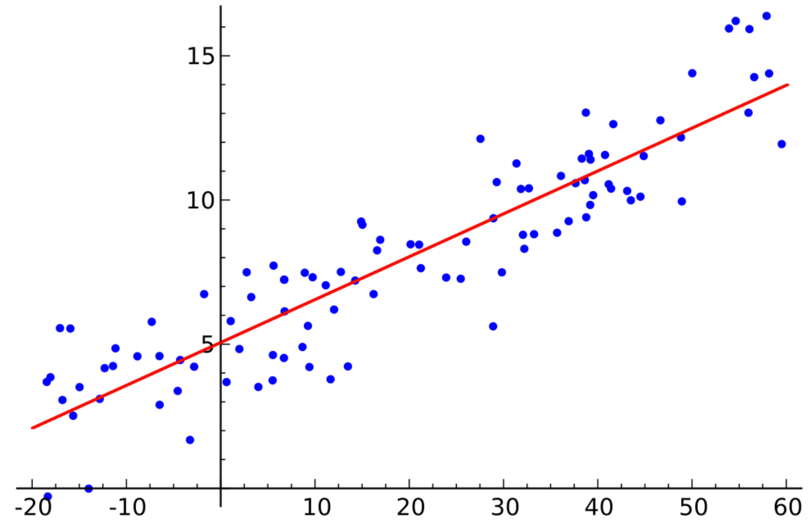
EECE5698
Parallel Processing for Data Analytics

Lecture 11: Sparsity and Parallelism

Loss Functions

$$\min_{\beta \in \mathbb{R}^d} F(\beta)$$

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$



- If ℓ is convex, so is $F(\beta)$
- Examples:
  - Squared loss $\ell(\beta; x, y) = (y - \beta^\top x)^2$
  - Logistic $\ell(\beta; x, y) = \log(1 + \exp(-y\beta^\top x))$
  - Hinge $\ell(\beta; x, y) = \max(0, 1 - y\beta^\top x)$

Northeastern

Parallel Computation of Gradient Descent

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$

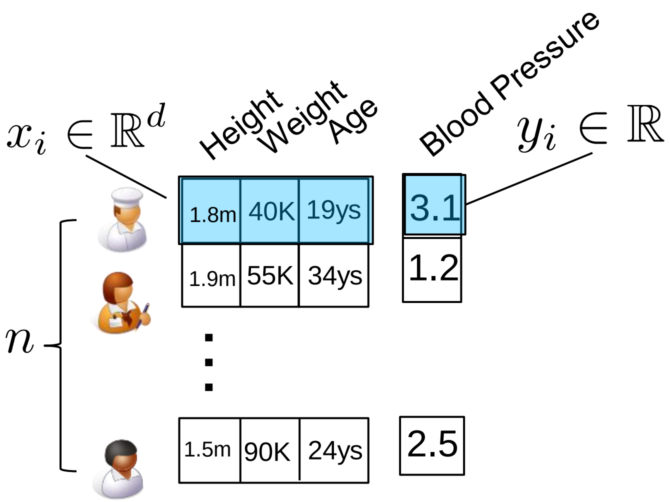
- ❑ Track Objective
- ❑ Backtracking Line Search

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i)$$

- ❑ GD step
- ❑ Convergence Criterion

Parallel Computation

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$

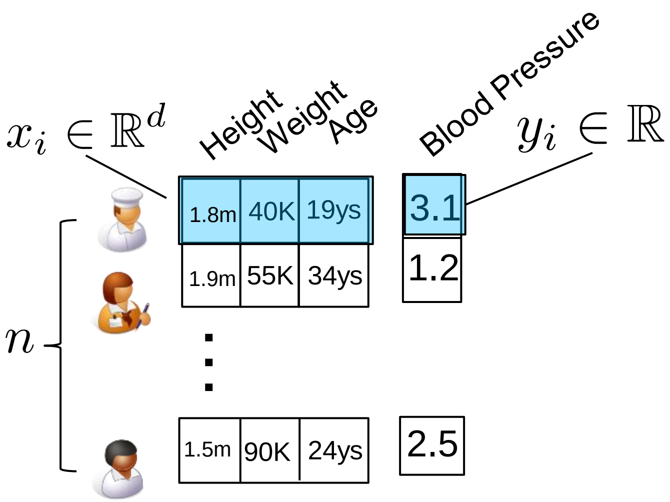



```
rdd = [(x1,y1),
        (x2,y2),
        …
        (xn,yn)]

beta = np.array([0.1,0.4,-2.0])

rdd.map( lambda (x,y):
                loss(beta,x,y))\
        .reduce(add)
```

- ❑ **Broadcasts O(d) sized vector**
- ❑ **Reduction msg size O(1)**

Parallel Computation

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i)$$




```
rdd = [(x1,y1),
       (x2,y2),
        …
       (xn,yn)]

beta = np.array([0.1,0.4,-2.0])

rdd.map( lambda (x,y):
                 gradLoss(beta,x,y))\
         .reduce(add)
```

- ❑ **Broadcasts O(d) sized vector**
- ❑ **Reduction msg size O(d)**

Let's be a bit more precise

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i)$$

m workers

myrdd

worker

```
[(x1,y1),
 (x2,y2),
 (x3,y2),
…,
(x333,y333) ]
```

worker

```
[(x334,y334),
 (x335,y335),
…,
(x665,y665)]
```

worker

```
[(x666,y666),
…,
(x1000,y1000)]
```

```
rdd = [(x1,y1),
       (x2,y2),
        …
       (xn,yn)]

beta = np.array([0.1,0.4,-2.0])

rdd.map( lambda (x,y):
             gradLoss(beta,x,y))\
      .reduce(add)
```

m messages of size d

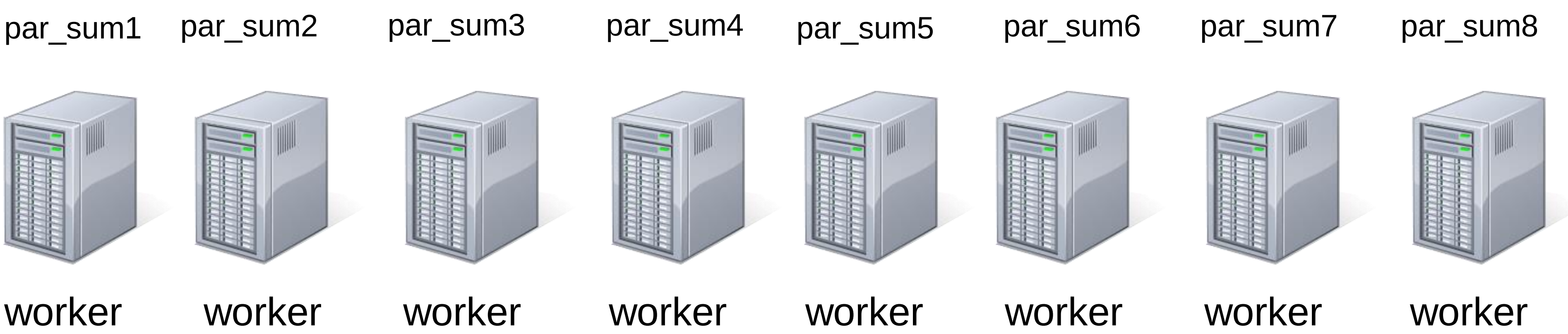
Computation parallelized over m machines

m messages of size d

Northeastern

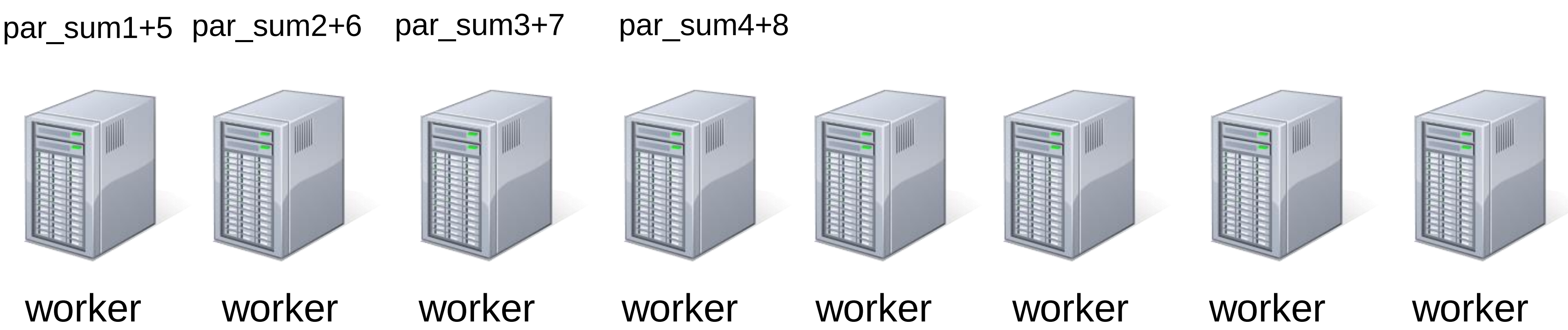
How does `reduce` work?

Round 1: Move results to n/2 processors



How does `reduce` work?

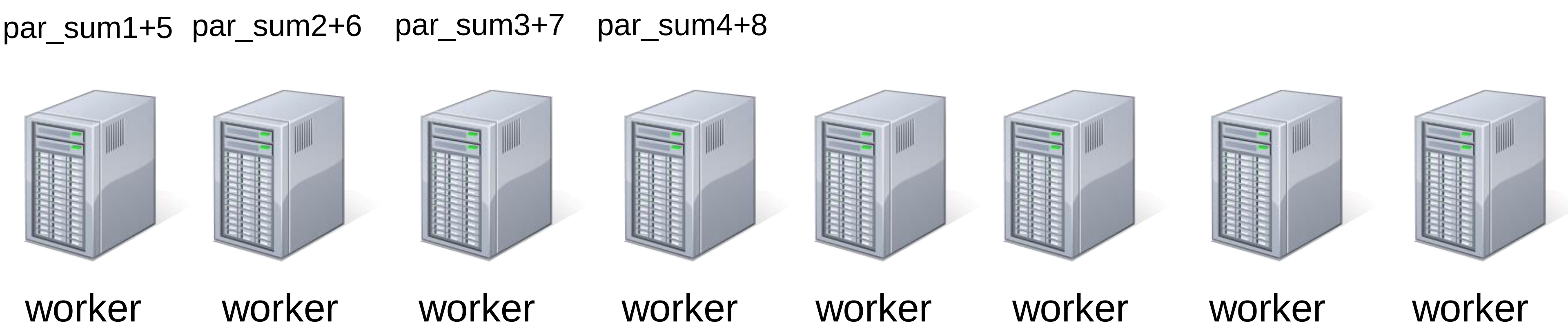
Round 1: Combine results



Northeastern

How does `reduce` work?

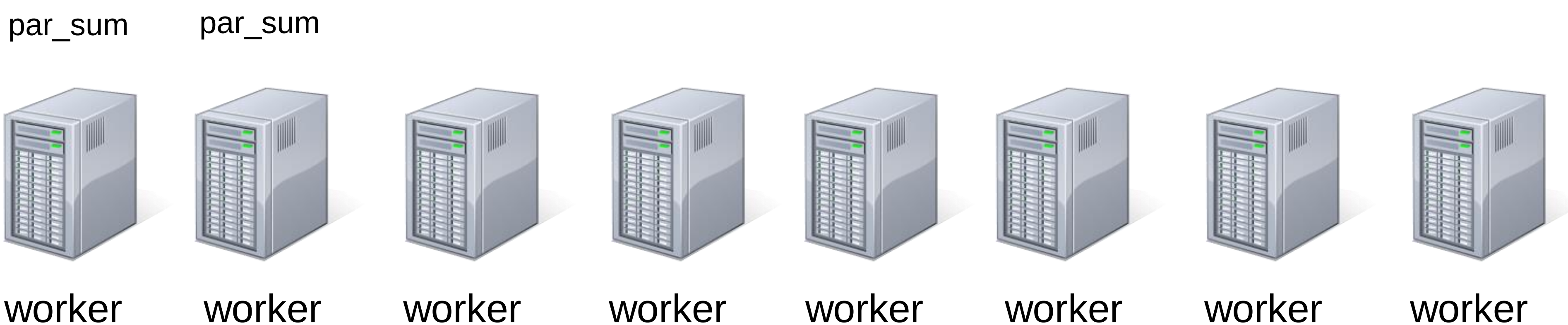
Round 2: Repeat



How does `reduce` work?

Round 3: Repeat

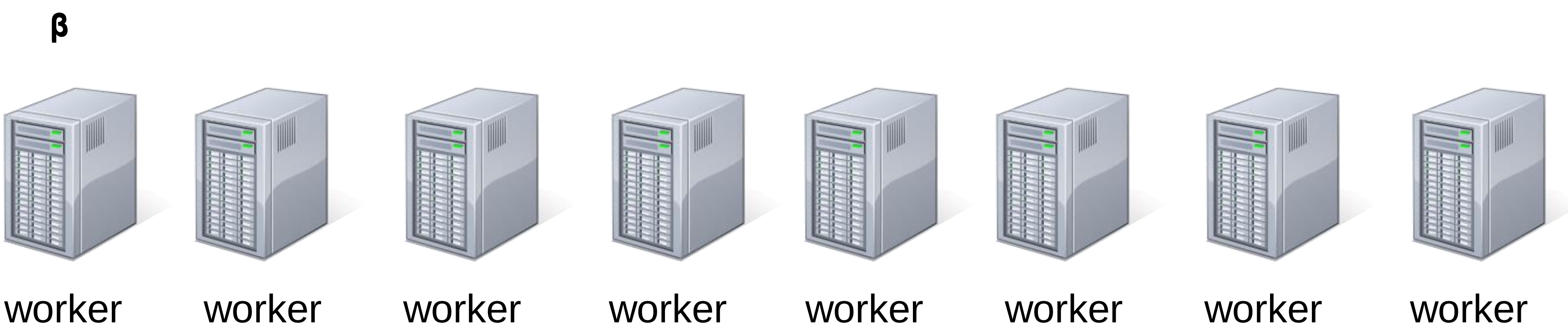
m processors:
m-1 messages
log m rounds



Northeastern

How does `broadcast` work? Reverse Process!

