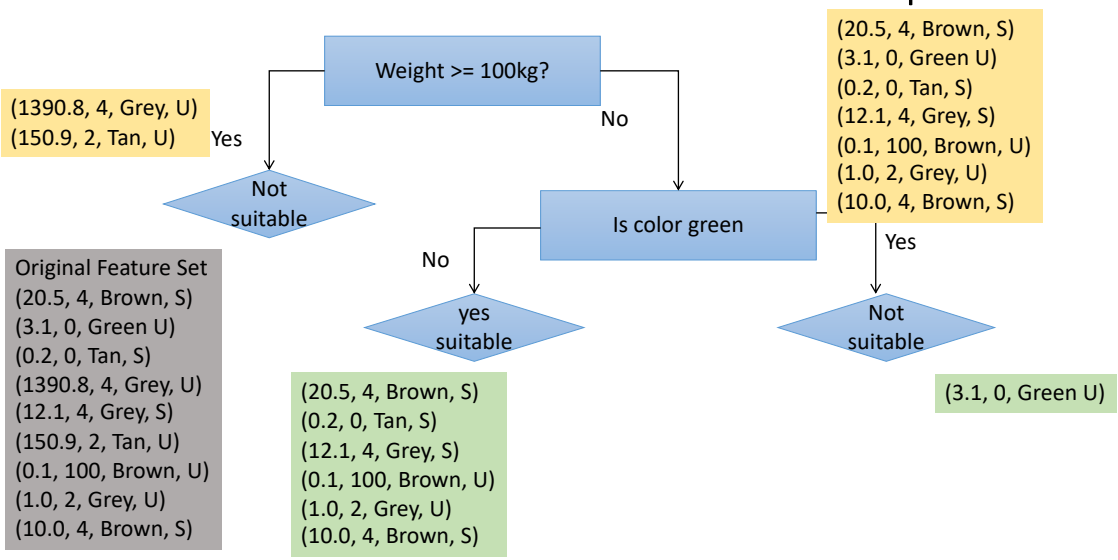
- Spark MLlib's DecisionTree and RandomForest implementation
- Each decision leads to one of two results
  - Prediction or another decision



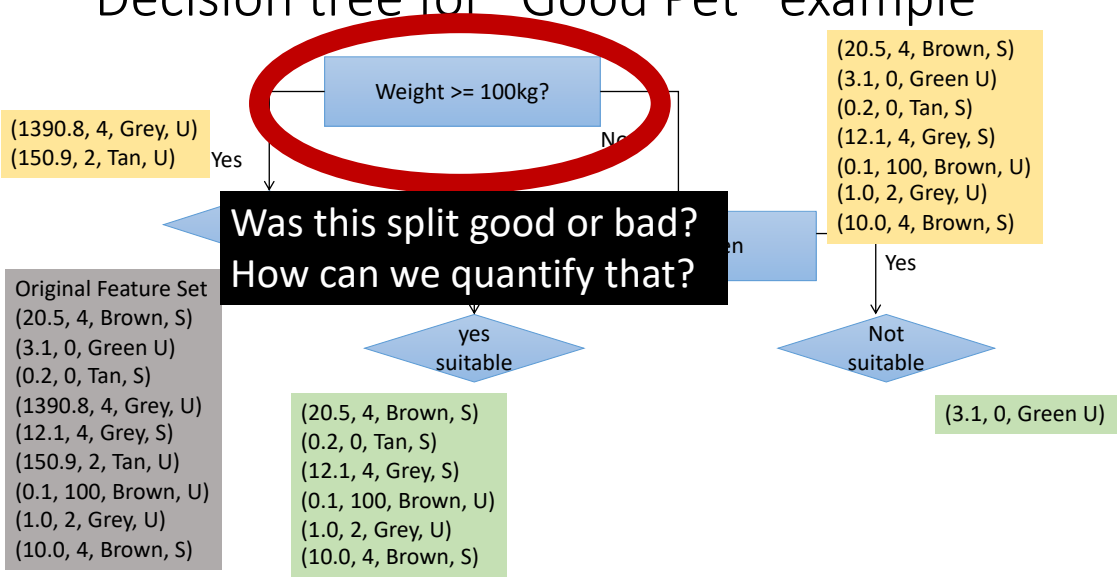
Example: Finding a good pet for the customer

name	Weight (hg)	# of legs	color	Satisfactory?
Fido	20.5	4	Brown	S
Mr. Slither	3.1	0	Green	U
Nemo	0.2	0	Tan	S
Dumbo	1390.8	4	Grey	U
Kitty	12.1	4	Grey	S
Jim	150.9	2	Tan	U
Millie	0.1	100	Brown	U
McPigeon	1.0	2	Grey	U
Spot	10.0	4	Brown	S

Decision tree for "Good Pet" example



Decision tree for "Good Pet" example



Gini impurity

- Gini impurity
  - Measuring impurity degree
  - Within a subset, the probability that a randomly chosen classification of a randomly chosen example is incorrect

Gini impurity – Example 1

- Example 1
  - Calculate the Gini impurity of our entire dataset
  - Random selection: Blue (50%), Green (50%)
  - Classification: it will be also "random"
  - What is the probability we classify our data points incorrectly?
  - **50%**

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

Gini impurity – Example 1: Random split

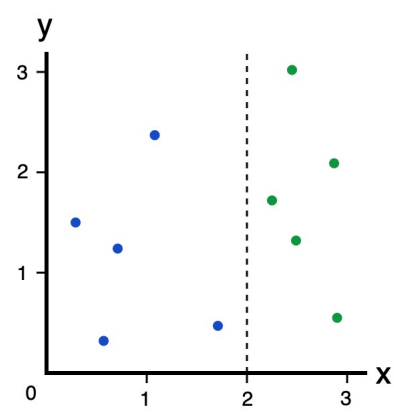
- If we have C total classes and $p(i)$ is the probability of picking a data point with class i
- Gini Impurity
  - $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
  - $C==2$ and $p(1)=p(2)=0.5$ (random classification)

$G=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
$=-0.5 \times (1-0.5)+0.5 \times (1-0.5)$
$=0.5$

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

Gini impurity – Example 2: Perfect split

- Left branch are only blues and right branch are only green
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=1 \times (1-1)+0 \times (1-0)$
  $=0$
- $G_{right}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=0 \times (1-0) + 1 \times (1-1)$
  $=0$
- **Both branches have 0 impurity! (Best Gini impurity)**

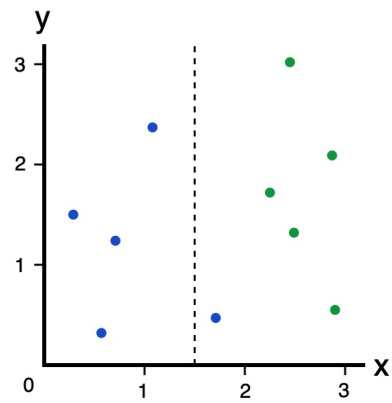


A Perfect Split

Gini impurity – Example 3: Imperfect split

- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $= 1 \times (1-1) + 0 \times (1-0)$
  $= 0$
- $G_{right} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $= 1/6 \times (1-1/6) + 5/6 \times (1-5/6)$
  $= 5/18 = 0.278$
- **Lower Gini impurity means better split**

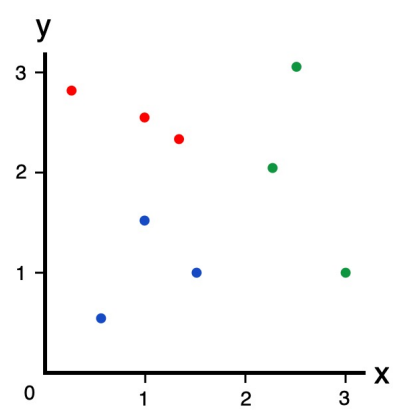
- **Gini Gain**
  - The quality of the split by weighting the impurity of each branch by how many elements it has
- $(0.4 \times 0) + (0.6 \times 0.278) = 0.167$
- **Higher Gini Gain means better split**



An Imperfect Split

Training a Decision Tree with Gini Impurity

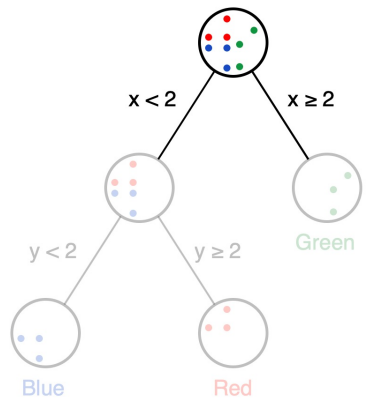
- The 3 class dataset
  - Red/blue/green with two features x and y



The Dataset v2

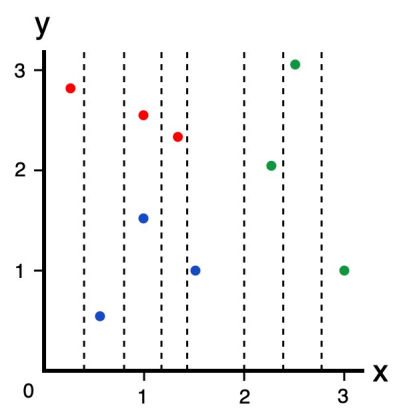
Training a Decision Tree with Gini Impurity : Root Node

- Finding the Root Node
  - Which feature do we have to use? X or Y?
  - What will be the test threshold? (x<2 or x<3)
- Example → see the picture
  - Using x feature with a test threshold of 2
- Good split
  - Separating different classes as much as possible



Training a Decision Tree with Gini Impurity : Root Node

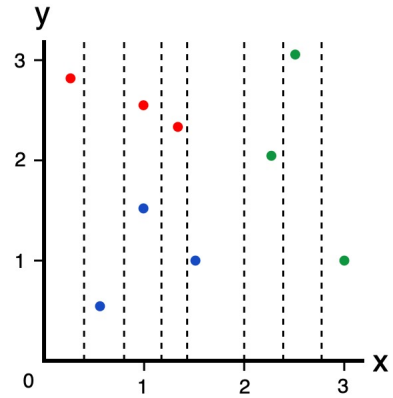
- Find the threshold that provides highest Gini Gain (lowest Gini Impurity)
- We will go over every possible split to find the best threshold
- First example: $x = 0.4$



Split	Left Branch	Right Branch
$x = 0.4$	●	● ● ● ● ● ● ● ●

Training a Decision Tree with Gini Impurity : Root Node

- Gini Impurity of the entire dataset
  - First example: $x = 0.4$
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{initial} = (1/3 \text{ x } 2/3) + (1/3 \text{ x } 2/3) + (1/3 \text{ x } 2/3) = 2/3$
- $G_{left} = 0 \text{ x } 1 + 1 \text{ x } 0 + 0 \text{ x } 1 = 0$
- $G_{right} = 3/8 \text{ x } 5/8 + 2/8 \text{ x } 6/8 + 3/8 \text{ x } 5/8 = 21/32$
- $Gain = G_{initial} - 1/9 G_{left} - 8/9 G_{right}$
  $= 2/3 - (1/9 \text{ x } 0) - (8/9 \text{ x } 21/32) = 0.083$

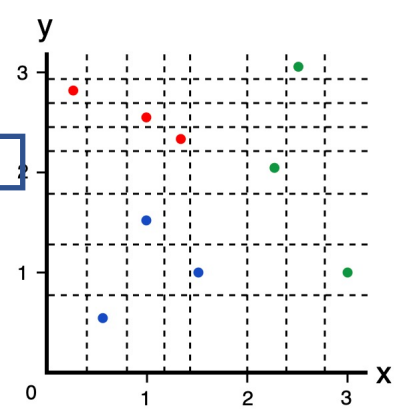


x Thresholds

Split	Left Branch	Right Branch
$x = 0.4$	●	●●●●●●●●

Training a Decision Tree with Gini Impurity : Root Node

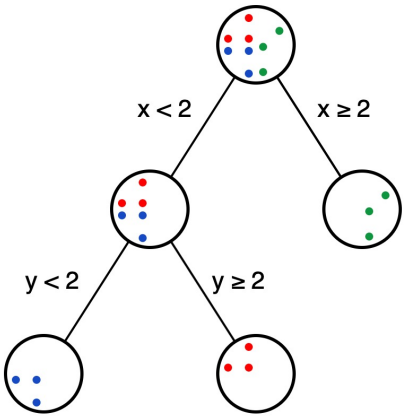
Split	Left Branch	Right Branch	Gini Gain
$x = 0.4$	●	●●●●●●●●	0.083
$x = 0.8$	●●	●●●●●●●●	0.048
$x = 1.1$	●●●●	●●●●	0.133
$x = 1.3$	●●●●●	●●●●	0.233
$x = 2$	●●●●●●	●●●	0.333
$x = 2.4$	●●●●●●●	●●	0.191
$x = 2.8$	●●●●●●●●	●	0.083
$y = 0.8$	●	●●●●●●●●	0.083
$y = 1.2$	●●●	●●●●●●	0.111
$y = 1.8$	●●●●	●●●●●	0.233
$y = 2.1$	●●●●●	●●●●	0.233
$y = 2.4$	●●●●●●	●●●	0.111
$y = 2.7$	●●●●●●●	●●	0.048
$y = 2.9$	●●●●●●●●	●	0.083



All Thresholds

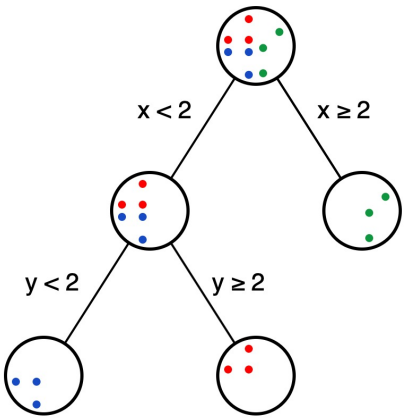
Training a Decision Tree with Gini Impurity : Second Node

- Do The Same Thing..



Training a Decision Tree with Gini Impurity : When to stop?

- Until have a Gini Gain of 0
- We trained a decision tree!



Entropy

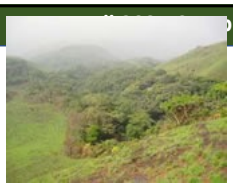
- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?

$$I_E(p) = \sum_{i=1}^{N} p_i \log(1/p) = -\sum_{i=1}^{N} p_i \log(p_i)$$

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests with MLlib



Covertype dataset

- Dataset with records of the types of forest covering parcels of land in Colorado, USA
  - Each example contains several features describing
    - Each parcel of land
    - Elevation, slope, distance to water, shade, and soil type
- The forest cover type is to be predicted
  - From the rest of features
  - Total 54 features
- Used in Kaggle competition
- It includes categorical and numeric features
- 581,012 examples
  - https://archive.ics.uci.edu/ml/datasets/Covertype

Attribute information

Name/ Data Type/ Measurement/ Description

Elevation **/ quantitative** /meters / Elevation in meters
Aspect / quantitative / azimuth / Aspect in degrees azimuth
Slope / quantitative / degrees / Slope in degrees
Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology / quantitative / meters / Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways / quantitative / meters / Horz Dist to nearest roadway
Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer soltice
Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns) **/ qualitative** / 0 (absence) or 1 (presence) / Wilderness area designation
Soil_Type (40 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Soil Type designation
Cover_Type (7 types) / integer **/ 1 to 7** / Forest Cover Type designation

Preparing data

- The `covtype.data` file should be extracted and copied into HDFS
  - File is available at `/user/ds/`

- `LabeledPoint`
  - The **Spark MLlib abstraction for a feature vector**
  - Consists of a Spark MLlib Vector of features, and a target value (label)
  - `LabeledPoint` is only for numeric features
    - It can be used with categorical features, with appropriate encoding

Using `LabeledPoint` for the categorical features

- **One-hot** coding
  - One categorical feature that takes on N distinct values becomes N numeric features, each taking on the value 0 or 1
  - Exactly one of the N values have value 1 and the others are 0
  - `Cloudy, rainy or clear`
    - `Cloudy: 1,0,0`
    - `Rainy: 0,1,0`
    - `Clear: 0,0,1`

- **1-of-n** coding
  - `Cloudy: 1`
  - `Rainy: 2`
  - `Clear: 3`

Question

Using `LabeledPoint` for the categorical features

- There are four categorical features, ”basketball” ,”soccer” ,”volleyball”, and “baseball”. What is the one-hot coding of “soccer”? Suppose that you use a feature vector of ((”basketball” ,”soccer” ,”volleyball”, “baseball”).

a) 1,0,0,0
b) 0,1,0,0
c) 0,0,1,0
d) 1,1,1,0

Question

Using `LabeledPoint` for the categorical features

- There are four categorical features, ”basketball” ,”soccer” ,”volleyball”, and “baseball”. What is the 1-of-n coding of “soccer”? Suppose that you use a code of ((”basketball”=1 ,”soccer”=2 ,”volleyball”=3, “baseball”=4).

a) 1
b) 2
c) 3
d) 4

Categorical values in `Covtype` data set

- The `covtype.info` file says that four of the columns are actually a one-hot encoding of a single categorical feature
  - `Wilderness_Type`, with four values
  - Likewise, 40 of the columns are really one `Soil_Type` categorical feature
- **The target itself is a categorical value encoded as the values 1 to 7**
- The remaining features are numeric features in various units, like meters, degrees, or a qualitative "index" value

