



# BUILDING STREAMING RECOMMENDATION ENGINES ON APACHE SPARK

Rui Vieira  
Software Engineer

# Overview

Collaborative Filtering

Batch Alternating Least Squares (ALS)

Streaming ALS

Apache Spark

Distributed Streaming ALS

# Collaborative Filtering

# Collaborative Filtering

Users, products and ratings

(user, product)  $\longmapsto$  rating

Collaborative

“Filtering”

# Collaborative Filtering

# Collaborative Filtering

A



B



# Collaborative Filtering

A



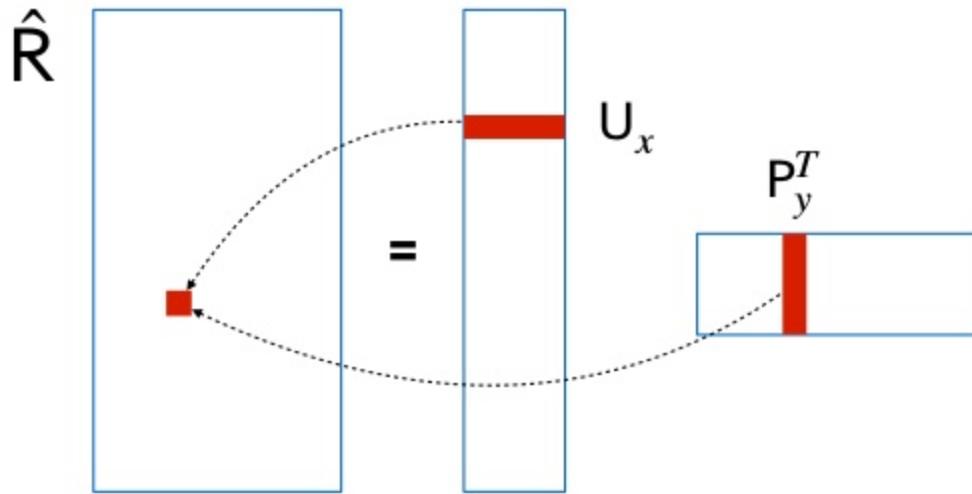
B



# Alternating Least Squares

$$R = \begin{bmatrix} \text{user 1} & \text{user 2} & \text{user 3} & \dots & \text{user N} \\ 1 & 4.5 & ? & \dots & 3 \\ ? & 3 & 3 & \dots & 4 \\ 5 & 3 & ? & \dots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \dots & ? \end{bmatrix} \begin{array}{l} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product M} \end{array}$$

# Alternating Least Squares



$$\hat{R}_{x,y} = U_x P_y^T$$

# Batch ALS

$$loss = \sum_{x,y} \left( \epsilon_{x,y} \right)^2 + \lambda_x + \lambda_y$$

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

# Alternating Least Squares

$$\hat{R} \quad \boxed{\text{ }} = \quad \textcolor{red}{U} \quad \boxed{\text{ }} \quad P^T \quad \boxed{\text{ }}$$

# Alternating Least Squares

$$\hat{R} \quad \boxed{\quad} = U \quad \boxed{\quad} P^T \quad \boxed{\quad}$$

# Alternating Least Squares

$$R = \begin{bmatrix} \text{user 1} & \text{user 2} & \text{user 3} & \dots & \text{user N} \\ 1 & 4.5 & 3.8 & \dots & 3 \\ 3.2 & 3 & 3 & \dots & 4 \\ 5 & 3 & 3.4 & \dots & 3.1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \dots & 2.7 \end{bmatrix} \begin{array}{l} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product M} \end{array}$$

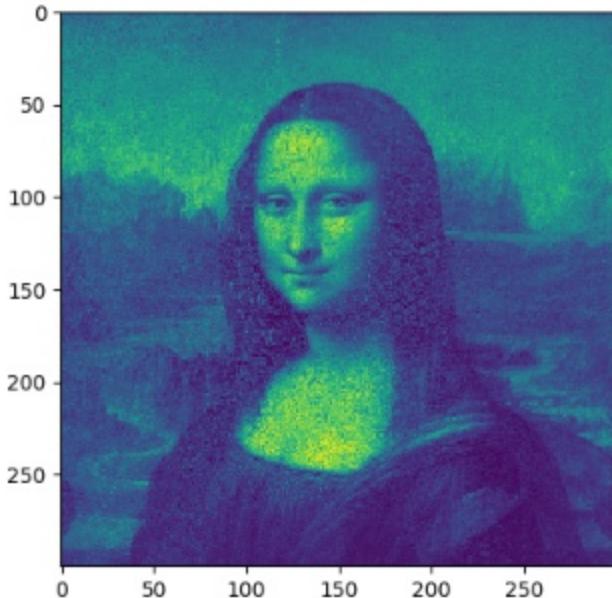
# Batch ALS

	1	2	3	4	... 300
1	70	82	60	54	65
2	70	86	68	67	72
3	96	103	82	82	77
4	90	87	68	93	82
...					
300	38	48	44	51	35

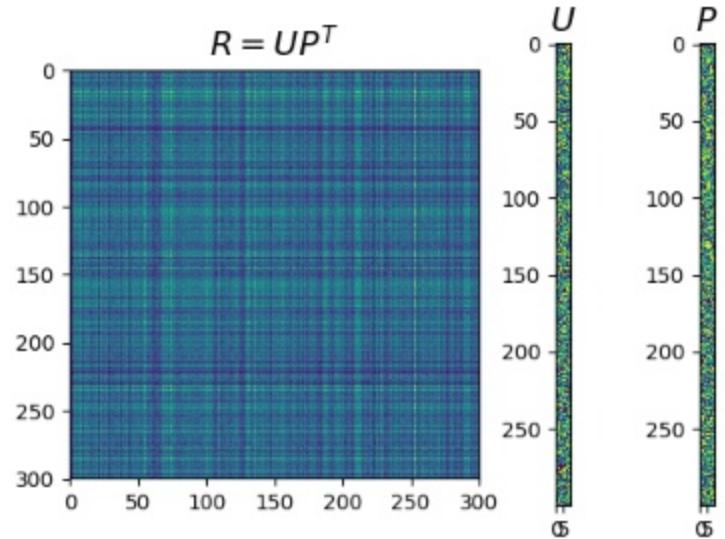
# Batch ALS

	1	2	3	4	... 300
1	70	82	60	54	65
2	70	86	68	67	72
3	96	103	82	82	77
4	90	87	68	93	82
...					
300	38	48	44	51	35

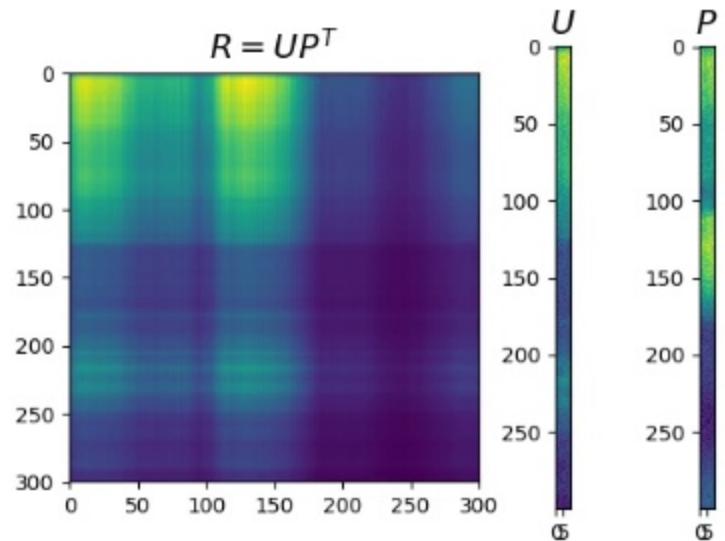
# Batch ALS



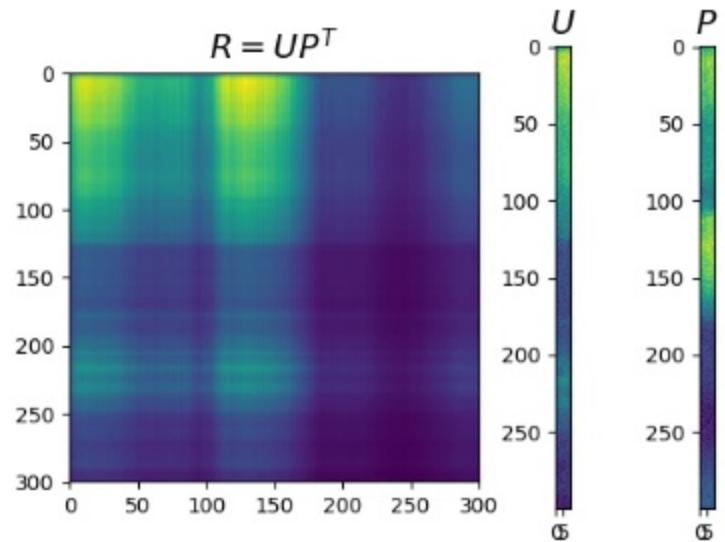
# Batch ALS



# Batch ALS



# Batch ALS



# Streaming ALS

**Can we update the model with a data stream?**

**Stochastic Gradient Descent (SGD)**

**Bias SGD (B-SGD)**

# Streaming ALS

$$b_{x,y} = \mu + b_x + b_y$$

$$\hat{r}_{x,y}^{\star} = b_{x,y} + \hat{r}_{x,y}$$

# Streaming ALS

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathbf{U}_x \cdot \mathbf{P}_y^T$$

$$\text{loss} = \sum_{x,y} \left( \underbrace{r_{x,y} - \hat{r}_{x,y}}_{\epsilon_{x,y}} \right)^2 + \dots$$