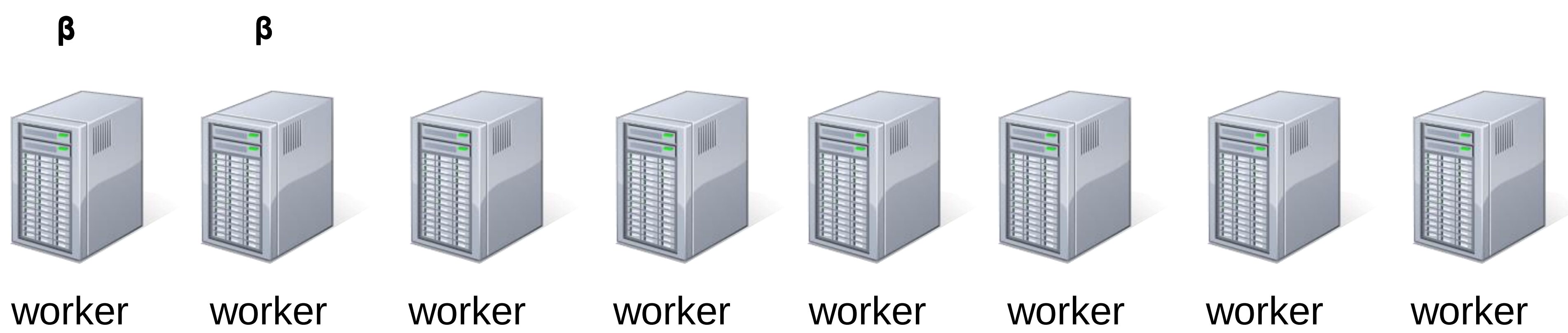
Round 3: Repeat



Northeastern

How does `broadcast` work? Reverse Process!

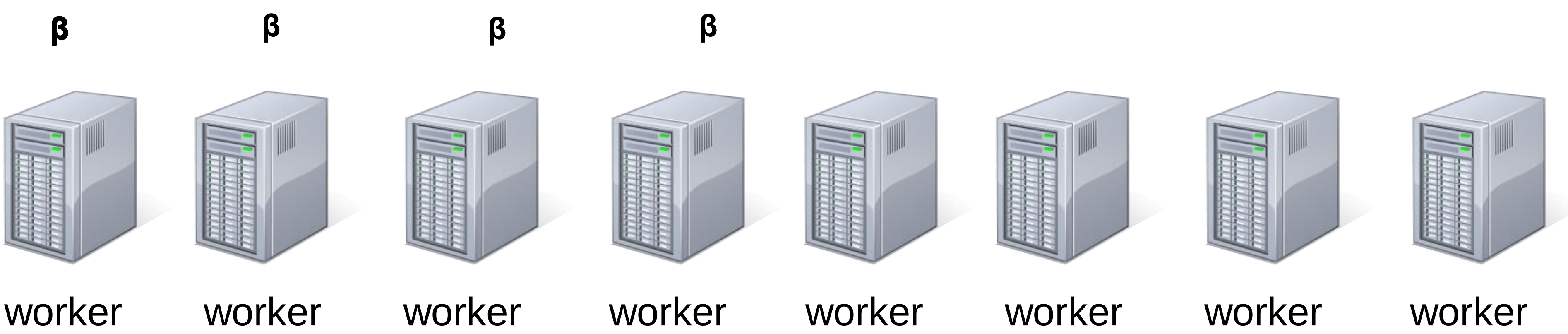
Round 3: Repeat



How does `broadcast` work? Reverse Process!

Round 3: Repeat

m processors:
m-1 messages
log m rounds



Communication and Computation

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i)$$

Suppose that the time necessary to compute

$$\nabla_\beta \ell(\beta; x, y)$$

on a single point is **c**

Serial Time (no spark, single machine)

Total computation time: **n * c**

Total communication time: 0

Parallel time with m workers

Total computation time**: n * c/m**

Total communication: **2***(**m-1) * d** messages

Total communication time: ~**log m *d** (ideal)

in reality, it is higher

Increasing **m** decreases computation, but increases communication!

For large **d**, this can be quite high

Problem with Generic Model

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i)$$

m workers

myrdd

worker

```
[(x1,y1),
 (x2,y2),
 (x3,y2),
…,
(x333,y333) ]
```

worker

```
[(x334,y334),
 (x335,y335),
…,
(x665,y665)]
```

worker

```
[(x666,y666),
…,
(x1000,y1000)]
```

When d is large, beta should be an RDD!

```
rdd = [(x1,y1),
       (x2,y2),
       …
       (xn,yn)]

beta = np.array([0.1,0.4,…,-2.0])

rdd.map( lambda (x,y):
              gradLoss(beta,x,y))\
      .reduce(add)
```

Works well for **n large** and **d small**, but not well when **both n** and **d are big**.

Northeastern

Main Challenge

Reduce the number of messages to something less than

m*d

Northeastern

Sparsity Revisited

$$\min_{\beta \in \mathbb{R}^d} F(\beta) \qquad\qquad F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$

$$\ell(\beta; x, y) = \ell(\beta^\top x, y)$$

- Examples:
  - Squared loss $\qquad \ell(\beta; x, y) = (y - \beta^\top x)^2$
  - Logistic $\qquad \ell(\beta; x, y) = \log(1 + \exp(-y\beta^\top x))$
  - Hinge $\qquad \ell(\beta; x, y) = \max(0, 1 - y\beta^\top x)$

Suppose x is sparse

0	0.2	0	0	0	0	-0.3	0	0	0	1.4	0	0	2.1	0	0	0	-2.7

Then, both ℓ and $\nabla_\beta \ell$ depend only on a small subset of coordinates of β!

Northeastern

Sparsity: Computing Loss at a Datapoint

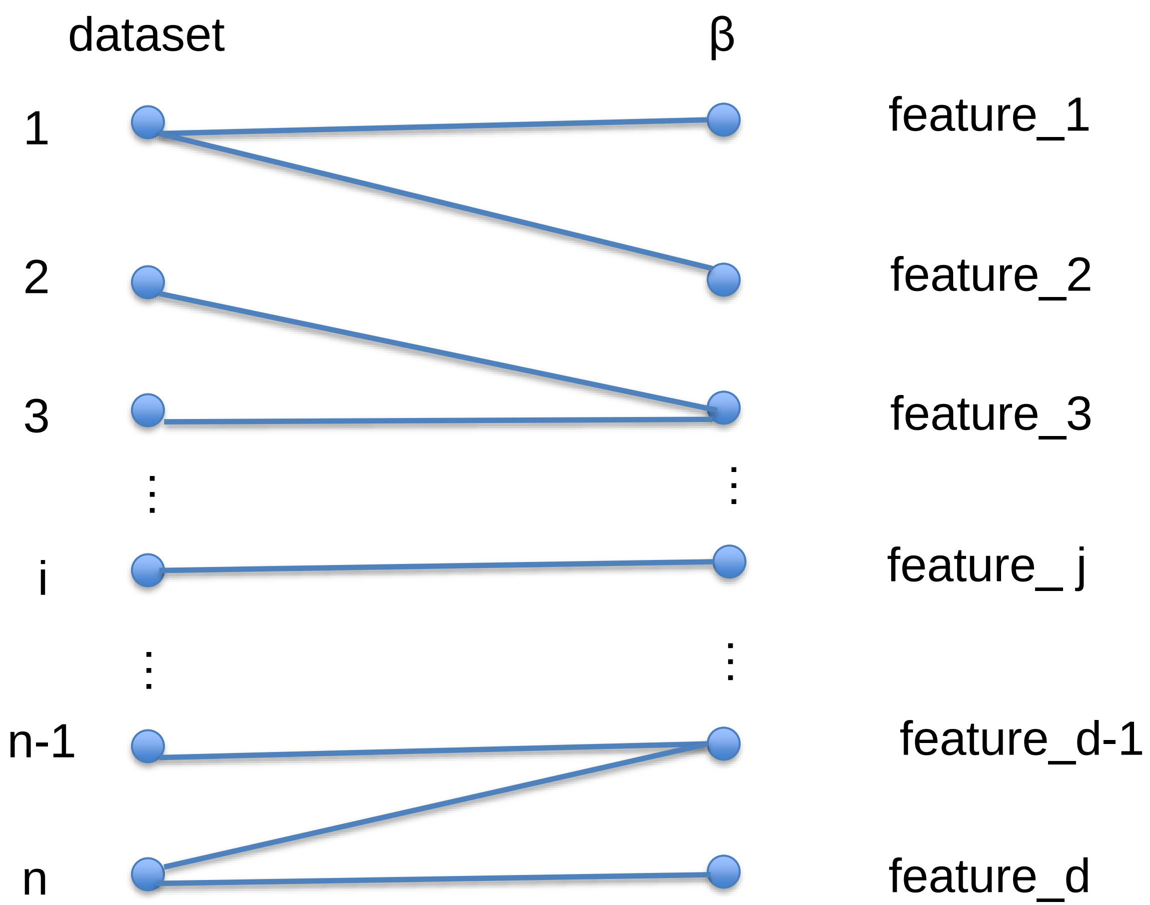
$$\ell_i(\beta) = \ell(\beta^\top x_i; y_i)$$

$$= \ell\Big(\sum_{j : x_{ij} \neq 0} \beta_j x_{ij}; y_i\Big)$$

x_i	0	0.2	0	0	0	0	-0.3	0	0	0	1.4	0	0	2.1	0	0	0	2.7
β	1	4.2	-1	3	2.2	.4	0.3	0.2	1	0	2.0	4	2.1	3.1	1.0	4.	5.0	-1.2

Sparsity: Dependence Graph

Suppose that each feature vector has at most k non-zero features



Dependence Graph

Connect datapoint i to feature j iff x_ij is not 0

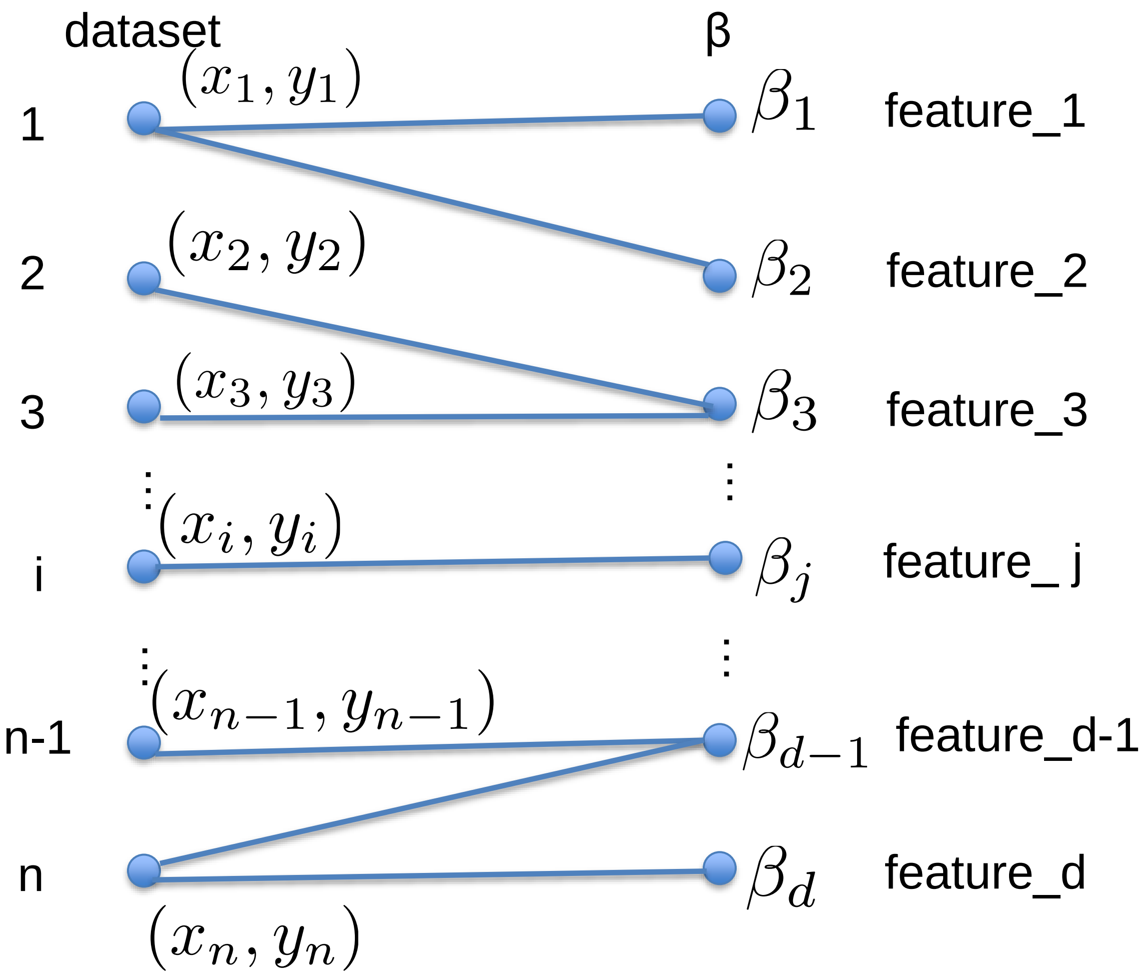
n nodes on the left
d nodes on the right

