MLlib: Spark's Machine Learning Library

Ameet Talwalkar

LIBLINEAR?

Vowpal Wabbit?

R?

Mahout?

scikit-learn?

Weka?

Matlab?

- Performance / Scalability?
- Simple development environment?
- Integration with other data processing components?

LIBLINEAR?

Vowpal Wabbit?

R?

Mahout?

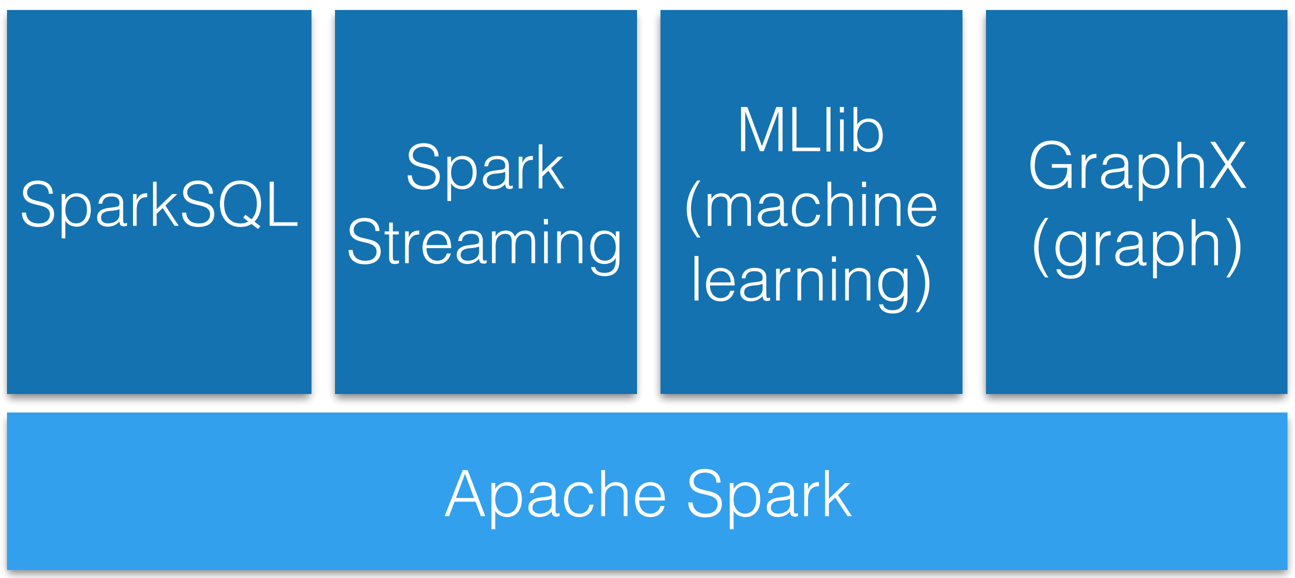
scikit-learn?

Weka?

Matlab?

MLlib

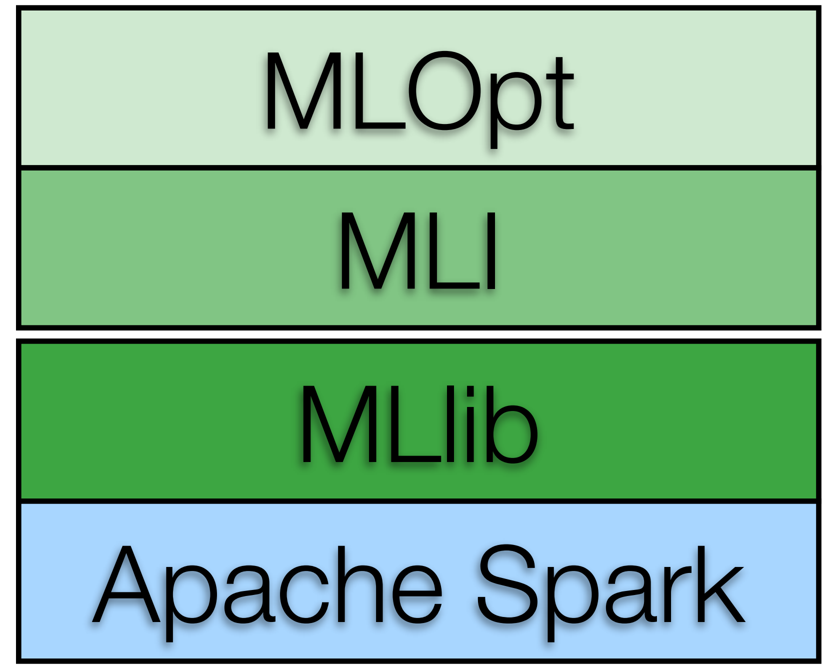
+ Simple development environment (Spark)
+ Scalable and fast
+ Part of Apache Spark Ecosystem



Overview
Examples
Roadmap

MLbase and MLlib

MLbase Goal:
Simplify development and deployment of ML pipelines



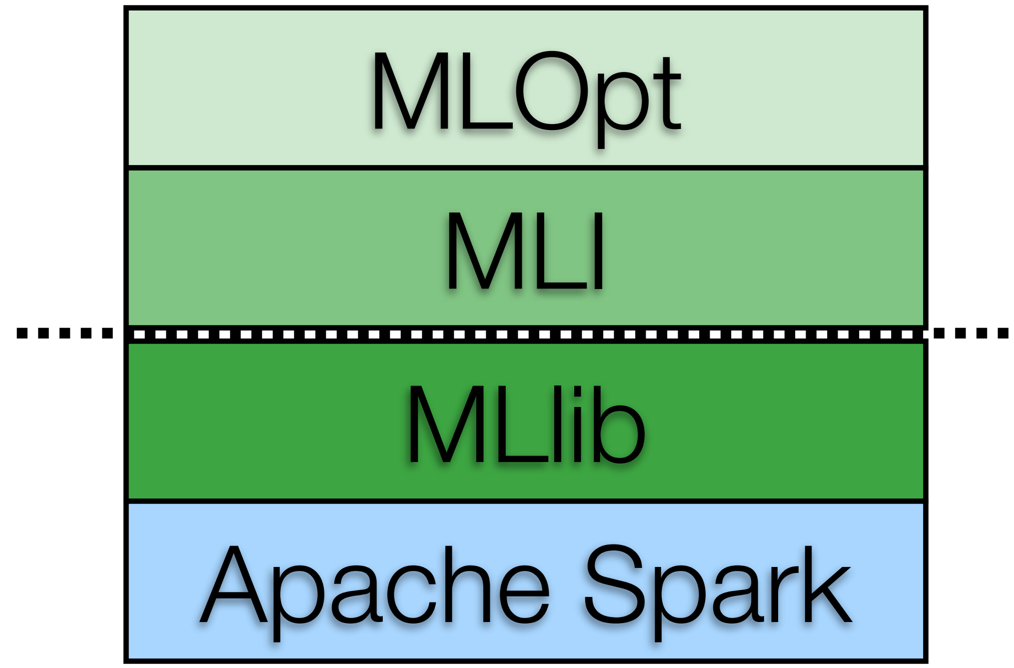
MLOpt: Autotuners for ML pipelines

MLI: Experimental API to simplify ML development

MLlib: Spark's core ML library

MLbase and MLlib

MLbase Goal:
Simplify development and deployment of ML pipelines



MLOpt and MLI are experimental testbed

See video of Evan Spark's talk from Day 2 of Spark Summit

MLOpt: Autotuners for ML pipelines

MLI: Experimental API to simplify ML development

MLlib: Spark's core ML library

Active Development

Initial Release

- Developed by AMPLab (11 contributors)
- Shipped with Spark v0.8 (Sep 2013)

DATABRICKS

Active Development

Initial Release

- Developed by AMPLab (11 contributors)
- Shipped with Spark v0.8 (Sep 2013)

Current Version

- 48 contributors from various organizations
- Shipped with Spark v1.0 (May 2014)

Algorithms in v0.8

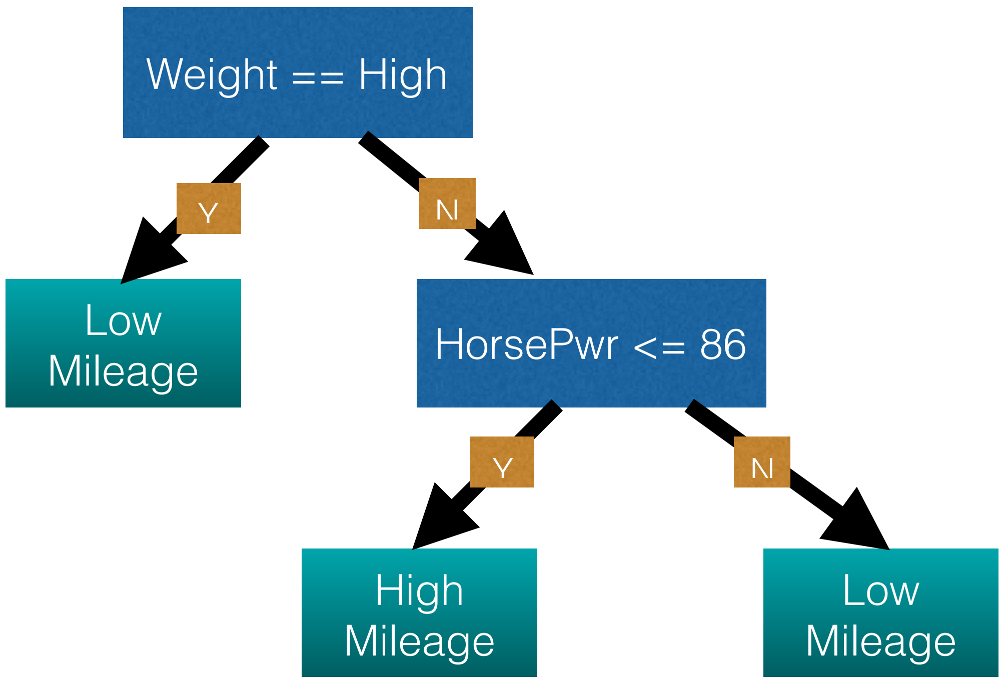
- **classification:** logistic regression, linear support vector machines (SVM)
- **regression:** linear regression
- **collaborative filtering:** alternating least squares (ALS)
- **clustering:** k-means
- **optimization**: stochastic gradient descent (SGD)

Algorithms in v1.0

- **classification:** logistic regression, linear support vector machines (SVM), naive Bayes, decision trees
- **regression:** linear regression, regression trees
- **collaborative filtering:** alternating least squares (ALS)
- **clustering:** k-means
- **optimization**: stochastic gradient descent (SGD), limited-memory BFGS (L-BFGS)
- **dimensionality reduction:** singular value decomposition (SVD), principal component analysis (PCA)

Distributed Decision Trees

- Interpretable, supports categorical variables / missing data, ensembles are top performers
- Classification and regression
- Scales to massive datasets
- See video of Manish Amde's talk from Day 1 of Spark Summit



DATABRICKS

What else is new in v1.0?