# Streaming ALS

**bias**

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

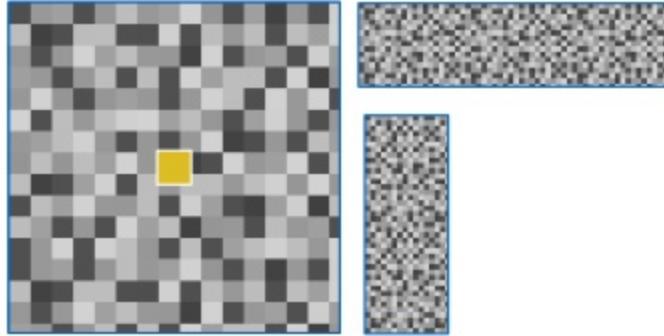
$$b_y \leftarrow b_y + \gamma (\epsilon_{x,y} - \lambda_y b_y)$$

**factors**

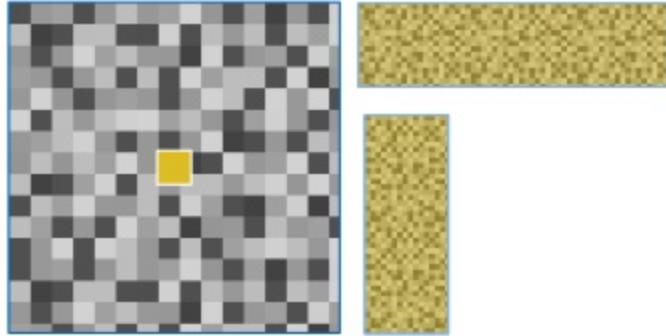
$$\mathbf{U}_x \leftarrow \mathbf{U}_x + \gamma (\epsilon_{x,y} \mathbf{P}_y - \lambda'_x \mathbf{U}_x)$$

$$\mathbf{P}_y \leftarrow \mathbf{P}_y + \gamma (\epsilon_{x,y} \mathbf{U}_x - \lambda'_y \mathbf{P}_y)$$

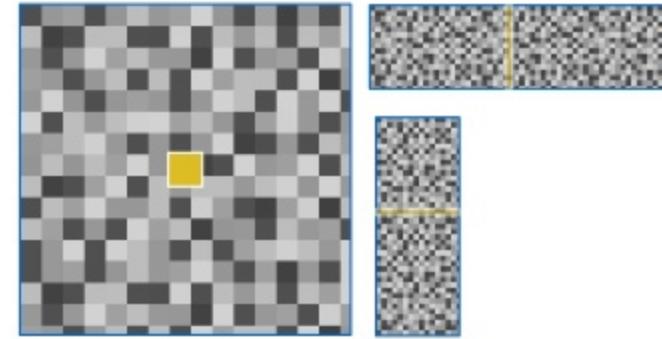
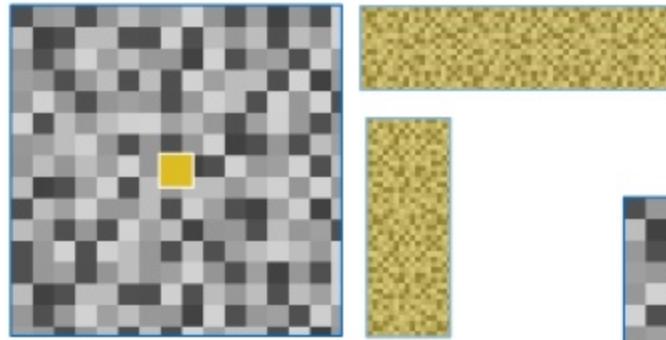
# Streaming ALS