At most k*n edges

Northeastern

Sparsity: Dependence Graph

Suppose that each feature vector has at most k non-zero features



Dependence Graph

Connect datapoint i to feature j iff x_ij is not 0

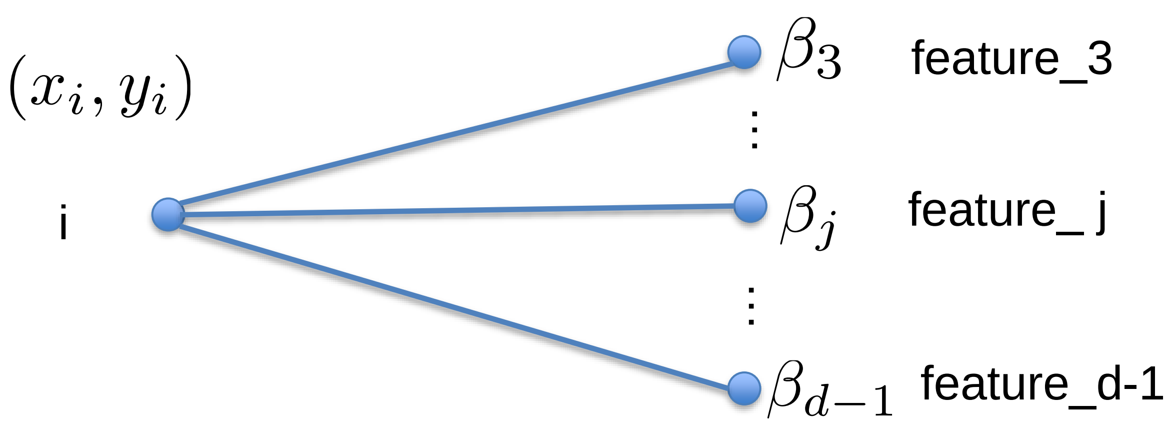
n nodes on the left
d nodes on the right

At most k*n edges

Northeastern

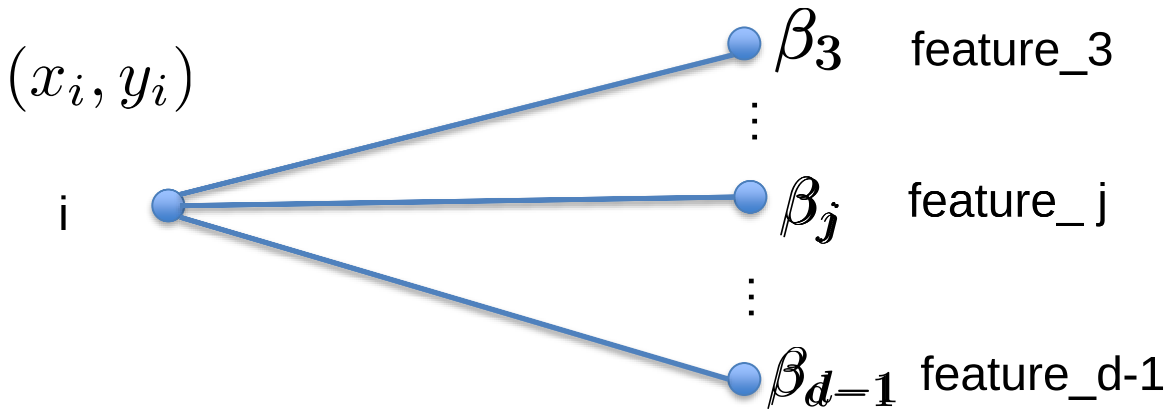
Computing Loss over Dependence Graph

$$\ell_i(\beta) = \ell(\beta^\top x_i; y_i)$$
$$= \ell\Big(\sum_{j:x_{ij}\neq 0} \beta_j x_{ij}; y_i\Big)$$



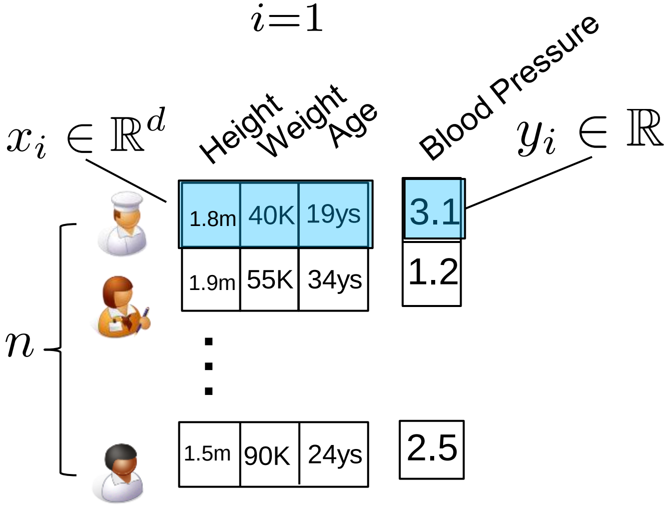
Computing Loss over Dependence Graph

$$\ell_i(\beta) = \ell(\beta^\top x_i; y_i)$$
$$= \ell\Big(\sum_{j:x_{ij}\neq 0} \beta_j x_{ij}; y_i\Big)$$



Parallel Computation Through Dependence Graph

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$

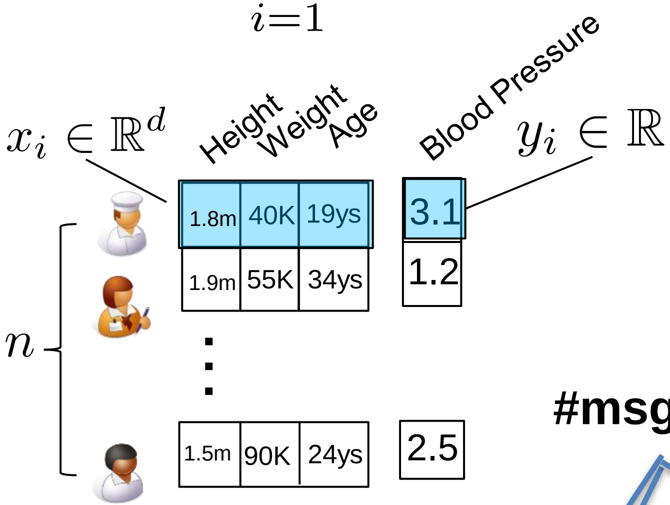



```
dataRDD = [ (1,(x1,y1)),
            (2,(x2,y2)),
            …
            (n,(xn,yn))]

betaRDD = [ (feature1, val1),          # β is an rdd!
            (feature2, val2),
            …
            (featured, vald)]

GRDD= [ (1, feature1), (1, feature7), …
        (2, feature16), (2, feature22),…
        …
        (n, feature2),(n,feature99)]

tmp = GRDD.map(swap).join(betaRDD)\                                           # (feature, (i, value))
      .map(lambda (f,(i,val)): (i, (f,val))\                                   # (i, (feature,value))
      .mapValues(lambda  (f,val): SparseVector({f:val})).reduceByKey(add)       # (i, SparseVector())

return dataRDD.join(tmp).values()\                                             # ( (xi,yi) , SparseVector() )
              .mapValues(evaluateLoss).reduce(add)
```


Northeastern

Parallel Computation Through Dependence Graph

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$



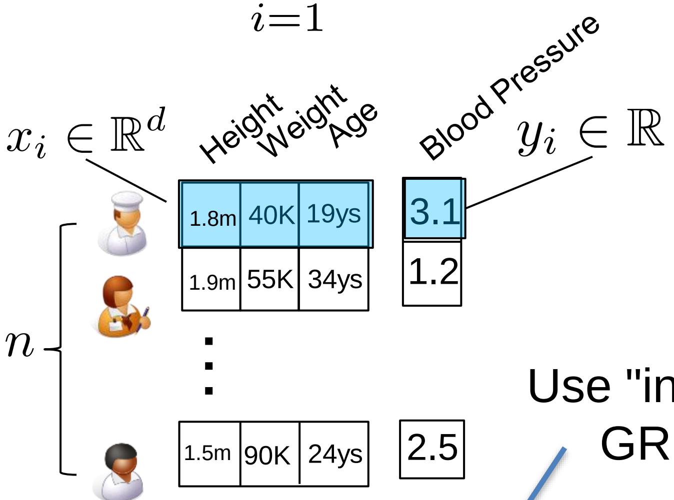

```
dataRDD = [ (1,(x1,y1)),
            (2,(x2,y2)),
            …
            (n,(xn,yn))]

betaRDD = [ (feature1: val1),          # β is an rdd!
            (feature2: val2),
            …
            (featured: vald)]

GRDD= [ (1, feature1), (1, feature7), …
        (2, feature16), (2, feature22),…
        …
        (n, feature2),(n,feature99)]
```

#msg < n*k+d

```
tmp = GRDD.map(swap).join(betaRDD)\                                   # (feature, (i, value))
      .map(lambda (f,(i,val)): (i, (f,val))\                          # (i, (feature,value))
      .mapValues(lambda  (f,val): SparseVector({f:val})).reduceByKey(add)   # (i, SparseVector())
return dataRDD.join(tmp).values()\                                    # ( (xi,yi) , SparseVector() )
       .mapValues(evaluateLoss).reduce(add)
```

Improvements

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$



dataRDD = [(1,(x1,y1)),
(2,(x2,y2)),
…
(n,(xn,yn))]

betaRDD = [(feature1: val1), # β is an rdd!
(feature2: val2),
…
(featured: vald)]

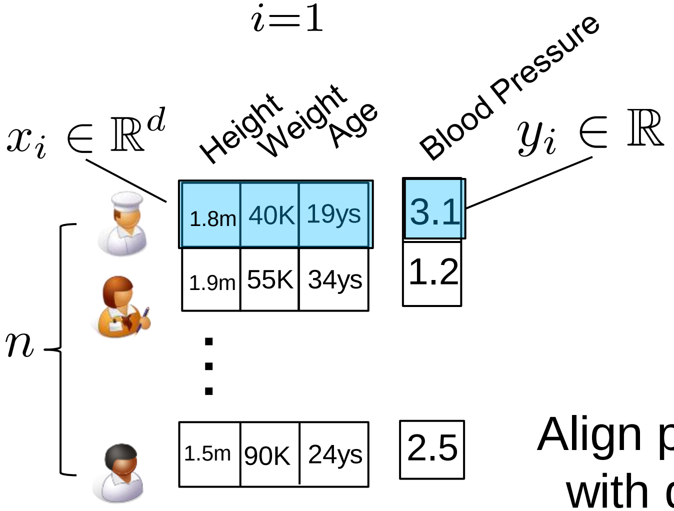
Use "inverse" GRDD

invGRDD= [(feature1,1), (feature7,1), …
(feature16,2), (feature22,2),…
…
(feature2,n),(feature99,n)]

```
tmp = invGRDD.join(beta)\                                             # (feature, (i, value))
    .map(lambda (f,(i,val)): (i, (f,val))\                            # (i, (feature,value))
    .mapValues(lambda  (f,val): SparseVector({f:val})).reduceByKey(add)   # (i, SparseVector())

return dataRDD.join(tmp).values()\ .                                  # ( (xi,yi) , SparseVector() )
        .mapValues(evaluateLoss).reduce(add)
```

Improvements

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$



dataRDD = [(1,(x1,y1)),
(2,(x2,y2)),
…
(n,(xn,yn))]

betaRDD = [(feature1: val1), # β is an rdd!
(feature2: val2),
…
(featured: vald)]

invGRDD= [(feature1,1), (feature7,1), …
(feature16,2), (feature22,2),…
…
(feature2,n),(feature99,n)]