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

A First Decision Tree

- Spark MLlib requires input in the form of `LabeledPoint` objects

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression._

val rawData = sc.textFile("hdfs:///user/ds/covtype.data")

val data = rawData.map { line = >
   val values = line.split(',').map(_.toDouble)
   val featureVector = Vectors.dense(values.init)

   val label = values.last - 1
   LabeledPoint(label, featureVector)
}
```


Splitting data

- Training, cross-validation, and test
  - 80% of data for **training** and 10% each **for cross-validation** and **test**
  - Training and CV sets are used to choose a good setting of **hyperparameters** for this data set
  - Test set is used to produce an unbiased evaluation of the expected accuracy of a model built with those **hyperparameters**

```
val Array(trainData, cvData, testData) =
                data.randomSplit(Array(0.8, 0.1, 0.1))

trainData.cache()

cvData.cache()

testData.cache()
```

Building a `DecisionTreeModel` on the training set

- Building a `DecisionTreeModel` on the training set with some default arguments
- Compute some metrics about the resulting model using the CV set

Building a `DecisionTreeModel` on the training set

```
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.tree.model._
import org.apache.spark.rdd._

def getMetrics(model: DecisionTreeModel, data:
RDD[LabeledPoint]):
     MulticlassMetrics = {
   val predictionsAndLabels = data.map(example
  = >
   (model.predict(example.features), example.label))
  new MulticlassMetrics(predictionsAndLabels)
}

val model = DecisionTree.trainClassifier(trainData, 7,
Map[Int, Int](), "gini", 4, 100)

val metrics = getMetrics(model, cvData)
```

Confusion matrix

- 7 x 7 matrix
- The row number corresponds to an actual correct value
- The column number corresponds to a predicted value

```
metrics.confusionMatrix

...
14019.0 6630.0  15.0    0.0   0.0  1.0   391.0
5413.0  22399.0 438.0   16.0  0.0  3.0   50.0
0.0     457.0   2999.0  73.0  0.0  12.0  0.0
0.0     1.0     163.0   117.0 0.0  0.0   0.0
0.0     872.0   40.0    0.0   0.0  0.0   0.0
0.0     500.0   1138.0  36.0  0.0  48.0  0.0
1091.0  41.0    0.0     0.0   0.0  0.0   891.0

metrics.precision
…
0.7030630195577938
```


Precision in multiclass metrics

- Binary classification
  - **Positive vs. negative** class
  - Precision is the fraction of examples that the classifier marked positive that are actually positive
    - $PPV = TP/(TP+FP)$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - $TPR = TP/P=TP/(TP+FN)$
- Multiclass problem
  - **Positive class vs. negative (all else)**

Is 70% accuracy good?

- Classifier that classifies at random in proportion to its prevalence in the training set
- What is the baseline ?
  - A broken clock will be correct twice a day
- Randomly guessing a classification would also be occasionally correct

Decision Tree Hyperparameters [1/2]

- **Hyperparameters**
  - Values we have to choose by building models
  - **Maximum depth, maximum bins, and impurity measure**
- **Maximum depth**
  - Limits the number of levels in the decision tree
  - Useful to avoid overfitting the training data
- **Maximum bins**
  - feature <= value
  - feature in (value1, value 2 ,…)
  - A larger number of bins requires more processing time
    - More optimal decision rule

Decision Tree Hyperparameters [2/2]

- Good rule should **distinguish examples more meaningfully**
- Example
  - E.g. a rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
- A good rule divides the training data's target values into relatively homogeneous or "pure" subsets
- **Minimizing the impurity of the two subsets**

Tuning Decision Trees

- Spark tries a number of combinations of impurity measure, maximum depth or number of bins and reports the results

```
val evaluations =
   for (impurity <- Array(" gini", "entropy");
        depth <- Array( 1, 20);
        bins <- Array( 10, 300))
     yield {
       val model = DecisionTree.trainClassifier(trainData, 7,
                Map[Int, Int](), impurity, depth, bins)
       val predictionsAndLabels = cvData.map(example = >
         (model.predict( example.features), example.label) )
       val accuracy =
          new MulticlassMetrics(predictionsAndLabels).
          precision ((impurity, depth, bins), accuracy)
      }
      evaluations.sortBy(_._2). reverse.foreach( println) …
```

Tuning Decision Trees

- continued

```
(( entropy, 20,300), 0.9125545571245186)
(( gini, 20,300), 0.9042533162173727)
(( gini, 20,10), 0.8854428754813863)
(( entropy, 20,10), 0.8848951647411211)
(( gini, 1,300), 0.6358065896448438)
(( gini, 1,10), 0.6355669661959777)
(( entropy, 1,300), 0.4861446298673513)
(( entropy, 1,10), 0.4861446298673513)
```


Categorical Features Revisited

- `Map[Int, Int]()`
  - Keys
    - **Indices of features** in the input Vector
  - Values
    - Distinct **value counts**
- Empty `Map()`
  - No features should be treated as categorical
  - All are numeric
- Numeric representation of categorical features
  - It can cause errors
  - The algorithm would be trying to learn from an ordering that has no meaning

Treating the categorical features with one-hot encoding

- Encodes the categorical features as several binary 0/1 values
- Any decision rule on the "numeric" features will choose thresholds between 0 and 1
  - All are equivalent since all values are 0 or 1
- Considers the values of the underlying categorical feature **individually**
  - Increases memory usage

Converting one-hot encoding to 1-n encoding [1/3]

```
val data = rawData.map { line = >
  val values = line.split(','). map(_.toDouble)
  val wilderness = values.slice(10, 14).indexOf(1.0). toDouble
  val soil = values.slice(14, 54).indexOf(1.0).toDouble
  val featureVector = Vectors.dense(values.slice(0, 10) :+
wilderness :+ soil)
  val label = values.last - 1
  LabeledPoint( label, featureVector)
}
```

- 4 "wilderness" features
- 40 "soil" features
- Add derived features back to first 10

Converting one-hot encoding to 1-n encoding [2/3]

```
val evaluations =
  for (impurity <- Array("gini", "entropy"); depth <- Array(
10, 20, 30); bins <- Array(40, 300))
  yield {
     val model =
   DecisionTree.trainClassifier(trainData,7,Map(10->4,11->40),
impurity, depth, bins)
     val trainAccuracy = getMetrics(model, trainData).
precision val cvAccuracy = getMetrics( model,cvData).
precision ((impurity, depth, bins),(trainAccuracy,cvAccuracy))
}
```

- Specify value count for categorical features 10, 11
  - Causes these features to be treated as categorical

Converting one-hot encoding to 1-n encoding [3/3]

```
(( entropy, 30,300),( 0.9996922984231909,0.9438383977425239))
(( entropy, 30,40),( 0.9994469978654548,0.938934581368939)) ((
gini, 30,300),( 0.9998622874061833,0.937127912178671)) ((
gini, 30,40),( 0.9995180059216415,0.9329467634811934)) ((
entropy, 20,40),( 0.9725865867933623,0.9280773598540899)) ((
gini, 20,300),( 0.9702347139020864,0.9249630062975326)) ((
entropy, 20,300),( 0.9643948392205467,0.9231391307340239)) ((
gini, 20,40),( 0.9679344832334917,0.9223820503114354)) ((
gini, 10,300),( 0.7953203539213661,0.7946763481193434)) ((
gini, 10,40),( 0.7880624698753701,0.7860215423792973))
…
```

- Tree-building process completes several times **faster**
- By treating categorical features as categorical features, it **improves accuracy by almost 3%**

Ensemble Approach: 1. Bagged Decision Tree

- Bagging
  - Bootstrap Aggregating

Step 1) Sample n data points with replacement

Step 2) Train with n data points

Step 3) repeat step 1 and 2, t times

We have t decision trees

- Take the **majority vote** if our trees produce class labels (like colors)
- Take the **average** if our trees produce numerical values (e.g. when predicting temperature, price, etc)

Ensemble Approach: 2. Random Forests

- Number of trees and **number of features to try when finding the best split**
- Dataset with p features
- Select a subset of features for each tree
  - Usually $\sqrt{p}$ or $p/3$
  - Feature bagging
- This provides randomness that makes
  - Individual tree more unique
  - Reduces correlation between trees

RandomForest

```
val forest = RandomForest.trainClassifier(
       trainData, 7, Map( 10 -> 4, 11 -> 40), 20,
       "auto", "entropy", 30, 300)
```

- Number of trees to build
  - Here, 20
- "auto"
  - The strategy for choosing which features to evaluate at each level of the tree
  - The random decision forest implementation will NOT even consider every feature as the basis of a decision rule
    - Only a subset of all features

Making predictions

- The results of the `DecisionTree` and `RandomForest` training
  - `DecisionTreeModel` and `RandomForestModel` objects
  - `predict()` method
    - Accepts a Vector object
  - We can classify a new example by converting it to a feature vector in the same way and predicting its target class

```
val input = "2709,125,28,67,23,3224,253,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble))
forest.predict(vector)
```

Part 2. Large Scale Data Storage Systems

Distributed File System

Distributed file systems

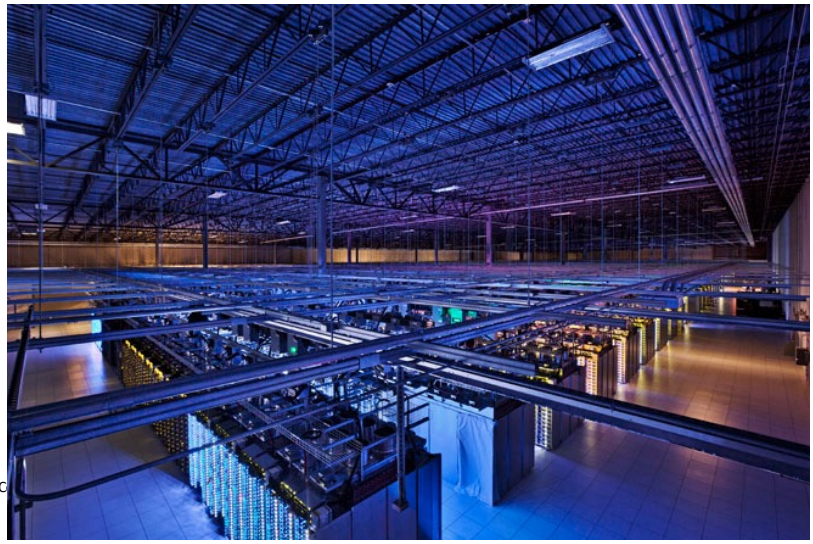
- Hadoop vs. Google File System?

Part 1. Large scale data analysis using MapReduce

Distributed File System

Google File System (GFS)

A server room in Council Bluffs, Iowa



- Photo: Google/C

Data Center

- https://youtube.googleapis.com/v/avP5d16wEp0%26fs=1%26autoplay=1

Data centers

- US data centers consumes about 70 billion kilowatt-hours of electricity in 2014
  - 2 percent of country's total energy consumption
  - 6.4 million average American homes

Questions?

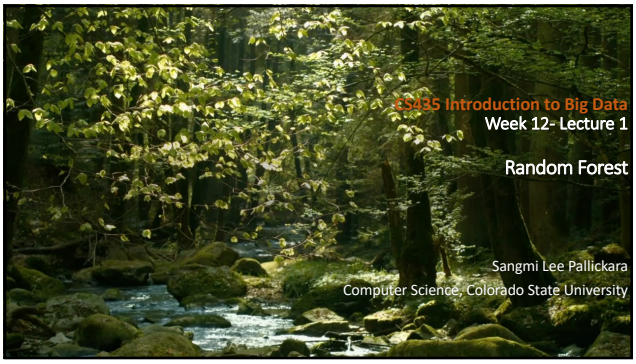
Questions?

84

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 11-B Fall 2021 Computer Science

Questions?

85



0

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 12-A Fall 2021 Computer Science

FAQs

- Please sign up for the PA3 demo
- Term project
  - Regular meetings are important!

1

1

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 12-A Fall 2021 Computer Science

Topics discussed in the previous class

- In-memory cluster computing - Continued
  - Partitions
  - Why RDD? Contrasting RDD to the traditional distributed memory approach
- Introduction to Random Forest

2

2

Topics to be covered

- Decision Tree and Random Forest

3

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

4

REVIEW

Features and Feature Vectors

- Feature vector
  - These features together, in order
  - `(13.1, 19.0, 0.73, cloudy, 1)`
  - Features do NOT need to be the same type
- **Categorical features**
  - Features that have no ordering
    - Cloudy, clear
- **Numeric features**
  - Features that can be quantified by a number and have a meaningful ordering
    - 23F, 56F (23F < 56F)

5

REVIEW

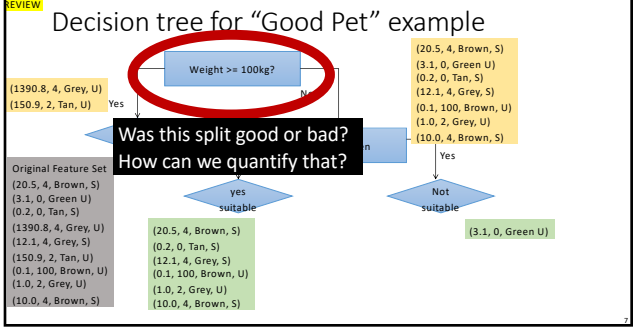
Training Examples

- A learning algorithm needs to train on data in order to make predictions
- Inputs
- Correct outputs (from historical data)
  - "One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was **17.2 degrees**"
  - **Target (output)**
    - 17.2 degree
- Feature vector often includes the target value
  - `(13.1, 19.0, 0.73, cloudy, 1, 17.2)`

6

REVIEW

Decision tree for "Good Pet" example



7

REVIEW

Gini impurity

- Gini impurity
  - Measuring impurity degree
    - Does your classifier collect the similar items?
  - Within a subset, the probability that a randomly chosen classification of a randomly chosen example is incorrect

8

REVIEW

Gini impurity – Example 1

- Example 1
  - Calculate the Gini impurity of our entire dataset
  - Random selection: Blue (50%), Green (50%)
  - Classification: it will be also "random"
  - What is the probability we classify our data points incorrectly?
  - **50%**

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

9

Gini impurity – Example 1: Random split

- If we have C total classes and $p(i)$ is the probability of picking a data point with class i within the split
  - *In this case, C==2 (blue or green)*
  - The probability of picking a data point with class *blue = 0.5 (random)*
  - The probability of picking a data point with class *green = 0.5 (random)*
- Gini Impurity
  - $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
  - $C==2$ and $p(blue)=p(green)=0.5$ (random classification)

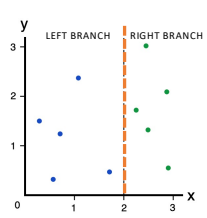
$G=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
$=0.5 \times (1-0.5)+0.5 \times (1-0.5)$
$=0.5$

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

10

Gini impurity – Example 2: Perfect split

- Left branch are only blues and right branch are only green
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=1 \times (1-1)+0 \times (1-0)$
  $=0$
- $G_{right}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=0 \times (1-0) + 1 \times (1-1)$
  $=0$
- **Both branches have 0 impurity! (Best Gini impurity)**



A Perfect Split

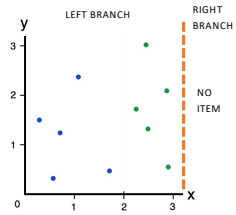
11

Gini impurity – Example 3: Poor split

- Left branch are all the blues and green
- Right branch has no item

- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$

- G_{left}=*p(1)* x *(1 - p(1)) + p(2)* x *(1 - p(2))*
  =5/10 x *(1-5/10)+4/10* x *(1-4/10)*
  =5/10 x *5/10 + 5/10* x *5/10 = 50/100 = 0.5*
- G_{right}=*p(1)* x *(1 - p(1)) + p(2)* x *(1 - p(2))*
  *=0*x *(1-0) + 0* x *(1-0)*
  =0



A Perfect Split

12

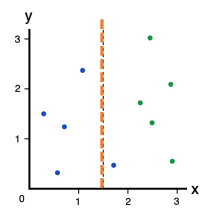
Gini impurity – Example 3: Imperfect split

- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$

- G_{left}=*p(1)* x *(1 - p(1)) + p(2)* x *(1 - p(2))*
  =1 x *(1-1)+0*x *(1-0)*
  =0
- G_{right}=*p(1)* x *(1 - p(1)) + p(2)* x *(1 - p(2))*
  =1/6 x *(1-1/6) + 5/6* x *(1-5/6)*
  =5/18 =0.278
- **Lower Gini impurity means better split**

- **Gini Gain**
  - The quality of the split by weighting the impurity of each branch by how many elements it has
  - *(0.4* x *0) + (0.6* x *0.278) = 0.167*
  - ~~Lower Gini Gain means better split~~
  - Higher Gini Gain means better split
- **NOTE: Spark MLlib uses Information Gain:** the difference between the parent node impurity and the weighted sum of the two child node impurities

$$IG(D,s)=Impurity(D)-NleftNImpurity(Dleft)-NrightNImpurity(Dright)$$

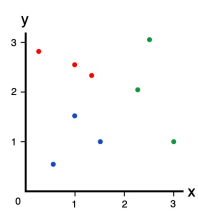


An Imperfect Split

13

Training a Decision Tree with Gini Impurity

- The 3 class dataset
  - Red/blue/green with two features x and y

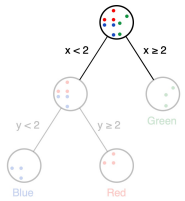


The Dataset v2

14

Training a Decision Tree with Gini Impurity : Root Node

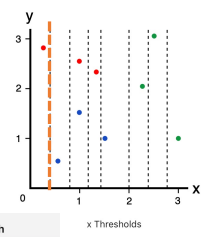
- Finding the Root Node
  - Which feature do we have to use? X or Y?
  - What will be the test threshold? (x<2 or x<3)
- Example → see the picture
  - Using x feature with a test threshold of 2
- Good split
  - Separating different classes as much as possible



15

Training a Decision Tree with Gini Impurity : Root Node

- Find the threshold that provides highest Gini Gain (lowest Gini Impurity)
- We will go over every possible split to find the best threshold
- First example: $x = 0.4$



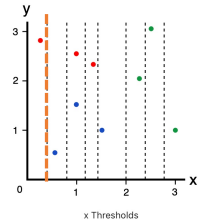
Split	Left Branch	Right Branch
$x = 0.4$	●	●●●●●●●●

16

Training a Decision Tree with Gini Impurity : Root Node

- Gini Impurity of the entire dataset
  - First example: $x = 0.4$
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{initial}$= (1/3 x 2/3) + (1/3 x 2/3) + (1/3 x 2/3) =2/3
- G_{left}=0 x 1 + 1 x 0 + 0 x 1 = 0
- G_{right}=3/8 x 5/8 + 2/8 x 6/8 + 3/8 x 5/8 =21/32
- $Gain = G_{initial} - 1/9 G_{left} - 8/9 G_{right}$

  =2/3-(1/9 x 0)-(8/9 x 21/32)=0.083

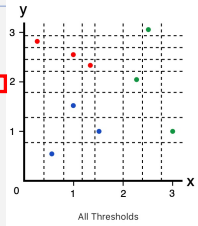


Split	Left Branch	Right Branch
$x = 0.4$	●	●●●●●●●●

17

Training a Decision Tree with Gini Impurity : Root Node (pick the highest Gini Gain)

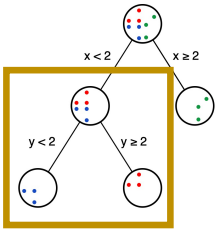
	Split	Left Branch	Right Branch	Gini Gain
Try the feature x	$x = 0.4$	1	7	0.083
	$x = 0.8$	2	6	0.048
	$x = 1.1$	4	4	0.133
	$x = 1.3$	5	3	0.233
	$x = 2$	6	3	0.333
	$x = 2.4$	7	2	0.191
	$x = 2.8$	8	1	0.083
Try the feature y	$y = 0.8$	1	8	0.083
	$y = 1.2$	3	6	0.111
	$y = 1.8$	4	5	0.233
	$y = 2.1$	5	4	0.233
	$y = 2.4$	5	3	0.111
	$y = 2.7$	7	2	0.048
	$y = 2.9$	8	1	0.083



18

Training a Decision Tree with Gini Impurity : Second Node

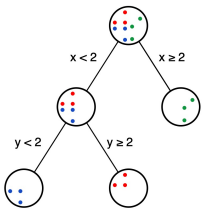
- Do The Same Thing for the child node



19

Training a Decision Tree with Gini Impurity : When to stop?

- Until have a Gini Gain of 0
- We trained a decision tree!



20

Entropy

- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?

$$I_E(p) = \sum_{i=1}^{N} p_i \log(1/p) = -\sum_{i=1}^{N} p_i \log(p_i)$$

- **Information Gain** is the Entropy of the parent node minus the entropy of the child nodes

21

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests with MLlib

22

`Covertype` dataset

- Dataset with records of the types of forest covering parcels of land in Colorado, USA
  - Each example contains several features describing
    - Each parcel of land
    - Elevation, slope, distance to water, shade, and soil type
- The forest cover type is to be predicted
  - From the rest of features
  - Total 54 features
- Used in Kaggle competition
- It includes categorical and numeric features
- 581,012 examples
  - https://archive.ics.uci.edu/ml/datasets/Covertype

23

Attribute information

Name/ Data Type/ Measurement/ Description

Elevation / **quantitative** /meters / Elevation in meters
Aspect / quantitative / azimuth / Aspect in degrees azimuth
Slope / quantitative / degrees / Slope in degrees
Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology / quantitative / meters / Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways / quantitative / meters / Horz Dist to nearest roadway
Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer soltice
Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Wilderness area designation
Soil_Type (40 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Soil Type designation
Cover_Type (7 types) / integer / **1 to 7** / Forest Cover Type designation

24

Preparing data

- The `covtype.data` file should be extracted and copied into HDFS
  - File is available at `/user/ds/`
- `LabeledPoint` (for the RDD version)
  - The **Spark MLlib abstraction for a feature vector**
  - Consists of a Spark MLlib Vector of features, and a target value (label)
  - `LabeledPoint` is only for numeric features
    - It can be used with categorical features, with appropriate encoding

25

Using `LabeledPoint` for the categorical features

- **One-hot** coding
  - One categorical feature that takes on N distinct values becomes N numeric features, each taking on the value 0 or 1
  - Exactly one of the N values have value 1 and the others are 0
  - `Cloudy, rainy or clear`
    - `Cloudy: 1,0,0`
    - `Rainy: 0,1,0`
    - `Clear: 0,0,1`
- **1-of-n** coding
  - `Cloudy: 1`
  - `Rainy: 2`
  - `Clear: 3`

26

Question

Using `LabeledPoint` for the categorical features

- There is a categorical feature with four values, "basketball" ,"soccer" ,"volleyball", and "baseball". What is the one-hot coding of "soccer"? Suppose that you use an encoding scheme of (("basketball" ,"soccer" ,"volleyball", "baseball").

a) 1,0,0,0
b) 0,1,0,0
c) 0,0,1,0
d) 1,1,1,0

27

Question

Using `LabeledPoint` for the categorical features

- There is a categorical feature with four values, "basketball" ,"soccer" ,"volleyball", and "baseball". What is the 1-of-n coding of "soccer"? Suppose that you use an encoding scheme of (("basketball"=1 ,"soccer"=2 ,"volleyball"=3, "baseball"=4).

a) 1
b) 2
c) 3
d) 4

28

Categorical features in `Covtype` data set

- The `covtype.info` file says that four of the columns are actually a one-hot encoding of a single categorical feature
  - `Wilderness_Type`, with four values
  - Likewise, 40 of the columns are really one `Soil_Type` categorical feature
- **The target itself is a categorical value encoded as the values 1 to 7**
- The remaining features are numeric features in various units, like meters, degrees, or a qualitative "index" value

29

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Tree and Random Forest

(1) Decision Tree

30

A First Decision Tree

- Spark MLlib requires input in the form of `LabeledPoint` objects

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression._

val rawData = sc.textFile("hdfs:///user/ds/covtype.data")

val data = rawData.map { line = >
   val values = line.split(',').map(_.toDouble)
   val featureVector = Vectors.dense(values.init)

   val label = values.last - 1
   LabeledPoint(label, featureVector)
}
```