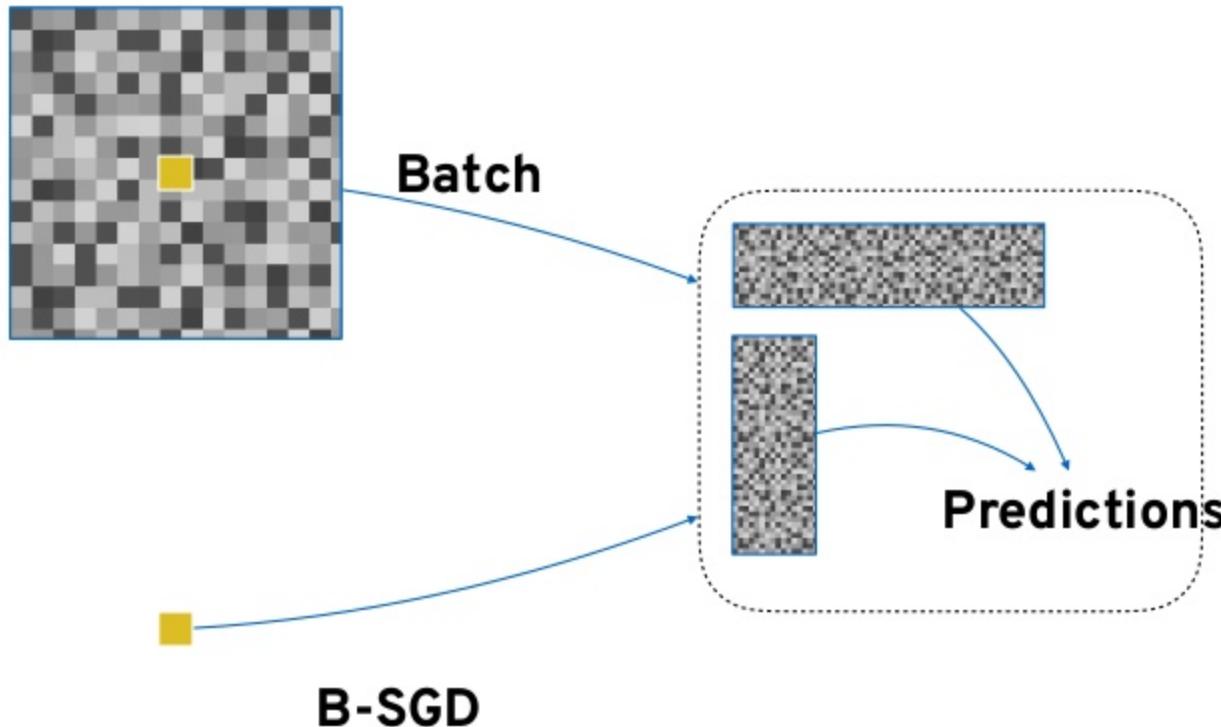
# Streaming ALS



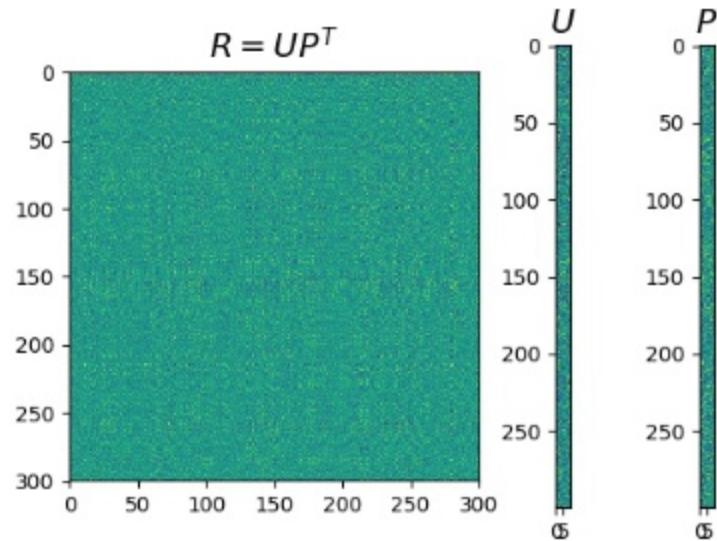
# Streaming ALS



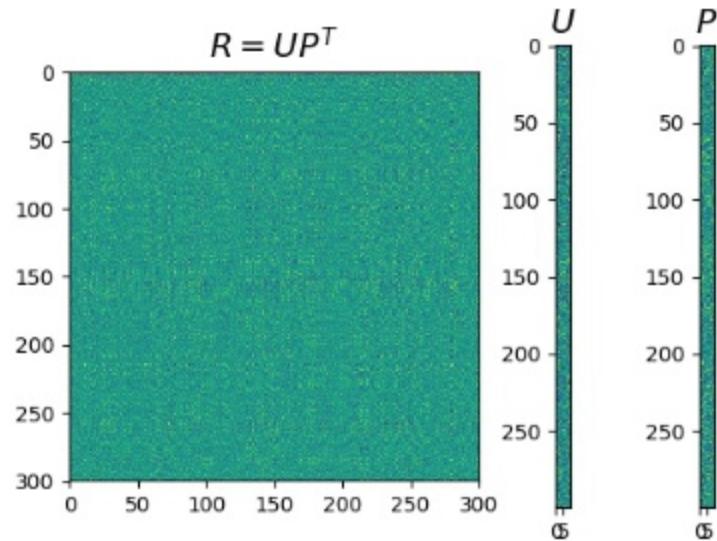
# Streaming ALS



# Streaming ALS



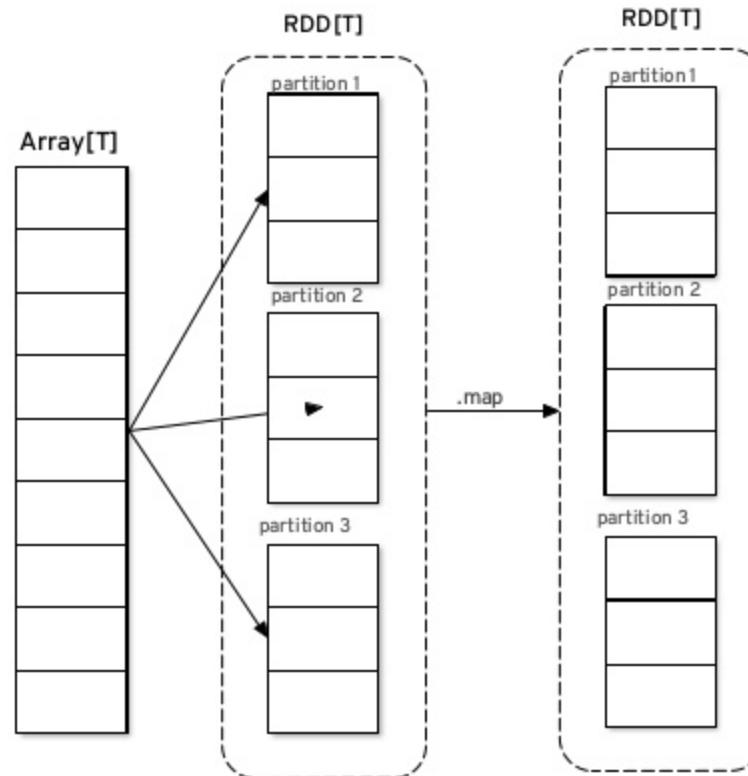
# Streaming ALS





# APACHE SPARK

# Apache Spark



# MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

# MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)  
val ratings: RDD[Rating]
```

# MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)
```

```
val ratings: RDD[Rating]
```

```
val rank: int
```

# MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)
```

```
val ratings: RDD[Rating]
```

```
val rank: int
```

```
val iterations: int
```

# MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)
```

```
val ratings: RDD[Rating]
```

```
val rank: int
```

```
val iterations: int
```

```
val lambda: Double
```

# MLlib ALS

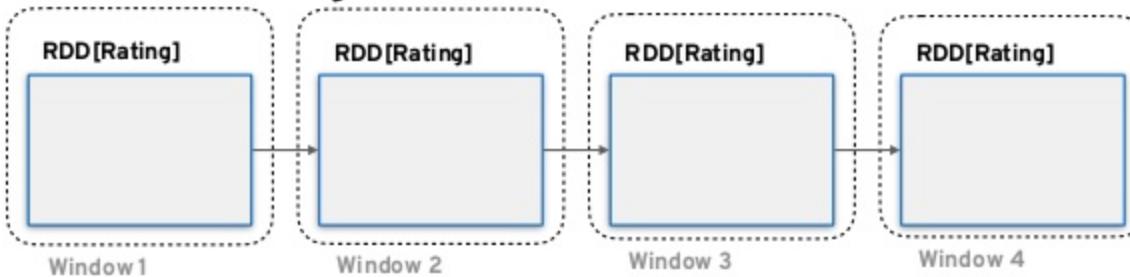
```
> val model = ALS.train(ratings, rank, iterations, lambda)  
model: MatrixFactorizationModel  
  
class MatrixFactorizationModel {  
  
    val userFeatures: RDD[(Int, Array[Double])]  
    val productFeatures: RDD[(Int, Array[Double])]  
  
}
```

# Spark Streaming ALS

RDD[Rating]



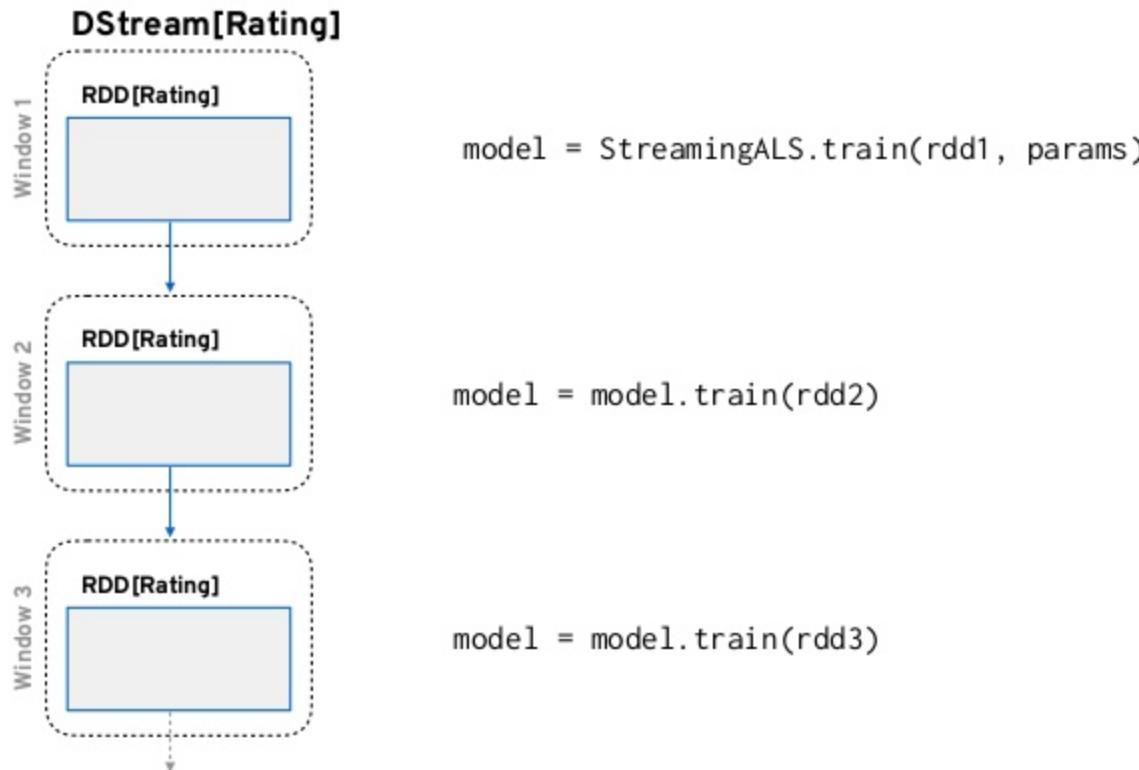
DStream[Rating]



# Spark Streaming ALS

DStream[Rating]