- Improved user guide
- Code examples / templates
- API stability
- Sparse data support
- Distributed matrices
- Binary classification model evaluation

User guide: Improved Organization

Machine Learning Library (MLlib)

MLlib is a Spark implementation of some common machine learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives:

- Basics
  - data types
  - summary statistics
- Classification and regression
  - linear support vector machine (SVM)
  - logistic regression
  - linear least squares, Lasso, and ridge regression
  - decision tree
  - naive Bayes
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Optimization
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)

User guide: Improved Organization

Scala **Java** **Python**

NaiveBayes implements multinomial naive Bayes. It takes an RDD of LabeledPoint and an optionally smoothing parameter `lambda` as input, and output a NaiveBayesModel, which can be used for evaluation and prediction.

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import NaiveBayes

# an RDD of LabeledPoint
data = sc.parallelize([
  LabeledPoint(0.0, [0.0, 0.0])
  ... # more labeled points
])

# Train a naive Bayes model.
model = NaiveBayes.train(data, 1.0)

# Make prediction.
prediction = model.predict([0.0, 0.0])
```

Code examples

Useful as templates for standalone applications

Code examples

Useful as templates for standalone applications

```
MovieLensALS: an example app for ALS on MovieLens data.
Usage: MovieLensALS [options] <input>