31

Splitting data

- Training, cross-validation, and test
  - 80% of data for **training** and 10% each **for cross-validation** and **test**
  - Training and CV sets are used to choose a good setting of **hyperparameters** for this data set
  - Test set is used to produce an unbiased evaluation of the expected accuracy of a model built with those **hyperparameters**

```
val Array(trainData, cvData, testData) =
                                data.randomSplit(Array(0.8, 0.1, 0.1))

trainData.cache()

cvData.cache()

testData.cache()
```


32

Building a `DecisionTreeModel` on the training set

- Building a `DecisionTreeModel` on the training set with some default arguments
- Compute some metrics about the resulting model using the CV set

33

Building a `DecisionTreeModel` on the training set

```
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.tree.model._
import org.apache.spark.rdd._

def getMetrics(model: DecisionTreeModel, data: RDD[LabeledPoint]):
     MulticlassMetrics = {
   val predictionsAndLabels = data.map(example
  = >
   (model.predict(example.features), example.label))
  new MulticlassMetrics(predictionsAndLabels)
}

val model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int](), "gini",
4, 100)

val metrics = getMetrics(model, cvData)
```


34

Confusion matrix

- 7 x 7 matrix
- The row number corresponds to an actual correct value
- The column number corresponds to a predicted value

```
metrics.confusionMatrix

...
14019.0 6630.0  15.0    0.0   0.0  1.0   391.0
5413.0  22399.0 438.0   16.0  0.0  3.0   50.0
0.0     457.0   2999.0  73.0  0.0  12.0  0.0
0.0     1.0     163.0   117.0 0.0  0.0   0.0
0.0     872.0   40.0    0.0   0.0  0.0   0.0
0.0     500.0   1138.0  36.0  0.0  48.0  0.0
1091.0  41.0    0.0     0.0   0.0  0.0   891.0

metrics.precision
…
0.7030630195577938
```

Example

TP/FP/TN/FN of the type 1

In the first column, out of 20523 items,
14019 items were classified correctly.

5413 items were classified as the second type
1091 items were classified as the seventy type

True positive of the type 1: 14019
False positive of the type 1: (5413+1091)
True negative of the type 1: neither the prediction nor the label take on the value of a given class
- All the items except the first row and the first column
False negative of the type 1:
# of items listed in the first row except the correct label (14019)

Precision in multiclass metrics

- Binary classification
  - **Positive vs. negative** class
  - Precision is the fraction of examples that the classifier marked positive that are actually positive
    - $PPV = TP/(TP+FP)$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - $TPR = TP/P=TP/(TP+FN)$
- Multiclass problem
  - **Positive class vs. negative (all else)**
  - **Precision by labels**

$$PPV(\ell') = \frac{TP}{TP+FP} = \frac{\sum_{i=0}^{N-1} \delta(\hat{y}_i=\ell')\delta(y_i=\ell')}{\sum_{i=0}^{N-1} \delta(\hat{y}_i=\ell')}$$

  - https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html

36

Is 70% accuracy good?

- Classifier that classifies at random in proportion to its prevalence in the training set
- What is the baseline ?
  - A broken clock will be correct twice a day
- Randomly guessing a classification would also be occasionally correct

37

Decision Tree Hyperparameters [1/2]

- **Hyperparameters**
  - Values we have to choose by building models
  - **Maximum depth, maximum bins, and impurity measure**
- **Maximum depth**
  - Limits the number of levels in the decision tree
  - Useful to avoid overfitting the training data
- **Maximum bins**
  - feature <= value
  - feature in (value1, value 2 ,…)
  - A larger number of bins requires more processing time
    - More optimal decision rule

38

Decision Tree Hyperparameters [2/2]

- Good rule should **distinguish examples more meaningfully**
- Example
  - E.g. a rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
- A good rule divides the training data's target values into relatively homogeneous or "pure" subsets
- **Minimizing the impurity of the two subsets**

39

Tuning Decision Trees

- Spark tries a number of combinations of impurity measure, maximum depth or number of bins and reports the results

```
val evaluations =
   for (impurity <- Array(" gini", "entropy");
        depth <- Array( 1, 20);
        bins <- Array( 10, 300))
     yield {
       val model = DecisionTree.trainClassifier(trainData, 7,
               Map[Int, Int](), impurity, depth, bins)
       val predictionsAndLabels = cvData.map(example = >
         (model.predict( example.features), example.label) )
       val accuracy =
          new MulticlassMetrics(predictionsAndLabels).
          precision ((impurity, depth, bins), accuracy)
     }
     evaluations.sortBy( . 2). reverse.foreach( println) …
```


40

Tuning Decision Trees

- continued

```
(( entropy, 20,300), 0.9125545571245186)
(( gini, 20,300), 0.9042533162173727)
(( gini, 20,10), 0.8854428754813863)
(( entropy, 20,10), 0.8848951647411211)
(( gini, 1,300), 0.6358065896448438)
(( gini, 1,10), 0.6355669661959777)
(( entropy, 1,300), 0.4861446298673513)
(( entropy, 1,10), 0.4861446298673513)
```


41

Treating the categorical features with one-hot encoding

- Encodes the categorical features as several binary 0/1 values
- Any decision rule on the "numeric" features will choose thresholds between 0 and 1
  - All are equivalent since all values are 0 or 1
- Considers the values of the underlying categorical feature **individually**
  - Increases memory usage

42

Converting one-hot encoding to 1-n encoding [1/3]

```
val data = rawData.map { line = >
  val values = line.split(','). map(_.toDouble)
  val wilderness = values.slice(10, 14).indexOf(1.0). toDouble
  val soil = values.slice(14, 54).indexOf(1.0).toDouble
  val featureVector = Vectors.dense(values.slice(0, 10) :+
wilderness :+ soil)
  val label = values.last - 1
  LabeledPoint( label, featureVector)
}
```

- 4 "wilderness" features
- 40 "soil" features
- Add derived features back to first 10

43

Converting one-hot encoding to 1-n encoding [2/3]

```
val evaluations =
  for (impurity <- Array("gini", "entropy"); depth <- Array(
10, 20, 30); bins <- Array(40, 300))
  yield {
     val model =
    DecisionTree.trainClassifier(trainData,7,Map(10->4,11->40),
impurity, depth, bins)
     val trainAccuracy = getMetrics(model, trainData).
precision val cvAccuracy = getMetrics( model,cvData).
precision ((impurity, depth, bins),(trainAccuracy,cvAccuracy))
}
```

- Specify value count for categorical features 10, 11
  - Causes these features to be treated as categorical

44

Converting one-hot encoding to 1-n encoding [3/3]

```
(( entropy, 30,300),( 0.9996922984231909,0.9438383977425239))
(( entropy, 30,40),( 0.9994469978654548,0.938934581368939)) ((
gini, 30,300),( 0.9998622874061833,0.937127912178671)) ((
gini, 30,40),( 0.9995180059216415,0.9329467634811934)) ((
entropy, 20,40),( 0.9725865867933623,0.9280773598540899)) ((
gini, 20,300),( 0.9702347139020864,0.9249630062975326)) ((
entropy, 20,300),( 0.9643948392205467,0.9231391307340239)) ((
gini, 20,40),( 0.9679344832334917,0.9223820503114354)) ((
gini, 10,300),( 0.7953203539213661,0.7946763481193434)) ((
gini, 10,40),( 0.7880624698753701,0.7860215423792973))
…
```

- Tree-building process completes several times faster
- By treating categorical features as categorical features, it **improves accuracy by almost 3%**

45

Large Scale Data Analytics

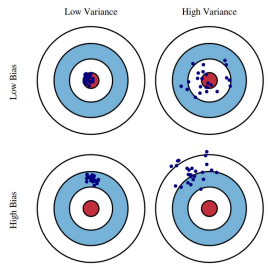
In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Tree and Random Forest
(2) Random Forest

46

Ensemble models

- Combine multiple other models in the prediction process
  - Base estimators
- High variance
  - The model is sensitive to the input values
- Low accuracy
  - One model or one algorithm to fit the entire training data might not be good enough to meet expectations
- Features noise and bias
  - The model relies heavily on one or a few features while making a prediction



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

47

Ensemble Approach: 1. Bagged Decision Tree

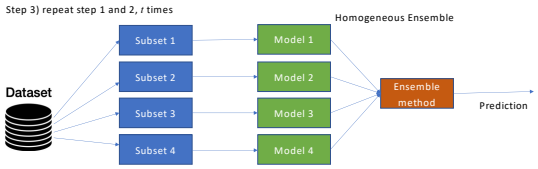
Bagging

Bootstrap Aggregating

Step 1) Sample n data points with replacement

Step 2) Train with n data points

Step 3) repeat step 1 and 2, t times



We have t decision trees

Take the **majority vote** if our trees produce class labels (like colors)

Take the **average** if our trees produce numerical values (e.g. when predicting temperature, price, etc)

48

Ensemble Approach: 2. Random Forests

- Number of trees and **number of features to try when finding the best split**
- Dataset with p features
- Select a subset of features for each tree
  - Usually $\sqrt{p}$ or $p/3$
  - Feature bagging
- This provides randomness that makes
  - Individual tree more unique
  - Reduces correlation between trees

49

RandomForest

```
val forest = RandomForest.trainClassifier(
      trainData, 7, Map( 10 -> 4, 11 -> 40), 20,
      "auto", "entropy", 30, 300)
```

- Number of trees to build
  - Here, 20
- "auto"
  - The strategy for choosing which features to evaluate at each level of the tree
  - The random decision forest implementation will NOT even consider every feature as the basis of a decision rule
    - Only a subset of all features

50

Making predictions

- The results of the `DecisionTree` and `RandomForest` training
  - `DecisionTreeModel` and `RandomForestModel` objects
  - `predict()` method
    - Accepts a Vector object
  - We can classify a new example by converting it to a feature vector in the same way and predicting its target class

```
val input = "2709,125,28,67,23,3224,253,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble))
forest.predict(vector)
```


51

Questions?

52

0

COLORADO STATE UNIVERSITY | CS435 Introduction to Big Data: Week 12-A | Fall 2021 Computer Science

FAQs

- How does Random Forest achieve parallelism with Spark?

1

1

Topics discussed in the previous class

- Classification model
  - Decision Tree and Random Forest

Topics to be covered

- Predictive analytics
  - Linear Regression
  - MapReduce and Apache Spark

Predictive Analysis

Linear Regression at Scale

This material is developed based on,

- Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. 2006. Map-reduce for machine learning on multicore. In Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06), B. Schölkopf, J. C. Platt, and T. Hoffman (Eds.). MIT Press, Cambridge, MA, USA, 281-288.

Predictive Analysis

Linear Regression: Concepts

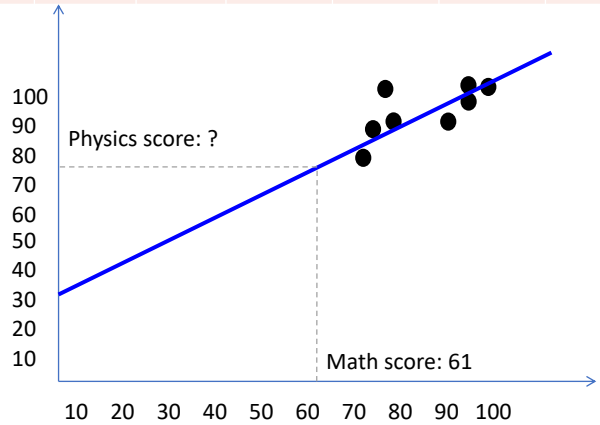
Linear regression [1/3]

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80

- Is there **any correlation** between a student's Math score and Physic score?
- If a student's Math score is known, can we predict his/her Physic score?

