

ST446 Distributed Computing for Big Data

Lecture 8

Scalable machine learning Part II



Milan Vojnovic

<https://github.com/lse-st446/lectures2021>

Goals of this lecture

- Learn about distributed computing for machine learning tasks used for collaborative filtering (recommender systems) and topic modelling, in particular:

- Matrix completion (alternating squares algorithm)

Collaborative
filtering

- Latent Semantic Analysis (LSA) / Singular Value Decomposition (SVD)

Topic models

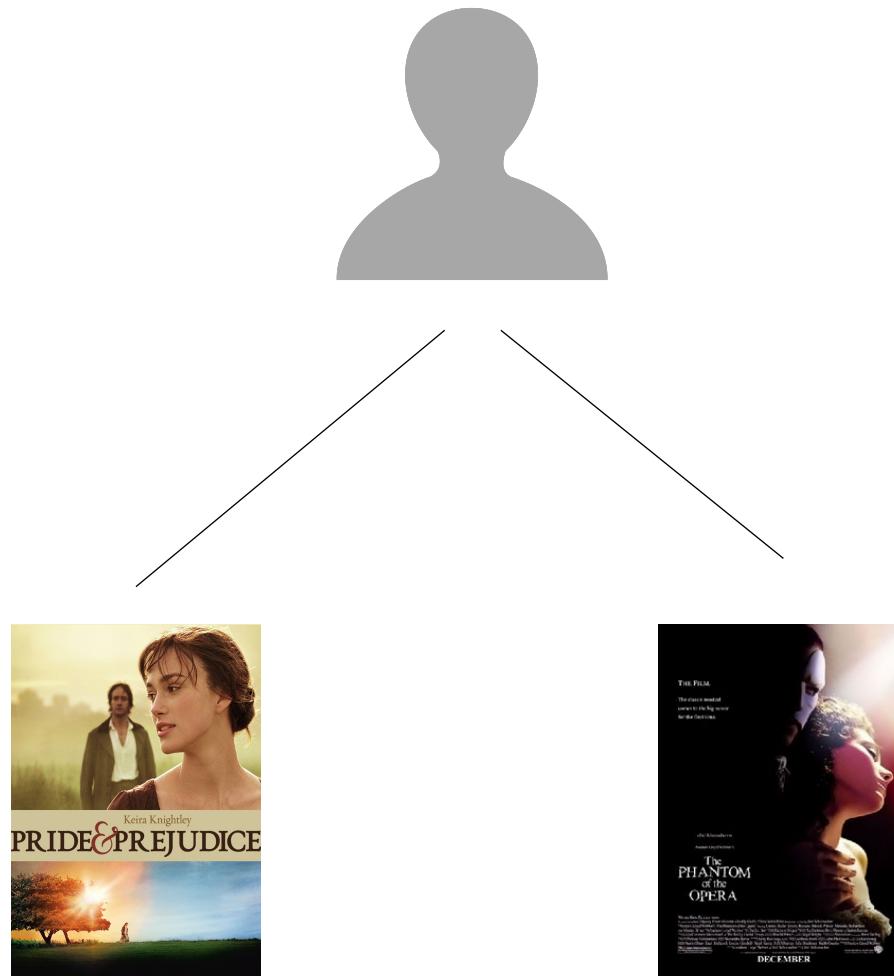
- Latent Dirichlet Allocation (LDA)

Topics of this lecture

- Matrix completion: introduction
- Matrix completion: alternating least squares algorithm
- Latent Semantic Analysis (LSA)
- Latent Dirichlet Allocation (LDA)

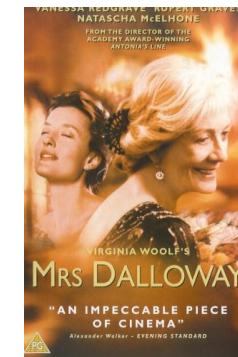
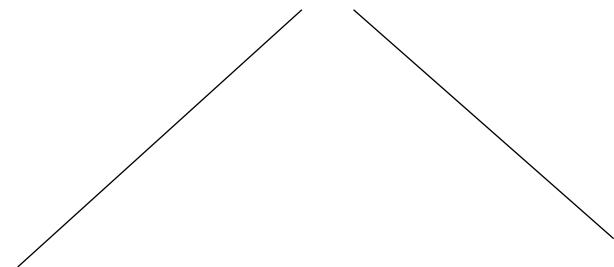
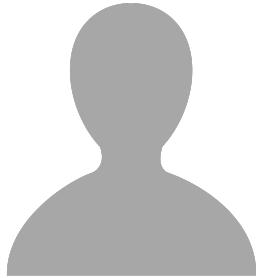
Matrix completion: introduction