# Spark Streaming ALS



# What do we need?

user latent factors

product latent factors

calculate the global bias

calculate user specific bias

calculate product specific bias

# What do we need?

user latent factors

product latent factors

calculate the global bias

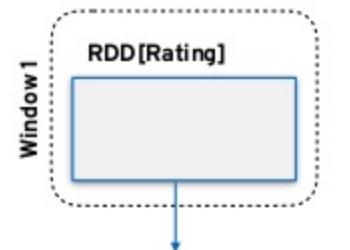
calculate user specific bias

calculate product specific bias

# Spark Streaming ALS

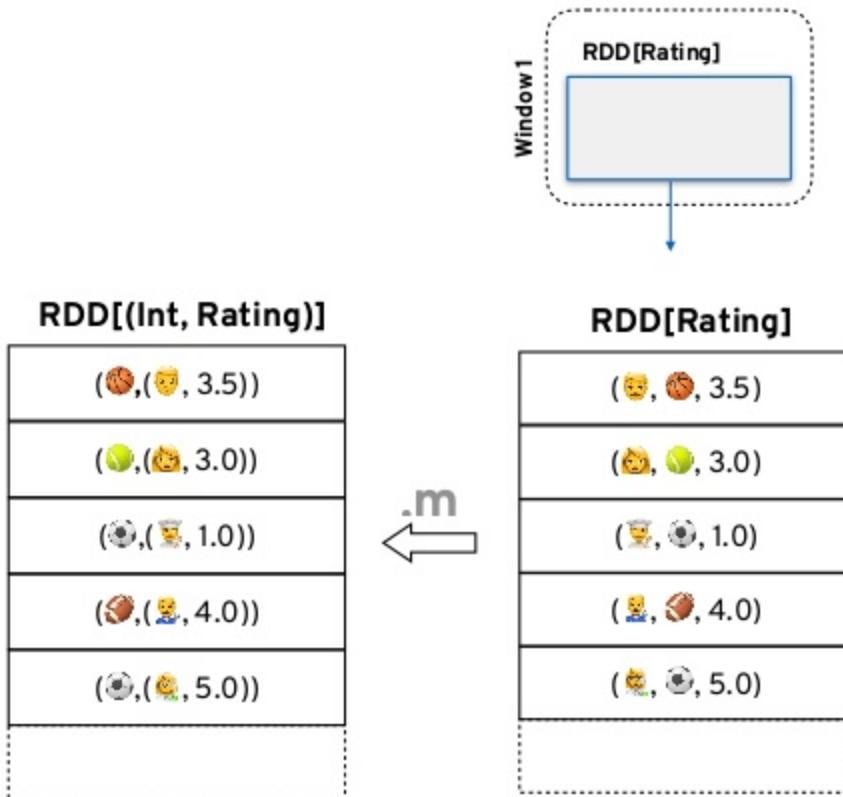


# Spark Streaming ALS

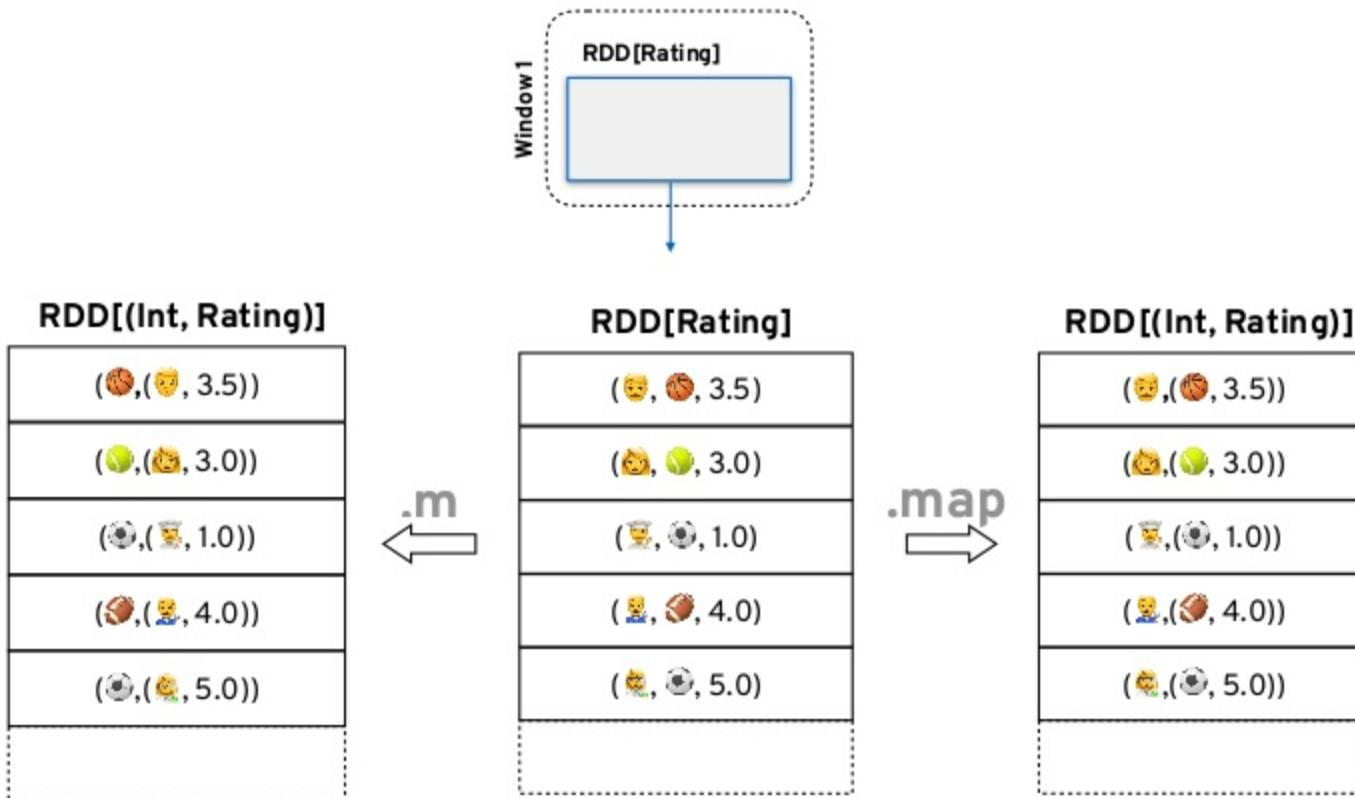


RDD[Rating]
(😊, 🍞, 3.5)
(😊, 🎾, 3.0)
(😊, 🎪, 1.0)
(😊, 🏈, 4.0)
(😊, 🎩, 5.0)

# Spark Streaming ALS



# Spark Streaming ALS



# Spark Streaming ALS

# Spark Streaming ALS

RDD[(Int, Rating)]

(观影人ID, (电影ID, 3.5))
(观影人ID, (电影ID, 3.0))
(观影人ID, (电影ID, 1.0))
(观影人ID, (电影ID, 4.0))
(观影人ID, (电影ID, 5.0))

# Spark Streaming ALS

RDD[(Int, Rating)]

(👨, (🏀, 3.5))
(👨, (⚽, 3.0))
(👨, (⌚, 1.0))
(👨, (🏈, 4.0))
(👨, (⌚, 5.0))
...

.map  
→

RDD[(Int, Factor)]

(👨, [0.123, -0.234, ...]))
(👨, [0.934, 0.526, ...]))
(👨, [0.421, -0.594, ...]))
(👨, [0.034, 0.661, ...]))
(👨, [0.713, -0.335, ...]))
...

# Spark Streaming ALS

# Spark Streaming ALS

RDD[(Int, Rating)]

(😊, (🏀, 3.5))
(🎃, (⚽, 3.0))
(🧙, (⊗, 1.0))
(🧙, (🏀, 4.0))
(🦉, (⊗, 5.0))

# Spark Streaming ALS

RDD[(Int, Rating)]

(😊, (🏀, 3.5))
(🎃, (⚽, 3.0))
(🧙, (⌚, 1.0))
(🧙, (🏀, 4.0))
(🧙, (⌚, 5.0))

RDD[(Int, Factor)]

(😊, [0.123, -0.234, ...])
(🎃, [0.934, 0.526, ...])
(🧙, [0.421, -0.594, ...])
(🧙, [0.034, 0.661, ...])
(🧙, [0.713, -0.335, ...])

# Spark Streaming ALS

RDD[(Int, Rating)]

(😊, (🏀, 3.5))
(🎃, (⚽, 3.0))
(🧙, (⊗, 1.0))
(🧙, (🏀, 4.0))
(🧙, (⊗, 5.0))

join  
↔

RDD[(Int, Factor)]

(😊, [0.123, -0.234, ...])
(🎃, [0.934, 0.526, ...])
(🧙, [0.421, -0.594, ...])
(🧙, [0.034, 0.661, ...])
(🧙, [0.713, -0.335, ...])

# Spark Streaming ALS

RDD[(Int, (Int, Double, Factor))]

(🏀, (👨, 3.5, [0.123, -0.234, ...]))

(⚽, (👨, 3.0, [0.934, 0.526, ...]))

(⚽, (👨, 1.0, [0.421, -0.594, ...]))

(🏀, (👨, 4.0, [0.034, 0.661, ...]))