  --rank <value>
        rank, default: 10
  --numIterations <value>
        number of iterations, default: 20
  --lambda <value>
        lambda (smoothing constant), default: 1.0
  --kryo
        use Kryo serialization
  --implicitPrefs
        use implicit preference
  <input>
        input paths to a MovieLens dataset of ratings
```

Code examples

Useful as templates for standalone applications

In “examples/” folder with sample datasets

- binary classification (SVM and logistic regression)
- decision tree
- naive Bayes
- k-means
- linear regression
- tall-and-skinny PCA and SVD
- collaborative filtering

API Stability

- Following Spark core, MLlib is guaranteeing binary compatibility for all 1.x releases on stable APIs
- For changes in experimental and developer APIs, we will provide migration guide between releases
- Unified API docs for various Spark components

DATABRICKS

Exploiting Sparsity

Sparse data is prevalent

- Text processing: bag-of-words, n-grams
- Collaborative Filtering: ratings matrix
- Graphs: adjacency matrix
- Genomics: SNPs, variant calling

Exploiting Sparsity

Sparse data is prevalent

- Text processing: bag-of-words, n-grams
- Collaborative Filtering: ratings matrix
- Graphs: adjacency matrix
- Genomics: SNPs, variant calling

MLlib supports sparse storage and computation

- classification
- k-means
- summary statistics

$$\text{dense} : \; 1. \;\; 0. \;\; 0. \;\; 0. \;\; 0. \;\; 0. \;\; 3.$$

$$\text{sparse} : \begin{cases} \text{size} : 7 \\ \text{indices} : 0 \;\; 6 \\ \text{values} : 1. \;\; 3. \end{cases}$$

Exploiting Sparsity

Sparse data is prevalent

- Text processing: bag-of-words, n-grams
- Collaborative Filtering: ratings matrix
- Graphs: adjacency matrix
- Genomics: SNPs, variant calling

MLlib supports sparse storage and computation

- classification
- k-means
- summary statistics

dense : 1. 0. 0. 0. 0. 0. 3.

sparse :
- size : 7
- indices : 0 6
- values : 1. 3.

See video of Xiangrui Meng's talk from Day 1 of Spark Summit

Exploiting sparsity in k-means

Training set:

- 12 million examples
- 500 features
- sparsity: 10%

	dense	sparse
storage	47GB	7GB
time	240s	58s

40GB savings in storage, 4x speedup in computation

Overview

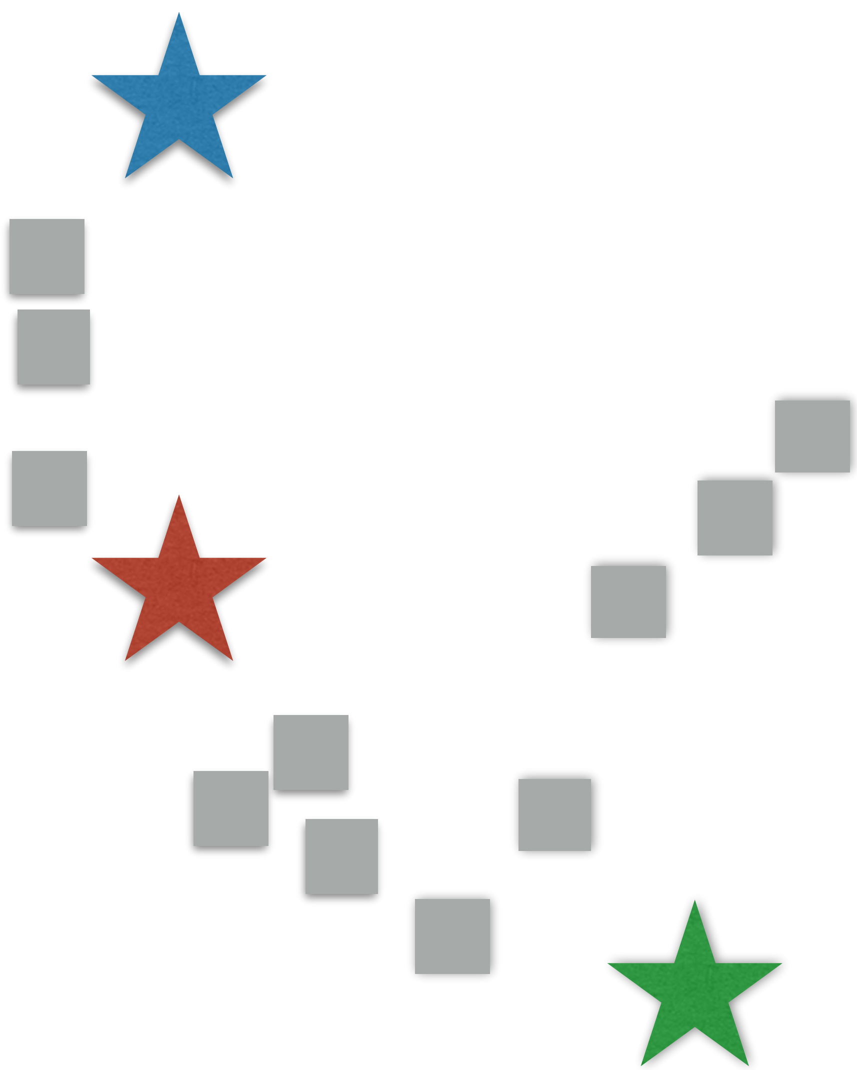
Examples

Roadmap

K-means clustering: Partition observations into k clusters

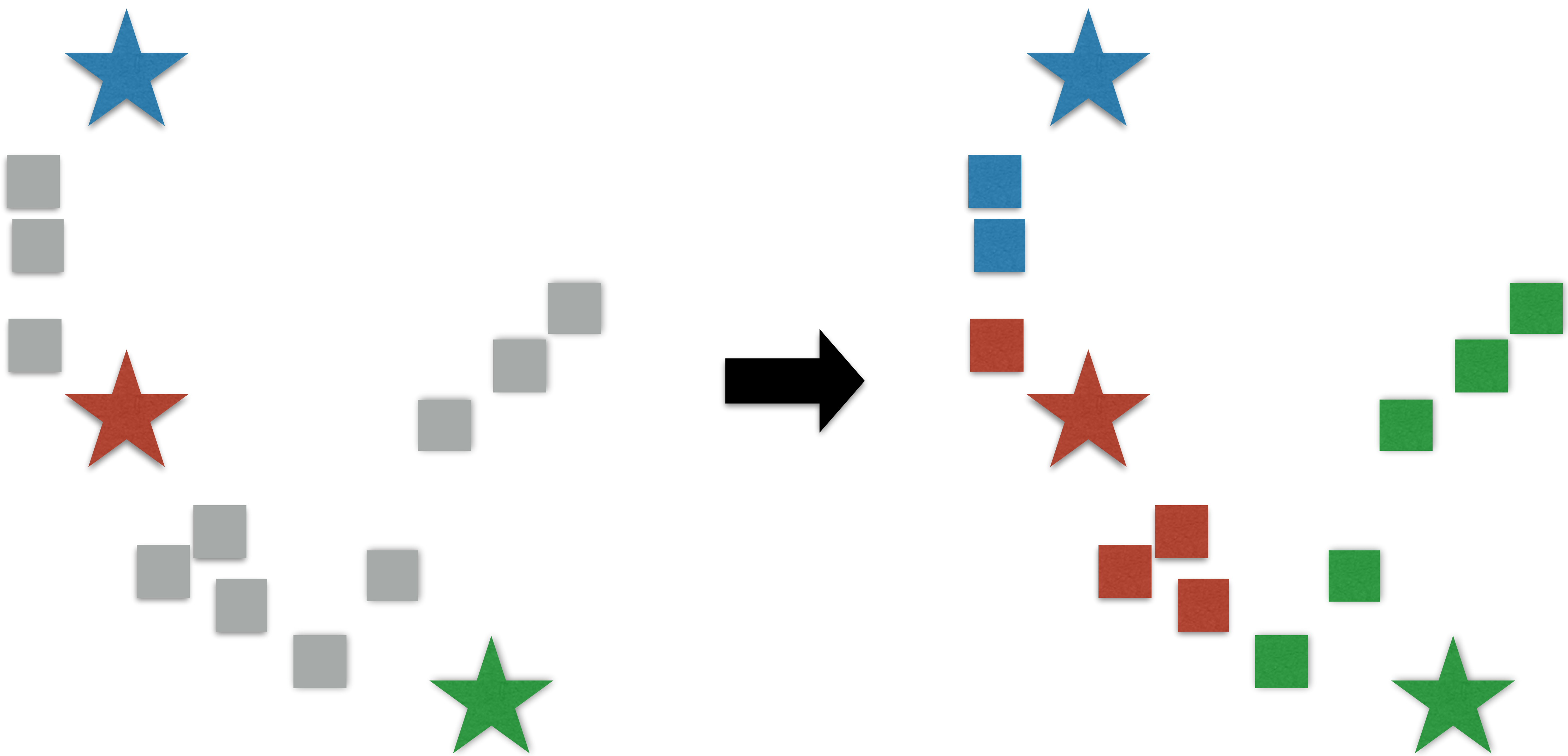
K-means clustering: Partition observations into k clusters

K-means clustering: Partition observations into k clusters



Compute/initialize
cluster centers

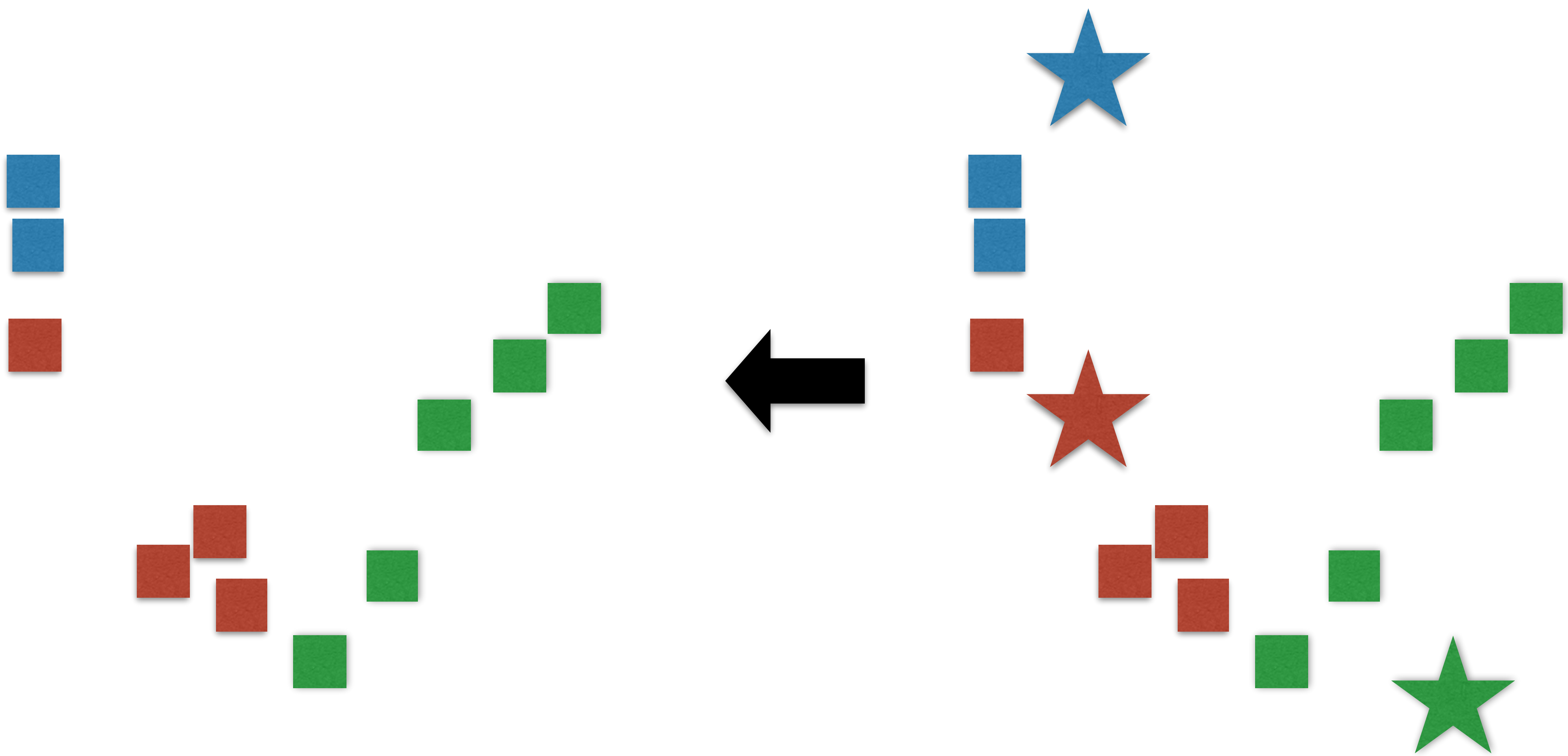
K-means clustering: Partition observations into k clusters



Compute/initialize cluster centers

Assign cluster membership

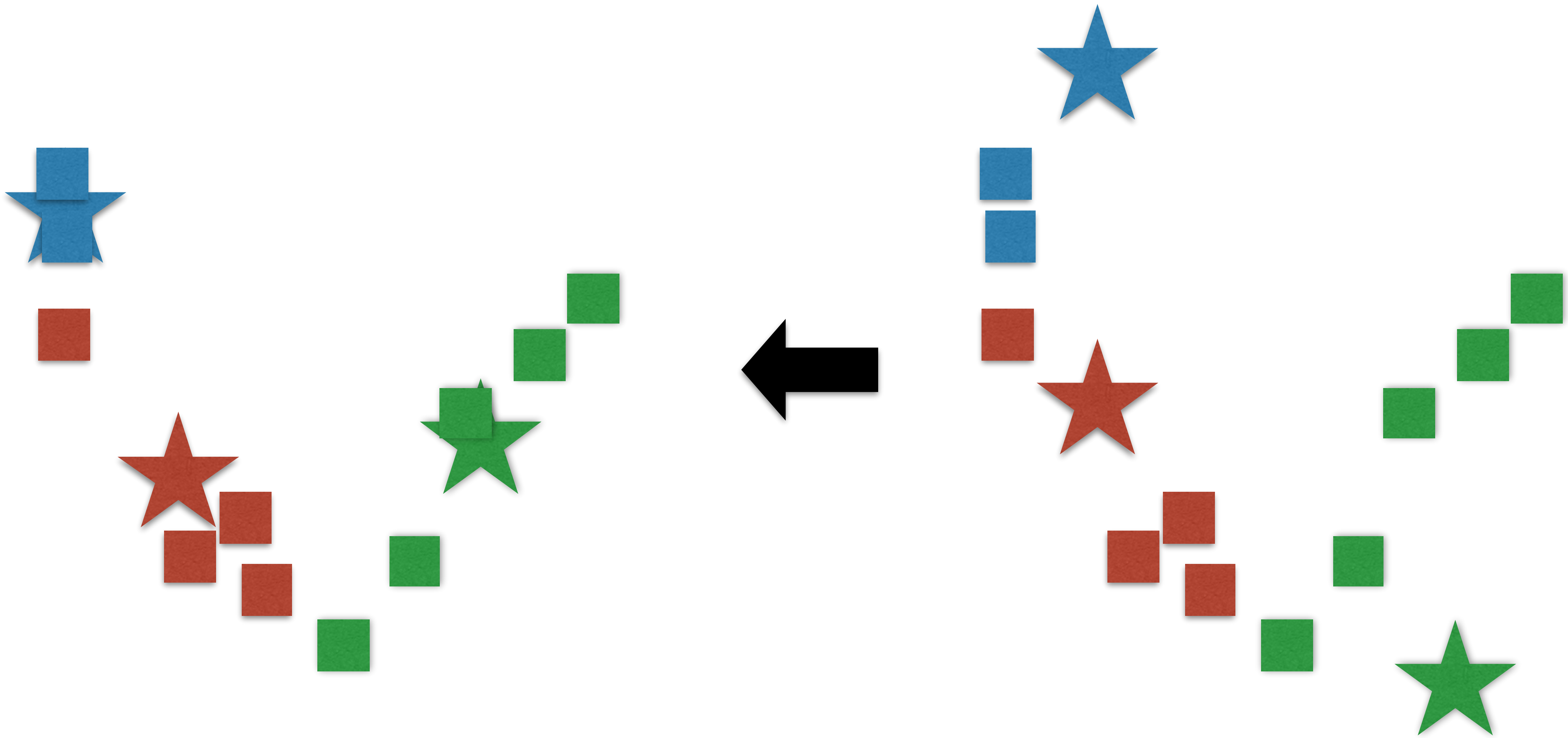
K-means clustering: Partition observations into k clusters



Compute/initialize cluster centers

Assign cluster membership

K-means clustering: Partition observations into k clusters

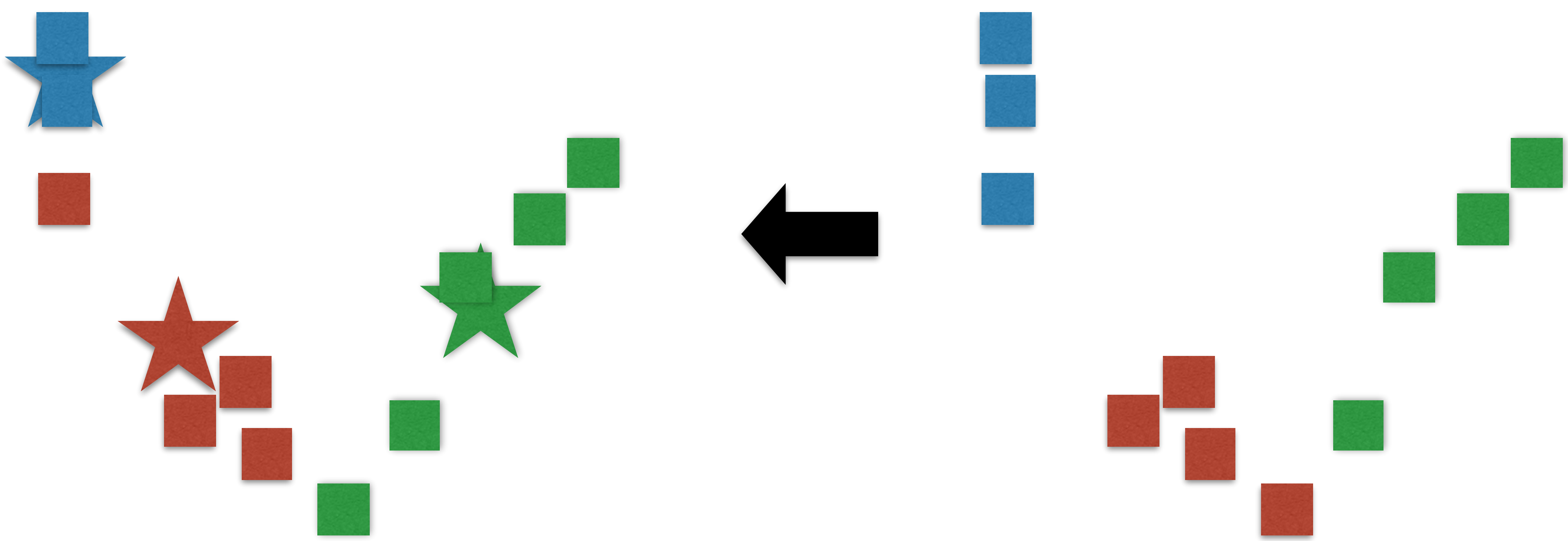


Compute/initialize cluster centers

Assign cluster membership

DATABRICKS

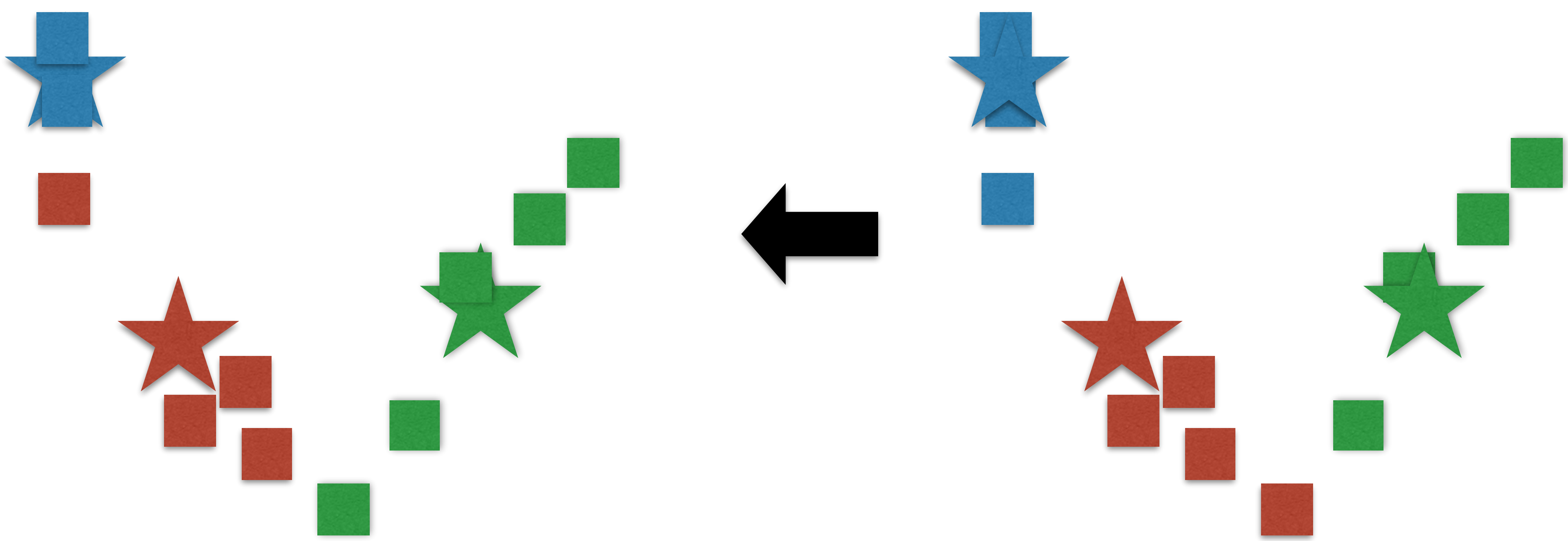
K-means clustering: Partition observations into k clusters



Compute/initialize cluster centers

Assign cluster membership

K-means clustering: Partition observations into k clusters



Compute/initialize cluster centers

Assign cluster membership

K-means (scala)