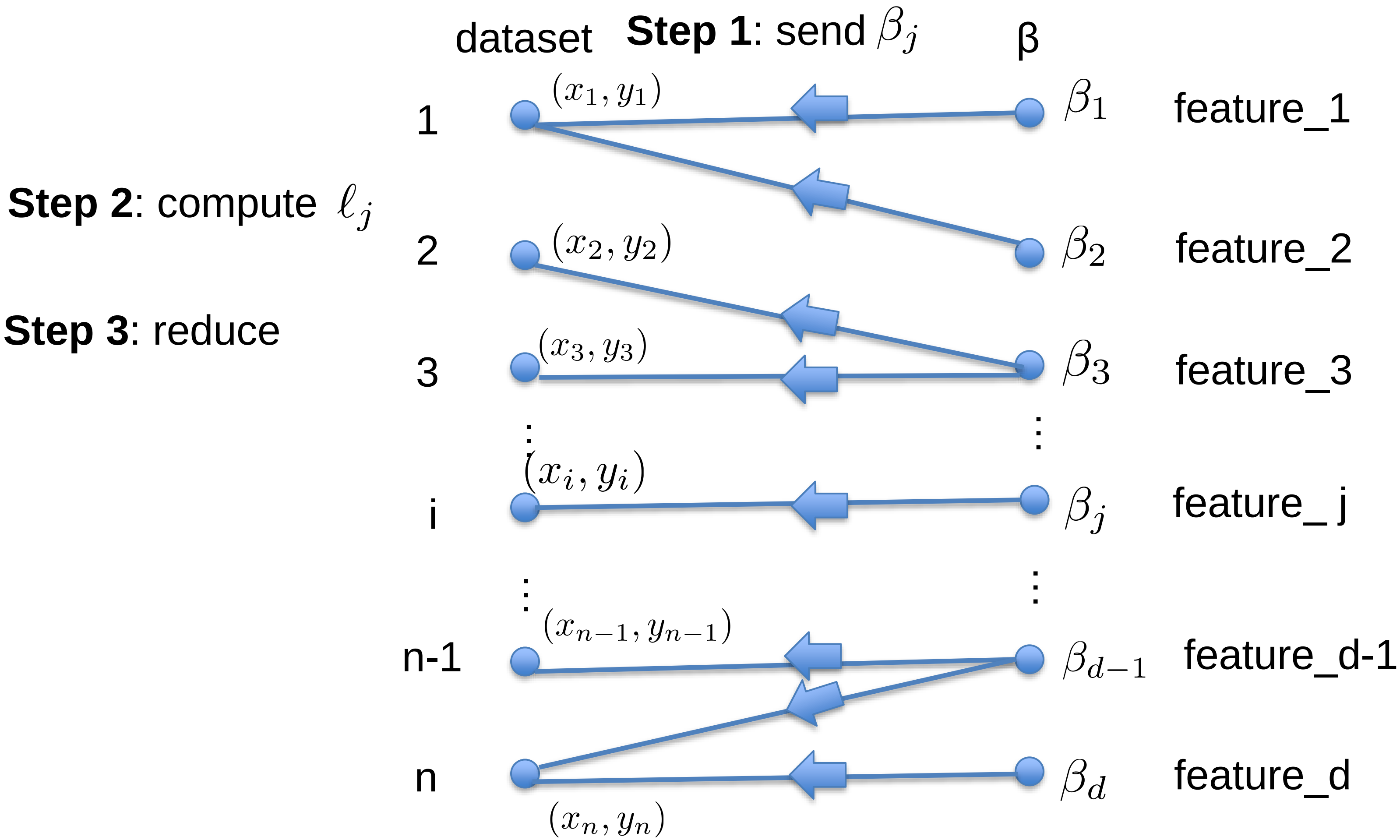
Align partitioning with dataRDD

```
tmp = invGRDD.join(beta)\                                                    # (feature, (i, value))
        .map(lambda (f,(i,val)): (i, (f,val))\                               # (i, (feature,value))
        .mapValues(lambda  (f,val): SparseVector({f:val})).reduceByKey(add, N)# (i, SparseVector())

return dataRDD.join(tmp).values()\ .                                         # ( (xi,yi) , SparseVector() )
            .mapValues(evaluateLoss).reduce(add)
```


Northeastern

In Summary: Function Evaluation

Suppose that each feature vector has at most k non-zero features



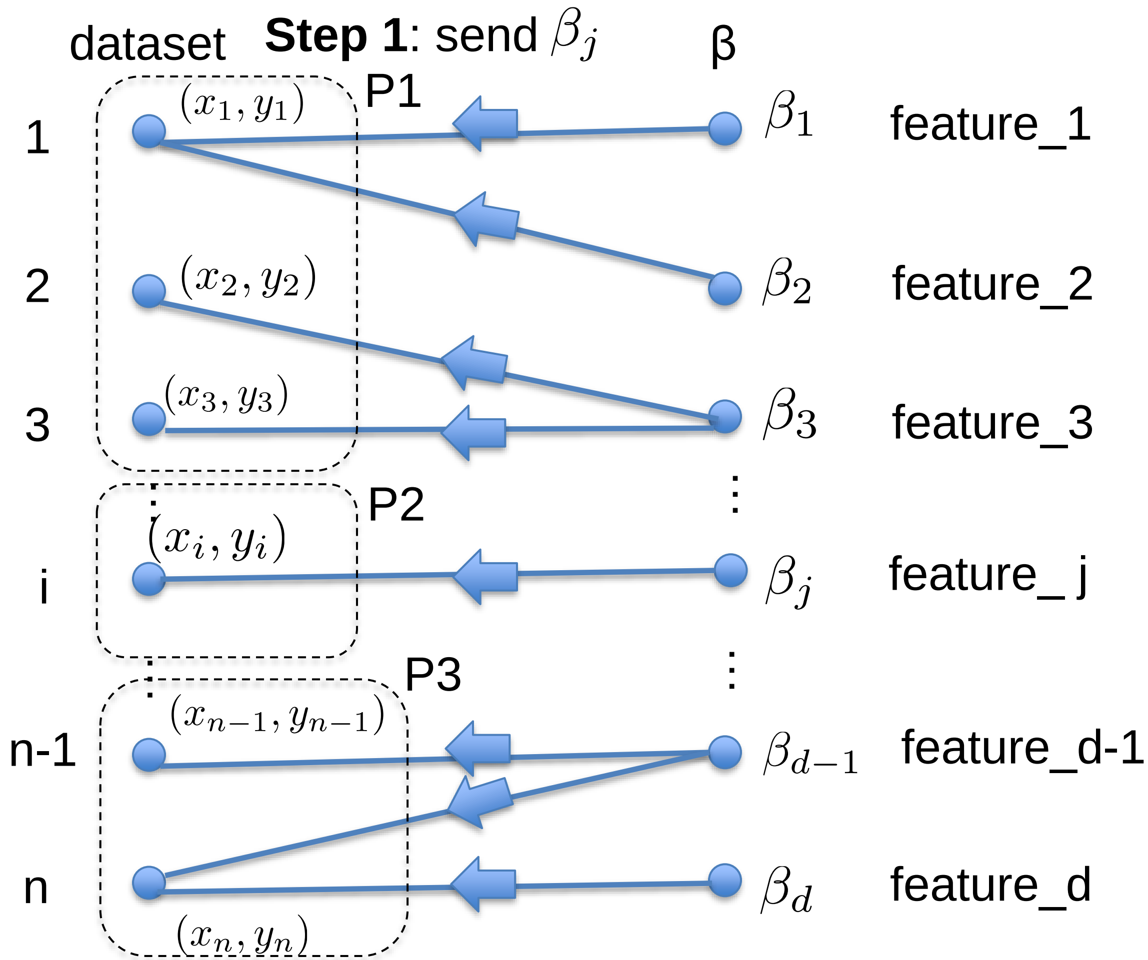
Northeastern

Another Improvement: Partition-Aware (HW3)

Suppose that each feature vector has at most k non-zero features

Step 2: compute ℓ_j

Step 3: reduce



Northeastern

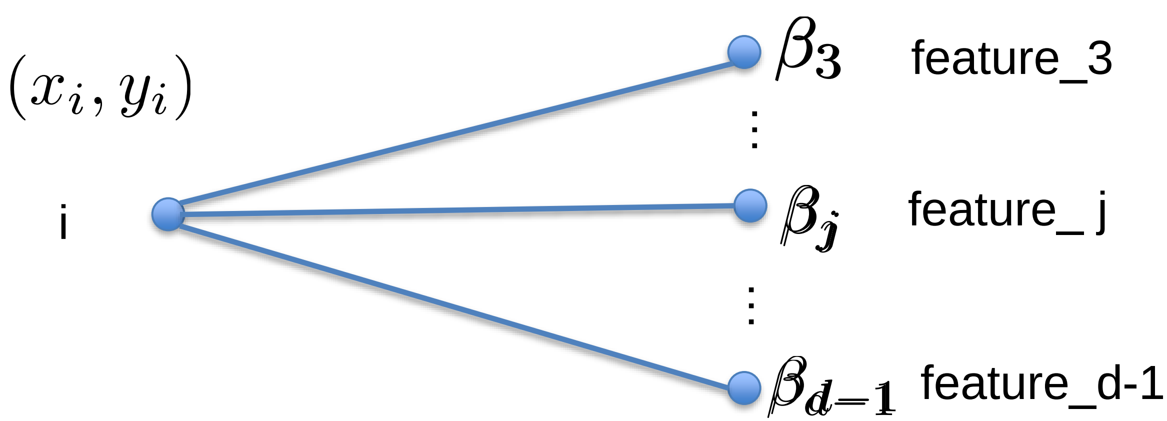
Computing Gradient

$$\nabla \ell_i(\beta) = \nabla \ell(\beta^\top x_i; y_i) = \ell'(\beta^\top x_i; y_i) \cdot x_i$$

$$= \ell'(\sum_{j: x_{ij} \neq 0} \beta_j x_{ij}; y_i) \cdot x_i$$

is sparse!!

depends only on k values of β



Northeastern

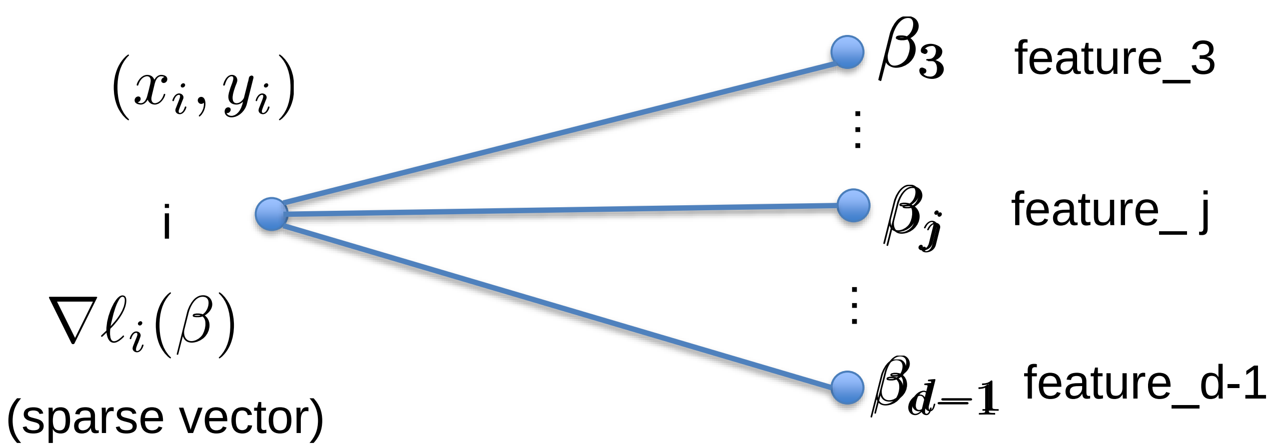
Computing Gradient

$$\nabla \ell_i(\beta) = \nabla \ell(\beta^\top x_i; y_i) = \ell'(\beta^\top x_i; y_i) \cdot x_i$$

$$= \ell'\Big(\sum_{j: x_{ij} \neq 0} \beta_j x_{ij}; y_i\Big) \cdot x_i$$

is sparse!!

depends only on k values of β



Computing Gradient

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla \ell_i(\beta)$$

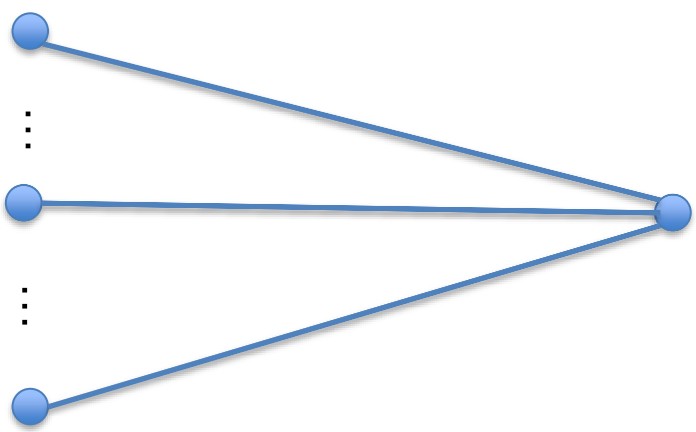
$$\frac{\partial F(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$
$$= \sum_{i: x_{ij} \neq 0}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

0 if ℓ_i does not depend on β_j !