(⚽, (👨, 5.0, [0.713, -0.335, ...]))

user latent factors

# Spark Streaming ALS

# Spark Streaming ALS

RDD[(Int, (Int, Double, Factor))]

(🏀, (👤, 3.5, [0.123, -0.234, ...]))
(⚽, (🧙, 3.0, [0.934, 0.526, ...]))
(⌚, (🧙, 1.0, [0.421, -0.594, ...]))
(🏀, (🧙, 4.0, [0.034, 0.661, ...]))
(⌚, (🧙, 5.0, [0.713, -0.335, ...]))
.....

# Spark Streaming ALS

RDD[(Int, (Int, Double, Factor))]

(🏀, (👤, 3.5, [0.123, -0.234, ...]))
(⚽, (👤, 3.0, [0.934, 0.526, ...]))
(🏀, (👤, 1.0, [0.421, -0.594, ...]))
(🏀, (👤, 4.0, [0.034, 0.661, ...]))
(🏀, (👤, 5.0, [0.713, -0.335, ...]))

.join  
→

RDD[(Int, Factor)]

(🏀, [0.764, 0.254, ...]))
(⚽, [0.136, 0.933, ...]))
(⚽, [0.663, -0.134, ...]))
(⚽, [0.811, 0.535, ...]))
(🏀, [0.234, -0.579, ...]))

# Spark Streaming ALS

RDD[(Int, (Int, Double, Factor, Factor))]

(🏀, (👤, 3.5, [0.123, ...], [0.764, ...]))
(🎾, ( %[<], 3.0, [0.934, 0.526, ...], [0.933, ...]))
(⚽, ( %[>], 1.0, [0.421, -0.594, ...], [0.663, ...]))
(🏈, ( %[<], 4.0, [0.034, 0.661, ...], [0.811, ...]))
(🎱, ( %[>], 5.0, [0.713, -0.335, ...], [0.234, ...]))

# What do we need?

user latent factors

product latent factors

**calculate the global bias**

calculate user specific bias

calculate product specific bias

# Spark Streaming ALS

RDD[(Int, (Int, Double))]

(🏀, (👨, 3.5, [0.123, ...], [0.764, ...]))
(⚽, (🧙, 3.0, [0.934, 0.526, ...], [0.933, ...])))
(⌚, (🧙, 1.0, [0.421, -0.594, ...], [0.663, ...])))
(🏈, (🧙, 4.0, [0.034, 0.661, ...], [0.811, ...])))
(⌚, (🧙, 5.0, [0.713, -0.335, ...], [0.234, ...])))

# Spark Streaming ALS

RDD[PointType, Double]
(🏀, (👤, 3.5, [0.123, ...], [0.764, ...]))
(🎾, (👤, 3.0, [0.934, 0.526, ...], [0.933, ...]))
(⚽, (👤, 1.0, [0.421, -0.594, ...], [0.663, ...]))
(🏀, (👤, 4.0, [0.034, 0.661, ...], [0.811, ...]))
(⚽, (👤, 5.0, [0.713, -0.335, ...], [0.234, ...]))

# What do we need?

user latent factors

product latent factors

calculate the global bias

calculate user specific bias

calculate product specific bias

# Spark Streaming ALS

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathbf{U}_x \cdot \mathbf{P}_y^T$$

# Spark Streaming ALS

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathbf{U}_x \cdot \mathbf{P}_y^T$$

# Spark Streaming ALS

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathbf{U}_x \cdot \mathbf{P}_y^T$$

# Spark Streaming ALS

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathbf{U}_x \cdot \mathbf{P}_y^T$$

RDD[(Int, (Int, Double, Factor, Factor))]

(��, (��, 1.0, [0.421, -0.594, ...], [0.663, ...]))

predicted(��,��) =  $\mu + b_x + b_y + [0.421, -0.594, \dots] \times [0.663, \dots]^T = 2.3$

# Spark Streaming ALS

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

# Spark Streaming ALS

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

# Spark Streaming ALS

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

RDD[(Int, (Int, Double, Factor))]

(��, (��, 1.0, [0.421, -0.594, ...], [0.663, ...]))

rating(��,��) = 1.0

predicted(��,��) = 2.3

error(��,��) = rating(��,��) - predicted(��,��) = -1.3

# Spark Streaming ALS

## gradients

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

$$b_y \leftarrow b_y + \gamma (\epsilon_{x,y} - \lambda_y b_y)$$

# Spark Streaming ALS

## gradients

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

RDD[(Int, Double, Factor)]

({:}, 0.932, [0.123, -0.140, ...])	$\leftarrow b_y + \gamma (\epsilon_{x,y} - \lambda_y b_y)$
({:}, 0.101, [0.334, 0.273, ...])	
({:}, 0.128, [0.957, -0.247, ...])	
({:}, 0.242, [0.038, 0.883, ...])	
({:}, 0.245, [0.283, -0.953, ...])	

# Spark Streaming ALS

## gradients

$$b_x \leftarrow b_x + \gamma (\epsilon_{x,y} - \lambda_x b_x)$$

RDD[(Int, Double, Factor)]

(😊, 0.932, [0.123, -0.140, ...])
(😊, 0.101, [0.334, 0.273, ...])
(😺, 0.128, [0.957, -0.247, ...])
(😺, 0.242, [0.038, 0.883, ...])
(😺, 0.245, [0.283, -0.953, ...])

RDD[(Int, Double, Factor)]

(🏀, 0.274, [0.445, -0.233, ...])
(🎾, 0.483, [0.843, 0.023, ...])
(⚽, 0.595, [0.284, -0.987, ...])
(🏈, 0.103, [0.340, 0.328, ...])
(⚾, 0.253, [0.472, -0.274, ...])

# Spark Streaming ALS

# Spark Streaming ALS

RDD[(Int, Double, Factor)]

({:thinking:, 0.932, [0.123, -0.140, ...]})

({:thinking:, 0.101, [0.334, 0.273, ...]})

# Spark Streaming ALS

RDD[(Int, Double, Factor)]

(😊, 0.932, [0.123, -0.140, ...])

(😊, 0.101, [0.334, 0.273, ...])

RDD[(Int, Double)]

(😊, 0.932)

(😊, 0.101)

$\nabla b_x$

# Spark Streaming ALS

RDD[(Int, Double, Factor)]

(😊, 0.932, [0.123, -0.140, ...])

(😊, 0.101, [0.334, 0.273, ...])

RDD[(Int, Double)]    RDD[(Int, Factor)]

(😊, 0.932)

(😊, 0.101)

(😊, [0.123, ...])

(😊, [0.334, ...])

$\nabla b_x$

$\nabla U_x$

# Spark Streaming ALS

RDD[(Int, Double, Factor)]

(😊, 0.932, [0.123, -0.140, ...])

(😊, 0.101, [0.334, 0.273, ...])

RDD[(Int, Double, Factor)]

(🏀, 0.274, [0.445, -0.233, ...])

(⚽, 0.483, [0.843, 0.023, ...])

RDD[(Int, Double)]

(😊, 0.932)

(😊, 0.101)

RDD[(Int, Factor)]

(😊, [0.123, ...])

(😊, [0.334, ...])

$\nabla b_x$

$\nabla U_x$

# Spark Streaming ALS

RDD[(Int, Double, Factor)]

(😊, 0.932, [0.123, -0.140, ...])

(😊, 0.101, [0.334, 0.273, ...])

RDD[(Int, Double, Factor)]

(🏀, 0.274, [0.445, -0.233, ...])

(⚽, 0.483, [0.843, 0.023, ...])

RDD[(Int, Double)]

(😊, 0.932)

(😊, 0.101)

RDD[(Int, Factor)]

(😊, [0.123, ...])

(😊, [0.334, ...])

RDD[(Int, Double)]

(🏀, 0.274)

(⚽, 0.483)

$\nabla b_x$

$\nabla U_x$

$\nabla b_y$

# Spark Streaming ALS

RDD[(Int, Double, Factor)]

(😊, 0.932, [0.123, -0.140, ...])

(😊, 0.101, [0.334, 0.273, ...])

RDD[(Int, Double, Factor)]

(🏀, 0.274, [0.445, -0.233, ...])

(⚽, 0.483, [0.843, 0.023, ...])

RDD[(Int, Double)]

(😊, 0.932)

(😊, 0.101)

RDD[(Int, Factor)]

(😊, [0.123, ...])

(😊, [0.334, ...])

RDD[(Int, Double)]

(🏀, 0.274)

(⚽, 0.483)

RDD[(Int, Factor)]

(🏀, [0.445, ...])

(⚽, [0.843, ...])