```
// Load and parse the data.
val data = sc.textFile("kmeans_data.txt")
val parsedData = data.map(_.split(' ').map(_.toDouble)).cache()

// Cluster the data into five classes using KMeans.
val clusters = KMeans.train(parsedData, 5, numIterations = 20)

// Compute the sum of squared errors.
val cost = clusters.computeCost(parsedData)
println("Sum of squared errors = " + cost)
```

K-means (python)

```
# Load and parse the data
data = sc.textFile("kmeans_data.txt")
parsedData = data.map(lambda line:
            array([float(x) for x in line.split(' ‘)])).cache()

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 5, maxIterations = 20,
           runs = 1, initialization_mode = "kmeans||")

# Evaluate clustering by computing the sum of squared errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

cost = parsedData.map(lambda point: error(point))
        .reduce(lambda x, y: x + y)
print("Sum of squared error = " + str(cost))
```

Dimensionality reduction + K-means

```
// compute principal components
val points: RDD[Vector] = ...
val mat = RowMatrix(points)
val pc = mat.computePrincipalComponents(20)

// project points to a low-dimensional space
val projected = mat.multiply(pc).rows

// train a k-means model on the projected data
val model = KMeans.train(projected, 10)
```

Streaming + MLlib

```
// collect tweets using streaming

// train a k-means model
val model: KMmeansModel = ...

// apply model to filter tweets
val tweets = TwitterUtils.createStream(ssc, Some(authorizations(0)))
val statuses = tweets.map(_.getText)
val filteredTweets =
  statuses.filter(t => model.predict(featurize(t)) == clusterNumber)

// print tweets within this particular cluster
filteredTweets.print()
```

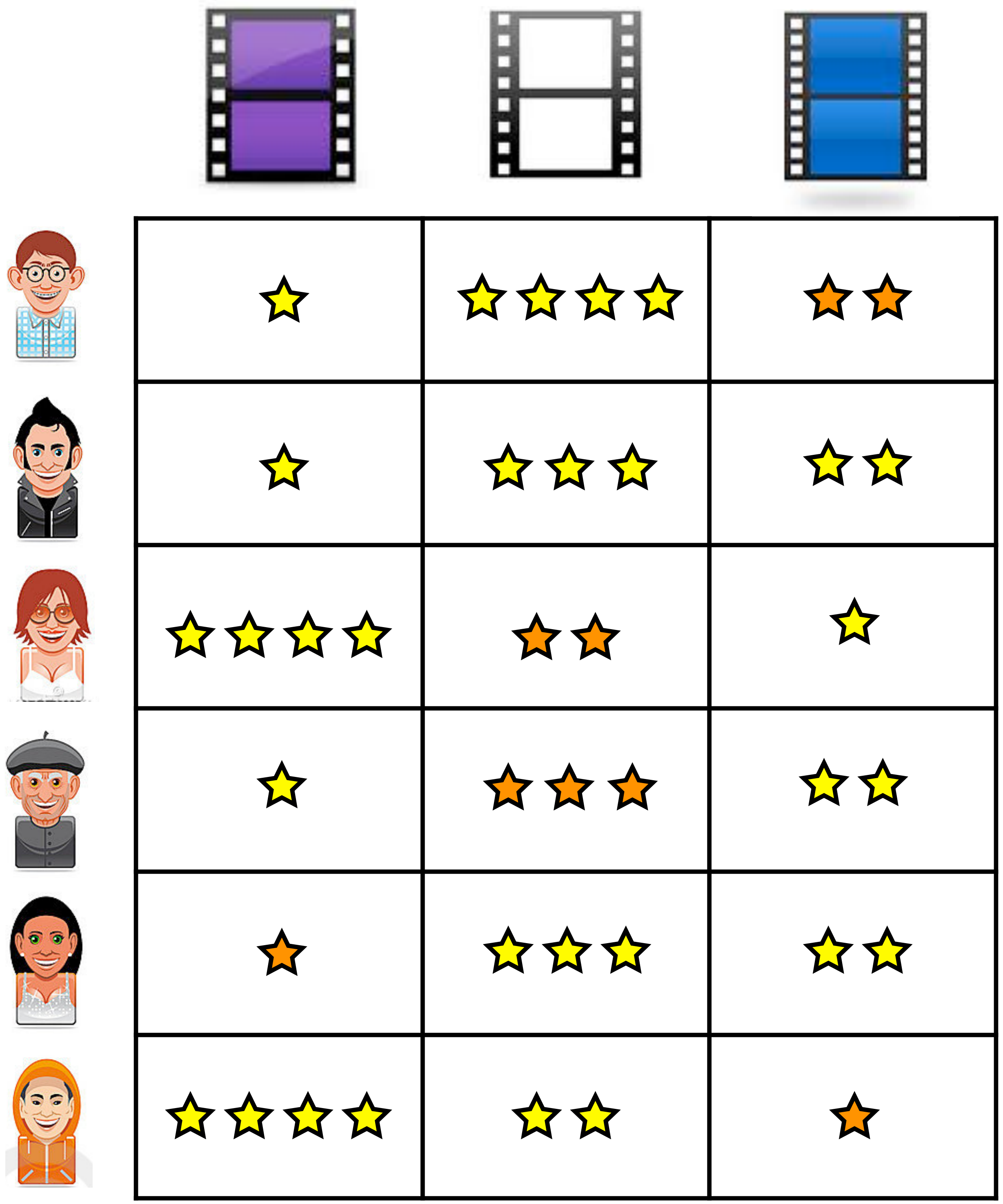
Streaming + MLlib

```
// collect tweets using streaming

// train a k-means model
val model: KMmeansModel = ...