$\nabla \ell_1(\beta)$ 1

$\nabla \ell_i(\beta)$ i

$\nabla \ell_n(\beta)$ n



feature_ j

Computing Gradient

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla \ell_i(\beta) \qquad \frac{\partial F(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

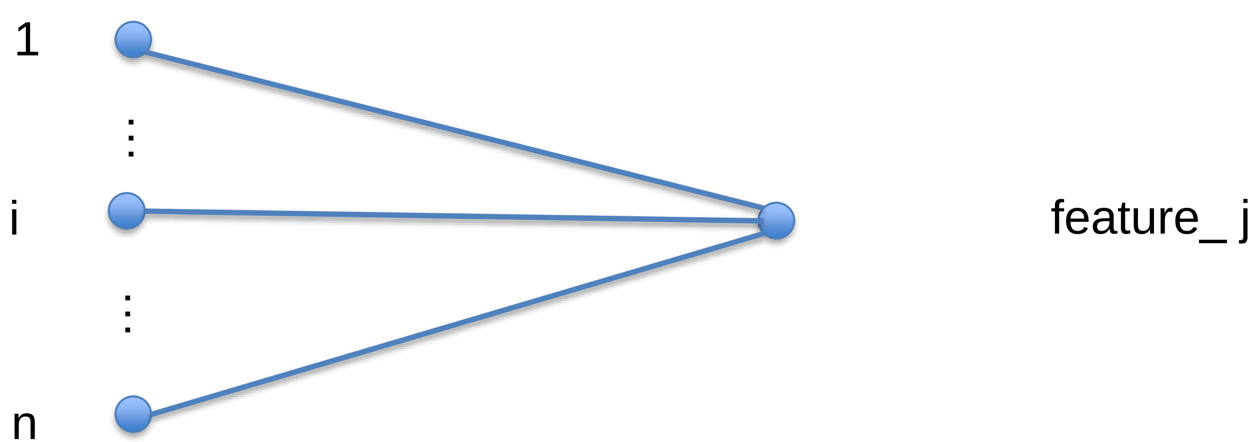
$$= \sum_{i:x_{ij} \neq 0}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

0 if ℓ_i does not depend on β_j !

$$\frac{\partial \ell_1(\beta)}{\partial \beta_j}$$

$$\frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

$$\frac{\partial \ell_n(\beta)}{\partial \beta_j}$$



Computing Gradient

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla \ell_i(\beta)$$

$$\frac{\partial F(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

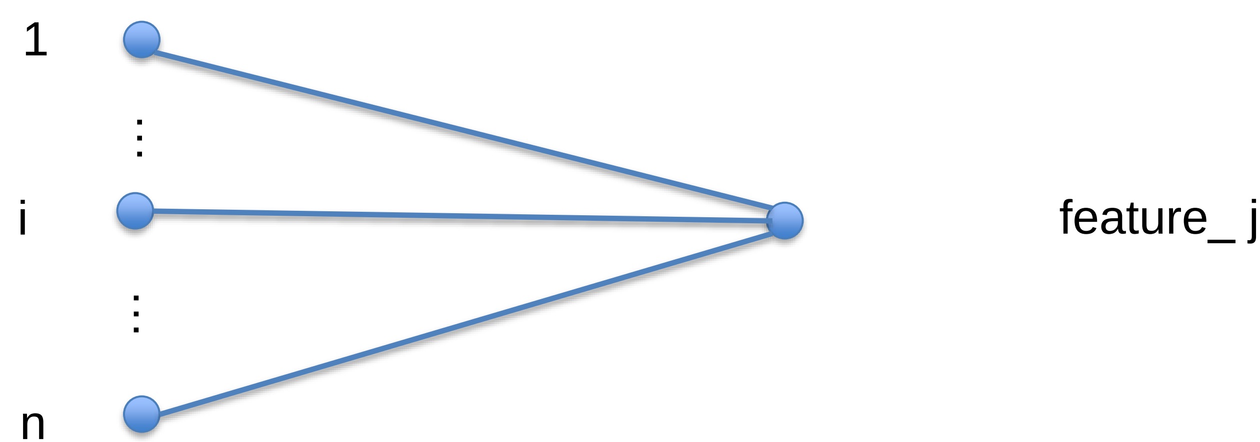
$$= \sum_{i:x_{ij} \neq 0} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

0 if ℓ_i does not depend on β_j !

$$\frac{\partial \ell_1(\beta)}{\partial \beta_j}$$

$$\frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

$$\frac{\partial \ell_n(\beta)}{\partial \beta_j}$$

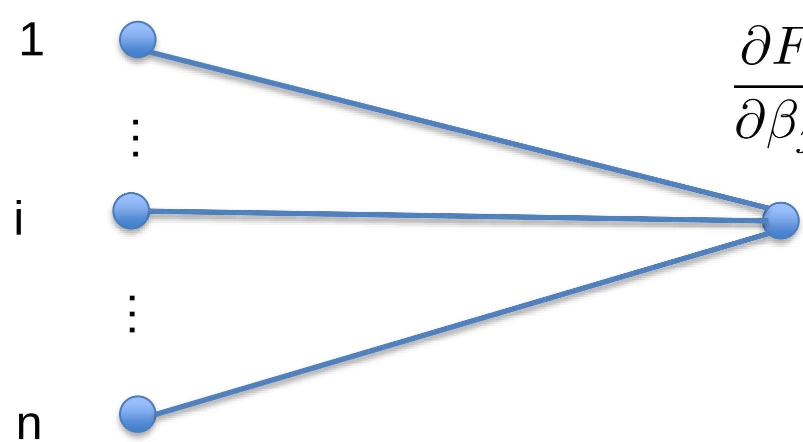


Computing Gradient

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla \ell_i(\beta)$$

$$\frac{\partial F(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j} = \sum_{i:x_{ij} \neq 0}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

0 if ℓ_i does not depend on β_j !



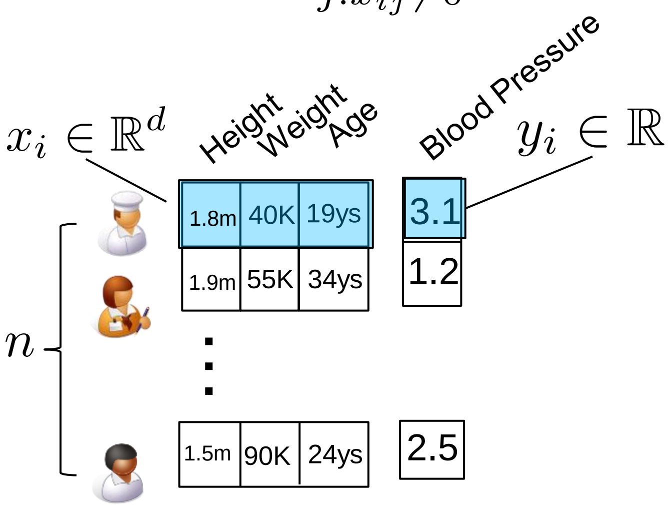
$$\frac{\partial F}{\partial \beta_j} = \sum_{i:x_{ij} \neq 0}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

feature_ j

Northeastern

Stage 1: Compute Gradients per Data Point

$$\nabla \ell_i(\beta) = \ell'\Big(\sum_{j:x_{ij}\neq 0} \beta_j x_{ij}; y_i\Big)\cdot x_i$$




```
dataRDD = [ (1,(x1,y1)),
            (2,(x2,y2)),
            …
            (n,(xn,yn))]

betaRDD = [ (feature1: val1),          # β is an rdd!
            (feature2: val2),
            …
            (featured: vald)]

invGRDD= [ (feature1,1), (feature7,1), …
           (feature16,2), (feature22,2),…
           …
           (feature2,n),(feature99,n)]

tmp = invGRDD.join(beta)\                                                   # (feature, (i, value))
      .map(lambda (f,(i,val)): (i, (f,val))\                                # (i, (feature,value))
      .mapValues(lambda  (f,val): SparseVector({f:val})).reduceByKey(add)   # (i, SparseVector())

localGrad =dataRDD.join(tmp).values()\  .                                   # ( (xi,yi) , SparseVector() )
           .mapValues(evaluateGradient)                                     #  SparseVector() containing  ∇ℓi(β)
```


Northeastern

Stage 2: Compute Gradients per Data Point

$$\nabla \ell_i(\beta) = \ell'(\sum_{j: x_{ij} \neq 0} \beta_j x_{ij}; y_i) \cdot x_i$$

$$\frac{\partial F}{\partial \beta_j} = \sum_{i: x_{ij} \neq 0}^{n} \frac{\partial \ell_i(\beta)}{\partial \beta_j}$$

```
localGrad = [ … ,
              SparseVector( {feat:val,feat:val,feat:val}),
             … ]
```

```
grad = localGrad.flatMap(lambda x:x.items())\
             .reduceByKey(add)
```

items() returns list of (key,value) tuples, so this is a (feature,val) RDD!

(feature,val) containing $\frac{\partial F}{\partial \beta_j}$!

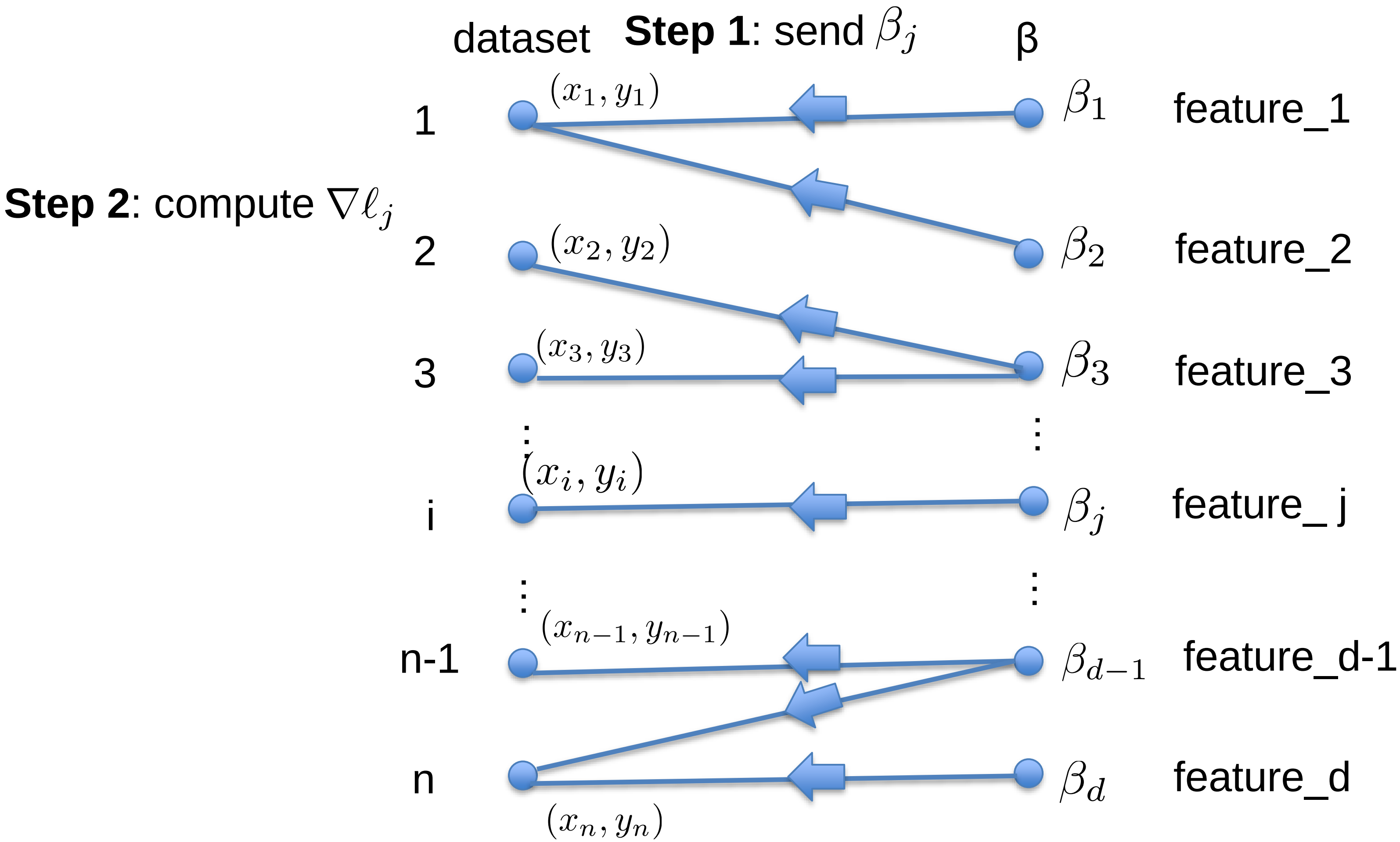
Gradient descent?

```
betaRDD = betaRDD.join(grad) \
             .mapValues(  lambda (valBeta,valGrad): valBeta - gamma valGrad)
```

(feature, (valBeta, valGrad))

In Summary: Computing Gradient

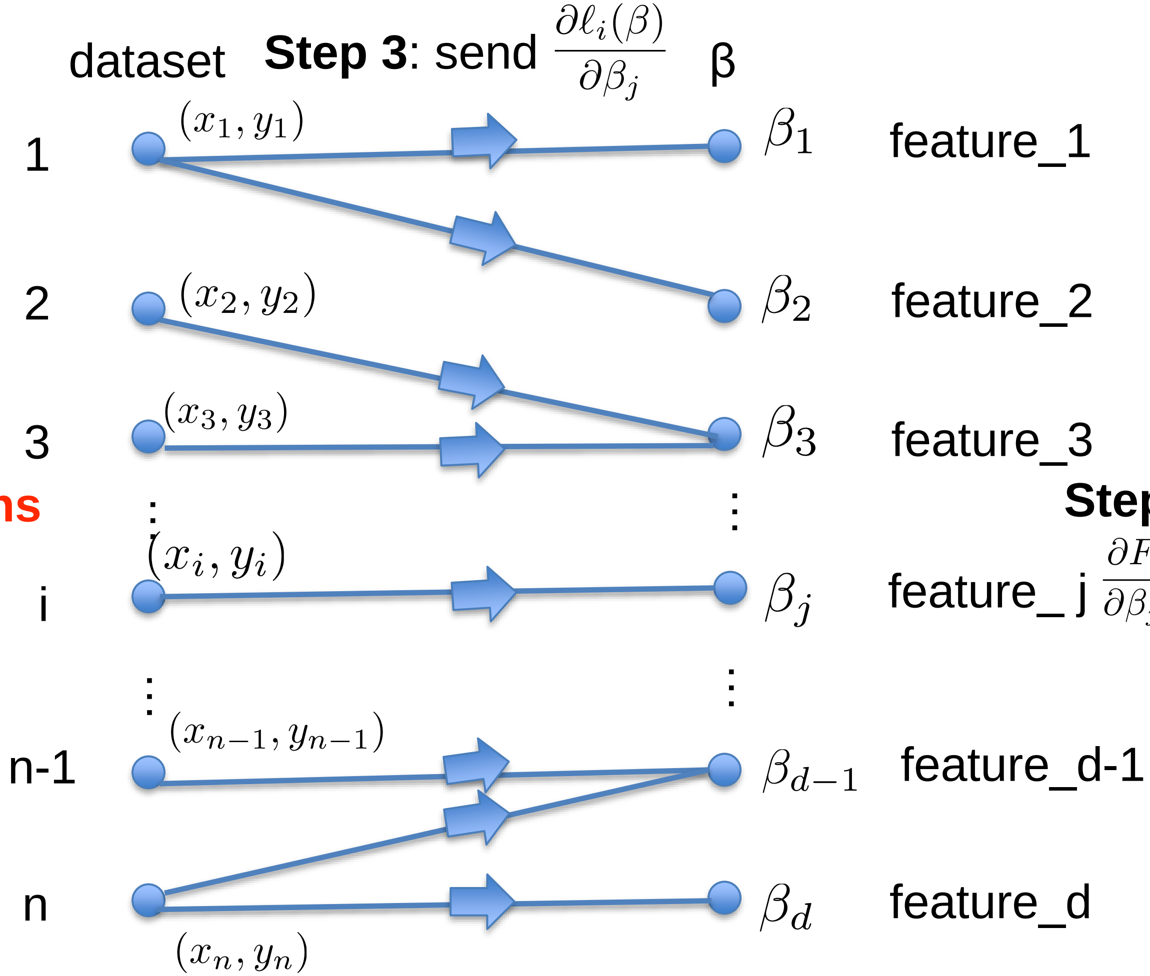
Suppose that each feature vector has at most k non-zero features



In Summary: Computing Gradient

Suppose that each feature vector has at most k non-zero features

Step 2: compute $\nabla \ell_j$



All communications of order n*k+d messages!