Example: movie recommendations



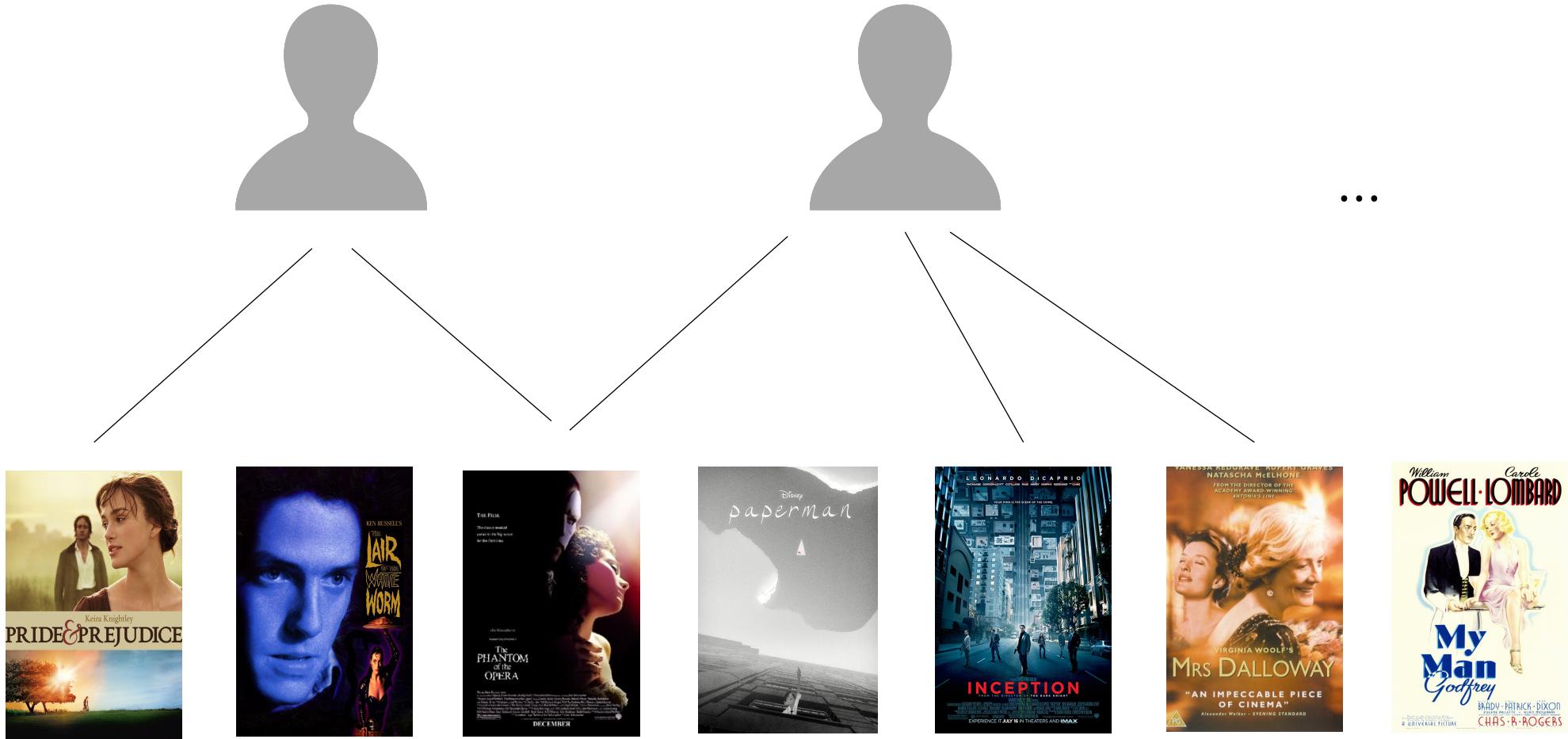
Example: movie recommendations (cont'd)

You may also like



Example: movie recommendations (cont'd)

Data



Example: movie recommendations (cont'd)

Data: matrix representation



Application scenarios

- Movie recommendations (ex Netflix, Amazon Prime, ...)
- Shopping product recommender systems (ex Amazon, ...)
- Travel (ex Booking.com,)
- Common to different application scenarios:
 - “users” on one side, “items” on other
 - Incomplete observations (reviews) of user preferences
 - Want to predict unseen preferences (reviews)
- General framework:
 - Framework of users, items and reviews used only for concreteness

Matrix completion problem

- Matrix completion problem: fill in missing entries of a matrix
 - Also referred as **matrix factorization** or **collaborative filtering**
- Input: $n \times m$ matrix \mathbf{M} with partially observed entries; entry $M_{i,j}$ observed only if $(i,j) \in \Omega$, for some set $\Omega \subseteq [n] \times [m]$
- Problem: fill in values of entries $M_{i,j}$ for $(i,j) \in [n] \times [m] \setminus \Omega$
- Standard approach: approximate \mathbf{M} by a low-rank matrix $\widehat{\mathbf{M}}$ (say of rank k)
$$\widehat{M}_{i,j} = \mathbf{u}_i^\top \mathbf{v}_j$$
where $\mathbf{u}_i \in \mathbf{R}^k$ is a *user parameter vector* and $\mathbf{v}_j \in \mathbf{R}^k$ is an *item parameter vector*
 - In matrix notation: $\widehat{\mathbf{M}} = \mathbf{U}^\top \mathbf{V}$ where $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_n]$ and $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_m]$

Loss function minimization

- Find \mathbf{U}, \mathbf{V} that minimize the squared error loss with L2 regularization

$$f(\mathbf{U}, \mathbf{V}) := \sum_{(i,j) \in \Omega} (\mathbf{M}_{i,j} - \mathbf{u}_i^\top \mathbf{v}_j)^2 + \lambda \left(\sum_{i=1}^n \|\mathbf{u}_i\|^2 + \sum_{j=1}^m \|\mathbf{v}_j\|^2 \right)$$

where $\lambda \geq 0$ is a regularization hyper-parameter

- Minimizing f is a *non-convex optimization problem* because of the quadratic terms $\mathbf{u}_i^\top \mathbf{v}_j$
 - Note however that f is convex in \mathbf{U} for every fixed \mathbf{V} and vice-versa (show this)
- Gradient descent algorithm can be slow: many iterations to converge to a (local) minima

Bayesian model

- Probabilistic generative model: assume \mathbf{M} is a random matrix with independent Gaussian entries and let \mathbf{M}_Ω be entries of $M_{i,j}$ for $(i,j) \in \Omega$

$$p(\mathbf{M}_\Omega | \mathbf{U}, \mathbf{V}) = \prod_{(i,j) \in \Omega} p(M_{i,j} | \mathbf{u}_i, \mathbf{v}_j)$$

and

$$M_{i,j} \sim N(\mathbf{u}_i^\top \mathbf{v}_j, 1/\alpha)$$

where α is a hyper-parameter

- By Bayes' rule, posterior distribution of \mathbf{U}, \mathbf{V} is given by $p(\mathbf{U}, \mathbf{V} | \mathbf{M}_\Omega) \propto p(\mathbf{M}_\Omega | \mathbf{U}, \mathbf{V}) p(\mathbf{U}, \mathbf{V})$ where $p(\mathbf{U}, \mathbf{V})$ is a given prior distribution
- Assume prior distribution is of product form $p(\mathbf{U}, \mathbf{V}) = \prod_{i=1}^n p(\mathbf{u}_i) \prod_{j=1}^m p(\mathbf{v}_j)$ with Gaussian marginals $\mathbf{u}_i \sim N(\mathbf{0}, \lambda \mathbf{I})$ and $\mathbf{v}_j \sim N(\mathbf{0}, \lambda \mathbf{I})$ where $\lambda > 0$ is a hyper-parameter

Bayesian model (cont'd)