// apply model to filter tweets
val tweets = TwitterUtils.createStream(ssc, Some(authorizations(0)))
val statuses = tweets.map(_.getText)
val filteredTweets =
  statuses.filter(t => model.predict(featurize(t)) == clusterNumber)
```

See video of Aaron Davidson's talk at last month's Hadoop Summit for extended demo: http://youtu.be/sPhyePwo7FA

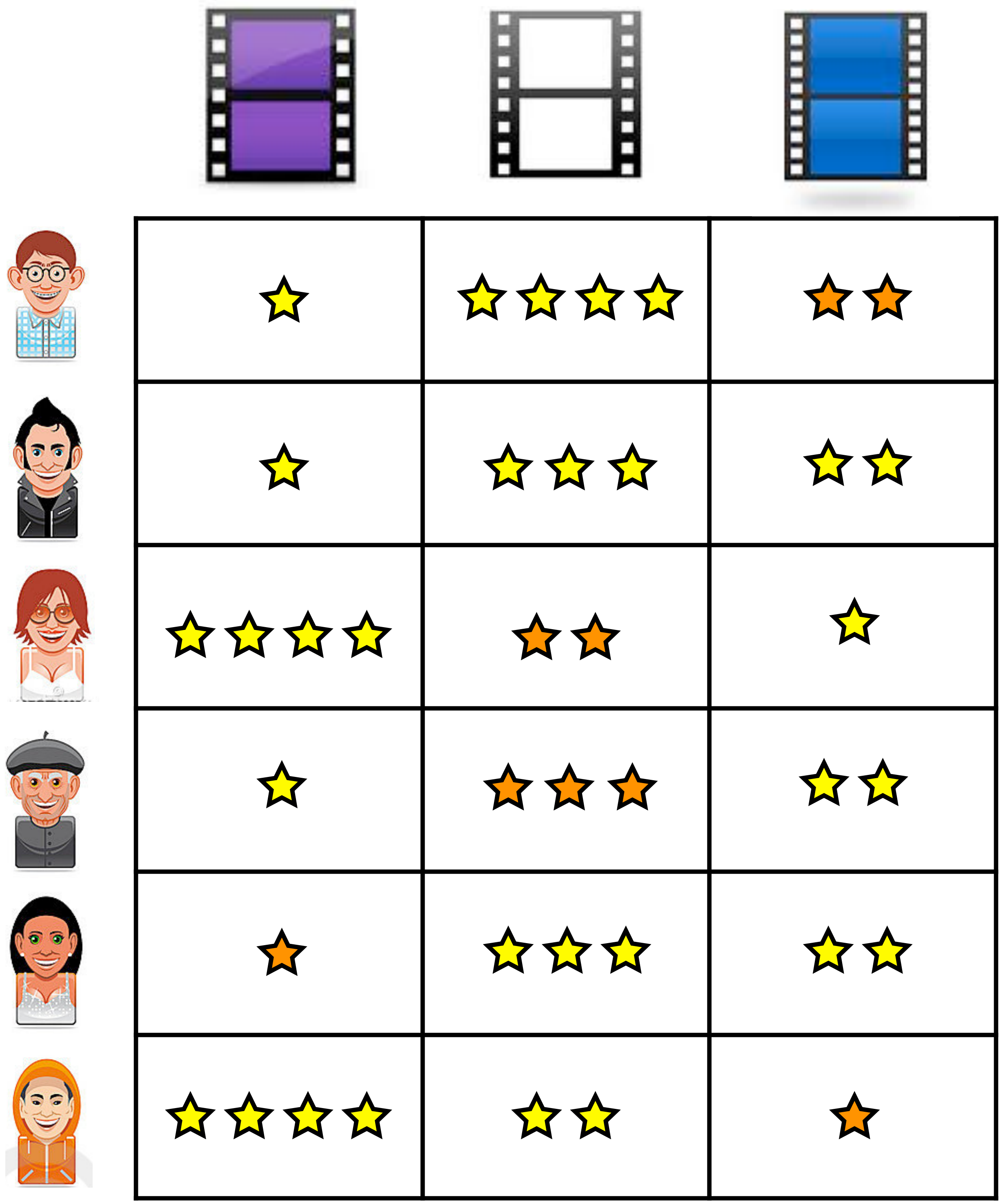
DATABRICKS

Collaborative Filtering



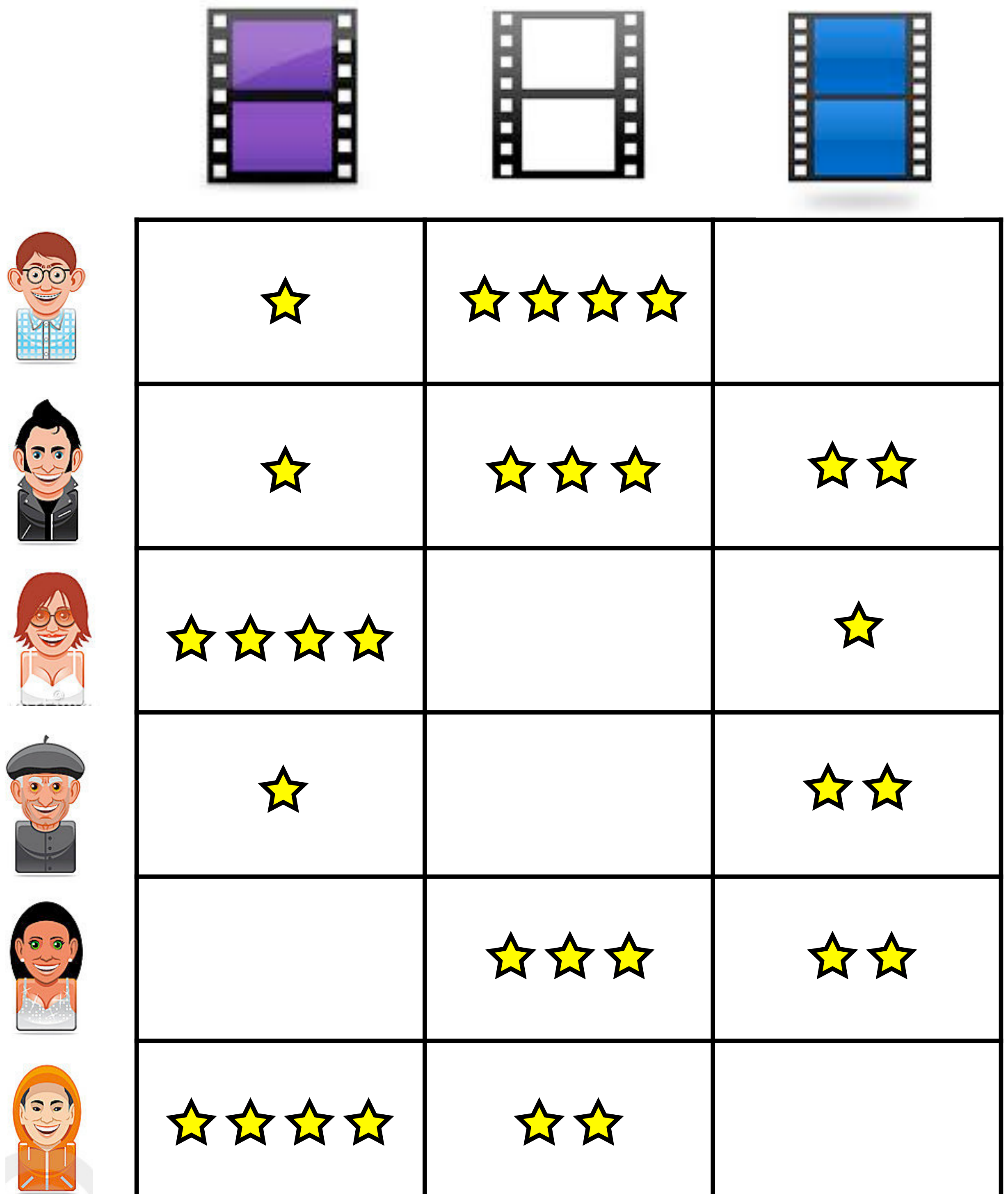
Goal: Recover a matrix from a subset of its entries

Collaborative Filtering

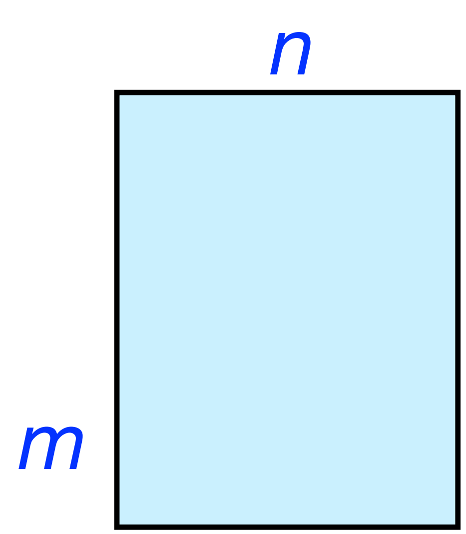


Goal: Recover a matrix from a subset of its entries

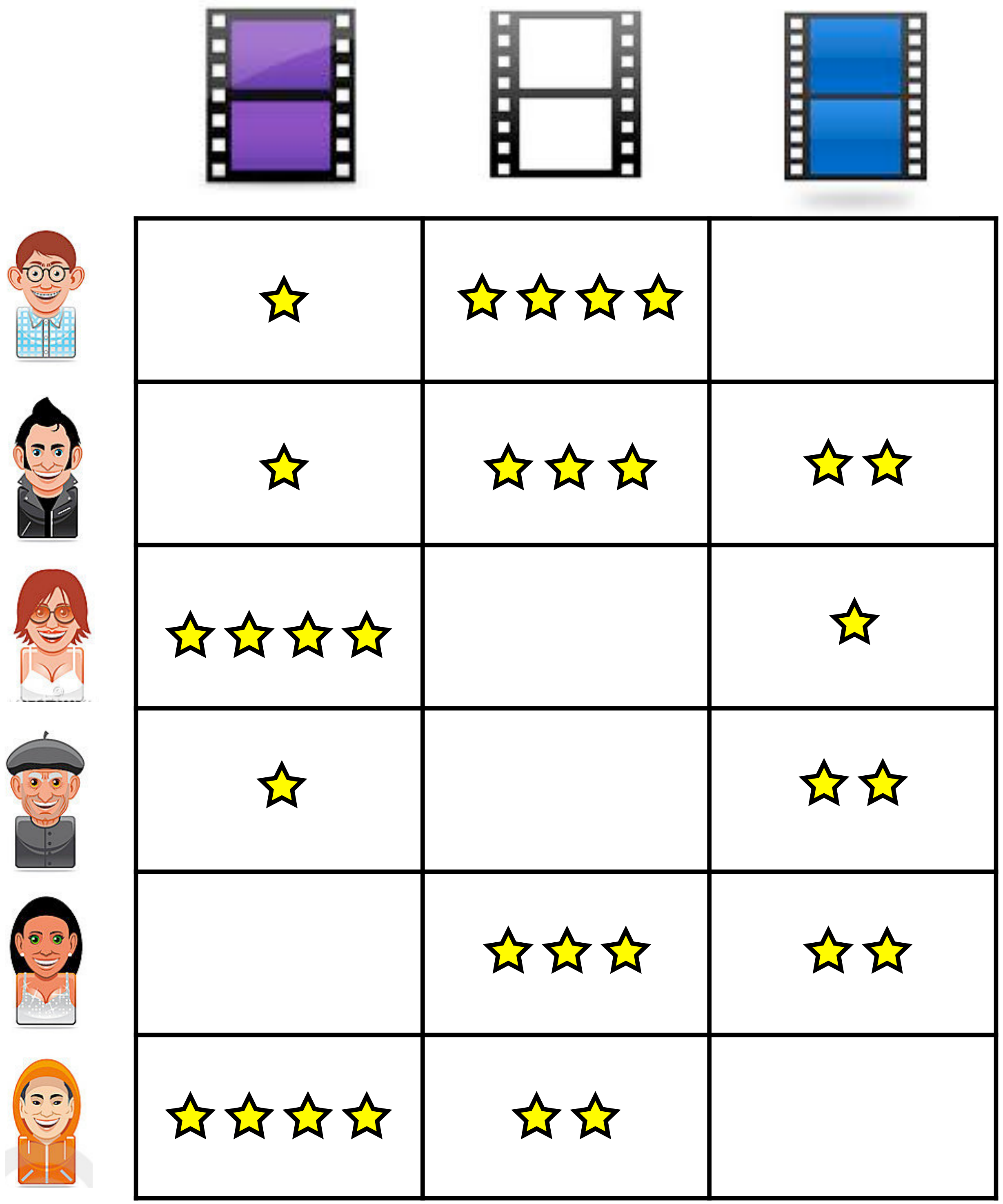
Reducing Degrees of Freedom



- Problem: Impossible without additional information
  - *mn* degrees of freedom

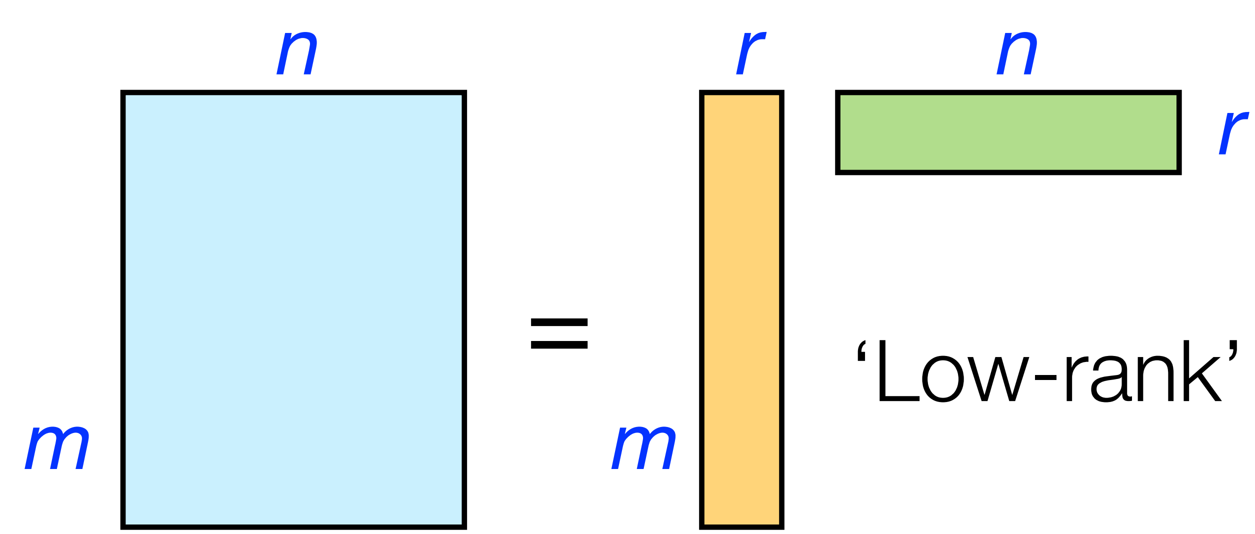