Linear regression [2/3]

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80



Linear regression [3/3]

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80

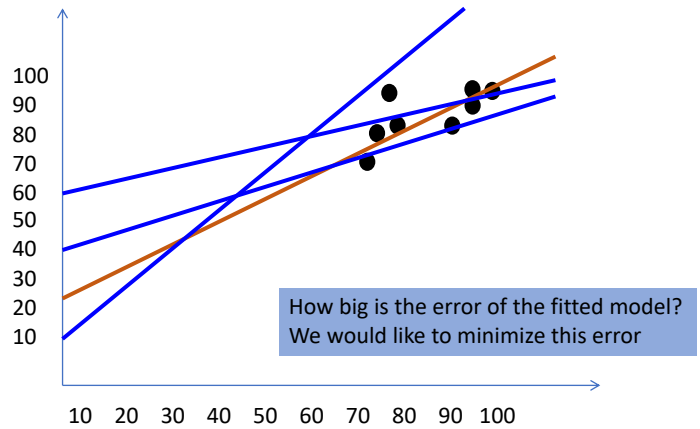
$$h_\theta(x) = \theta_0 + \theta_1 x$$

where, the θs are the **parameter vectors**
$h_\theta(x)$, (Estimated Physics score) is called the regressand or endogeneous variable
x_1 , (Math score) is called regressor, or exogeneous variable

Fitting the linear regression model [1/3]

- The structure

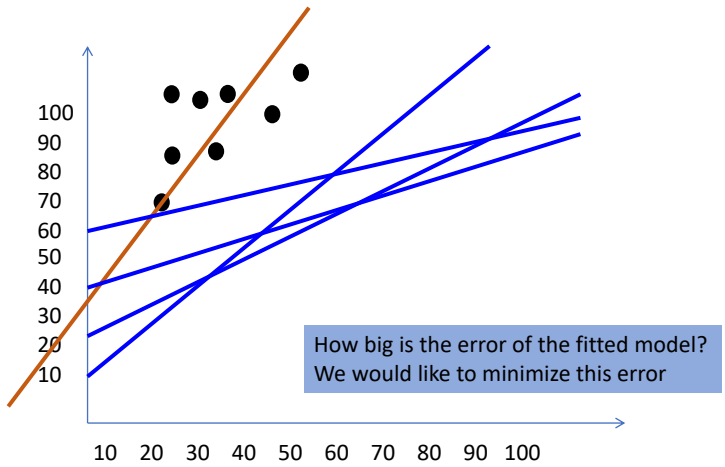
$$h_\theta(x) = \theta_0 + \theta_1 x$$



Fitting the linear regression model [2/3]

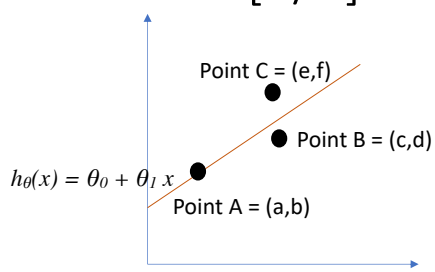
- The structure

$$h_\theta(x) = \theta_0 + \theta_1 x$$



Fitting the linear regression model [3/3]

- How **big is the error** of the fitted model?
  - We would like to minimize this error
  - Absolute error for the point A = 0
  - Absolute error for the point B = $|h_\theta(c) - d|$
  - Absolute error for the point C = $|h_\theta(e) - f|$



- The model that fits the data best
  - The model with **the minimum sum of errors** on the training data
  - e.g. The sum or mean of the squares of the errors
    - Least squares regression

Squared error

- Squared error
  - **Strongly penalizes very large errors**
  - Drawback
    - Is very **sensitive to the data**
    - Erroneous or outlying data points can severely skew the resultant linear function

- We should choose the objective function to optimize

Root Mean Squared Error

- Measures the differences between values predicted by model estimator and the values actually observed

$$RMSD = \sqrt{\frac{\sum_{i=0}^{n-1}(h_\theta(x^{(i)}) - y^{(i)})^2}{n}}$$

Mean Squared Error

- Measures the average of the squares of the errors

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Linear Regression

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \ldots$$

- To simplify the notation:

$$h(x) = \sum_{i=0}^{n} \theta_i x_i$$

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 12-A Fall 2021 Computer Science

Objective function (Cost function)

- For a given training set, how do we pick, or **learn the parameter θ?**
  - **What is the desirable condition of this regression?**
- **Make $h(x)$ close to y**
  - Make your **prediction** close to the **real observation**
- We define the **objective (cost) function**
  - Using Mean Squared Error (MSE) and multiplying ½ for convenience

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m-1} (h_\theta(x^{(i)}) - y^{(i)})^2$$

i is the index of the data points

Minimization problem

- We have an objective function $J(\theta_0, \theta_1)$
- Now, we want to find $\min_{\theta_o,\theta_1} J(\theta_o,\theta_1)$

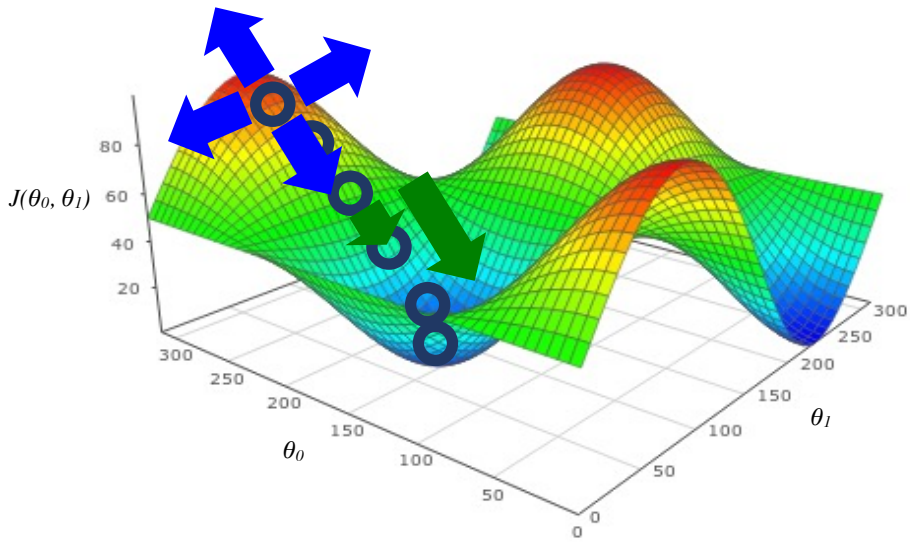
- **Goal: Find parameters to minimize the cost (output of the objective function)**

- Outline of our approach:
  - Start with some θ_0, θ_1
  - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum

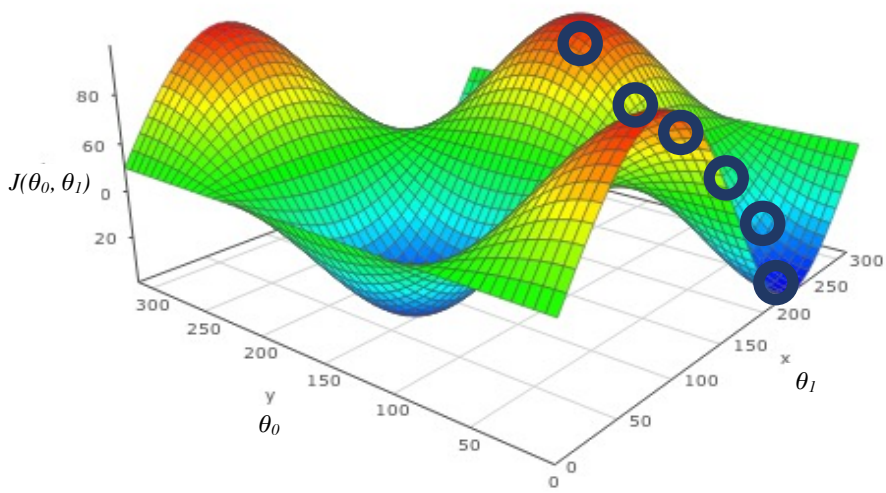
Predictive Analysis

Linear Regression: Gradient Descent Algorithm

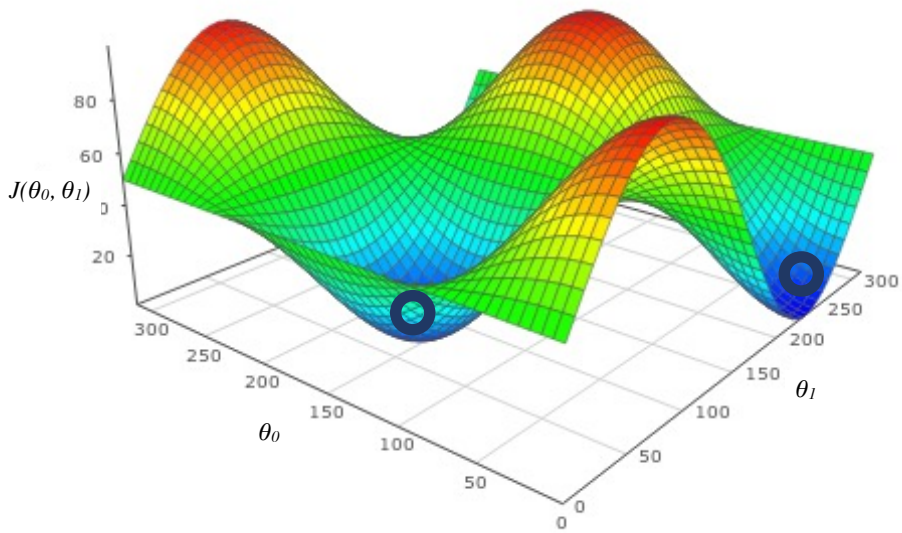
Concept of Gradient descent algorithm (1/2)



Concept of Gradient descent algorithm (2/2)

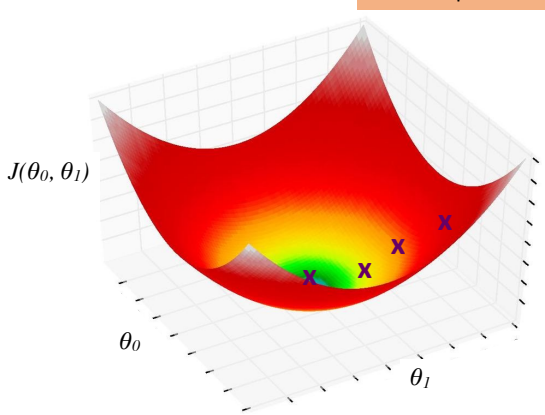


Multiple local optimal points



Convex function

Bowl-shaped function



Gradient descent algorithm

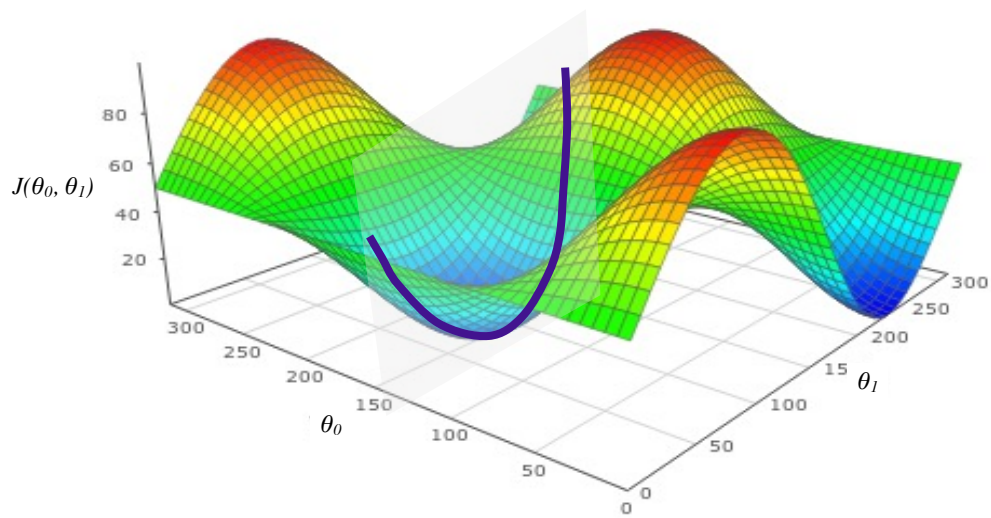
```
Repeat until convergence {
```

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

```
                    (for j=0 and j=1)
}
```

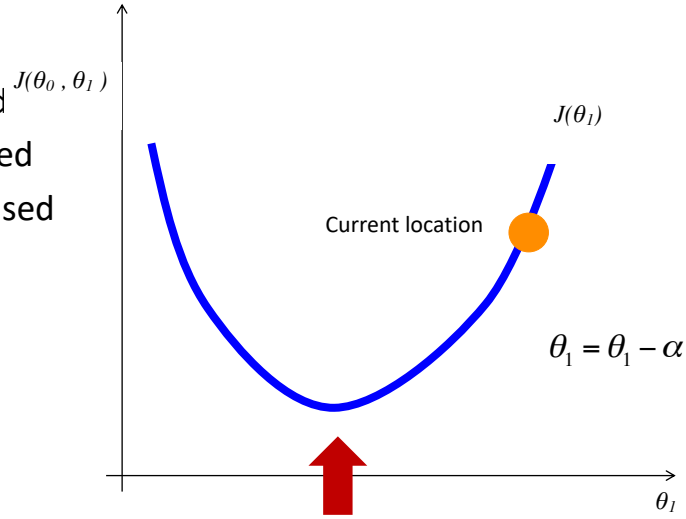
$$\begin{cases} \theta_0 := \theta_0 - \alpha \dfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_1 := \theta_1 - \alpha \dfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \end{cases}$$

1. Finding the direction to move for the next iteration: Decreasing and increasing θ_1



Question) **To reach the minimum point, what should be happened?**

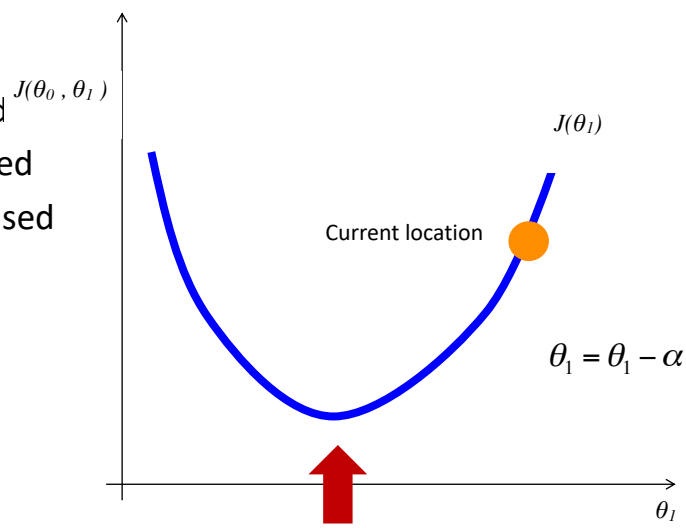
a) θ_1 should be reduced
b) θ_1 should be increased
c) *Alpha* should be reduced
d) *Alpha* should be increased



$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Question) **To reach the minimum point, what should be happened?**

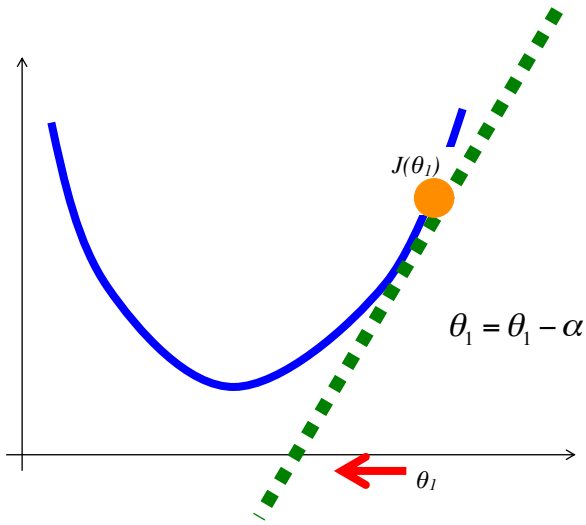
a) **θ_1 should be reduced**
b) θ_1 should be increased
c) *Alpha* should be reduced
d) *Alpha* should be increased



$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Decreasing θ_1

- **Positive** slope

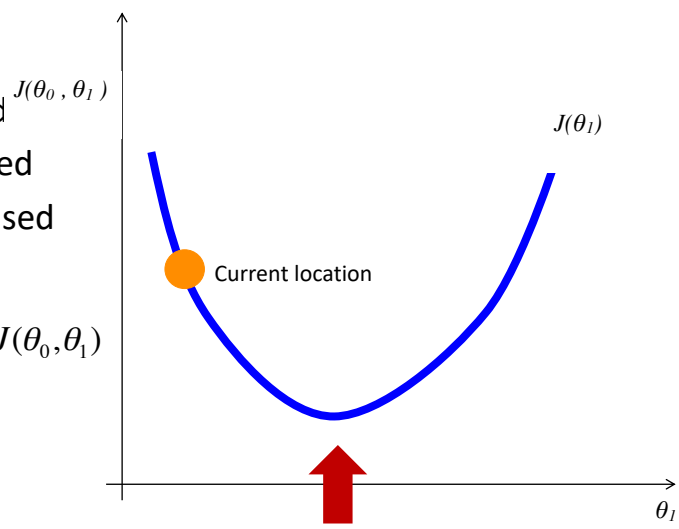


$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Question) **To reach the minimum point, what should be happened?**

a) θ_1 should be reduced
b) θ_1 should be increased
c) *Alpha* should be reduced
d) *Alpha* should be increased