- For the given Bayesian model, the *negative log-posterior distribution* is of the form

$$\begin{aligned} f(\mathbf{U}, \mathbf{V}) &:= -\log(p(\mathbf{U}, \mathbf{V} | \mathbf{M}_\Omega)) \\ &= \frac{1}{2} \alpha \sum_{(i,j) \in \Omega} (\mathbf{M}_{i,j} - \mathbf{u}_i^\top \mathbf{v}_j)^2 + \frac{1}{2} \lambda \left(\sum_{i=1}^n \|\mathbf{u}_i\|^2 + \sum_{j=1}^m \|\mathbf{v}_j\|^2 \right) \end{aligned}$$

- Hence, we obtain the *squared error loss function* with *L2 regularization*

Matrix completion: alternating least squares

Minimizing loss function: Alternating Least Squares

- Alternating least squares algorithm (ALS): alternation between two phases
 - Phase U: optimizing over \mathbf{U} for fixed \mathbf{V} , and
 - Phase V: optimizing over \mathbf{V} for fixed \mathbf{U}
- Gradient vector $\nabla_{\mathbf{U}} f(\mathbf{U}, \mathbf{V})$ entries are given by

$$\frac{\partial}{\partial u_{i,r}} f(\mathbf{U}, \mathbf{V}) = -2 \sum_{j:(i,j) \in \Omega} (\mathbf{M}_{i,j} - \mathbf{u}_i^\top \mathbf{v}_j) v_{j,r} + 2\lambda u_{i,r}$$

and similarly, for $\nabla_{\mathbf{V}} f(\mathbf{U}, \mathbf{V})$ we have

$$\frac{\partial}{\partial v_{j,r}} f(\mathbf{U}, \mathbf{V}) = -2 \sum_{i:(i,j) \in \Omega} (\mathbf{M}_{i,j} - \mathbf{u}_i^\top \mathbf{v}_j) u_{i,r} + 2\lambda v_{j,r}$$

First order optimality conditions

- Each phase corresponds to solving *a convex optimization problem*, hence the optimal solution of each such problem is a solution of *the first-order optimality conditions*
 - For Phase U: $\nabla_{\mathbf{U}} \mathbf{f}(\mathbf{U}, \mathbf{V}) = \mathbf{0}$
 - For Phase V: $\nabla_{\mathbf{V}} \mathbf{f}(\mathbf{U}, \mathbf{V}) = \mathbf{0}$
- Condition $\nabla_{\mathbf{U}} \mathbf{f}(\mathbf{U}, \mathbf{V}) = \mathbf{0}$ is equivalent to
$$(\lambda \mathbf{I} + \sum_{j:(i,j) \in \Omega} \mathbf{v}_j \mathbf{v}_j^\top) \mathbf{u}_i = \sum_{j:(i,j) \in \Omega} M_{i,j} \mathbf{v}_j \text{ for } i \in [n]$$
- Condition $\nabla_{\mathbf{V}} \mathbf{f}(\mathbf{U}, \mathbf{V}) = \mathbf{0}$ is equivalent to
$$(\lambda \mathbf{I} + \sum_{i:(i,j) \in \Omega} \mathbf{u}_i \mathbf{u}_i^\top) \mathbf{v}_j = \sum_{i:(i,j) \in \Omega} M_{i,j} \mathbf{u}_i \text{ for } j \in [m]$$

ALS pseudo-code

Initialization: U, V

Repeat until a termination criteria is met

For $i = 1$ to n

$$\mathbf{u}_i \leftarrow (\lambda I + \sum_{j:(i,j) \in \Omega} \mathbf{v}_j \mathbf{v}_j^\top)^{-1} (\sum_{j:(i,j) \in \Omega} M_{i,j} \mathbf{v}_j) \quad (\text{Phase U})$$

For $j = 1$ to m

$$\mathbf{v}_j \leftarrow (\lambda I + \sum_{i:(i,j) \in \Omega} \mathbf{u}_i \mathbf{u}_i^\top)^{-1} (\sum_{i:(i,j) \in \Omega} M_{i,j} \mathbf{u}_i) \quad (\text{Phase V})$$

- Computation costs:

- For each \mathbf{u}_i : $O(m_i k^2 + k^3)$ operations where $m_i = |\{j \in [m]: (i,j) \in \Omega\}|$
- For each \mathbf{v}_j : $O(n_j k^2 + k^3)$ operations where $n_j = |\{i \in [n]: (i,j) \in \Omega\}|$