Reducing Degrees of Freedom

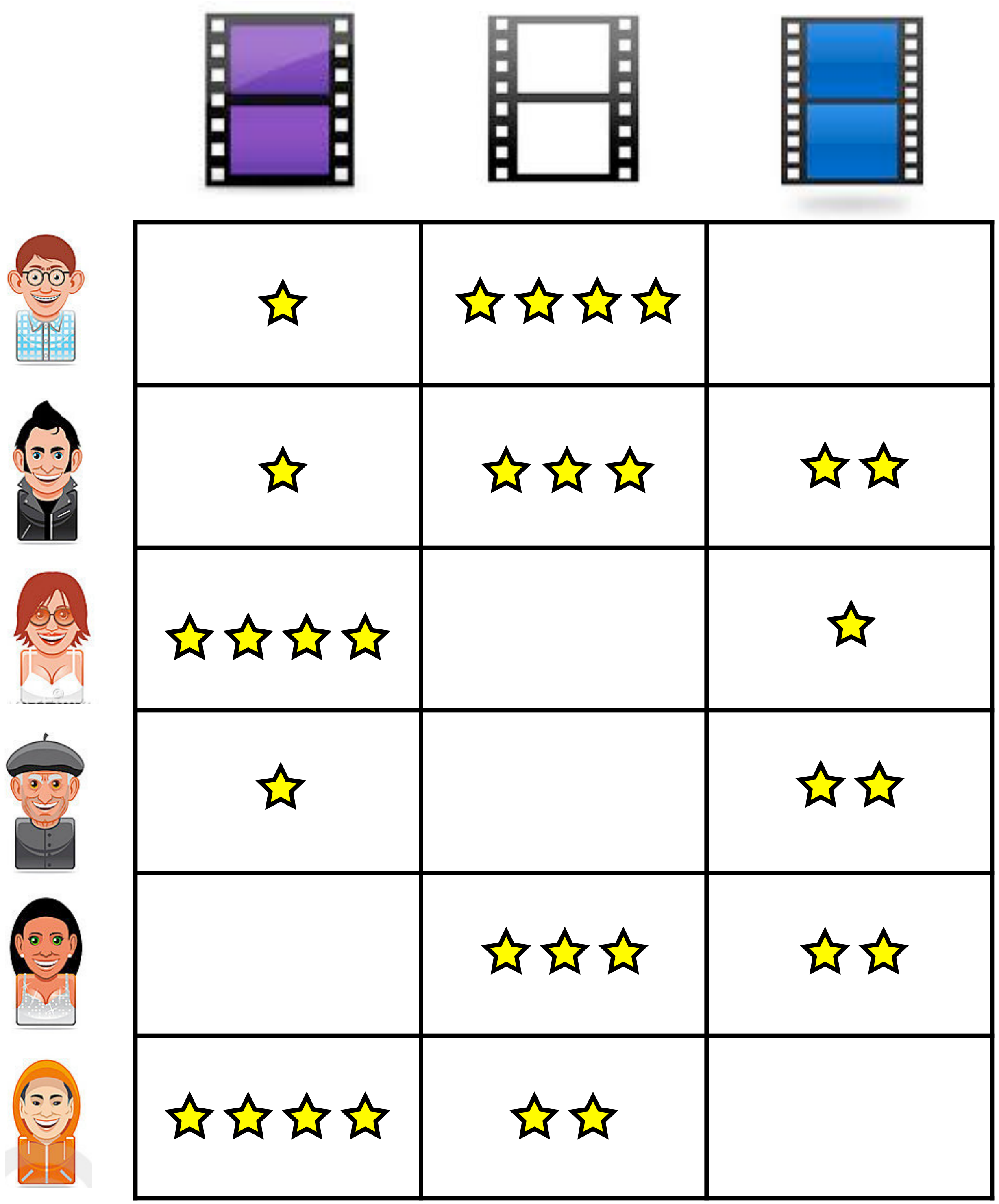


- Problem: Impossible without additional information
  - *mn* degrees of freedom

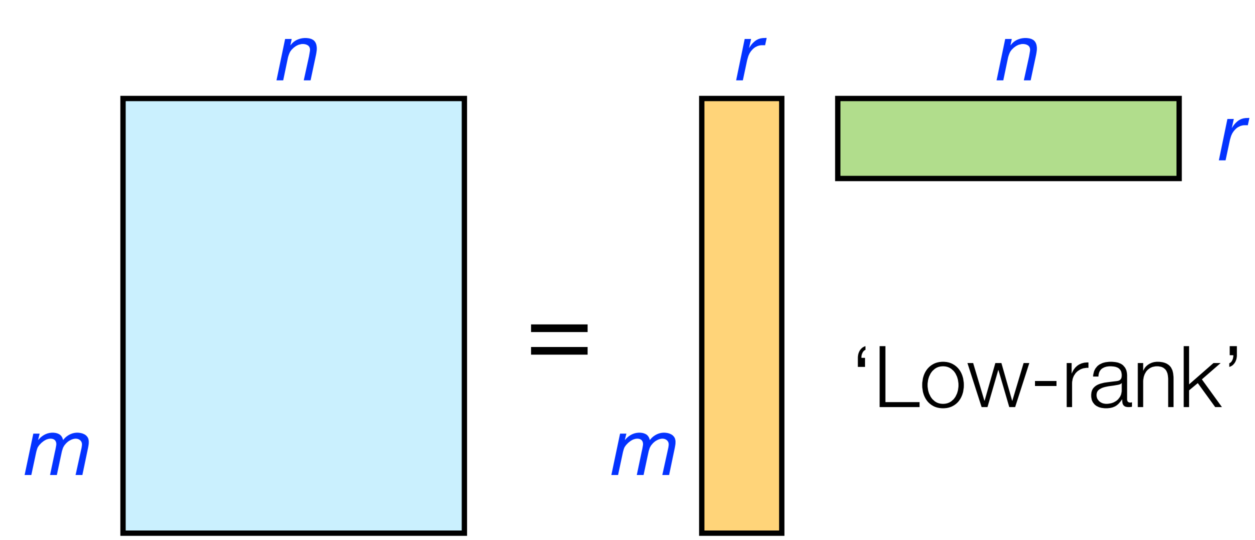
- Solution: Assume small # of factors determine preference



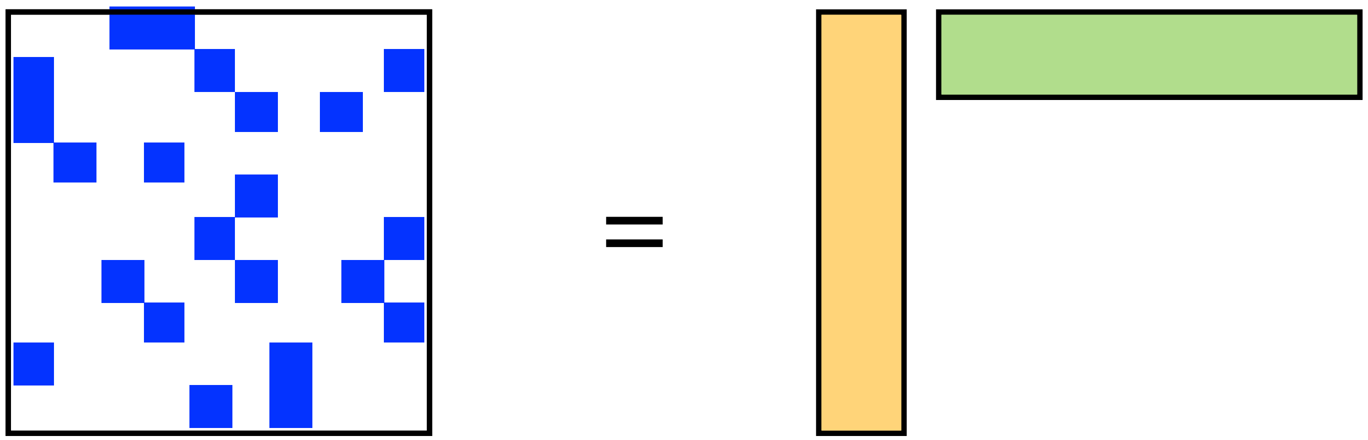
Reducing Degrees of Freedom



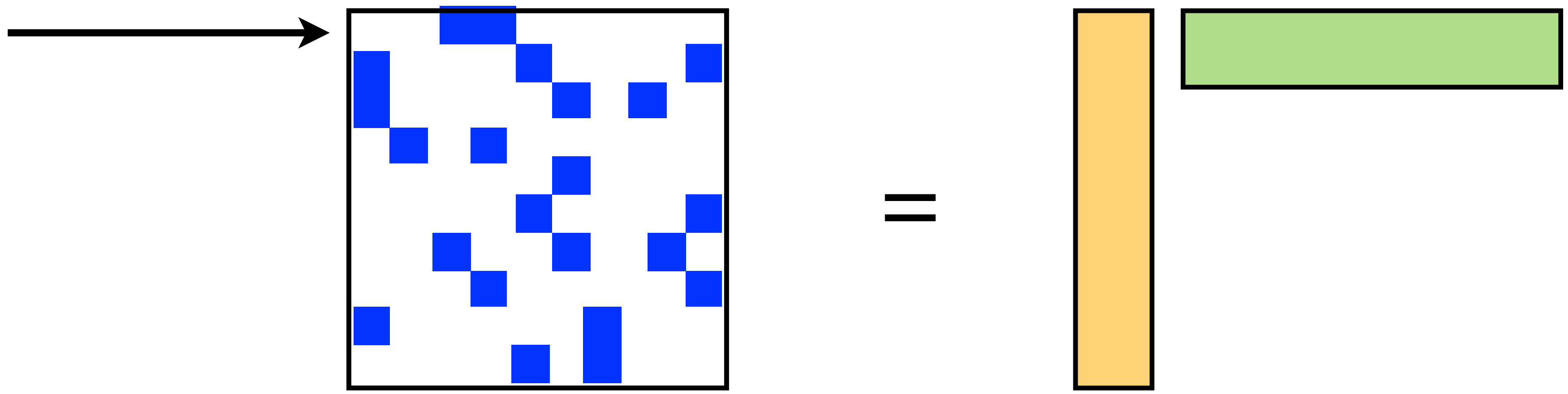
- Problem: Impossible without additional information
  - *mn* degrees of freedom
- Solution: Assume small # of factors determine preference
  - O(m + n) degrees of freedom
  - Linear storage costs



Alternating Least Squares

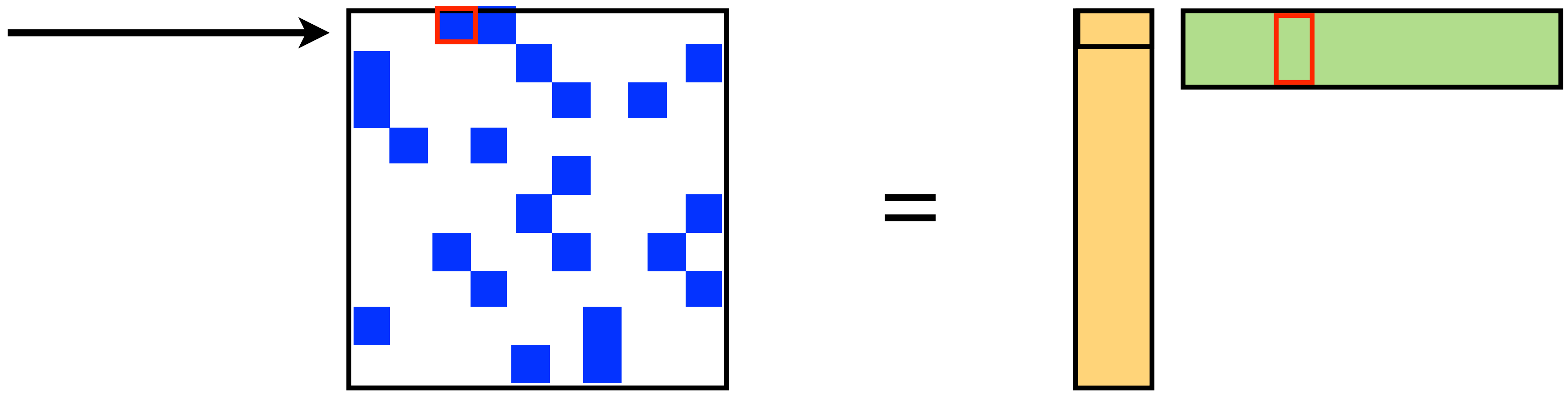


Alternating Least Squares

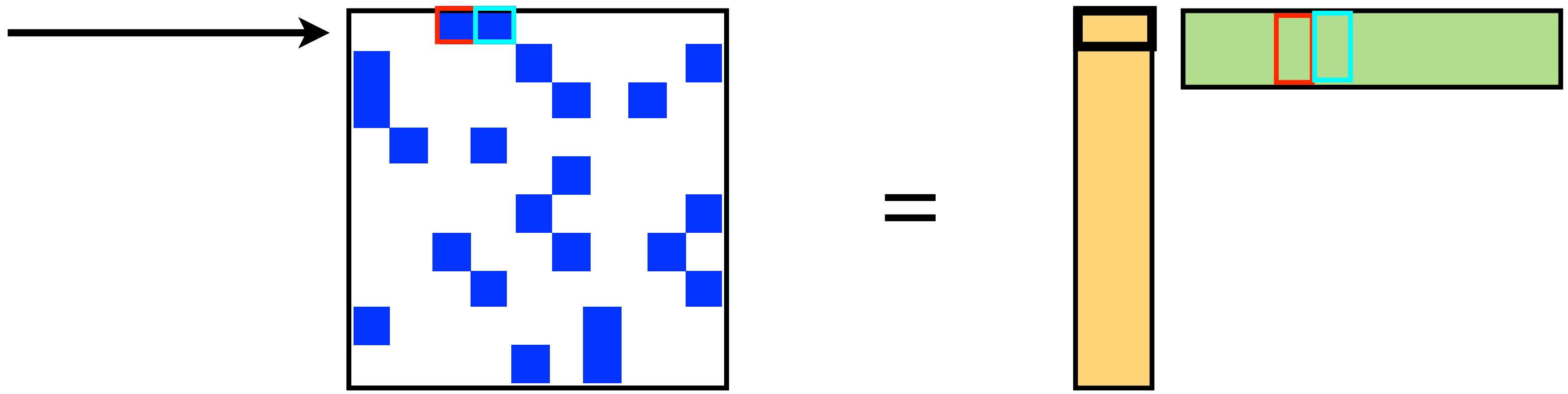


DATABRICKS

Alternating Least Squares

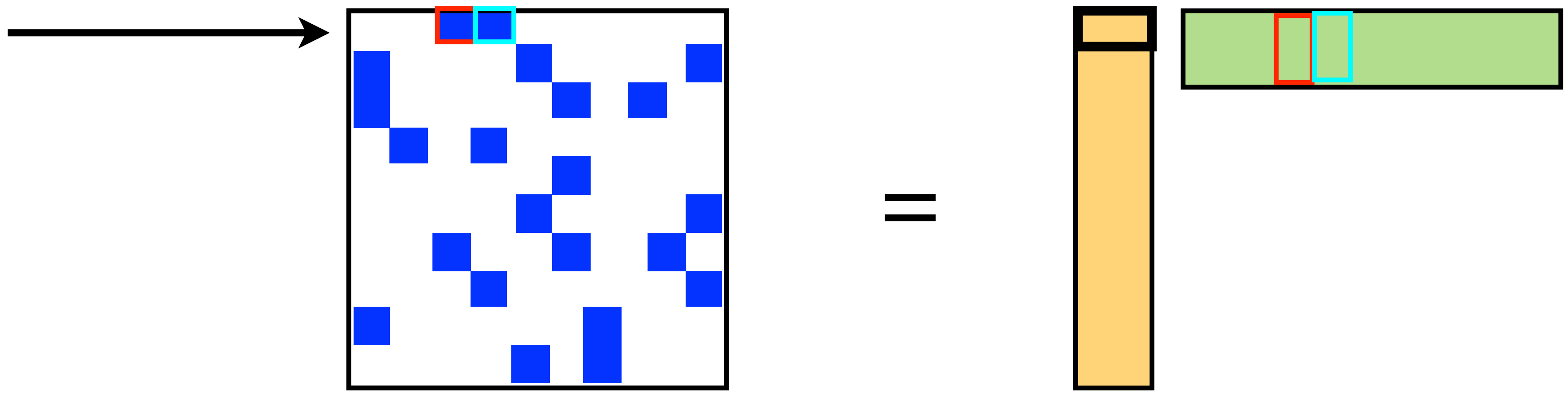


Alternating Least Squares