$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$



Question) **To reach the minimum point, what should be happened?**

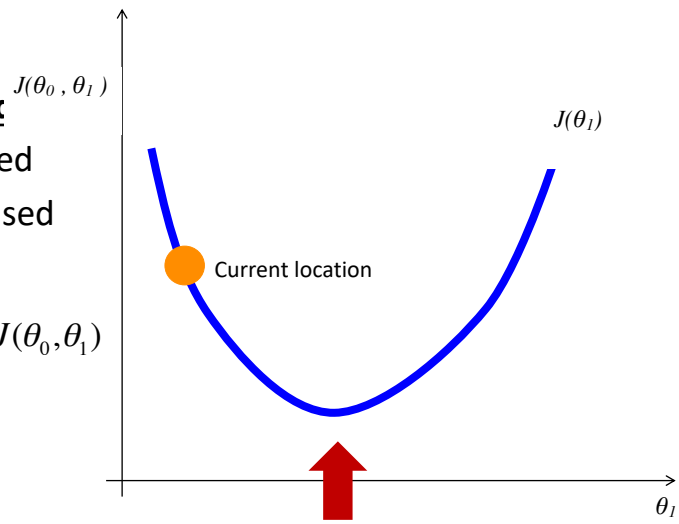
a) θ_1 should be reduced

b) **θ_1 should be increased**

c) *Alpha* should be reduced

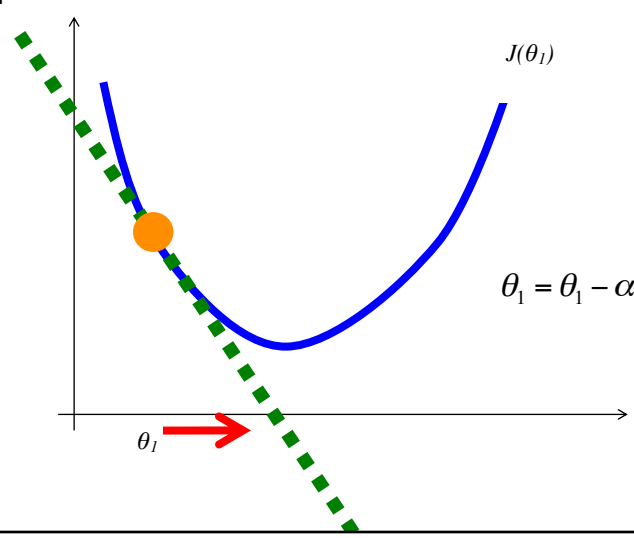
d) *Alpha* should be increased

$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$



Increasing θ_1

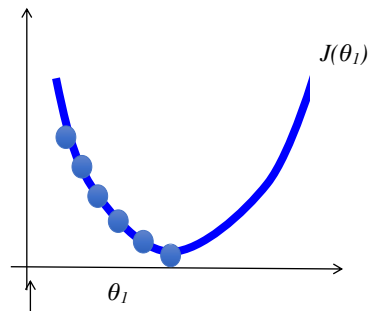
- **Negative** Slope



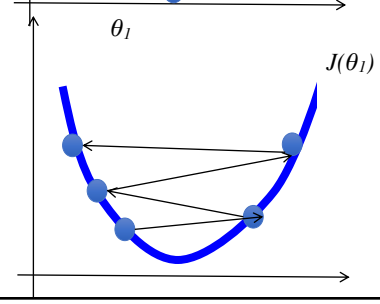
$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

2. Finding the size of steps: Learning rates

- If α(Learning Rate) is too small, gradient descent can be slow



- If α (Learning Rate) is too large, gradient descent can overshoot the minimum



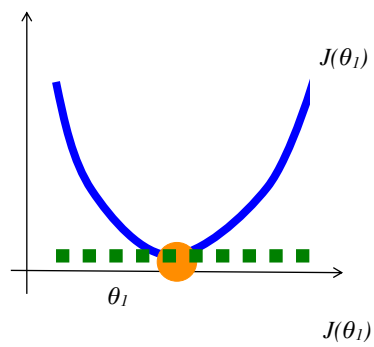
Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required

- **Question**

How will you detect the convergence?

a) Tracking θ_1
b) Tracking θ_0
c) Tracking the slope
d) Randomly testing



Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required

- **Question**

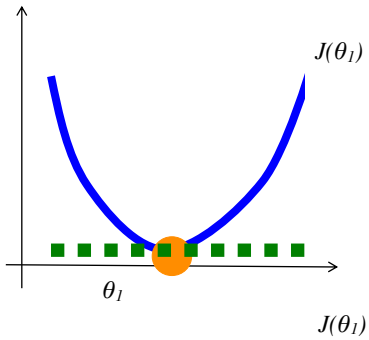
How will you detect the convergence?

a) Tracking θ_1

b) Tracking θ_0

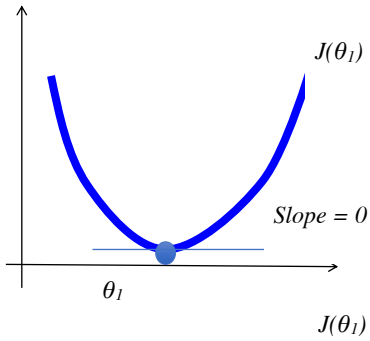
c) <u>Tracking the slope</u>

d) Randomly testing



Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required
  - **<u>Until the slope becomes 0</u>**



Fixed learning rate α

- Gradient descent can converge to a local minimum, even with a fixed learning rate
- As we approach a local minimum, gradient descent will automatically take smaller steps
  - **No need to decrease** α over time

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Simultaneous update

What is the difference between computing A and B?

A: Correct: Simultaneous update

temp0 := $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

temp1 := $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

θ_0 := temp0
θ_1 := temp1

temp1 is calculated before θ_0 is updated

B: Incorrect:

temp0 := $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

θ_0 := temp0

temp1 := $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

θ_1 := temp1

temp1 is calculated after θ_0 is updated

Predictive Analysis

Fitting a Linear Regression model using Gradient Descent Algorithm

Using Gradient Descent Algorithm for fitting a Linear Regression Model

Gradient descent algorithm

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for j=0 and j=1)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m-1} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Continued

$$\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$
$$= \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=0}^{m-1}\left(\theta_0 + \theta_0 x^{(i)} - y^{(i)}\right)^2$$

Case 1, θ_0 $(j = 0)$: $\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$

Case 2, θ_1 $(j = 1)$: $\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_1}\frac{1}{2m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$

Gradient descent for Linear Regression

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

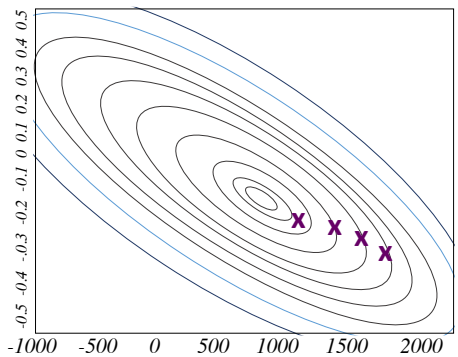
$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$$

Update θ_0 and θ_1 simultaneously

(for j=0 and j=1)

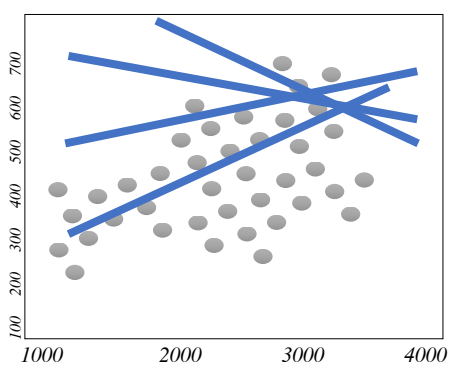
}

$J(\theta_0,\theta_1)$

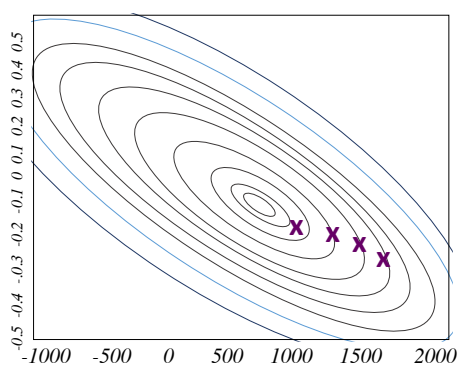


Fitting $h_\theta(x)$

$h_\theta(x)$



$J(\theta_0,\theta_1)$



Predictive Analysis

Linear Regression: Running with MapReduce

"Batch" Gradient Descent

- Batch
  - Each step of gradient descent uses all of the training example

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

Running with MapReduce

- For the sample size 1,000 ($m=1,000$)
- Batch gradient descent:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

- Using 4 machines

For θ_0

- Step 1. 4 input splits

$(x^{(0)},y^{(0)}),\ldots,((x^{(249)},y^{(249)})$
$(x^{(250)},y^{(250)}),\ldots,((x^{(499)},y^{(499)})$
$(x^{(500)},y^{(500)}),\ldots,((x^{(749)},y^{749})$
$(x^{(750)},y^{(750)}),\ldots,((x^{(999)},y^{(999)})$

$$Temp1 = \sum_{i=0}^{249} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$Temp2 = \sum_{i=250}^{499} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

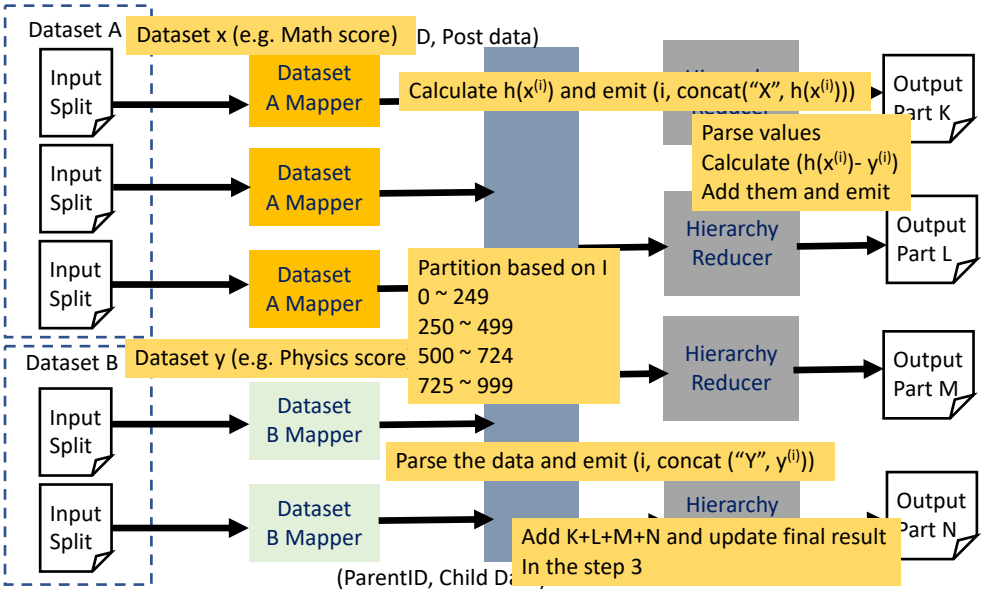
$$Temp3 = \sum_{i=500}^{749} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$Temp4 \sum_{i=750}^{999} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

- Step 2. Calculate temp1 ~ 4
- Step 3. Calculate final results

$$\theta_0 := \theta_0 - \alpha \frac{1}{1,000} (temp1 + temp2 + temp3 + temp4)$$

Structure of the structured to hierarchical pattern



For θ_1

- Step 1. 4 input splits

$(x^{(0)},y^{(0)}),\ldots,((x^{(249)},y^{(249)})$

$(x^{(250)},y^{(250)}),\ldots,((x^{(499)},y^{(499)})$

$(x^{(500)},y^{(500)}),\ldots,((x^{(749)},y^{749)})$

$(x^{(750)},y^{(750)}),\ldots,((x^{(999)},y^{(999)})$

$$Temp1 = \sum_{i=0}^{249} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

$$Temp2 = \sum_{i=250}^{499} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

$$Temp3 = \sum_{i=500}^{749} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

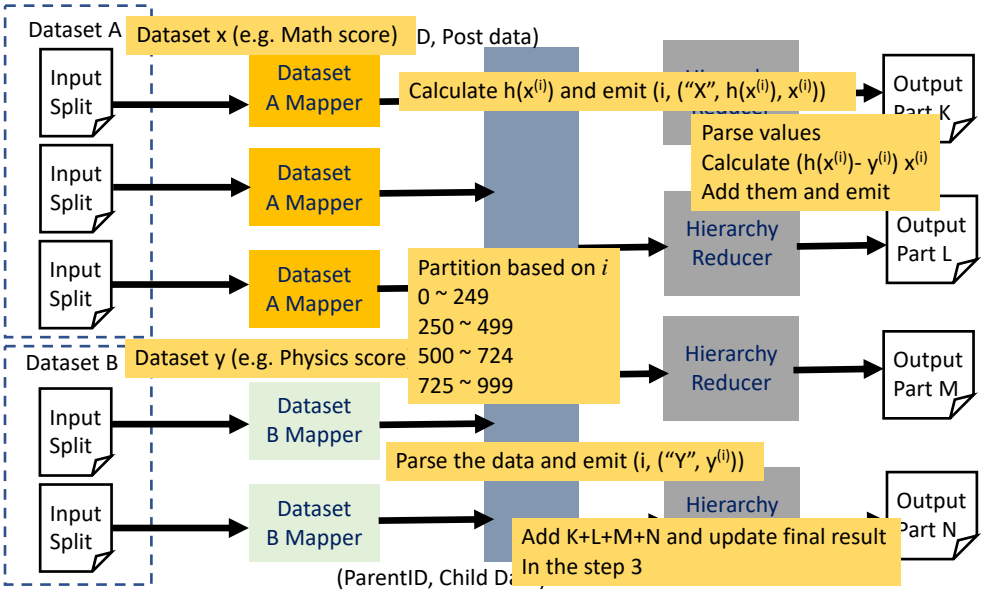
$$Temp4 \sum_{i=750}^{999} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

- Step 2. Calculate temp1 ~ 4

- Step 3. Calculate final results

$$\theta_1 := \theta_1 - \alpha \frac{1}{1{,}000}(temp1 + temp2 + temp3 + temp4)$$