$\nabla b_x$

$\nabla U_x$

$\nabla b_y$

$\nabla P_y$

# Spark Streaming ALS

# Spark Streaming ALS

RDD[(Int, Double)]

(	😊	,	0.932
(	😊	,	0.101

$$b(\text{😊}) += \sum \nabla b(\text{😊})$$

# Spark Streaming ALS

RDD[(Int, Double)]    RDD[(Int, Factor)]

({:)	0.932
({:)	0.101

({:)	[0.123, ...]
({:)	[0.334, ...]

$$b(\text{{:}}) += \sum \nabla b(\text{{:}})$$

$$U(\text{{:}}) += \sum \nabla U(\text{{:}})$$

# Spark Streaming ALS

RDD[(Int, Double)]

({:P, 0.932)
({:P, 0.101)

RDD[(Int, Factor)]

({:P, [0.123, ...])
({:P, [0.334, ...])

RDD[(Int, Double)]

({:B, 0.274)
({:B, 0.483)

$$b(\text{P}) += \sum \nabla b(\text{P})$$

$$U(\text{P}) += \sum \nabla U(\text{P})$$

$$b(\text{B}) += \sum \nabla b(\text{B})$$

# Spark Streaming ALS

RDD[(Int, Double)]

({:)	0.932
({:)	0.101

RDD[(Int, Factor)]

({:)	[0.123, ...]
({:)	[0.334, ...]

RDD[(Int, Double)]

({:)	0.274
({:)	0.483

RDD[(Int, Factor)]

({:)	[0.445, ...]
({:)	[0.843, ...]

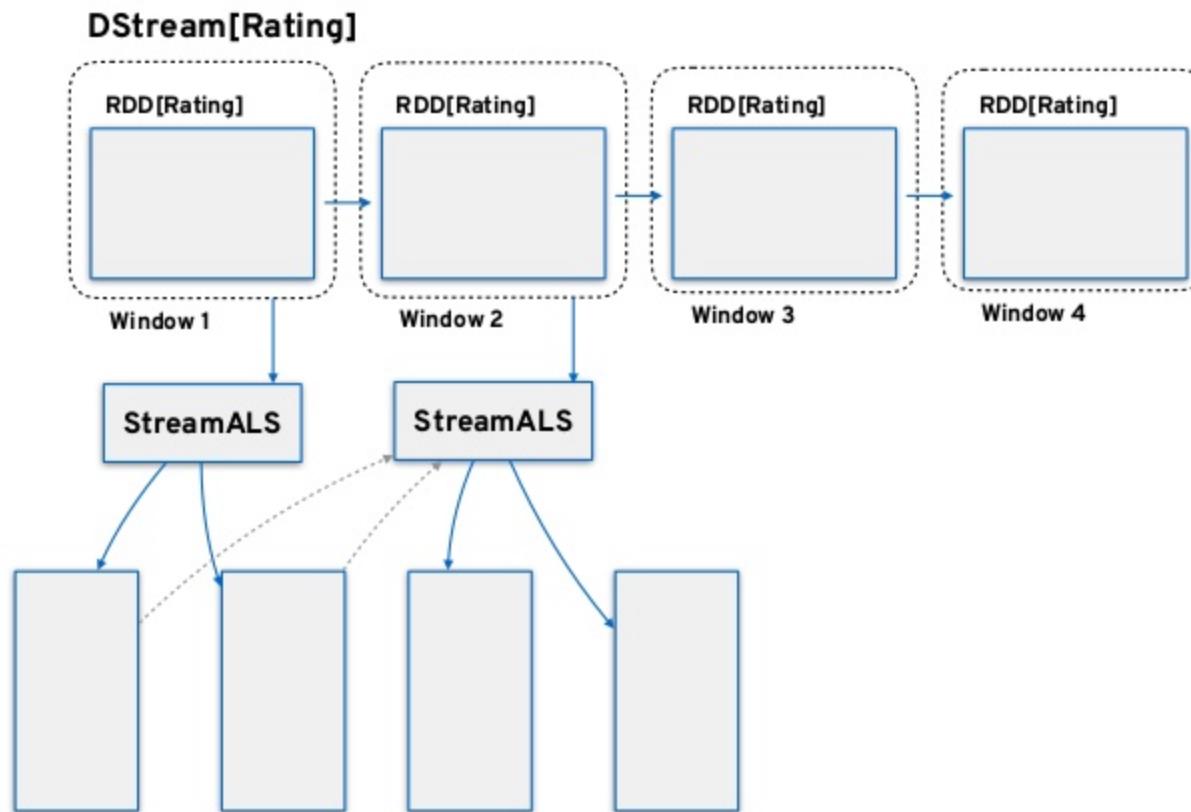
$$b({:}) += \sum \nabla b({:})$$

$$U({:}) += \sum \nabla U({:})$$

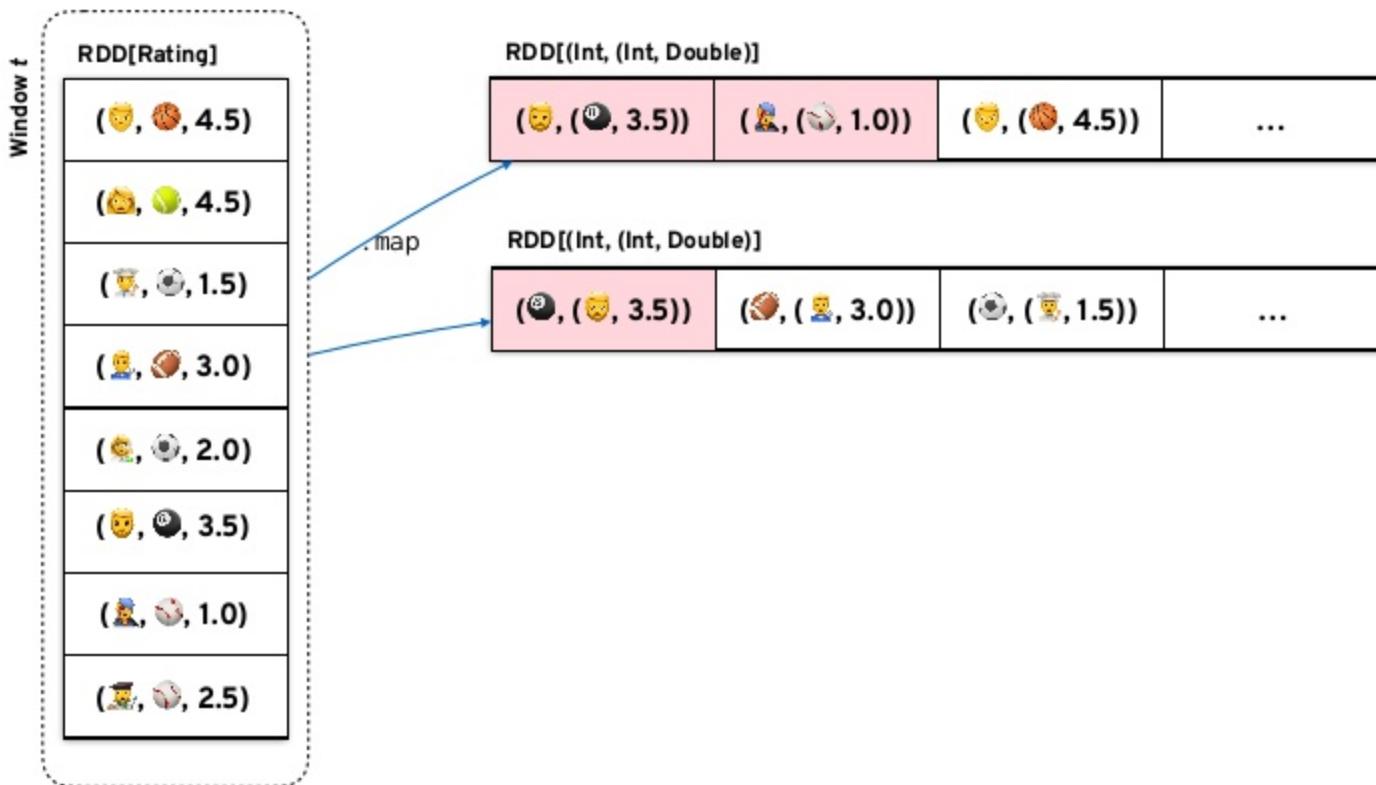
$$b({:}) += \sum \nabla b({:})$$

$$U({:}) += \sum \nabla U({:})$$

# Spark Streaming ALS



# Spark Streaming ALS



# Spark Streaming ALS

RDD[(Int, (Int, Double))]

(😊, (🏀, 4.5))	(😊, (⚽, 4.5))	(😊, (⌚, 1.5))	...
---------------	---------------	---------------	-----

.fullOuterJoin

RDD[(Int, (Int, Factor))]

(😊, (0.12, [0.9,...]))
(😊, (-0.1, [1.37,...])))
(⌚, (1, [0.123,...])))
...