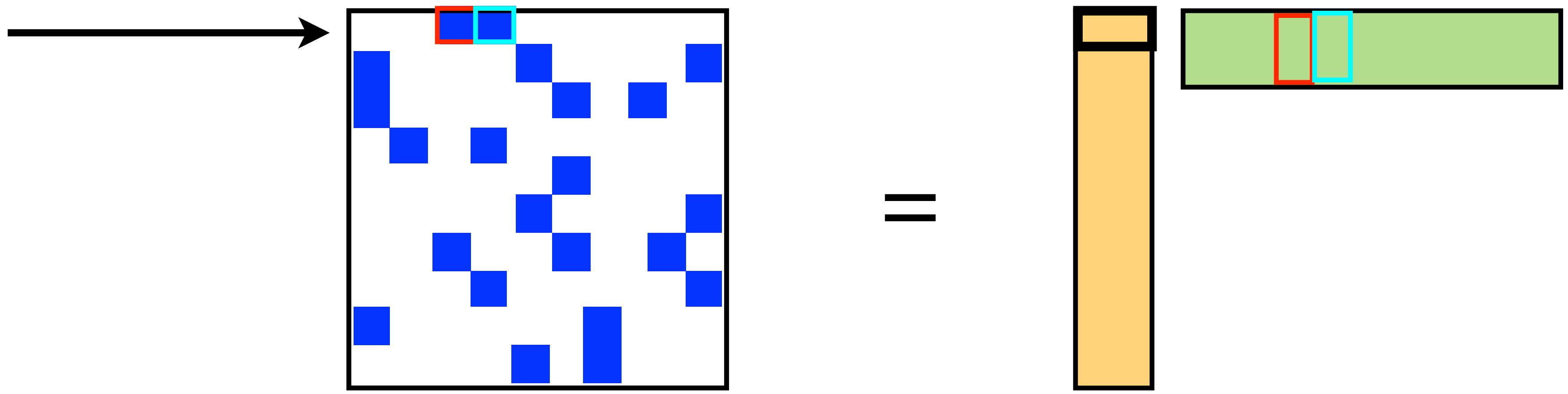
DATABRICKS

Alternating Least Squares



Training error for first user = (-) + (-)

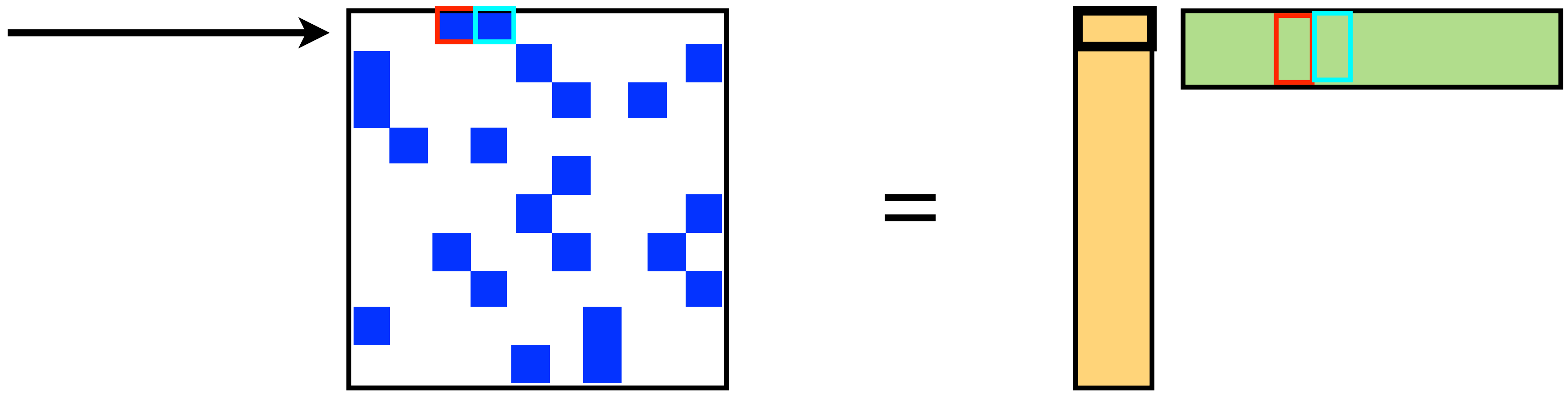
Alternating Least Squares



Training error for first user = (-) + (-)

ALS: alternate between updating user and movie factors

Alternating Least Squares

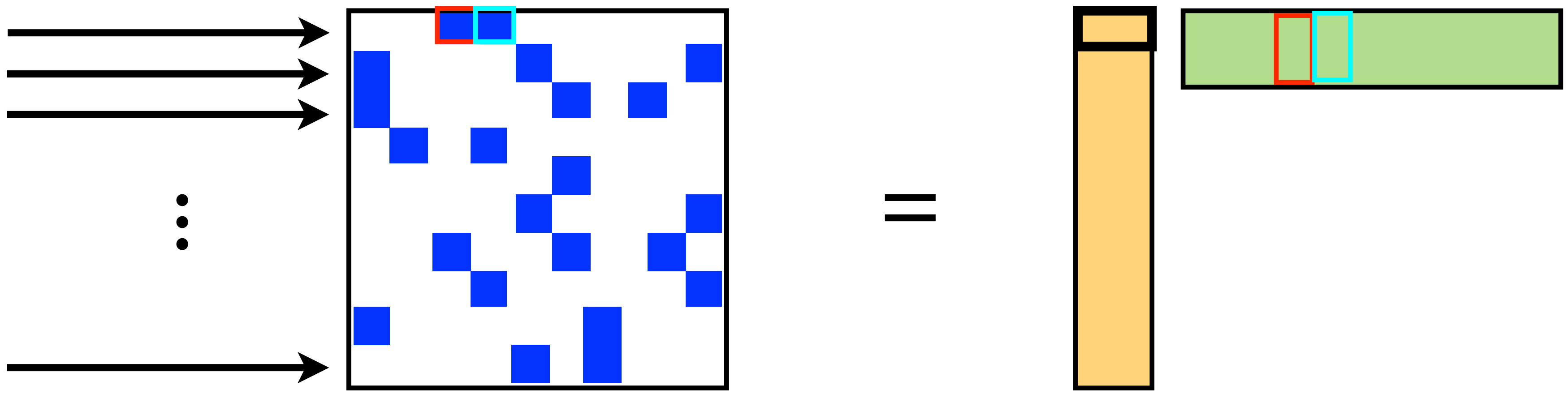


Training error for first user = (-) + (-)

ALS: alternate between updating user and movie factors

Update 1st user: find that minimizes training error
(reduces to standard linear regression problem)

Alternating Least Squares



Training error for first user = (-) + (-)

ALS: alternate between updating user and movie factors

Update 1st user: find that minimizes training error

(reduces to standard linear regression problem)

Can update all users in parallel!

Collaborative filtering