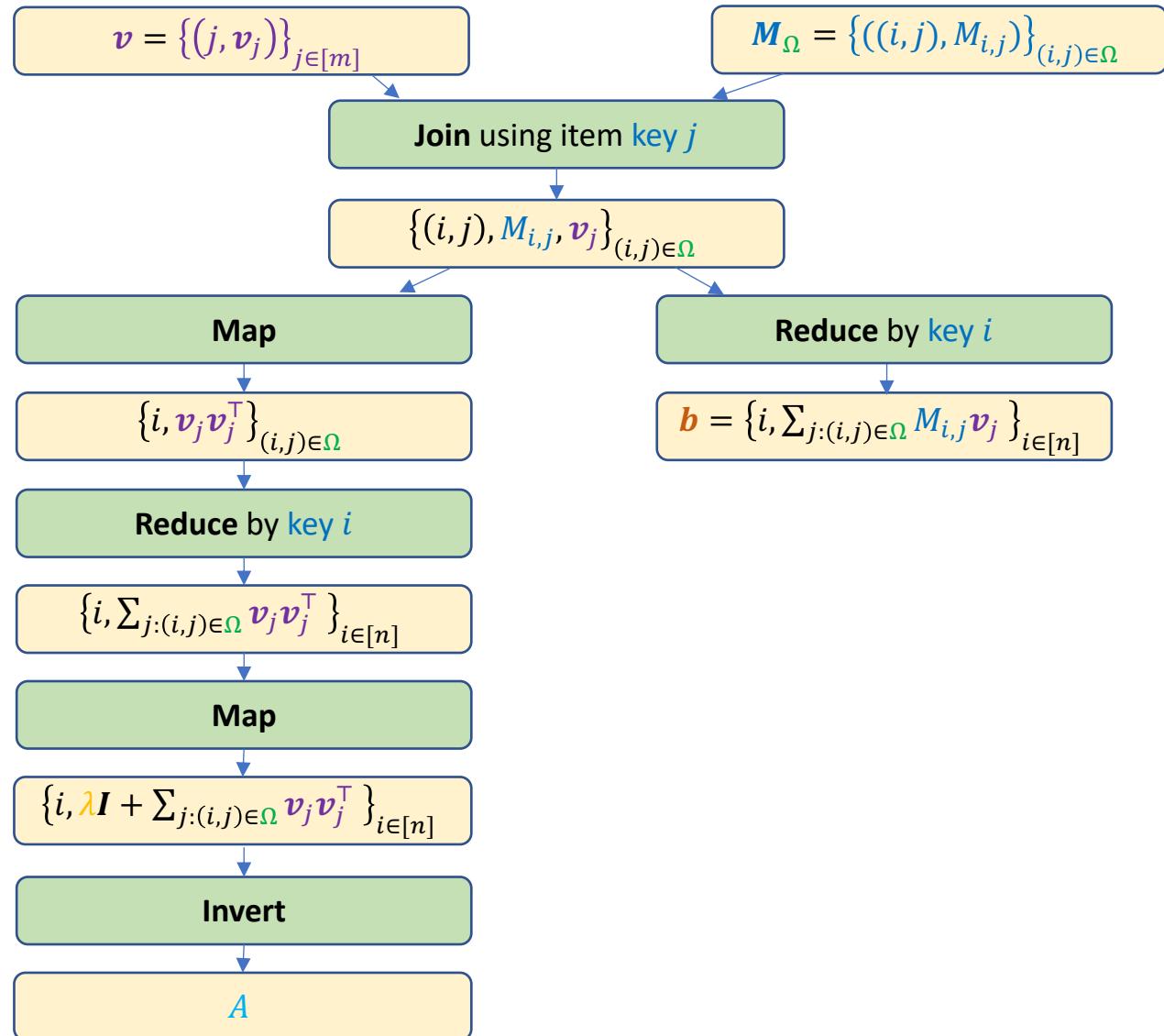
Distributed ALS using join

- Optimization over \mathbf{U} for fixed \mathbf{V} :

$$\mathbf{u}_i \leftarrow \mathbf{A}_i \mathbf{b}_i$$

$$\mathbf{A}_i = (\lambda \mathbf{I} + \sum_{j:(i,j) \in \Omega} \mathbf{v}_j \mathbf{v}_j^\top)^{-1}$$

$$\mathbf{b}_i = \sum_{j:(i,j) \in \Omega} M_{i,j} \mathbf{v}_j$$



Distributed ALS using broadcast

Partition \mathbf{M}_Ω by *user key* to create *user-key partition* $\mathbf{M}_\Omega^u = (\mathbf{M}_\Omega^{u,1}, \mathbf{M}_\Omega^{u,2}, \dots, \mathbf{M}_\Omega^{u,D})$

Partition \mathbf{M}_Ω by *item key* to create *item-key partition* $\mathbf{M}_\Omega^v = (\mathbf{M}_\Omega^{v,1}, \mathbf{M}_\Omega^{v,2}, \dots, \mathbf{M}_\Omega^{v,D})$

Assign user-key and item-key partition components to machines

// For each user $i \in [n]$, all entries $M_{i,j}$ for $(i,j) \in \Omega$ reside in one user-key partition

// For each item $j \in [m]$, all entries $M_{i,j}$ for $(i,j) \in \Omega$ reside in one item-key partition

Initialize \mathbf{U} and \mathbf{V} and broadcast \mathbf{V} to all machines

Repeat until a termination criteria is met:

 Compute \mathbf{u}_i 's locally on machines using user-key partition \mathbf{M}_Ω^u and \mathbf{V}

 Broadcast \mathbf{U} to all machines

 Compute \mathbf{v}_j 's locally on machines using item-key partition \mathbf{M}_Ω^v and \mathbf{U}

 Broadcast \mathbf{V} to all machines

Some recent theoretical results

- If \mathbf{M} is a symmetric positive semidefinite matrix, then

$$f(\mathbf{U}) = \sum_{(i,j) \in \Omega} (\mathbf{M}_{i,j} - \mathbf{u}_i^\top \mathbf{u}_j)^2$$

has no spurious local minima

- No spurious local minima: all local minima are globally optimal
- Stochastic gradient descent can provably solve matrix completion with arbitrary initialization in polynomial time
- To probe further:
 - Ge et al, [Matrix completion has no spurious local minimum](#), NIPS 2016
 - Bhojanapalli et al, [Dropping convexity for faster semi-definite optimization](#), JMLR 2016

Spark Python API

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Load and parse the data
data = sc.textFile("data/mllib/als/test.data")
ratings = data.map(lambda l: l.split(',')\
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

# Build the recommendation model using Alternating Least Squares
rank = 10
numIterations = 10
model = ALS.train(ratings, rank, numIterations)

# Evaluate the model on training data
testdata = ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error = " + str(MSE))

# Save and load model
model.save(sc, "target/tmp/myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc, "target/tmp/myCollaborativeFilter")
```

- Source: <https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>

Spark implementation

```
@DeveloperApi
def train[ID: ClassTag]( // scalastyle:ignore
    ratings: RDD[Rating[ID]],
    rank: Int = 10,
    numUserBlocks: Int = 10,
    numItemBlocks: Int = 10,
    maxIter: Int = 10,
    regParam: Double = 0.1,
    implicitPrefs: Boolean = false,
    alpha: Double = 1.0,
    nonnegative: Boolean = false,
    intermediateRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
    finalRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
    checkpointInterval: Int = 10,
    seed: Long = 0L)(
    implicit ord: Ordering[ID]): (RDD[(ID, Array[Float])], RDD[(ID, Array[Float])]) =
{
    ...
}
```

- Source: [ALS.scala](#)

Spark implementation (cont'd)