Structure of the structured to hierarchical pattern



Linear Regression in the Apache Spark MLlib

- Similar steps as in Decision Tree and Random Forest

1. Create a dataset
   1. Generating underlying RDD with a system compatible structure
2. Specify the hyperparameters
3. Train the model
4. Evaluate

```
// Load training data.
Dataset<Row> training = spark.read().format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt");

LinearRegression lr = new LinearRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8);

// Fit the model.
LinearRegressionModel lrModel = lr.fit(training);

// Print the coefficients and intercept for linear regression.
System.out.println("Coefficients: "
  + lrModel.coefficients() + " Intercept: " + lrModel.intercept());
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/ml/JavaLinearRegressionWithElasticNetExample.java" in the Spark repo.

More Models available in Spark MLlib

- Classification and Regression Models
  - https://spark.apache.org/docs/latest/ml-classification-regression.html
- ML Pipelines (e.g., Collaborative Filtering, Frequent Pattern Mining)
  - https://spark.apache.org/docs/latest/ml-pipeline.html
- If your team would like to construct Neural Networks, please consider using TensorFlow or PyTorch
  - Your project must use the distributed version of above framework

0

FAQs

- What is the "batch" gradient descent?
- What if my project does not have direct evaluation method?
- Please fill out the peer-evaluation form for the team participation of PA3
  - **Deadline: 11:59AM 11/16/2021 (see the canvas announcement)**
  - **This evaluation will take less than 2-3 minutes**
  - The same evaluation will be conducted after your team completes the term project. It will account for 3% of your final score
  - Contact me if you would like to improve your team effort/collaboration
    - I will help you to manage your team project

1

1

Topics discussed in the previous class

- Predictive analytics
  - Linear Regression with Gradient descent algorithm
  - Implementation with MapReduce and Apache Spark

Topics to be covered

- Google file system

Large Scale Data Storage Systems

Distributed File System

This material is built based on,

- *SANJAY GHEMAWAT, HOWARD GOBIOFF, SHUN-TAK LEUNG: The Google file system. Proceedings of SOSP 2003: 29-43*
- The link to this publication is listed on the course webpage (schedule)
- References: Tom White, "Hadoop: The definitive guide" , 2012

Distributed file systems

- Hadoop vs. Google File System?
- Where are they in our CS435 software stack?

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Introduction

Demand pulls in GFS [1/2]

- Files are huge by traditional standards
- File mutations predominantly through **appends**
  - Not overwrites
- Applications and File system API designed in **lock-step**
- Component failures are the **norm**
  - **Major reasons for the data center outage**
    - **UPS system failure**
    - **Accident and human error**
    - **Cooling system failure**

Backblaze Hard Drives Quarterly Failure Rates for Q1 2021
Reporting period: 1/1/2021 through 3/31/2021 inclusive

MFG	Model	Drive Size	Drive Count	Avg. Age (months)	Drive Days	Drive Failures	AFR
HGST	HMS5C4040ALE640	4TB	3,163	59.4	281,692	5	0.65%
HGST	HMS5C4040BLE640	4TB	12,738	53.4	1,146,496	10	0.32%
HGST	HUH728080ALE600	8TB	1,077	37.7	97,027	4	1.50%
HGST	HUH721212ALE600	12TB	2,599	18.0	233,948	5	0.78%
HGST	HUH721212ALE604	12TB	5,685	3.7	308,793	6	0.71%
HGST	HUH721212ALN604	12TB	10,825	24.0	974,310	9	0.34%
Seagate	ST4000DM000	4TB	18,882	65.3	1,701,967	59	1.27%
Seagate	ST6000DX000	6TB	886	71.8	79,740	-	0.00%
Seagate	ST8000DM002	8TB	9,744	54.1	878,106	26	1.08%
Seagate	ST8000NM0055	8TB	14,419	44.2	1,297,674	31	0.87%
Seagate	ST10000NM0086	10TB	1,200	41.6	108,057	6	2.03%
Seagate	ST12000NM0007	12TB	13,702	31.9	1,732,307	66	1.39%
Seagate	ST12000NM0008	12TB	20,085	12.3	1,764,318	41	0.85%
Seagate	ST12000NM001G	12TB	9,029	7.5	704,446	12	0.62%
Seagate	ST14000NM001G	14TB	5,977	5.9	538,401	13	0.88%
Seagate	ST14000NM0138	14TB	1,675	3.8	135,157	9	2.43%
Seagate	ST16000NM001G	16TB	2,459	1.7	54,177	1	0.67%
Toshiba	MD04ABA400V	4TB	99	70.3	8,910	-	0.00%
Toshiba	MG07ACA14TA	14TB	27,336	8.6	2,165,421	34	0.57%
Toshiba	MG07ACA14TEY	14TB	405	3.8	33,831	1	1.08%
Toshiba	MG08ACA16TEY	16TB	1,014	4.1	91,260	-	0.00%
WDC	WUH721414ALE6L4	14TB	8,400	3.8	640,767	10	0.57%
WDC	WUH721816ALE6L0	16TB	520	0.4	4,853	-	0.00%
		Totals	**171,919**		**14,981,658**	**348**	0.85%

BACKBLAZE

Source: https://www.backblaze.com/blog/backblaze-hard-drive-stats-q1-2021/

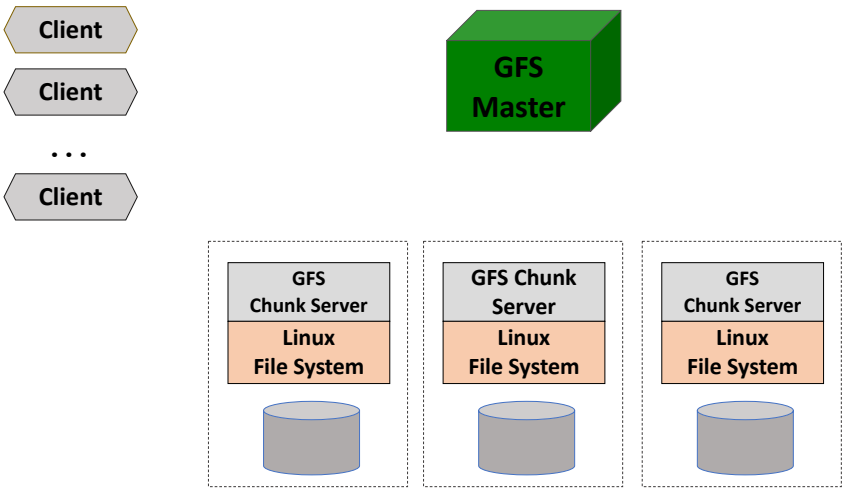
Demand pulls in GFS- [2/2]

- Hundreds of producers will **concurrently append** to a file
  - Many-way merging
- High **sustained bandwidth** is more important
  - than low latency

The file system interface

- Does not implement standard APIs such as POSIX
- Supports **`create, delete, open, close, read`** and **`write`**
- **`snapshot`**
  - Create a fast copy of file and directory tree
- **`record append`**
  - Multiple files can concurrently append records to the same file
    - Without additional locking

Architecture of GFS



Chunks

- Obvious reason
  - The file is too big
- **Set the stage** for computations that operate on this data
  - Parallel I/O
  - I/O seek times are 14 x 10^6 slower than CPU access times

Chunk size

- This is fixed at **128MB (it used to be 64MB)**
  - Much larger than typical FS block sizes
    - Disks are divided with blocks: smallest readable or writable unit
    - All major file systems use a block size of 4096 Bytes or more
    - 512 bytes used to be the standard
- Therefore, in GFS (or in HDFS), 128MB of data chuck is the minimum unit to read/write
  - You cannot randomly access a subset of a chunk
- **Lazy space allocation**
  - Stored as plain Linux file
  - Extended only as needed
  - Avoiding internal fragmentation

Chunk size: But why this big?-Advantage

- **Reduces client interaction** with the master
  - Can cache info for a multi-TB working set
- Reduce network **overhead**
  - With a large chunk, client performs more operations
  - Persistent connections
- Reduce **size of metadata** stored in the master
  - 64 bytes of metadata per 64 MB chunk

Large chunk size: Disadvantage

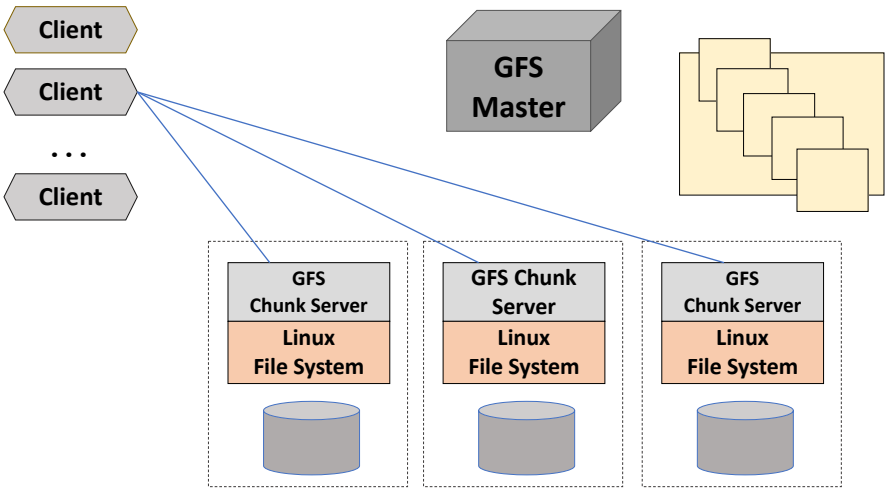
- Small files (with small number of chunks)
  - May **become hot spots**
  - E.g. popular executable files
  - Solution
    - **Assigning a higher replication factor**

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Master Operations

Architecture of GFS



Master operations

- Single master
- Manage system **metadata**
- **Leasing** of chunks
- **Garbage collection** of orphaned chunks
- Chunk **migrations**

Why have a single Master?

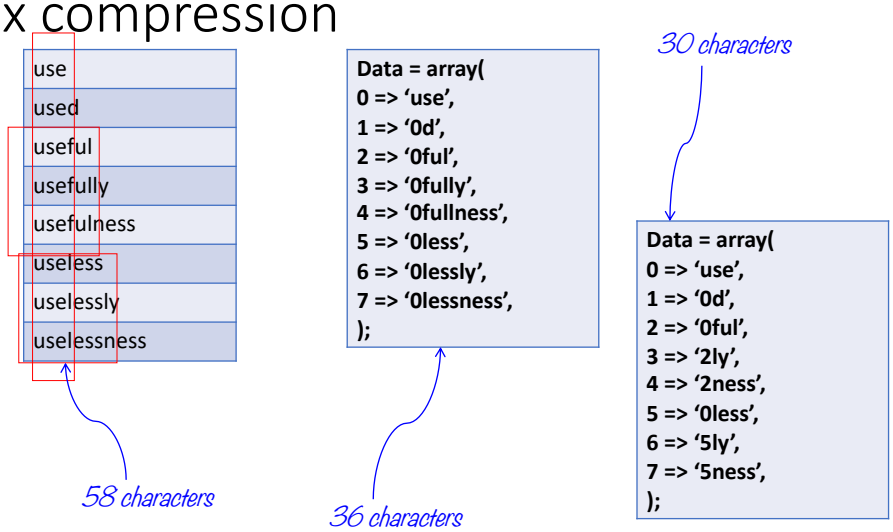
- Vastly **simplifies** design
- Easy to use global knowledge to **reason about**
  - Chunk placements
  - Replication decisions
- Disadvantage
  - **Single-point-of-error**

ALL system metadata is managed by the Master and stored in Main Memory

- File and chunk namespaces
- Mapping from files to chunks
- Location of chunks

Logs mutations into a permanent log

Prefix compression



Why keep the entire metadata in memory?

- **Speed**
- Master can **scan** its **state** in the background
  - Implement chunk garbage collection
  - Re-replicate if there are failures
  - Chunk migration to balance load and space
- **Add** extra memory to increase file system size

Size of the file system with 1 TB of RAM:

Assume file sizes are exact multiples of chunk sizes

- The **master** maintains less than 64 bytes of metadata for each 64MB chunk
- Number of entries = $2^{40}/2^{6}$
- MAXIMUM SIZE of the file system

  = **Number of entries x Chunk size**

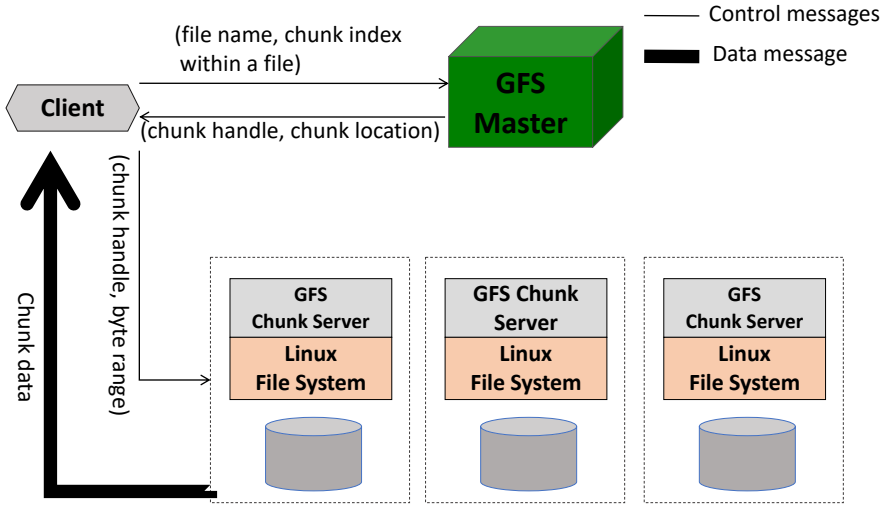
  $= \frac{2^{40}}{2^{6}} \times (2^{6} \times 2^{20})$

  $= 2^{60} = 1$ EB

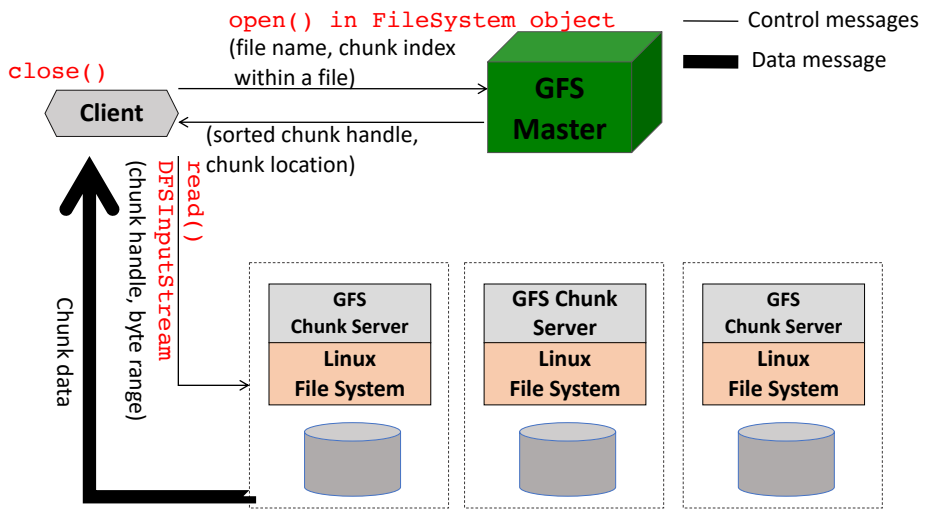
Tracking the chunk servers

- Master **does not** keep a persistent copy of the **location of chunk servers**
- List maintained via **heart-beats**
  - Allows list to be in *sync* with reality despite failures
  - Chunk server has final word on chunks it holds

Simple **read** example



Simple **read** example in HDFS

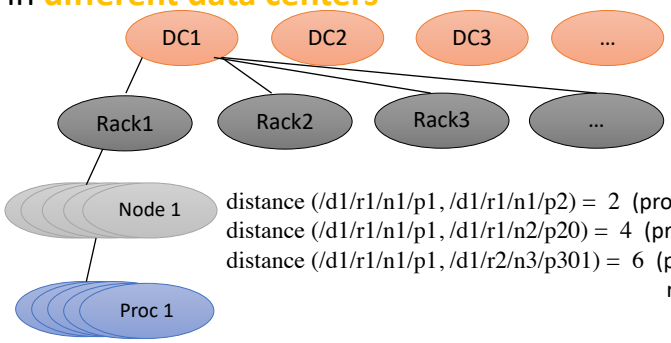


Network topology and HDFS [1/2]

- What does it mean for **two nodes in a local network to be "close"** to each other?
- **Bandwidth between nodes?**
- Hadoop's approach
  - The network is **represented as a tree**
  - The distance between two nodes is **the sum of their distances to their closest common ancestor**

Network topology and HDFS [2/2]

- Processes on the **same node**
- Different nodes on the **same rack**
- Nodes on different racks in the **same data center**
- Nodes in **different data centers**



distance (/d1/r1/n1/p1, /d1/r1/n1/p2) = 2 (processes on the same node)
distance (/d1/r1/n1/p1, /d1/r1/n2/p20) = 4 (processes on the same rack)
distance (/d1/r1/n1/p1, /d1/r2/n3/p301) = 6 (processes on the different rack but in the same data center)

Caching at the client/chunk servers

- Clients **do not cache** file data
  - At client the working set may be *too large*
  - Simplify client; eliminate *cache-coherence* problems
- Chunk servers **do not cache** file data either
  - Chunks are stored as local files
  - Linux's buffer cache *already keeps* frequently accessed data in memory

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Consistency in GFS

What is consistency? (1/4)

(1) **Strong consistency**
- Avoiding all inconsistences

(2) **Update consistency**
- **Write-write conflict**
  - Two people updating the same data
    - at the same time
- Without conflict handling
  - Server will serialize them
  - Decide to apply one then another

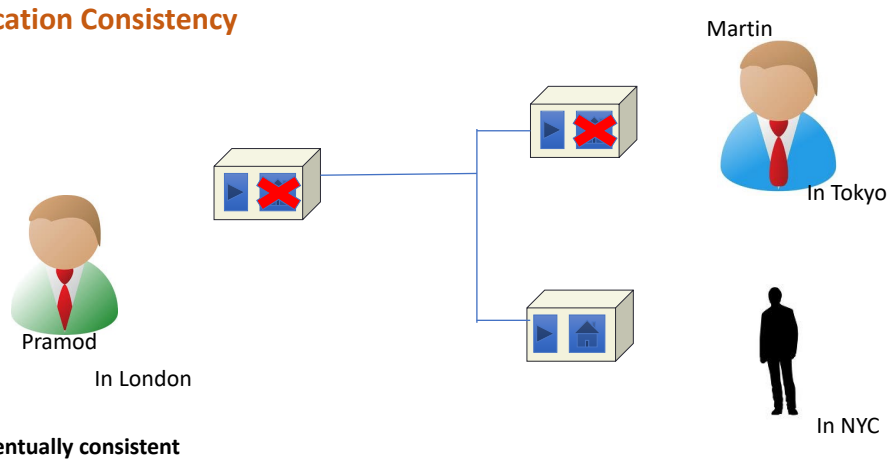
What is consistency? (2/4)

(3) Read consistency

Update consistency does NOT guarantee that readers of that data store will always get consistent responses

What is consistency? (3/4)

(4) Replication Consistency



Eventually consistent

At any time nodes may have replication inconsistencies

-- If there are no more updates, eventually all nodes will be updated to the same value

What is consistency? (4/4)

(5) Session Consistency

- Within a user's session there is read-your-writes consistency
- Sticky session
  - A session that's tied to one node (session affinity)
- Version stamps
  - Ensure every interaction with the data store includes the latest version stamp seen by a session
  - Server node must ensure that it has the updates that include that version stamp before responding to the request

Consistency in the ACID properties

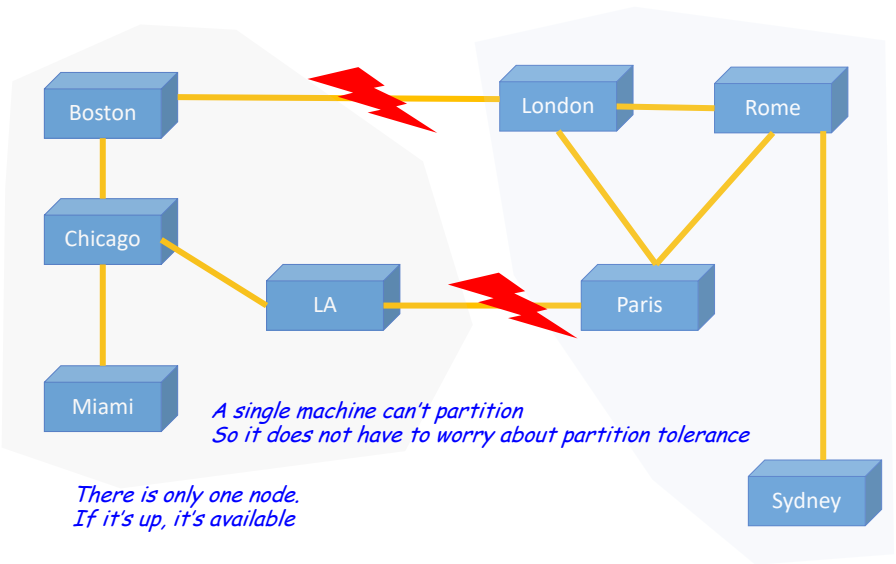
- **Relational databases** allow user to manipulate any combination of rows from any table in a single transaction
- ACID transaction is
  - **Atomic**
  - **Consistent**
  - **Isolated**
  - **Durable**

Relaxed consistency

- **The CAP theorem**
  - Eric Brewer, 2000
  - Seth Gilbert and Nancy Lynch, 2002
    - Formal proof
- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Partition: Two breaks in the communication lines



Relaxed consistency

- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Eventually consistent

- At any time nodes **may have replication inconsistencies**
- If there are no more updates (or updates can be ordered), **eventually all nodes will be updated to the same value**

Mutations in GFS

- **Mutation** changes the content or metadata of a chunk
  - Write
    - Writes data at an application-specific file offset
  - Append
    - Appends atomically *at least once* even in the presence of concurrent mutation
- Each mutation is performed at **all** chunk replicas
- Append is done based **on *record*** (1/4 of a chunk)
  - Offset is specified by GFS

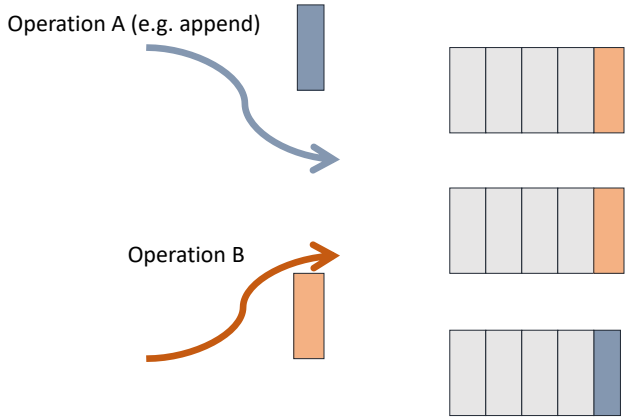
In GFS the state of file region after mutation depends on …

- **Type** of the mutation
- **Success/Failure** of the mutation
- Whether there were **concurren**t mutations

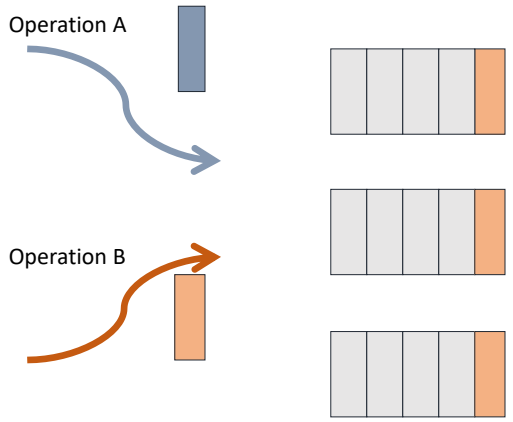
GFS has a relaxed consistency model

- **Consistent**: See the same data
  - On all replicas
- **Defined**: If it is consistent AND
  - Clients see mutation writes in its **entirety**

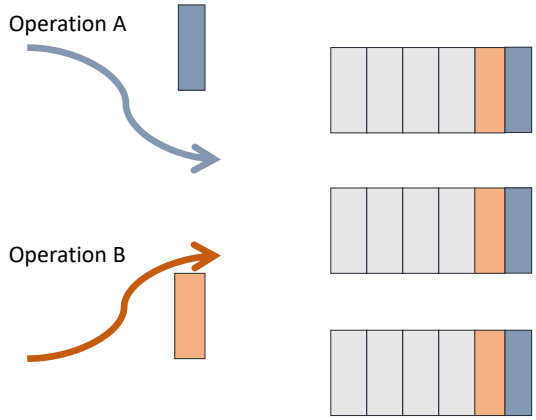
Inconsistent and undefined



Consistent but undefined



Defined



Data mutation

- Write
  - Data to be written at an application-specified **file offset**
- Record appends (in GFS)
  - Data ("record") to be appended **atomically at least once even in the presence of concurrent mutations**
  - **GFS chooses the offset**
  - No need for a distributed lock manger

File state region after a mutation

	Write	Record Append
Serial success	**defined**	**defined** interspersed with *inconsistent*
Concurrent success	Consistent but *undefined*	
Failure	Inconsistent	

For the sequence of successful mutations,

- GFS guarantees the mutated file to be **defined** and to contain the data written by **the last mutation**