```
// Load and parse the data
val data = sc.textFile("mllib/data/als/test.data")
val ratings = data.map(_.split(',') match {
  case Array(user, item, rate) =>
    Rating(user.toInt, item.toInt, rate.toDouble)
})

// Build the recommendation model using ALS
val numIterations = 20
val rank = 10
val regularizer = 0.01
val model = ALS.train(ratings, rank, numIterations, regularizer)

// Evaluate the model on rating data
val usersProducts = ratings.map { case Rating(user, product, rate) =>
  (user, product)
}
val predictions = model.predict(usersProducts)
```

Today's Exercise

- Load 1M/10M ratings from MovieLens
- Specify YOUR ratings on examples
- Split examples into training/validation
- Fit a model (Python or Scala)
- Improve model via parameter tuning
- Get YOUR recommendations

Overview

Examples

Roadmap

Next release (v1.1)

Spark has 3-month release cycle

July 25: cut-off for new features

MLlib roadmap for v1.1

Standardize interfaces (MLbase/MLI)

Parallel model training for autotuning (MLbase/MLOpt)

Statistical toolbox

- descriptive statistics, sampling, hypothesis testing

Learning algorithms

- Non-negative matrix factorization, Sparse SVD, Multiclass decision tree, Random Forests?, …

Optimization algorithms

- ADMM, Accelerated gradient methods

Beyond v1.1?

Scalable implementations of standard ML algorithms and optimization primitives

User-friendly documentation and consistent APIs

Support for machine learning pipeline development

- Autotuning (MLbase/MLOpt), feature extractors, code examples

Beyond v1.1?

Scalable implementations of standard ML algorithms and optimization primitives

User-friendly documentation and consistent APIs

Support for machine learning pipeline development

- Autotuning (MLbase/MLOpt), feature extractors, code examples

Feedback and Contributions Encouraged!

http://spark.apache.org/docs/latest/mllib-guide.html

Contributors: Ameet Talwalkar, Andrew Or, Andrew Tulloch, Chen Chao, Cheng Lian, DB Tsai, Doris Xin, Evan Sparks, Frank Dai, Gang Bai, Ginger Smith, Guoqiang Li, Henry Saputra, Holden Karau, Hossein Falaki, Jerry Shao, Jey Kottalam, Marcelo Vanzin, Marek Kolodziej, Mark Hamstra, Martin Jaggi, Martin Weindel, Matei Zaharia, Nan Zhu, Neville Li, Nick Pentreath, Patrick Wendell, Piotr Szul, Prashant Sharma, Reynold Xin, Reza Zadeh, Ryan LeCompte, Sandeep Singh, Sandy Ryza, Sean Owen, Shaocun Tian, Shivaram Venkataraman, Shixiong Zhu, Shuo Bai, Shuo Xiang, Syed Hashmi, Takuya Ueshin, Tor Myklebust, Xiangrui Meng, Xinghao Pan, Xusen Yin

DATABRICKS

Thank You!