```
for (iter <- 0 until maxIter) {
    itemFactors = computeFactors(userFactors, userOutBlocks, itemInBlocks, rank, regParam,
        userLocalIndexEncoder, solver = solver)
    if (shouldCheckpoint(iter)) {
        val deps = itemFactors.dependencies
        itemFactors.checkpoint()
        itemFactors.count() // checkpoint item factors and cut lineage
        ALS.cleanShuffleDependencies(sc, deps)
        deletePreviousCheckpointFile()
        previousCheckpointFile = itemFactors.getCheckpointFile
    }
    userFactors = computeFactors(itemFactors, itemOutBlocks, userInBlocks, rank, regParam,
        itemLocalIndexEncoder, solver = solver)
}
```

Spark implementation (cont'd)

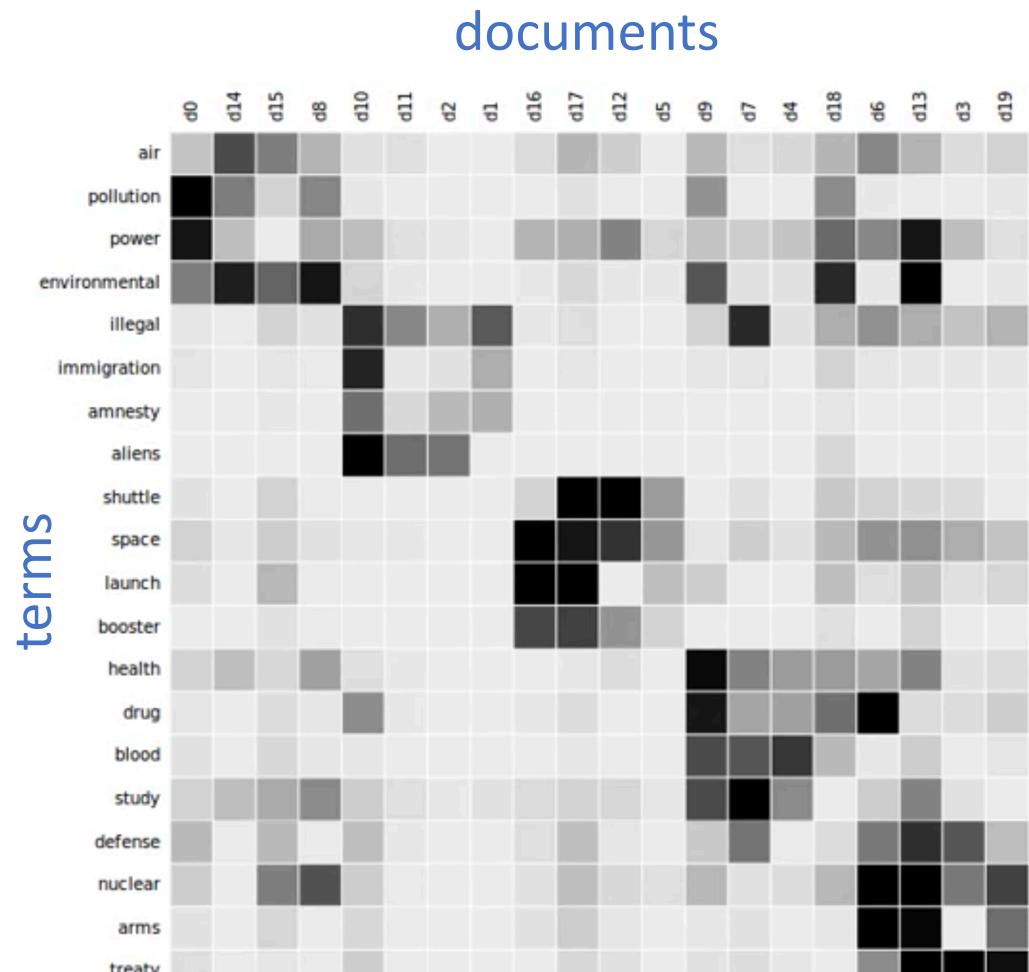
```
/** Compute dst factors by constructing and solving least square problems.
 * @param srcFactorBlocks src factors
 * @param srcOutBlocks src out-blocks
 * @param dstInBlocks dst in-blocks
 * @param rank rank
 * @param regParam regularization constant
 * @param srcEncoder encoder for src local indices
 * @param implicitPrefs whether to use implicit preference
 * @param alpha the alpha constant in the implicit preference formulation
 * @param solver solver for least squares problems
 * @return dst factors */