Step 4: compute $\frac{\partial F}{\partial \beta_j}$ through a reduceByKey

Northeastern

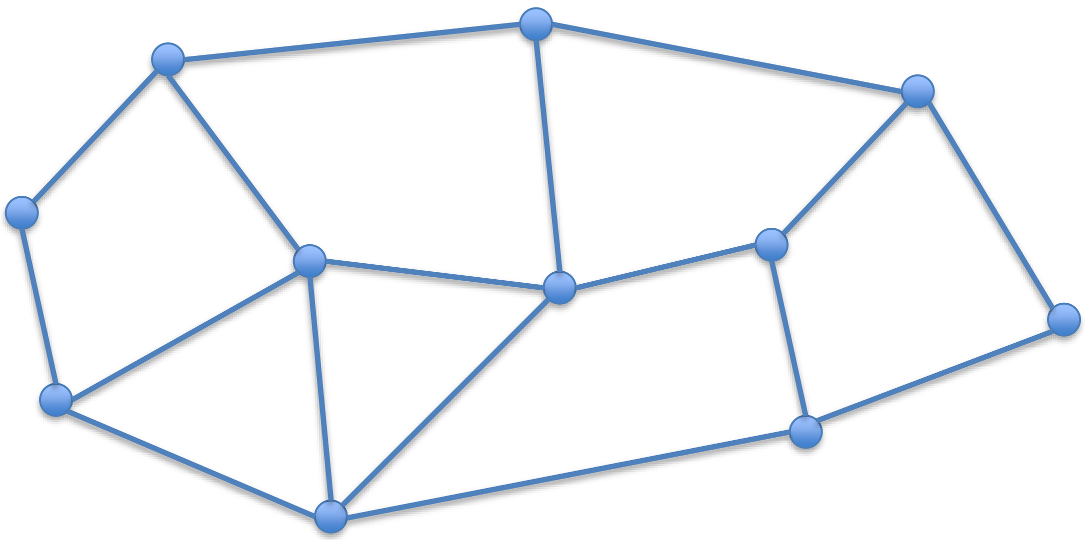
So What? What is the Big Picture?

- Does all this remind you of anything?

PageRank!

Graph-Parallel Algorithms

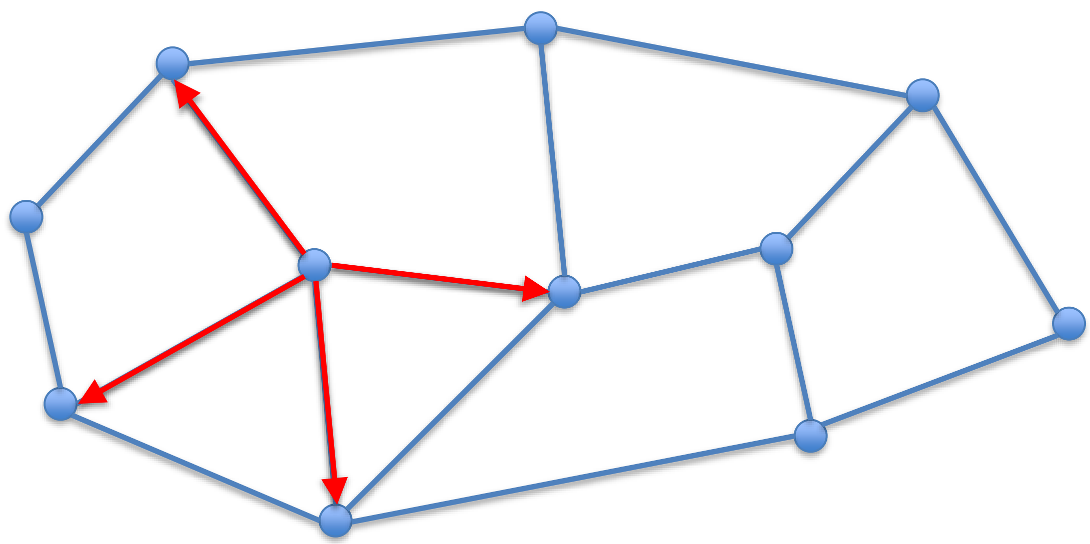
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Graph-Parallel Algorithm: Computation happens **over a directed graph** through **scatter**, **gather** and **apply** operations.

Graph-Parallel Algorithms

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]

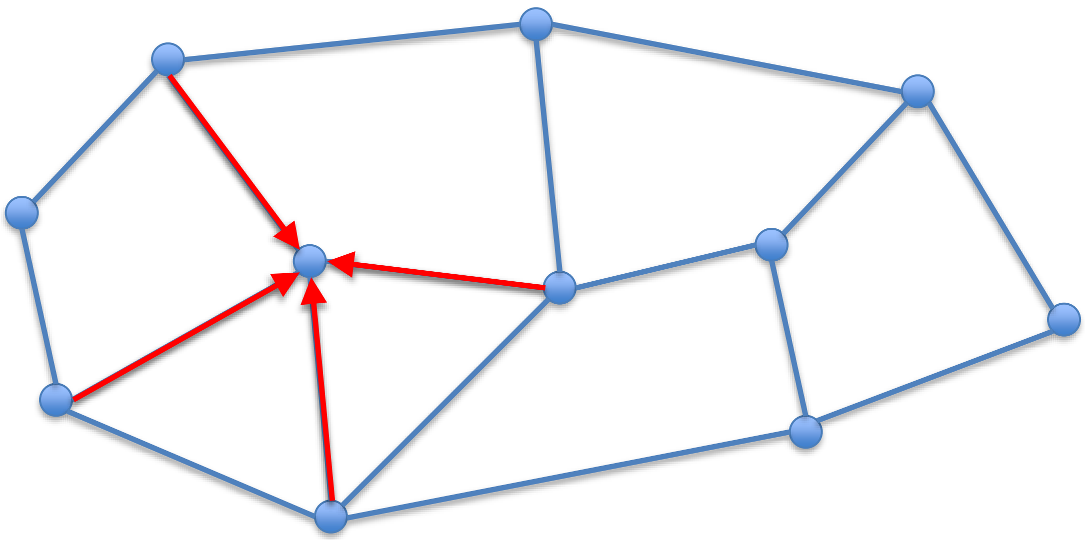


Scatter: Every node sends data to its neighbors

Northeastern

Graph-Parallel Algorithms

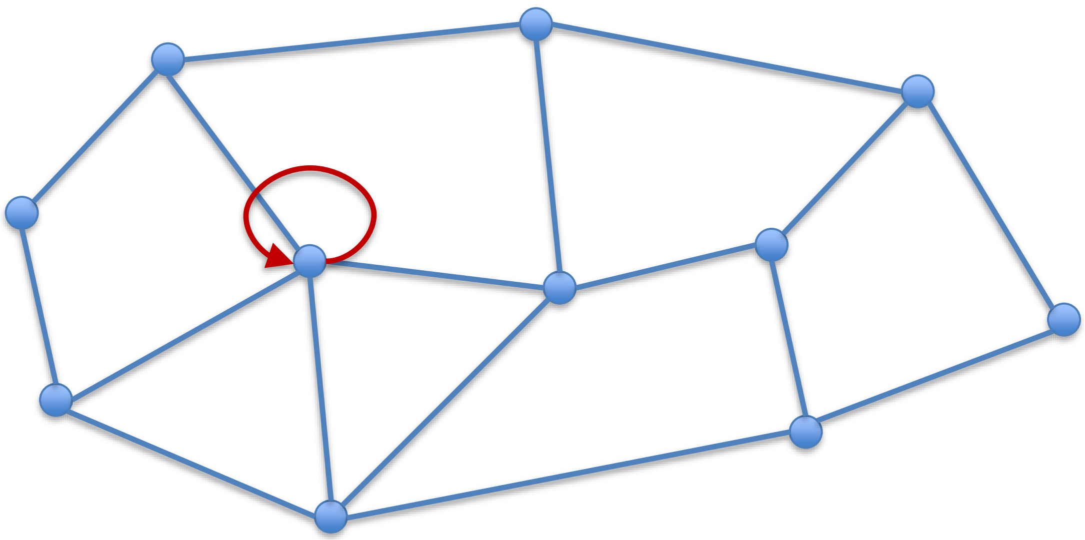
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Gather: Every node collects data from its neighbors and aggregates it.

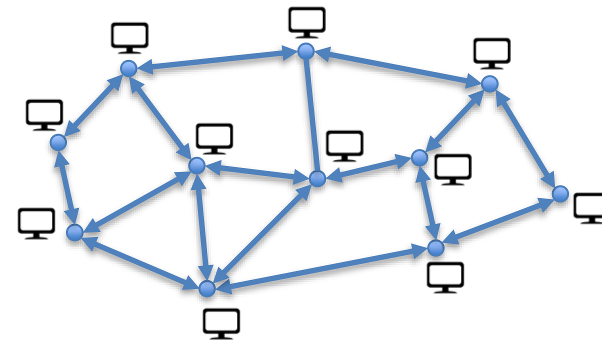
Graph-Parallel Algorithms

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Apply: Every node transforms its data locally

Graph-Parallel Algorithms



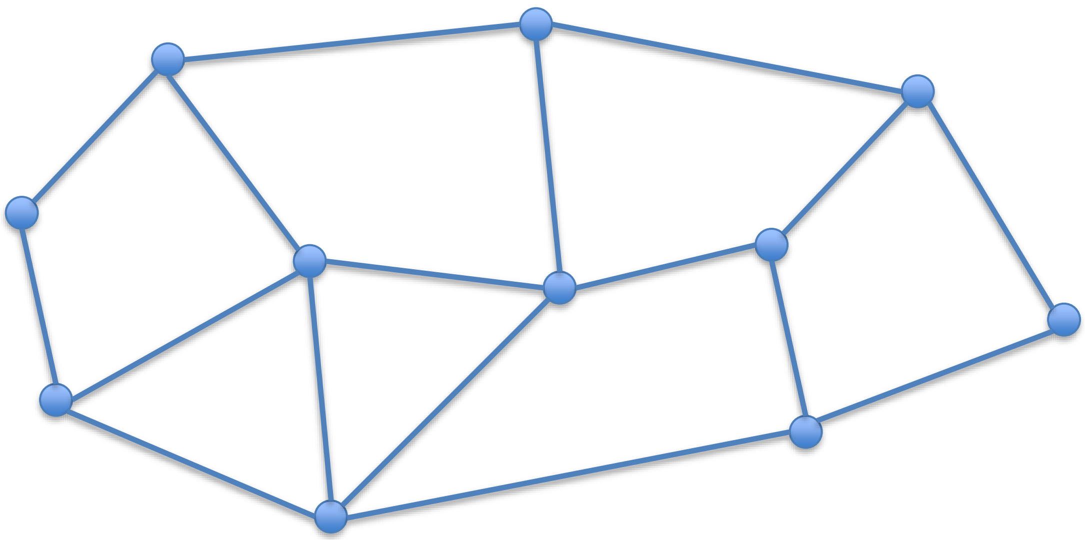
- ❑ There exist programming frameworks (e.g., GraphLab, Giraph) for parallelizing the execution of graph-parallel algorithms.

- ❑ Many interesting data mining, ML, and graph algorithms are graph-parallel:
  - ❑ Shortest paths
  - ❑ PageRank
  - ❑ Triangle counting
  - ❑ Graph coloring
  - ❑ Matrix Factorization through GD/ALS
  - ❑ ERM through GD
  - ❑ DNNs
  - ❑ …

Key Intution: Scatter, gather and apply can be implemented through appropriate join, reduceByKey, and map operations!!

These are efficient when the graph is SPARSE!!!