(😊, (🏀, 4.5), 😊, (0.12, [0.9,...])))

(😊, (⚽, 4.5), (😊, (-0.1, [1.37,...])))

(😊, (⌚, 1.5), None)

...

RDD[(Int, (Int, Double), Int, Factor)]

```
userRatings.fullOuterJoin(userFactors).map {  
    case (userId, (_, userFactors)) =>  
        (userId, userFactors(featureGenerator.nextIntValue()))  
    }  
}
```

# Data

MovieLens

Widely used in recommendation engine research

Variants

Small - 100,000 ratings / 9,000 movies / 700 users

Full - 26 million ratings / 45,000 movies / 270,000

# Training batch ALS

```
val split: Array[RDD[Rating]] = ratings.randomSplit(0.8, 0.2)  
val model = ALS.train(split(0), rank, iter, lambda)
```

# Training batch ALS

```
val split: Array[RDD[Rating]] = ratings.randomSplit(0.8, 0.2)

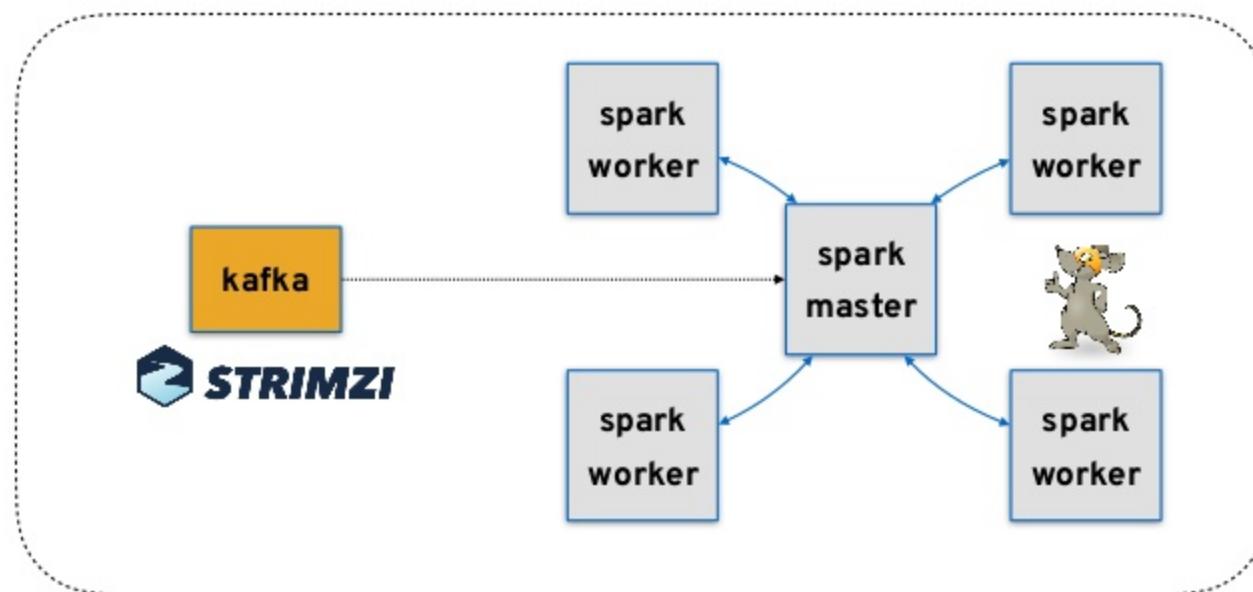
val model = ALS.train(split(0), rank, iter, lambda)

val predictions: RDD[Rating] = model.predict(split(1).map { x =>
    (x.user, x.product)
}

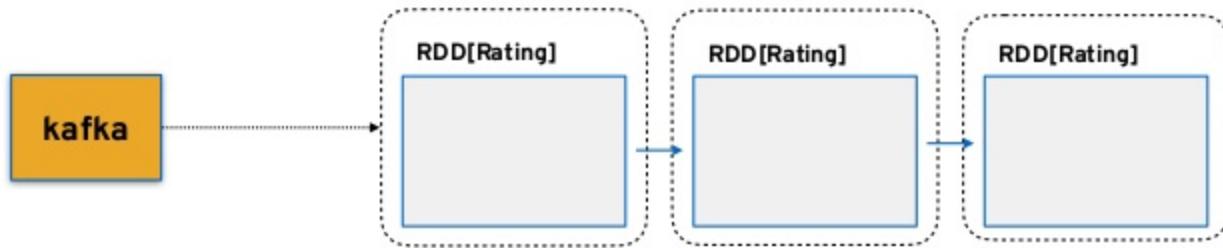
val pairs = predictions.map(x => ((x.user, x.product), x.rating))
    .join(split(1).map(x => ((x.user, x.product), x.rating)))
    .values

Val RMSE = math.sqrt(pairs.map(x => math.pow(x._1 - x._2, 2)).mean())
```

# Training streaming ALS

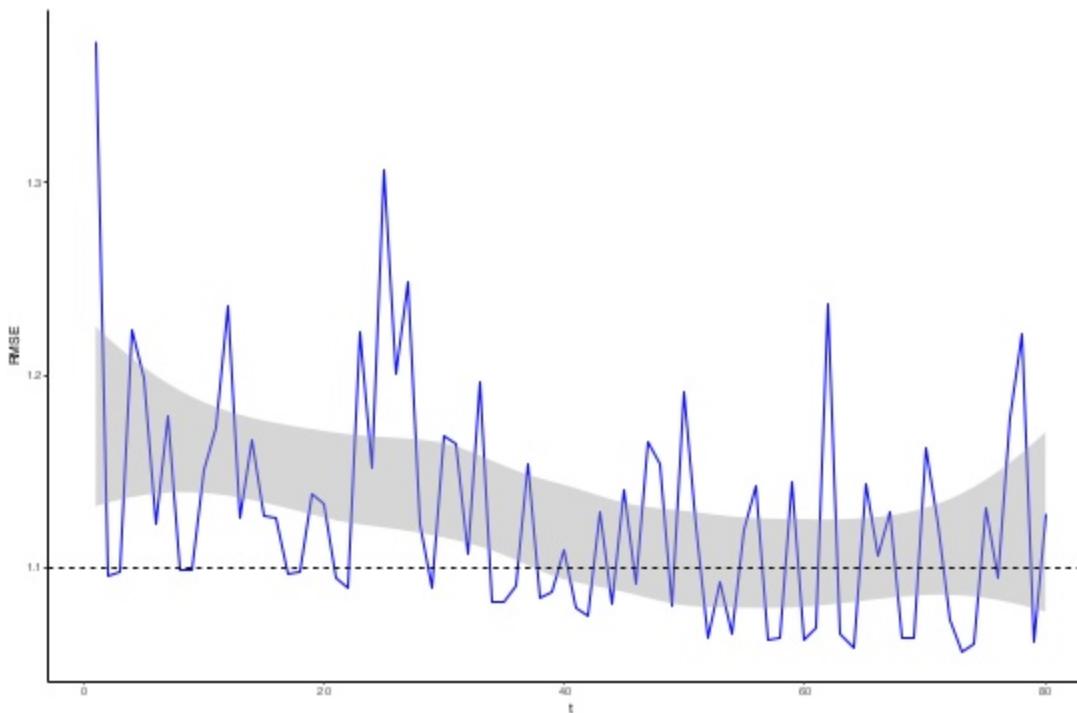


# Training streaming ALS



```
val model = StreamingALS(rank, iterations, lambda, gamma)  
trainingStreamSet.foreachRDD { rdd =>  
    model.train(rdd)  
    val RMSE = calculateRMSE(model, validation)  
}
```

# Comparison





**TO CONSIDER...**

# To consider

“Cold start”