private def computeFactors[ID](
    srcFactorBlocks: RDD[(Int, FactorBlock)],
    srcOutBlocks: RDD[(Int, OutBlock)],
    dstInBlocks: RDD[(Int, InBlock[ID])],
    rank: Int,
    regParam: Double,
    srcEncoder: LocalIndexEncoder,
    implicitPrefs: Boolean = false,
    alpha: Double = 1.0,
    solver: LeastSquaresNESolver): RDD[(Int, FactorBlock)]
```

Latent Semantic Analysis (LSA) - Singular Value Decomposition (SVD)

Latent Semantic Analysis

- Latent Semantic Analysis (LSA): a method for analyzing relationships between a set of terms and documents
 - Referred also as Latent Semantic Indexing
 - Introduced by Deerwester et al in 1990
- Input: $m \times n$ term-document matrix \mathbf{M} where $M_{i,j}$ is the number of occurrences of term i in document j
- Goal: find numerical vector representations of terms and documents
- Solution method: matrix factorization by using singular value decomposition



Source: [wikipedia](#)

Singular value decomposition

- Singular value decomposition of a real $m \times n$ matrix \mathbf{M} of rank k is given by

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$$

where

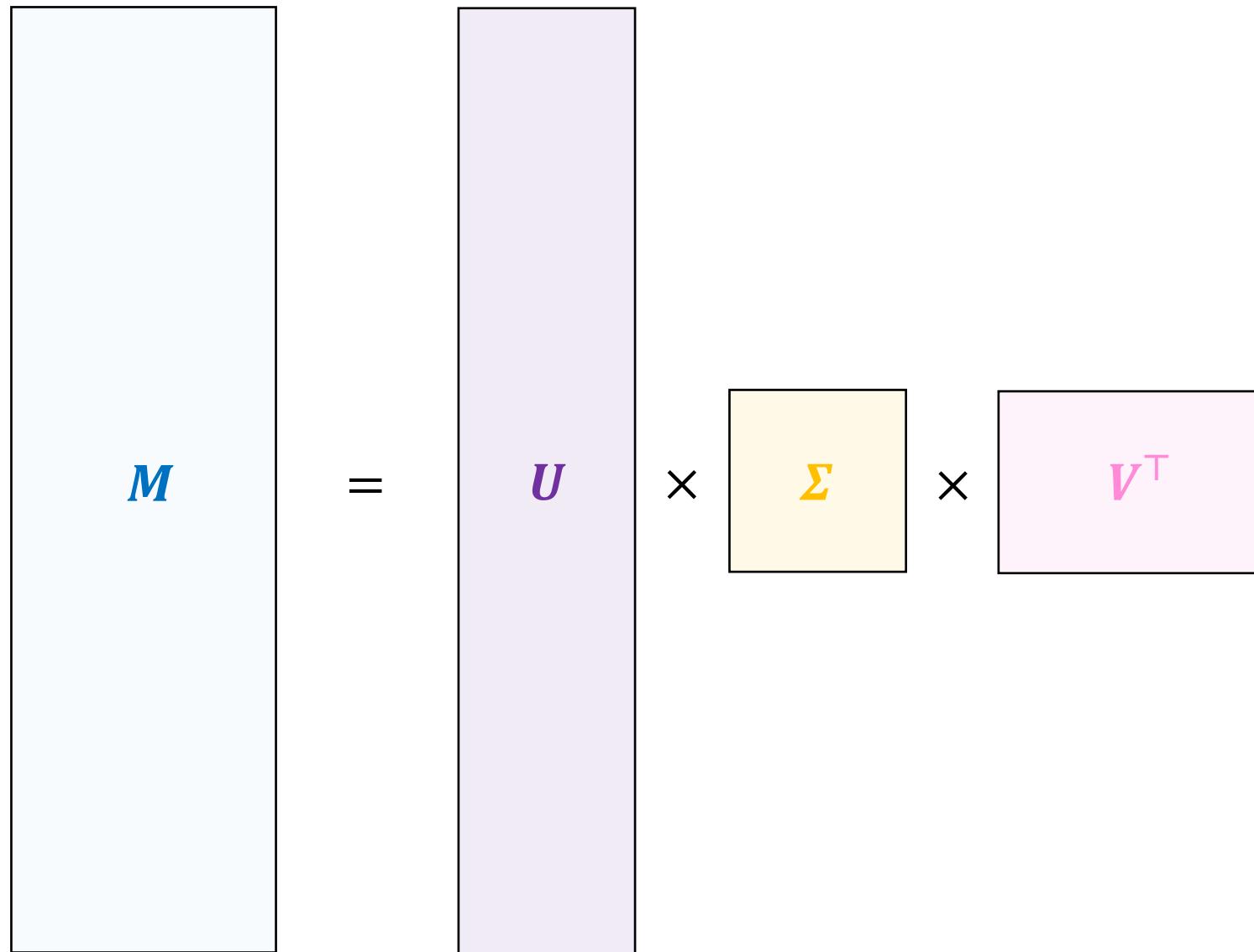
\mathbf{U} is a $m \times k$ unitary matrix

Σ is a $k \times k$ diagonal matrix with real-valued non-negative diagonal elements

\mathbf{V} is a $n \times k$ unitary matrix

- Two cases:
 - \mathbf{M} is tall and skinny: $m \gg n$ (we will only consider this case)
 - \mathbf{M} is (approximately) a square matrix

Tall and skinny case

$$\mathbf{M} = \mathbf{U} \times \Sigma \times \mathbf{V}^\top$$


Distributed computing: tall and skinny case

- Key idea: separation in two steps
 - Step 1: computation of Σ and V
 - Step 2: computation of U
- Step 1: compute Σ and V
 - Compute $M^T M$ using all-to-one communication to driver node
 - $M^T M$ is a $n \times n$ matrix that can fit in memory of a single node
 - Compute Σ and V by eigen-decomposition $M^T M = V \Sigma^2 V^T$ locally on the driver node
 - $M^T M$ is a positive semidefinite matrix with eigenvalues equal to squares of singular values of M

Distributed computing: tall and skinny case (cont'd)

- Step 2: compute \mathbf{U}
 - Use the fact $\mathbf{U} = \mathbf{M}\mathbf{V}\Sigma^{-1}$
 - Broadcast $\mathbf{V}\Sigma^{-1}$ to all nodes that hold rows of \mathbf{M}
 - For each $i \in [m]$, compute $\mathbf{u}_i = \mathbf{m}_i\mathbf{V}\Sigma^{-1}$ where \mathbf{u}_i and \mathbf{m}_i are rows of \mathbf{U} and \mathbf{M}
- Remarks:
 - $\mathbf{U} = \mathbf{M}\mathbf{V}\Sigma^{-1}$ follows from SVD $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$ as
$$\mathbf{M}\mathbf{V}\Sigma^{-1} = \mathbf{U}\Sigma\mathbf{V}^\top\mathbf{V}\Sigma^{-1} = \mathbf{U}\Sigma\Sigma^{-1} = \mathbf{U}$$
 - $\mathbf{V}\Sigma^{-1}$ is a $n \times k$ matrix that can fit into memory of a single machine
 - $\mathbf{u}_i = \mathbf{m}_i\mathbf{V}\Sigma^{-1}$ can be computed locally on machine holding row i

Spark SVD API

```
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.linalg.distributed import RowMatrix

rows = sc.parallelize([
    Vectors.sparse(5, {1: 1.0, 3: 7.0}),
    Vectors.dense(2.0, 0.0, 3.0, 4.0, 5.0),
    Vectors.dense(4.0, 0.0, 0.0, 6.0, 7.0)
])

mat = RowMatrix(rows)