  - Applying mutations to a chunk in the **same order on all its replicas**
  - Using chunk version numbers to detect any replica that has become stale
    - If the chunkserver was down

But, clients cache chunk locations…

- What if the chunk location points to a **stale replica** before that information is refreshed?
  - **Cache entry's timeout**
    - Next open to the file will purge all chunk information for that file from the cache
- Most of files are append-only
  - Stale replica usually returns a **premature end** of chunk rather than outdated data

Implications for applications

- Rely on **appends** instead of **overwrites**
- Checkpoint
- Write **records** that are
  - Self-validating
  - Self-identifying

What if many writers concurrently append?

- Append-at-least-once semantics **preserves each writer's output**
- Each record contains extra information like checksums so that its validity can be verified
  - Reader can identify and discard extra padding

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Handling writes and record-appends to a file

GFS uses leases to maintain consistent mutation order across replicas

- Master grants **lease** to one of the replicas
  - **PRIMARY**
- Primary picks **serial-order**
  - For all mutations to the chunk
  - Other replicas *follow* this order
    - When applying mutations

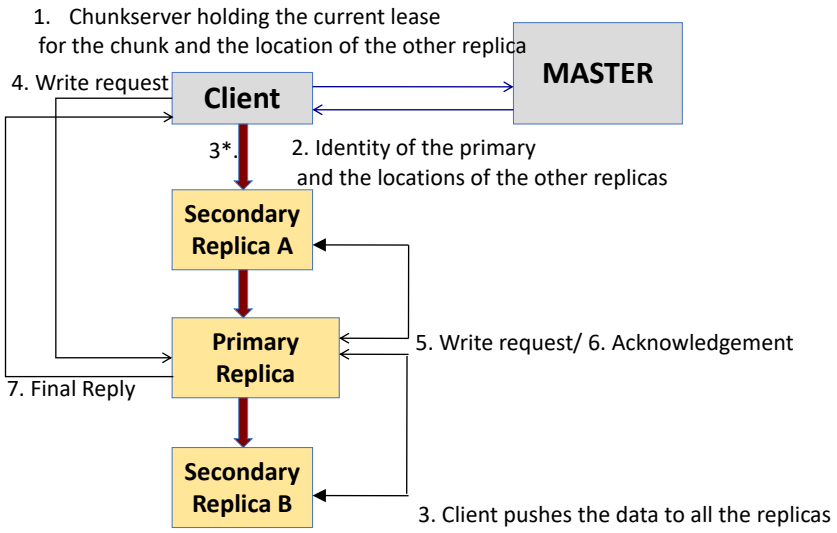
Lease mechanism designed to minimize communications with the master

- Lease has **initial *timeout*** of 60 seconds
- As long as chunk is being mutated
  - Primary can request and receive **extensions**
- Extension requests/grants piggybacked over **heart-beat messages**

Revocation and transfer of leases

- Master may **revoke** a lease before it expires
- If communications lost with primary
  - Master can safely give lease to another replica
    - **ONLY AFTER** the lease period for old primary *elapses*

How a **`write`** is actually performed



Client pushes data to all the replicas [1/2]

- While the data is written, each chunk server stores data in an **LRU buffer** until
  - Data is used
  - Aged out

Client pushes data to all the replicas [2/2]

- When chunk servers acknowledge receipt of data
  - Client sends a ***write*** *request to primary*
- Primary assigns *consecutive serial numbers* to mutations
  - Forwards to replicas

Data flow is decoupled from the control flow to utilize network efficiently [1/2]

- Utilize each machine's network bandwidth
- Avoid network bottlenecks
- Avoid high-latency links
- **Leverage network topology**
  - Estimate distances from IP addresses (in Google's data center)

Data flow is decoupled from the control flow to utilize network efficiently [2/2]

- Pipeline the data transfer
  - Once a chunkserver receives some data, it starts forwarding immediately.
  - For transferring B bytes to R replicas
    - ideal elapsed time will be B/T+RL
    - Where, T is the network throughput
    - L is latency to transfer bytes between two machines

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Record Append and Inconsistency

The control flow for record appends is similar to that of writes

- Google's record appends
  - **Atomic record appends**
- Client pushes data to replicas of the ***last*** chunk of file
- Primary replica checks if the record **fits** in this chunk (64MB)

Primary replica checks if the record append will breach the size (64MB) threshold

- If chunk size would be breached
  - Pad the chunk to maximum size
  - Tell client, that operation should be retried on *next chunk*
- If the record fits, the primary
  - Appends data to its replica
  - Notifies secondaries to **write at the exact offset**

Record sizes and fragmentation

- Size is restricted to ¼ the chunk size (16MB→ 128MB)
- Minimizes worst-case fragmentation
  - Internal fragmentation in each chunk …

What if the secondary replicas could not finish the write operation?

- Client request is considered **failed**
- Modified region is **inconsistent**
  - *No attempt* to delete this from the chunk
  - Client must handle this inconsistency
- Client **retries** the failed mutation

Inconsistent Regions

What if record append fails at one of the replicas

- Client **must retry** the operation
- Replicas of same chunk may contain
  - Different data
  - **Duplicates** of the same record
    - In whole or in part
- Replicas of chunks are **not bit-wise identical**!
  - In most systems, replicas are identical

GFS only guarantees that the data will be written at least once as an atomic unit

- For an operation to return *success*
  - Data must be **written at the same offset** on **all** the replicas
- After the write, all replicas are *as long as* the end of the record
  - Any future record will be assigned a higher offset or a different chunk

GFS client code implements the file system API

- Communications with master and chunk servers done *transparently*
  - On behalf of apps that read or write data
- Interact with master for **metadata**
- **Data-bearing** communications directly to chunk servers

Handling failed "write" in Hadoop HDFS [1/2]

- Different from GFS

1. Pipeline is closed
   - Any packets in the ack queue are added to the front of the data queue
   - Datanodes that are downstream from the failed node will not miss any packets
2. The current block on the good datanodes is given a new identity
   - Reports to the namenode
   - To detect and delete partial block on the failed datanode later on

Handling failed "write" in Hadoop HDFS [2/2]

3. Remainder of the block's data is written to the two good datanodes in the pipeline
4. Namenode notices the block is under-replicated
   - It arranges for a further replica to be created on another node
   - Write quorum
     - `dfs.replication.min` (default to 1)

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Creating Snapshot

Snapshots

- Copying file or directory tree **almost instantaneously**
- Minimizing any **interruptions of ongoing mutations**
- Providing **checkpoint mechanism**
  - Users can commit later
  - Rollback

Snapshots allow you to make a copy of a file very fast

1. Master **revokes** outstanding leases for any chunks of the file (**source**) to be snapshot
2. **Log** the operation to disk
3. Update in-memory state
   1. Duplicate metadata of the source file
4. Newly created file points to the "**same chunks**" as the source

When a client wants to write to a chunk C after the snapshot operation

- Master sees "the *reference count* to **C" > 1**
- Pick **new chunk-handle C'**
- Ask chunk-server with current replica of **C**
  - Create new chunk **C'**
  - Data is *copied locally, not over the network*
- From this point chunk handling of **C'** is no different

GFS does not have a per-directory structure that lists files in the directory

- Name spaces represented as a **lookup** table
  - Maps **full pathnames** to metadata
- Each node has an associated read/write lock
- File creation does not require a lock **on the directory structure**
  - No inode needs to be protected from modification

Each master operation acquires a set of locks before it runs

- **Read lock** prevents a directory/file from being deleted, renamed, or snapshotted
  - Others can still read it
  - Others CANNOT mutate (write/append/delete) this file
- **Write lock** on directory/file names serialize attempts to create a file with the same twice
  - Others CANNOT mutate (write/append/delete) this file
  - Others CANNOT read this file
- If operation involves `/d1/d2/…/dn/leaf`
  - Acquire **read locks** on directory names
    - `/d1, /d1/d2, …, /d1/d2/…/dn`
  - **Read or write lock** on full pathname
    - `/d1/d2/…/dn/leaf`

Locks are used to prevent operations during snapshots

How do we prevent creating /home/user/foo
While /home/user is being snapshotted to /save/user ?

- `/home/user` is being snapshotted to `/save/user`
  - Read locks on `/home` and `/save`
  - Write lock on **`/home/user`** and **`/save/user`**
- To create file, /home/user/foo
  - Read lock on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`
- The two operations will be serialized
  - because they try to obtain **`/home/user`**
- File creation does not require write lock on parent directory … there is no "directory"
  - Read locks on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`

Part 2. Large scale data storage system

Distributed File System

Google File System (GFS): Deletion of files and garbage collection

Garbage collection in GFS

- After a file is deleted, GFS does not reclaim space immediately
- Done **lazily** during garbage collection at
  - File and chunk levels

Master logs a file's deletion immediately

- File is **renamed to a hidden name**
  - Includes deletion timestamp
- Master scans the file system namespace
  - Delete if hidden file existed for <u>more than 3 days</u>
- When file removed from namespace
  - *In memory metadata* is also removed
  - **Severs links** to all its chunks!

Garbage collection: When Master scans its chunk namespace

- Identifies **orphaned chunks**
  - Not reachable from any file
- Erase metadata for these chunks

The role of heart-beats in garbage collection

- Chunk server reports **subset of chunks** it currently has
- Master replies with identity <u>of chunks no longer present</u>
  - Chunk server free to delete its replica of such chunks

Stale chunks and issues

- If a chunk server fails
  - AND misses mutations to the chunk
  - The chunk replica becomes **stale**
- Working with a **stale replica** causes problems with:
  - Correctness
  - Consistency

Aiding the detection of stale chunks

- Master maintains a **chunk version number** for each chunk
  - Distinguish between stale and up-to-date chunks
- When master grants a new lease on chunk
  - Increase version number
  - Inform replicas
  - Record new version persistently

Occurs BEFORE any client can write to chunk

If a replica is unavailable its version number will not be advanced

- When a chunk server restarts, it reports to the Master with the following:
  - Set of Chunks
  - Corresponding version numbers
- Used to **detect** stale replicas
- **Remove stale replicas** in regular garbage collection

Additional safeguards against stale replicas

- Include chunk version number
  - When **client requests** chunk information
    - Client/Chunk server verify version to make sure things are up-to-date
  - During cloning operations
    - Clone the most up-to-date chunk
- Clients and chunk servers expected to **verify** versioning information

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Data Integrity

Data Integrity

- *Impractical* to detect chunk corruptions *across replicas*
  - Not bytewise identical in any case!
- Detection of corruption should be **self-contained**

Data Integrity

- Break chunks into **64 KB** data blocks
- Compute 32-bit checksum for block
  - Keep in chunk server memory
  - Store persistently, **separate** from the data
- Verify checksums of data blocks that **overlap** read range

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Inefficiencies

The master server is a single point of failure

- Master server **restart** takes several seconds
- **Shadow servers** exist
  - Can handle reads of files
    - In place of the master
  - But not writes
- Requires a **massive main memory**

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 13-A Fall 2021 Computer Science

The system is optimized for large files

- But **not for** a very large number of very **small files**
- Primary operation on files
  - Long, sequential reads/writes
  - Large number of **random overwrites** will **clog** things up quite a bit

Consistency Issues: GFS expects clients to resolve inconsistencies

- File chunks may have **gaps or duplicates** of some records
  - The client has to be able to deal with this
- Imagine doing this for a scientific application
  - Portions of a massive array are corrupted
    - Clients would have to detect this
- HDFS does **NOT** have this problem

Security model

- **None**
  - Operation is expected to be in a trusted environment

Questions?

96

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 13-A Fall 2021 Computer Science

Questions?

97

0

FAQs

- Can we use Hadoop/Spark and then process the reduced dataset with python or R or Scikit?
  - YES!

1

1

Topics discussed in the previous class

- Google file system
  - Overview
  - Consistency
  - Master operations

Topics to be covered

- Google file system
  - continued

Large Scale Data Storage Systems

Distributed File System

This material is built based on,

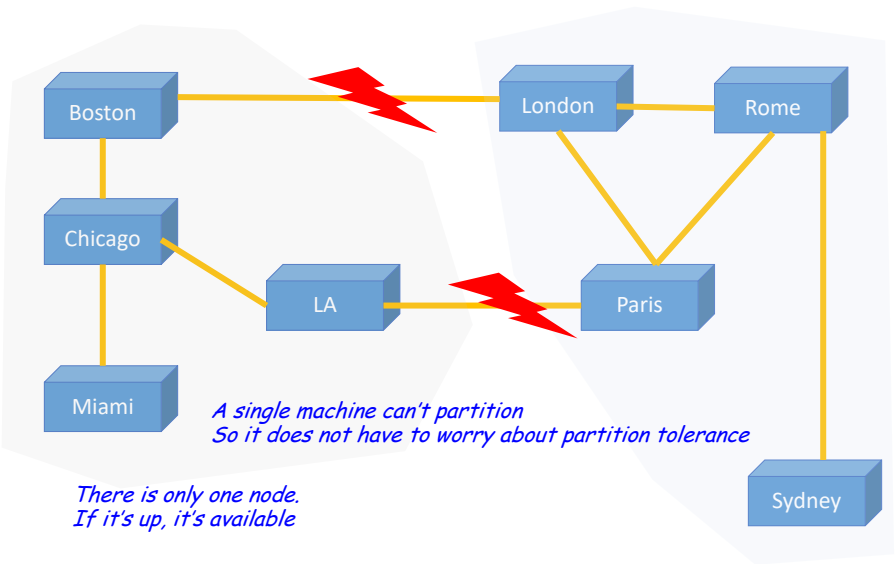
- *SANJAY GHEMAWAT, HOWARD GOBIOFF, SHUN-TAK LEUNG: The Google file system. Proceedings of SOSP 2003: 29-43*
- The link to this publication is listed on the course webpage (schedule)
- References: Tom White, "Hadoop: The definitive guide" , 2012