Graph-Parallel Algorithms: Formal definition



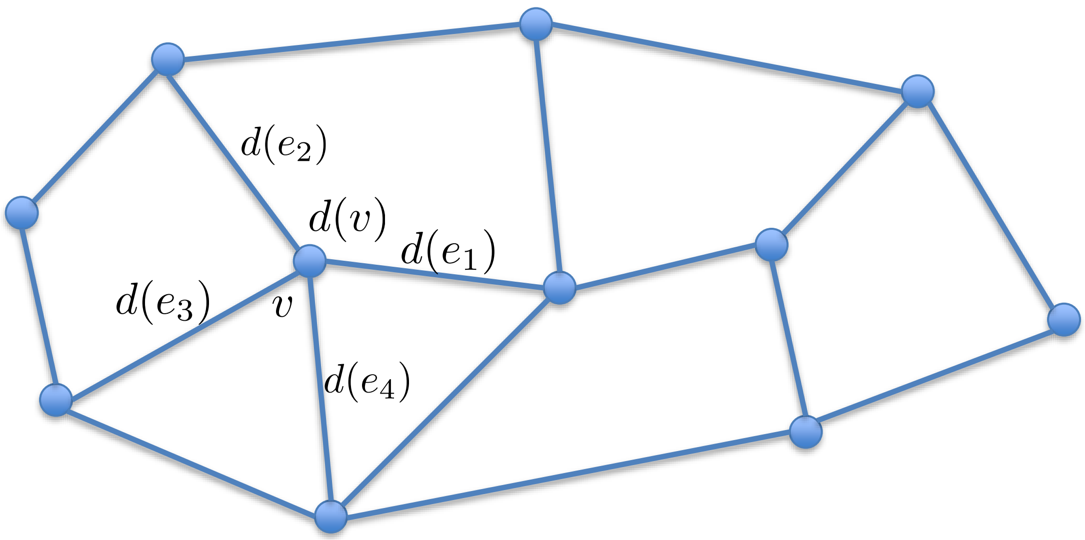
$$G(V, E)$$

$$|E| = O(|V|)$$

Consider a **directed, sparse graph** $G(V, E)$

Northeastern

Graph-Parallel Algorithms: Formal definition



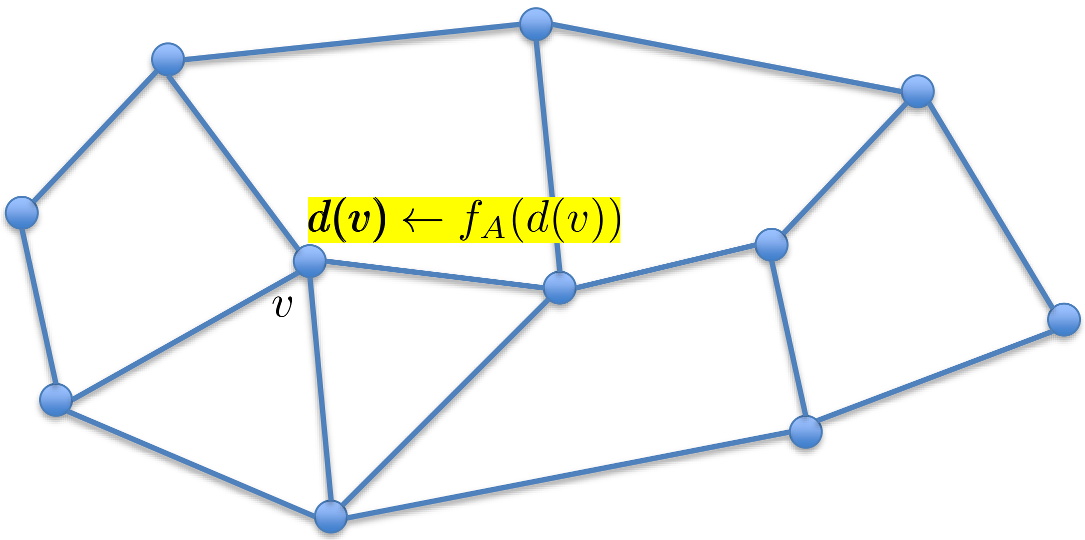
$$G(V, E)$$

$$|E| = O(|V|)$$

Both nodes **and** edges carry data.

Northeastern

Graph-Parallel Algorithms: Formal definition



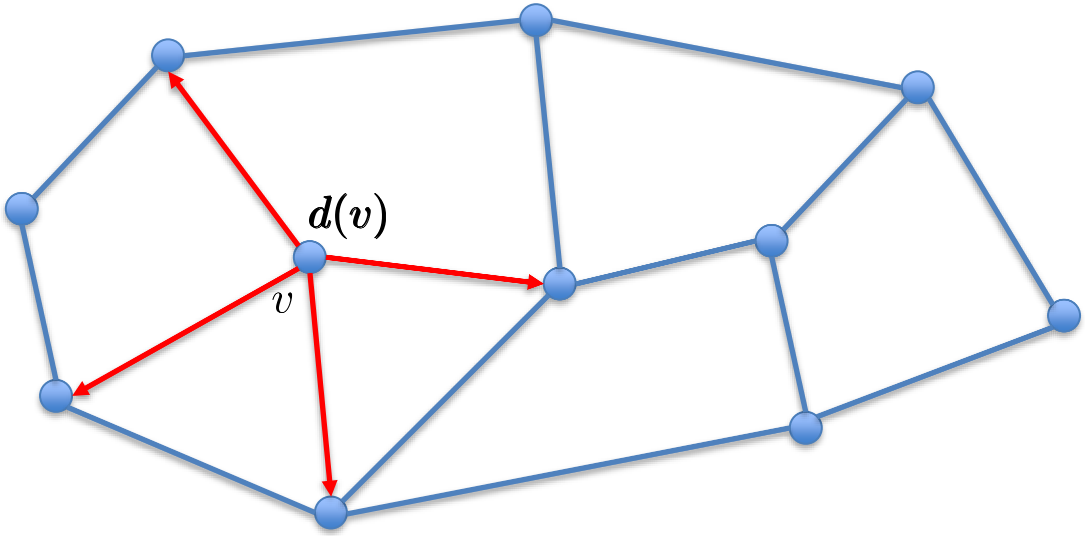
$G(V, E)$

$|E| = O(|V|)$

Apply(f_A): Every node $v \in V$ applies function f_A to their data $d(v)$ **in parallel**.

Northeastern

Graph-Parallel Algorithms: Formal Definition

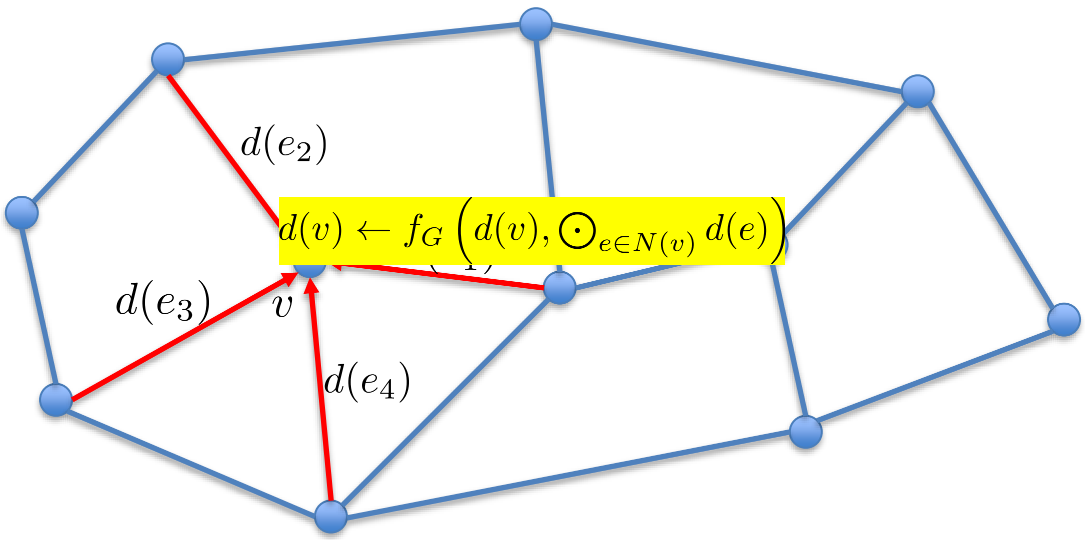


$G(V, E)$

$|E| = O(|V|)$

Scatter( ): Every node $v \in V$ sends their data $d(v)$ to adjoining edges

Graph-Parallel Algorithms: Formal Definition



$G(V, E)$

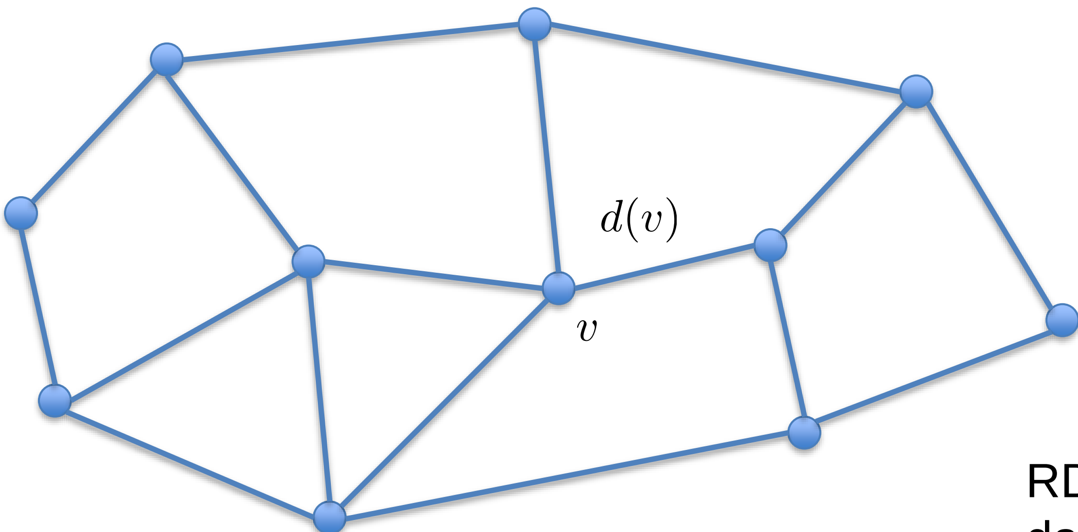
$|E| = O(|V|)$

Gather($f_G, \odot$): Every node $v \in V$ aggregates adjacent edge data through binary operator $\odot$ and combines it with its local data through f_G. I.e.:

$$d(v) \leftarrow f_G\left(d(v), \bigodot_{e\in N(v)} d(e)\right), \quad \text{where} \qquad \bigodot_{e\in N(v)} d(e) = \underbrace{d(e_1)\odot d(e_2)\odot\ldots\odot d(e_k)}_{e_i\in N(v)}$$

Graph-Parallel Algorithms via Spark

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



dataRDD = [(1, data1),
(2,data2),
…
(n,datan)]

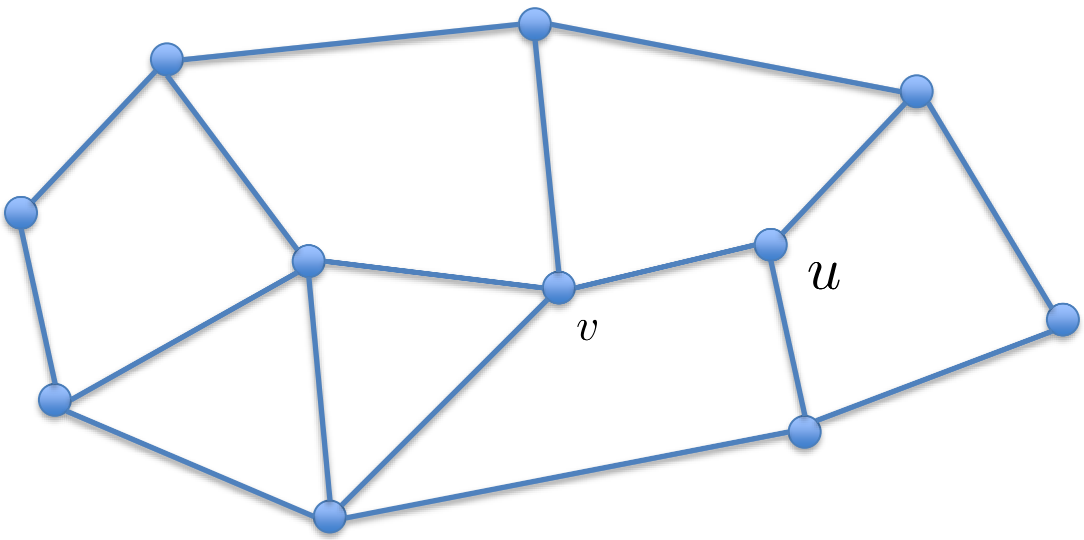
RDD representing
data at nodes with tuples (v,dv)

Data and graph represented through two RDDs

Northeastern

Graph-Parallel Algorithms via Spark

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



dataRDD = [(1, data1),
(2,data2),
…
(n,datan)]

graphRDD = [(1,2),
(1,3),
…
]

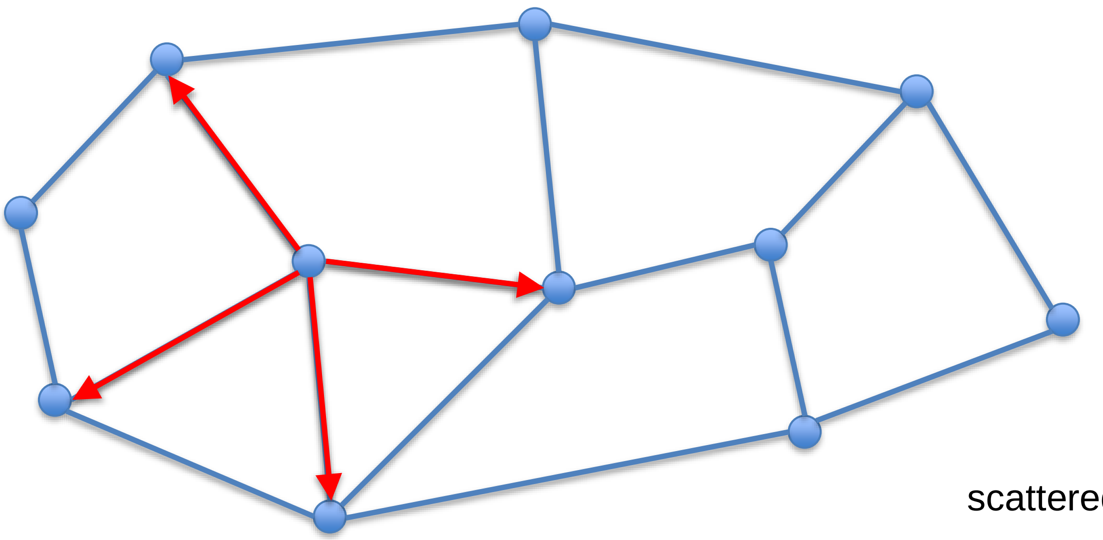
Data and graph represented through two RDDs