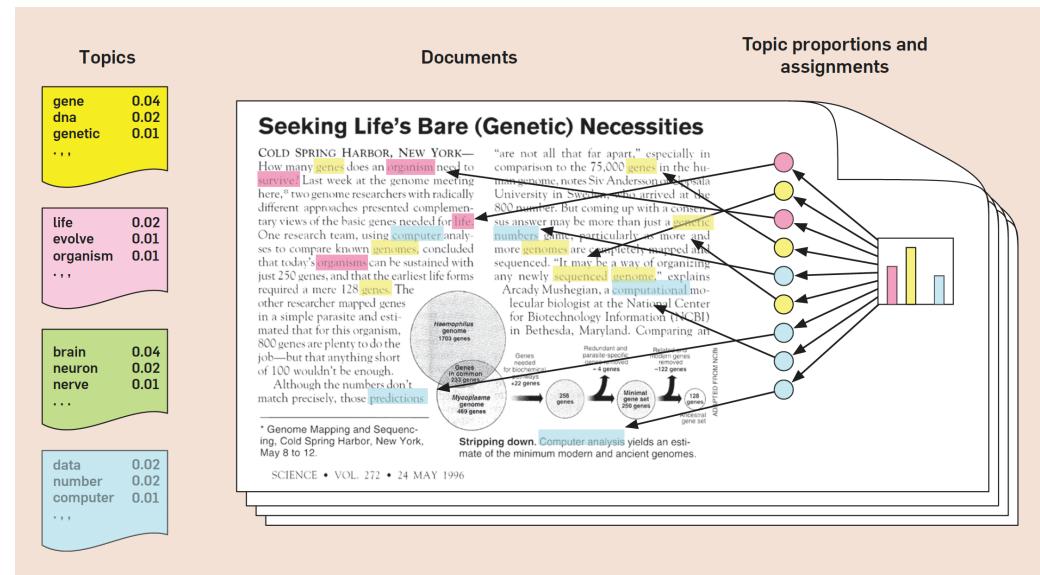
# Compute the top 5 singular values and corresponding singular vectors.
svd = mat.computeSVD(5, computeU=True)
U = svd.U      # The U factor is a RowMatrix.
s = svd.s      # The singular values are stored in a local dense vector.
V = svd.V      # The V factor is a local dense matrix.
```

- Example: <https://spark.apache.org/docs/2.2.0/mllib-dimensionality-reduction.html>
- Full example code: https://github.com/apache/spark/blob/master/examples/src/main/python/mllib/svd_example.py

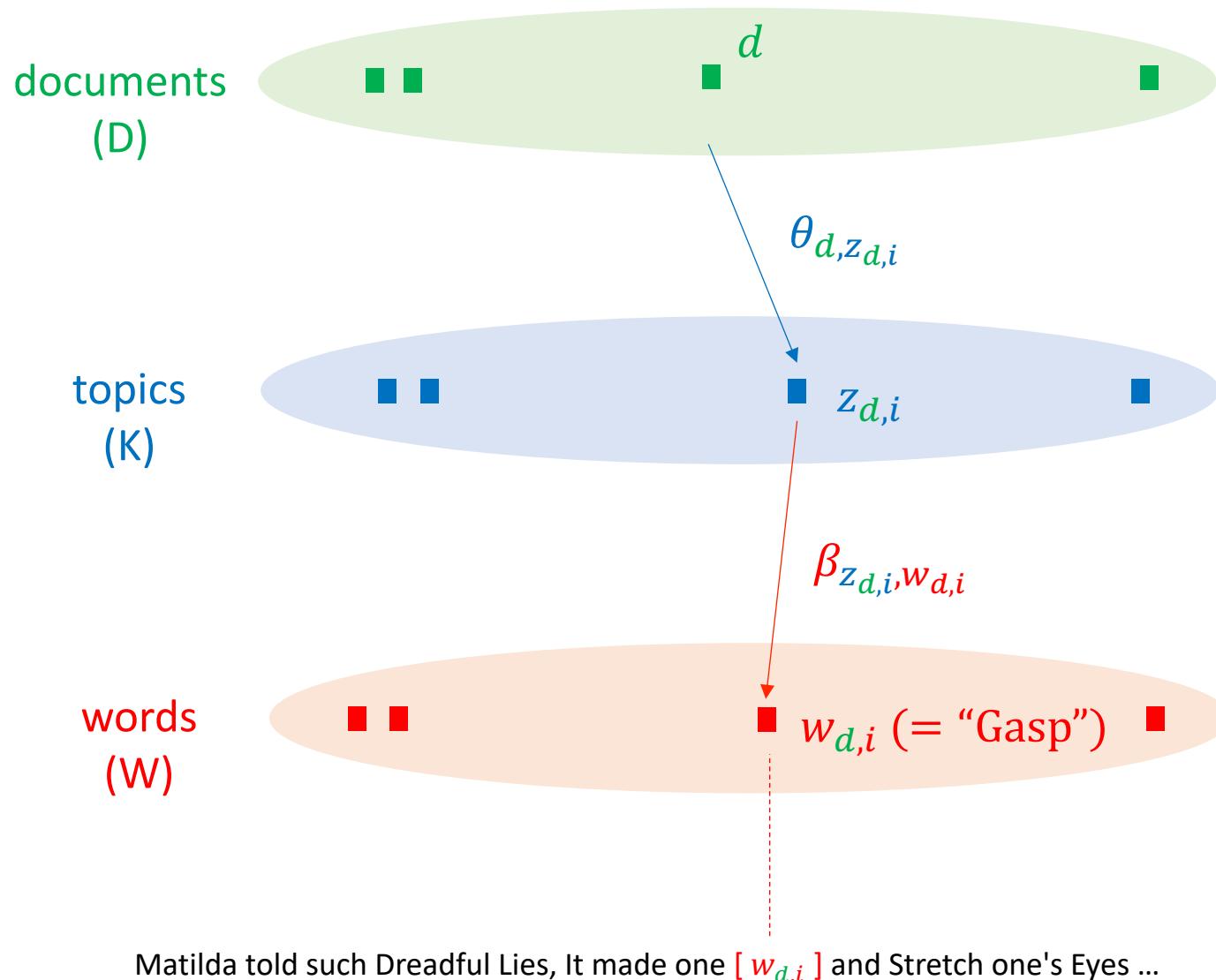
Latent Dirichlet Allocation

Probabilistic topic models

- Latent Dirichlet Allocation (LDA): a generative statistical model that allows sets of observations to be explained by latent groups that explain why some parts of the data are similar
 - Proposed by Blei et al (2003)
- For observations that consist of words collected in a corpus of documents:
 - Each **document** is associated with a distribution over topics
 - Each **topic** is associated with a multinomial distribution over words
 - Each **word** is drawn from a topic distribution



LDA probabilistic generative model



θ_d := distribution over topics for document d

$z_{d,i} \sim \theta_d$ (latent random variable)

$\beta_{z_{d,i}, w_{d,i}}$:= distribution over words for topic $z_{d,i}$

$w_{d,i} \sim \beta_{z_{d,i}}$ (observed random variable)

Matrix factorization interpretation:

$$P[w_{d,i} = w | \theta_d, \beta] = \sum_z \theta_{d,z} \beta_{z,w}$$

Bayesian models and inference

- Suppose we have a probabilistic generative model of a random variable \mathbf{Y} defined by

$$\mathbf{P}[\mathbf{Y} = \mathbf{y}] = f_{\boldsymbol{\theta}}(\mathbf{y})$$

for some given function $\mathbf{y} \mapsto f_{\boldsymbol{\theta}}(\mathbf{y})$ where $\boldsymbol{\theta}$ is a parameter vector

- Parameter vector $\boldsymbol{\theta}$ interpreted as a random variable with given prior distribution $p_0(\mathbf{x})$
- Bayesian inference: posterior distribution of $\boldsymbol{\theta}$ conditional on observed data $\mathbf{Y} = \mathbf{y}$
- By Bayes' rule: posterior distribution $p_1(\mathbf{x})$ of $\boldsymbol{\theta}$ is given by

$$p_1(\mathbf{x}) := \mathbf{P}[\boldsymbol{\theta} = \mathbf{x} \mid \mathbf{Y} = \mathbf{y}] = \frac{f_{\mathbf{x}}(\mathbf{y})p_0(\mathbf{x})}{\sum_{\mathbf{x}'} f_{\mathbf{x}'}(\mathbf{y})p_0(\mathbf{x}')}$$

Bayesian inference for LDA

- Prior distributions:

- $\theta_d \sim \text{Dir}(\alpha)$
- $\beta_z \sim \text{Dir}(\eta)$

where $\text{Dir}(\alpha)$ is the *Dirichlet distribution* with parameter α

- Dirichlet distribution of order $r \geq 2$ with parameter $\alpha = (\alpha_1, \dots, \alpha_r)^\top \in \mathbf{R}_+^r$ is a distribution over the simplex $S_r = \{x \in \mathbf{R}_+^r : \sum_{i=1}^r x_i = 1\}$ with the density function

$$p(x_1, \dots, x_r; \alpha_1, \dots, \alpha_r) = \frac{1}{B(\alpha_1, \dots, \alpha_r)} x_1^{\alpha_1-1} \cdots x_r^{\alpha_r-1}$$

where $B(\alpha_1, \dots, \alpha_r)$ is the normalizing constant (multivariate Beta function)

- Remarks:

- α is referred to as vector of *concentration parameters*
- For the special case $r = 2$, the Dirichlet distribution is Beta distribution

Dirichlet and multinomial distributions

- Multinomial distribution with parameters n, p_1, \dots, p_r has the density function

$$p(x_1, \dots, x_r; n, p_1, \dots, p_r) = \frac{n!}{x_1! \cdots x_r!} p_1^{x_1} \cdots p_r^{x_r}$$