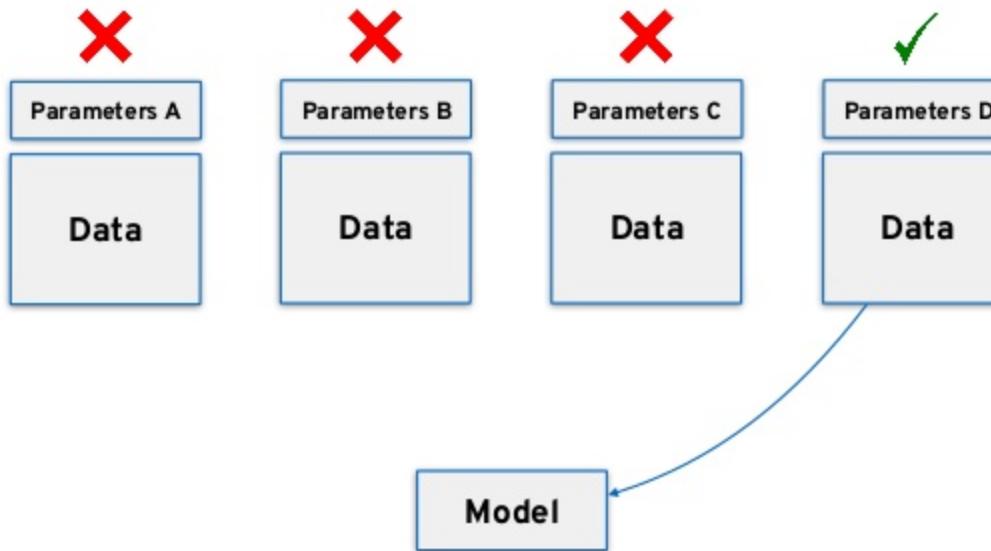
Same as batch ALS

Too few observations = meaningless

Train offline

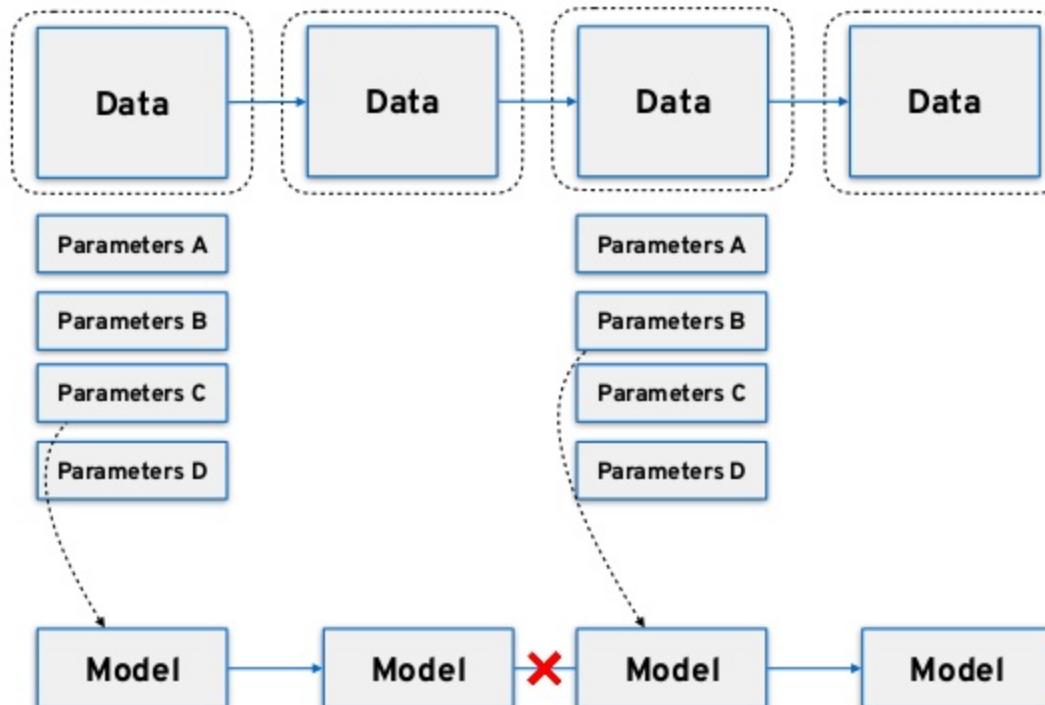
# To consider

## Hyper-parameter estimation



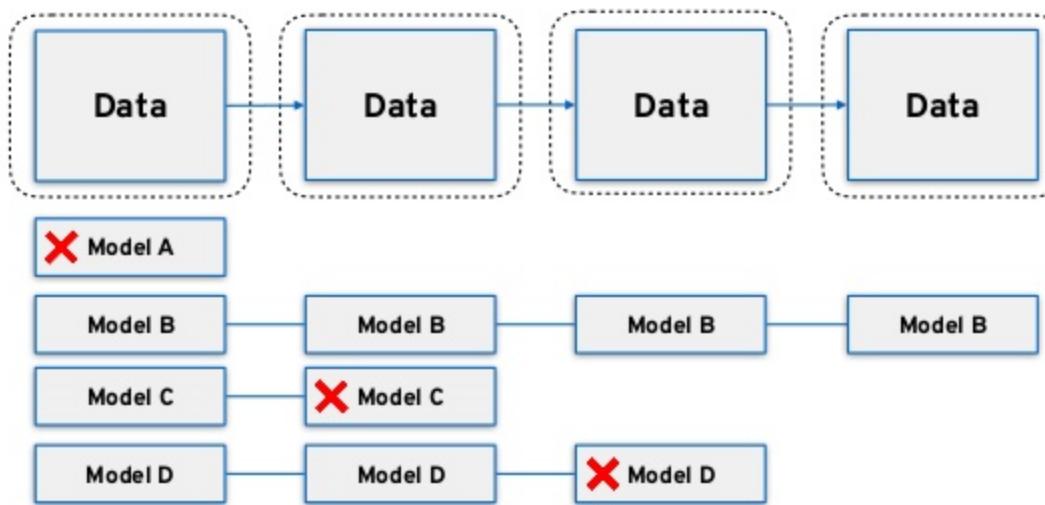
# To consider

## Hyper-parameter estimation



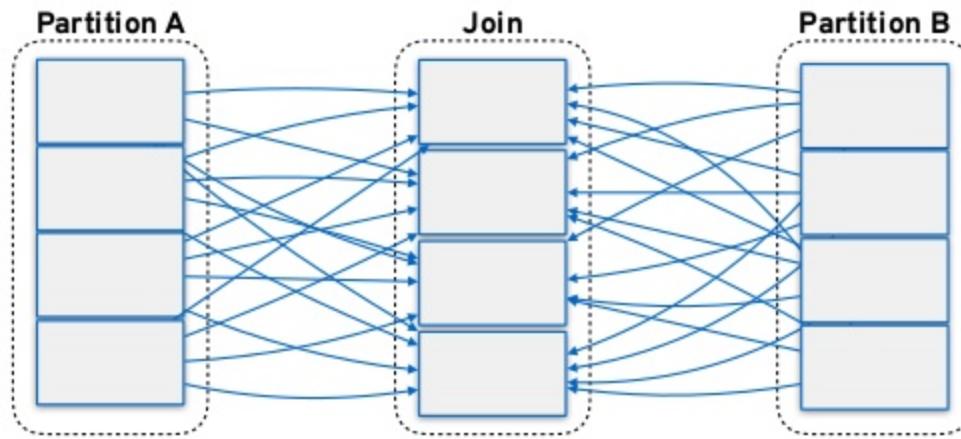
# To consider

## Hyper-parameter estimation



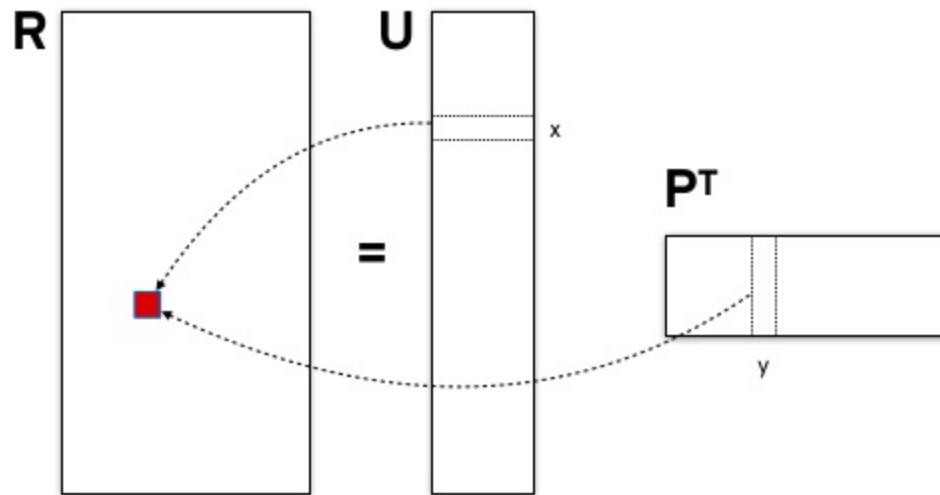
# To consider

## Partitioning



# To consider

RDD random access?



```
val u = userFeatures.lookup(userId)
val p = productFeatures.lookup(productId)
val predicted = model.predict(userId, productId, u, p, globalBias)
```

# Links

Blog post:

<https://ruivieira.github.io/a-streaming-als-implementation.html>

[radanalytics.io](http://radanalytics.io)



SPARK+AI  
SUMMIT EUROPE



THANK YOU