RDD representing
graph with tuples (v,u)

Northeastern

Graph-Parallel Algorithms

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]

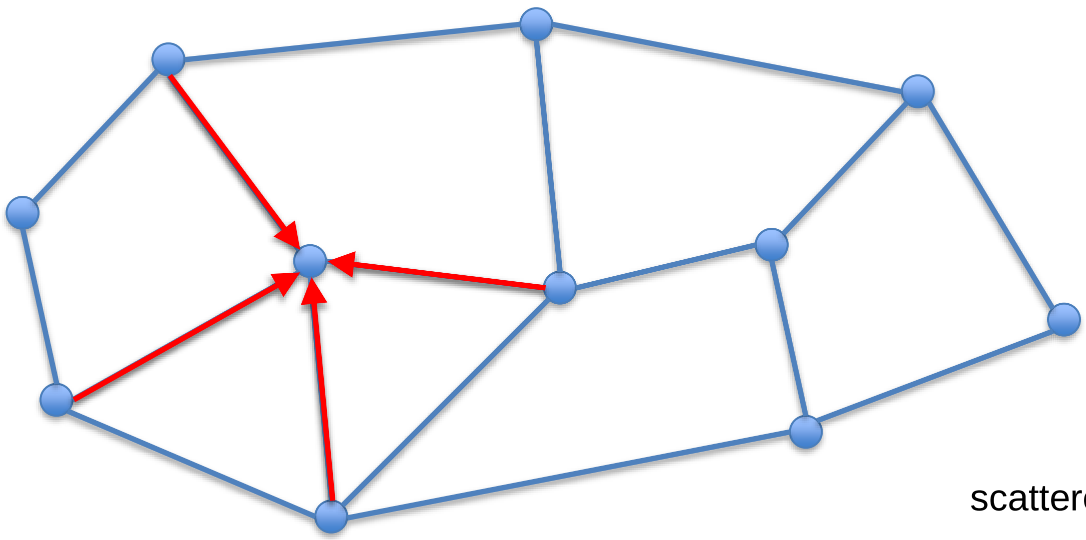


```
dataRDD = [(1, data1),
           (2,data2),
            …
           (n,datan)]

graphRDD = [(1,2),
            (1,3),
             …
           ]

scattered = graphRDD.join(dataRDD)\   #(v,(u,datav)
            .values()
```

Scatter: Every node sends data to its neighbors

Graph-Parallel Algorithms

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



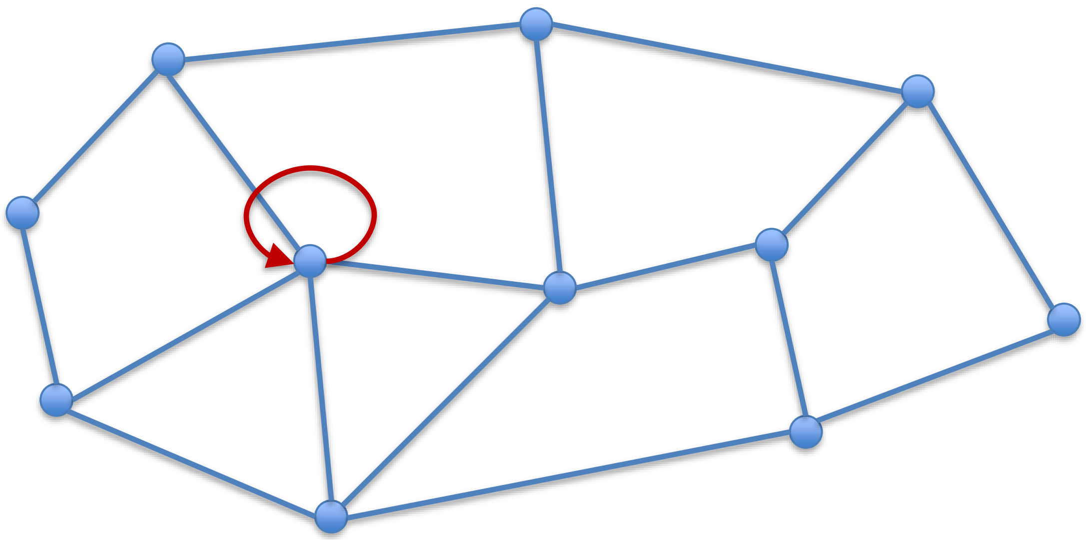
```
dataRDD = [(1, data1),
           (2,data2),
           …
           (n,datan)]

graphRDD = [(1,2),
            (1,3),
            …
           ]

scattered = graphRDD.join(dataRDD)\   #(v,(u,datav)
            .values()

gathered = scattered.reduceByKey(aggregate)
#dataRDD = gathered.join(dataRDD).mapValues(…)
```

Gather: Every node collects data from its neighbors and aggregates it.

Northeastern

Graph-Parallel Algorithms

[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



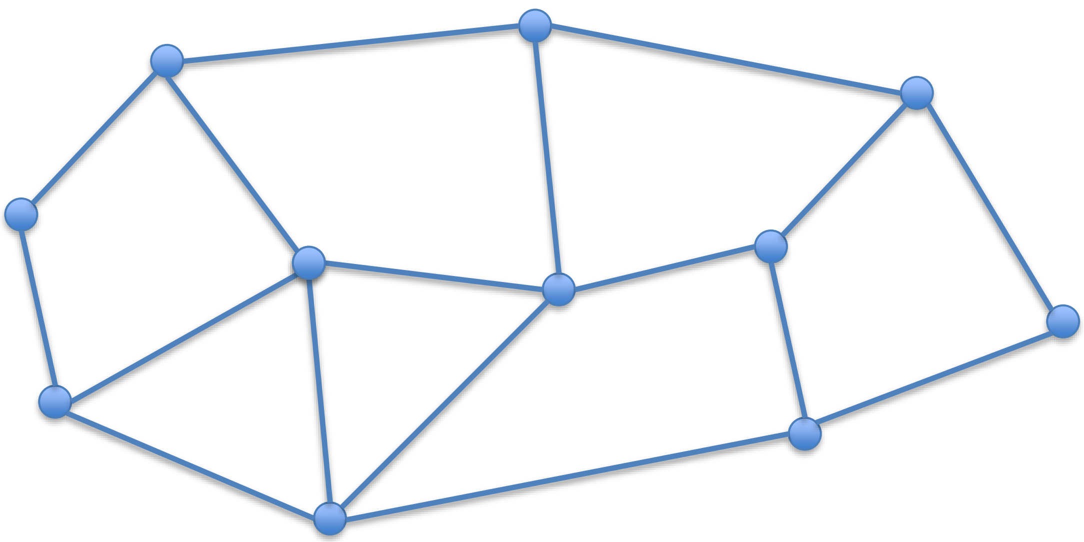
dataRDD = [(1, data1),
(2,data2),
…
(n,datan)]

graphRDD = [(1,2),
(1,3),
…
]

dataRDD = dataRDD.mapValues(…)

Apply: Every node transforms its data locally

Example 1: PageRank

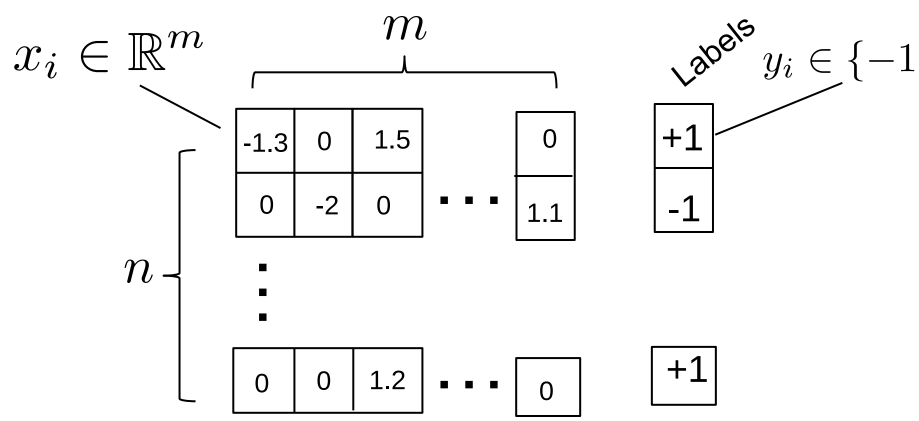


Each iteration implements

$$d(v) \leftarrow \gamma \frac{1}{|V|} + (1-\gamma) \sum_{(u,v)\in N(v)} \frac{d(u)}{|N(u)|}, \quad \forall v \in V$$

repeated until convergence

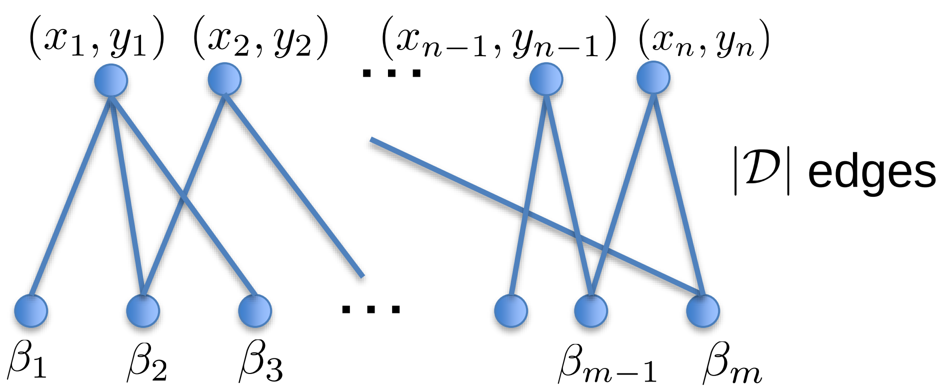
Example 2: High-Dimensional, Sparse Gradient Descent



$$\beta^* = \arg\min_{\beta \in \mathbb{R}^m} \sum_{i=1}^{n} \ell(\langle x_i, \beta \rangle, y_i) + \lambda \|\beta\|_2^2$$

Non-zero elements: $|\mathcal{D}| = O(n+m)$

Gradient Descent: $\beta_j \leftarrow \beta_j - \gamma \left(\sum_{i: x_{ij} \neq 0} \ell'(\langle x_i, \beta \rangle, y_i) \cdot x_{ik} + 2\lambda\beta_j \right), \qquad \forall j \in \{1, \ldots, m\}$



HW3!!!

Example 3: Matrix Factorization

Dataset $\mathcal{D}$

?	5	5	?	3	1
3	2	?	?	1	5
⋮					
3	?	1	2	?	5

r_{ij} : rating by user i to item j.

$r_{ij} = \langle u_i, v_j \rangle + \varepsilon_{ij}$, where $u_i \in \mathbb{R}^d$, $v_j \in \mathbb{R}^d$.

user profile

item profile

- Prediction: $\hat{r}_{ij} = \langle u_i, v_j \rangle$
- LSE: $(U, V) = \underset{U \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{m \times d}}{\arg\min} \sum_{(i,j,r_{ij}) \in \mathcal{D}} \left(r_{ij} - \langle u_i, v_j \rangle\right)^2$

Example 3: Matrix Factorization

Dataset $\mathcal{D}$

m columns, n rows:

?	5	5	?	3	1
3	2	?	?	1	5
⋮					
3	?	1	2	?	5

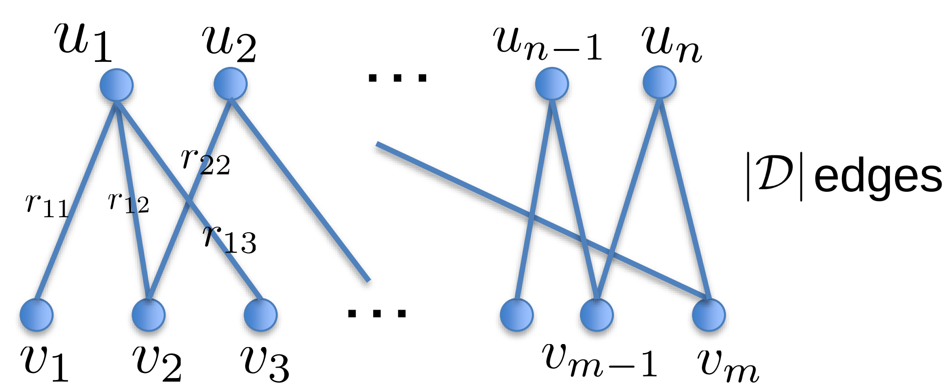
$$|\mathcal{D}| = O(n+m)$$

HW4!!!

- Gradient Descent:

$$u_i \leftarrow u_i + \gamma \cdot \sum_{j:(i,j,r_{ij})\in\mathcal{D}} (r_{ij} - \langle u_i, v_j \rangle) v_j$$

$$v_j \leftarrow v_j + \gamma \cdot \sum_{i:(i,j,r_{ij})\in\mathcal{D}} (r_{ij} - \langle u_i, v_j \rangle) u_i$$



Northeastern