Relaxed consistency

- **The CAP theorem**
  - Eric Brewer, 2000
  - Seth Gilbert and Nancy Lynch, 2002
    - Formal proof
- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Partition: Two breaks in the communication lines



Relaxed consistency

- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Eventually consistent

- At any time nodes **may have replication inconsistencies**
- If there are no more updates (or updates can be ordered), **eventually all nodes will be updated to the same value**

Mutations in GFS

- **Mutation** changes the content or metadata of a chunk
  - Write
    - Writes data at an application-specific file offset
  - Append
    - Appends atomically *at least once* even in the presence of concurrent mutation
    - HDFS: data will be appeared exactly once
- Each mutation is performed at **all** chunk replicas
- Append is done based **on *record*** (1/4 of a chunk)
  - Offset is specified by GFS

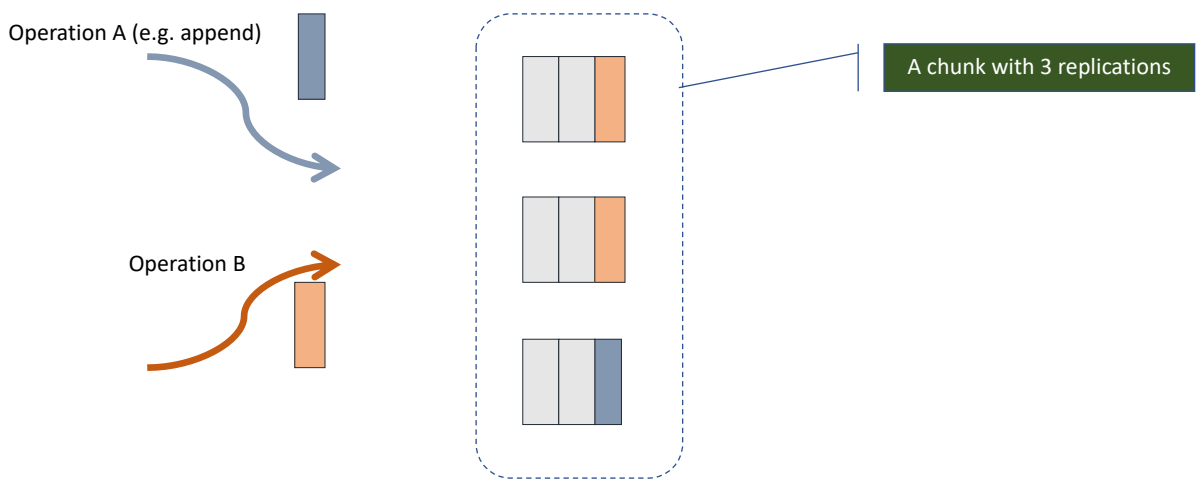
In GFS the state of file region after mutation depends on …

- **Type** of the mutation
- **Success/Failure** of the mutation
- Whether there were **concurren**t mutations

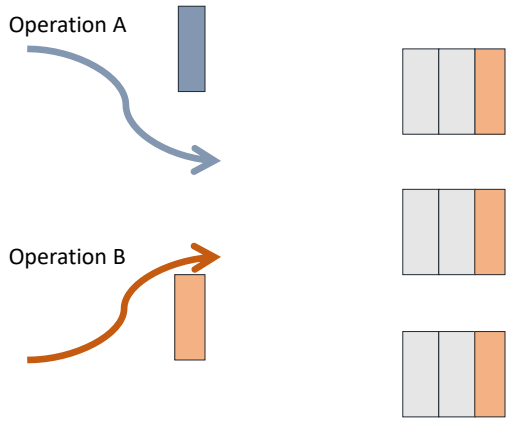
GFS has a relaxed consistency model

- **Consistent**: See the same data
  - On all replicas
- **Defined**: If it is consistent AND
  - Clients see mutation writes in its **entirety**

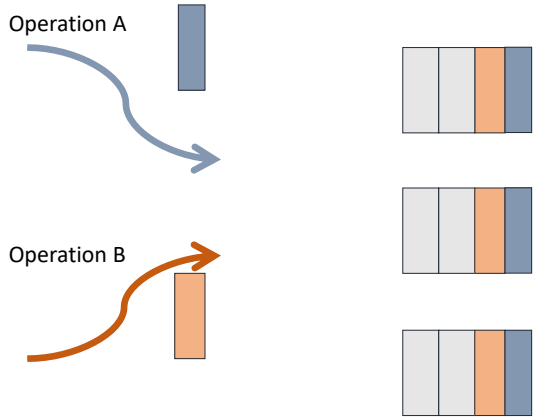
Inconsistent and undefined



Consistent but undefined



Defined



Data mutation

- Write
  - Data to be written at an application-specified **file offset**
- Record appends (in GFS)
  - Data ("record") to be appended **atomically at least once even in the presence of concurrent mutations**
  - **GFS chooses the offset**
  - No need for a distributed lock manger

File state region after a mutation in GFS

	Write	Record Append	HDFS
Serial success	**defined**	**defined** interspersed with *inconsistent*	Consistent and defined
Concurrent success	Consistent but *undefined*		
Failure	Inconsistent		inconsistent

For the sequence of successful mutations,

- GFS guarantees the mutated file to be **defined** and to contain the data written by **the last mutation**
  - Applying mutations to a chunk in the **same order on all its replicas**
  - Using chunk version numbers to detect any replica that has become stale
    - If the chunkserver was down

But, clients cache chunk locations…

- What if the chunk location points to a **stale replica** before that information is refreshed?
  - **Cache entry's timeout**
    - Next open to the file will purge all chunk information for that file from the cache
- Most of files are append-only
  - Stale replica usually returns a **premature end** of chunk rather than outdated data

Implications for applications

- Rely on **appends** instead of **overwrites**
- Checkpoint
- Write **records** that are
  - Self-validating
  - Self-identifying

What if many writers concurrently append?

- Append-at-least-once semantics **preserves each writer's output**
- Each record contains extra information like checksums so that its validity can be verified
  - Reader can identify and discard extra padding

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Handling writes and record-appends to a file

GFS uses leases to maintain consistent mutation order across replicas

- Master grants **lease** to one of the replicas
  - **PRIMARY**
- Primary picks **serial-order**
  - For all mutations to the chunk
  - Other replicas *follow* this order
    - When applying mutations

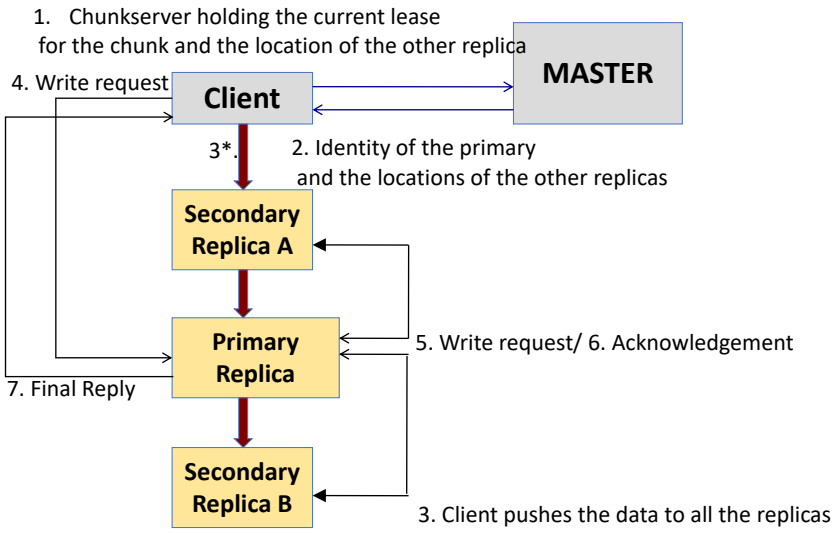
Lease mechanism designed to minimize communications with the master

- Lease has **initial *timeout*** of 60 seconds
- As long as chunk is being mutated
  - Primary can request and receive **extensions**
- Extension requests/grants piggybacked over **heart-beat messages**

Revocation and transfer of leases

- Master may **revoke** a lease before it expires
- If communications lost with primary
  - Master can safely give lease to another replica
    - **ONLY AFTER** the lease period for old primary *elapses*

How a **`write`** is actually performed



Client pushes data to all the replicas [1/2]

- While the data is written, each chunk server stores data in an **LRU buffer** until
  - Data is used
  - Aged out

Client pushes data to all the replicas [2/2]

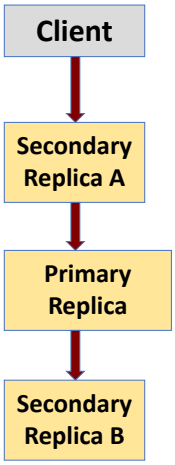
- When chunk servers acknowledge receipt of data
  - Client sends a ***write*** *request to primary*
- Primary assigns *consecutive serial numbers* to mutations
  - Forwards to replicas

Data flow is decoupled from the control flow to utilize network efficiently [1/2]

- Utilize each machine's network bandwidth
- Avoid network bottlenecks
- Avoid high-latency links
- **Leverage network topology**
  - Estimate distances from IP addresses (in Google's data center)

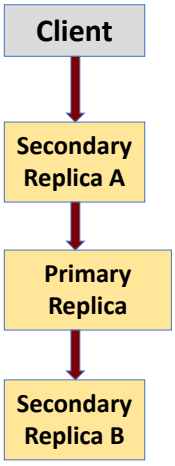
Data flow is decoupled from the control flow to utilize network efficiently [2/2]

- Pipeline the data transfer
  - Once a chunkserver receives some data, it starts forwarding immediately.
  - For transferring B bytes to R replicas
    - ideal elapsed time will be B/T+2RL
    - Where, T is the network throughput
    - L is latency to transfer bytes between two machines



Example

- Assume that the network links are 1 Gbit/sec and 3 replicas
- The latency to transfer bytes between two machines is far below 1ms
- To write 1MB of data
- 8x1MB/1G +2x1(ms) = 10 ms



Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Record Append and Inconsistency

The control flow for record appends is similar to that of writes

- Google's record appends
  - **Atomic record appends**
- Client pushes data to replicas of the ***last*** chunk of file
- Primary replica checks if the record **fits** in this chunk (64MB)

Primary replica checks if the record append will breach the size threshold

- If chunk size would be breached
  - Pad the chunk to maximum size
  - Tell client, that operation should be retried on *next chunk*
- If the record fits, the primary
  - Appends data to its replica
  - Notifies secondaries to **write at the exact offset**

Record sizes and fragmentation

- Size is restricted to ¼ the chunk size (16MB→ 128MB)
- Minimizes worst-case fragmentation
  - Internal fragmentation in each chunk …

What if the secondary replicas could not finish the write operation?

- Client request is considered **failed**
- Modified region is **inconsistent**
  - *No attempt* to delete this from the chunk
  - Client must handle this inconsistency
- Client **retries** the failed mutation

Inconsistent Regions

What if record append fails at one of the replicas

- Client **must retry** the operation
- Replicas of same chunk may contain
  - Different data
  - **Duplicates** of the same record
    - In whole or in part
- Replicas of chunks are **not bit-wise identical**!
  - In most systems, replicas are identical

GFS only guarantees that the data will be written at least once as an atomic unit

- For an operation to return *success*
  - Data must be **written at the same offset** on **all** the replicas
- After the write, all replicas are *as long as* the end of the record
  - Any future record will be assigned a higher offset or a different chunk

GFS client code implements the file system API

- Communications with master and chunk servers done *transparently*
  - On behalf of apps that read or write data
- Interact with master for **metadata**
- **Data-bearing** communications directly to chunk servers

Handling failed "write" in Hadoop HDFS [1/2]

- Different from GFS

1. Pipeline is closed
   - Any packets in the ack queue are added to the front of the data queue
   - Datanodes that are downstream from the failed node will not miss any packets
2. The current block on the good datanodes is given a new identity
   - Reports to the namenode
   - To detect and delete partial block on the failed datanode later on

Handling failed "write" in Hadoop HDFS [2/2]

3. Remainder of the block's data is written to the two good datanodes in the pipeline

4. Namenode notices the block is under-replicated
   - It arranges for a further replica to be created on another node
   - Write quorum
     - `dfs.replication.min` (default to 1)

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Creating Snapshot

Snapshots

- Copying file or directory tree **almost instantaneously**
- Minimizing any **interruptions of ongoing mutations**
- Providing **checkpoint mechanism**
  - Users can commit later
  - Rollback

Snapshots allow you to make a copy of a file very fast

1. Master **revokes** outstanding leases for any chunks of the file (**source**) to be snapshot
2. **Log** the operation to disk
3. Update in-memory state
   1. Duplicate metadata of the source file
4. Newly created file points to the "**same chunks**" as the source

When a client wants to write to a chunk C after the snapshot operation

- Master sees "the *reference count* to **C" > 1**
- Pick **new chunk-handle C'**
- Ask chunk-server with current replica of **C**
  - Create new chunk **C'**
  - Data is *copied locally, not over the network*
- From this point chunk handling of **C'** is no different

GFS does not have a per-directory structure that lists files in the directory

- Name spaces represented as a **lookup** table
  - Maps **full pathnames** to metadata
- Each node has an associated read/write lock
- File creation does not require a lock **on the directory structure**
  - No inode needs to be protected from modification

Each master operation acquires a set of locks before it runs

- **Read lock** prevents a directory/file from being deleted, renamed, or snapshotted
  - Others can still read it
  - Others CANNOT mutate (write/append/delete) this file
- **Write lock** on directory/file names serialize attempts to create a file with the same twice
  - Others CANNOT mutate (write/append/delete) this file
  - Others CANNOT read this file
- If operation involves `/d1/d2/…/dn/leaf`
  - Acquire **read locks** on directory names
    - `/d1, /d1/d2, …, /d1/d2/…/dn`
  - **Read or write lock** on full pathname
    - `/d1/d2/…/dn/leaf`

Locks are used to prevent operations during snapshots

How do we prevent creating /home/user/foo
While /home/user is being snapshotted to /save/user ?

- `/home/user` is being snapshotted to `/save/user`
  - Read locks on `/home` and `/save`
  - Write lock on **`/home/user`** and **`/save/user`**
- To create file, /home/user/foo
  - Read lock on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`
- The two operations will be serialized
  - because they try to obtain **`/home/user`**
- File creation does not require write lock on parent directory … there is no "directory"
  - Read locks on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Deletion of files and garbage collection

Garbage collection in GFS

- After a file is deleted, GFS does not reclaim space immediately
- Done **lazily** during garbage collection at
  - File and chunk levels

Master logs a file's deletion immediately

- File is **renamed to a hidden name**
  - Includes deletion timestamp
- Master scans the file system namespace
  - Delete if hidden file existed for more than 3 days
- When file removed from namespace
  - *In memory metadata* is also removed
  - **Severs links** to all its chunks!

Garbage collection: When Master scans its chunk namespace

- Identifies **orphaned chunks**
  - Not reachable from any file
- Erase metadata for these chunks

The role of heart-beats in garbage collection

- Chunk server reports **subset of chunks** it currently has
- Master replies with identity of chunks no longer present
  - Chunk server free to delete its replica of such chunks

Stale chunks and issues

- If a chunk server fails
  - AND misses mutations to the chunk
  - The chunk replica becomes **stale**
- Working with a **stale replica** causes problems with:
  - Correctness
  - Consistency

Aiding the detection of stale chunks

- Master maintains a **chunk version number** for each chunk
  - Distinguish between stale and up-to-date chunks
- When master grants a new lease on chunk
  - Increase version number
  - Inform replicas
  - Record new version persistently

Occurs BEFORE any client can write to chunk

If a replica is unavailable its version number will not be advanced

- When a chunk server restarts, it reports to the Master with the following:
  - Set of Chunks
  - Corresponding version numbers
- Used to **detect** stale replicas
- **Remove stale replicas** in regular garbage collection

Additional safeguards against stale replicas

- Include chunk version number
  - When **client requests** chunk information
    - Client/Chunk server verify version to make sure things are up-to-date
  - During cloning operations
    - Clone the most up-to-date chunk
- Clients and chunk servers expected to **verify** versioning information

Questions?