for x_1, \dots, x_r non-negative integers such that $x_1 + \cdots + x_r = n$

- Dirichlet distribution is the **conjugate prior** of multinomial distribution:

If (x_1, \dots, x_r) has multinomial distribution with parameters n, p_1, \dots, p_r

and (p_1, \dots, p_r) has $\text{Dir}(a_1, \dots, a_r)$ prior distribution

then, having observed (x_1, \dots, x_r) , posterior distribution of (p_1, \dots, p_r) is

$\text{Dir}(x_1 + a_1, \dots, x_r + a_r)$ (same family distribution as prior)

Posterior distribution for LDA

- Joint distribution of $(\mathbf{w}, \boldsymbol{\beta}, \mathbf{z}, \boldsymbol{\theta})$:

$$p(\mathbf{w}, \boldsymbol{\beta}, \mathbf{z}, \boldsymbol{\theta} \mid \boldsymbol{\alpha}, \boldsymbol{\eta}) = \prod_{\mathbf{z}} p(\boldsymbol{\beta}_{\mathbf{z}} \mid \boldsymbol{\eta}) \prod_{\mathbf{d}} p(\boldsymbol{\theta}_{\mathbf{d}} \mid \boldsymbol{\alpha}) \prod_i p(\mathbf{z}_{\mathbf{d},i} \mid \boldsymbol{\theta}_{\mathbf{d}}) p(\mathbf{w}_{\mathbf{d},i} \mid \boldsymbol{\beta}, \mathbf{z}_{\mathbf{d},i})$$

- Computing posterior distribution of $\boldsymbol{\alpha}$ and $\boldsymbol{\eta}$ requires marginalizing out the joint distribution of $(\mathbf{w}, \boldsymbol{\beta}, \mathbf{z}, \boldsymbol{\theta})$ over $(\boldsymbol{\beta}, \mathbf{z}, \boldsymbol{\theta})$ which is computationally too expensive
- A way around this computational difficulty is using variational Bayes inference
- Variational Bayes inference*: posterior approximated by a simpler distribution

Variational Bayes inference

- For any distribution q of (β, z, θ) , the following lower bound holds:

$$\log(p(\mathbf{w} | \alpha, \eta)) \geq \mathbb{E}_{(\beta, z, \theta) \sim q} \left[\log \left(\frac{p(\mathbf{w}, \beta, z, \theta | \alpha, \eta)}{q(\beta, z, \theta)} \right) \right] \quad (\text{ELBO})$$

- Proof of ELBO is given in the next slide (ELBO: Evidence Lower Bound)
- Maximizing ELBO is equivalent to minimizing the Kullback-Leibler divergence between $q(\beta, z, \theta)$ and $p(\mathbf{w}, \beta, z, \theta | \alpha, \eta)$
- *Kullback-Liebler divergence*: a commonly used measure of “distance” between two probability distributions
 - To probe further: see [wikipedia](#)

Proof of ELBO

- Suppose $p(\mathbf{x}, \mathbf{y})$ is a distribution on $X \times Y$ and $q(\mathbf{y})$ is a distribution on Y
- Then, for every $\mathbf{x} \in X$ and $\mathbf{y} \in Y$, we have

$$\begin{aligned}\log(p(\mathbf{x})) &= \log\left(\sum_{\mathbf{y} \in Y} p(\mathbf{x}, \mathbf{y})\right) \\ &= \log\left(\sum_{\mathbf{y} \in Y} \frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{y})} q(\mathbf{y})\right) \\ &\geq \sum_{\mathbf{y} \in Y} \log\left(\frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{y})}\right) q(\mathbf{y}) \quad (\text{Jensen's inequality}) \\ &= \mathbf{E}_{\mathbf{Y} \sim q} \left[\log\left(\frac{p(\mathbf{x}, \mathbf{Y})}{q(\mathbf{Y})}\right) \right]\end{aligned}$$

Variational Bayes inference for LDA (cont'd)

- Assume q to be a product-form distribution with marginal distributions:

$$q(z_{d,i} = z) = \phi_{d,w_{d,i},z}, q(\theta_d) = \text{Dir}(\theta_d; \gamma_d) \text{ and } q(\beta_z) = \text{Dir}(\beta_z; \lambda_z)$$

where ϕ, γ, λ are variational parameters

- Then, $L(w, \phi, \alpha, \eta) := E_{(\beta, z, \theta) \sim q} \left[\log \left(\frac{p(w, \beta, z, \theta | \alpha, \eta)}{q(\beta, z, \theta)} \right) \right]$ is given by

$$L(w, \phi, \alpha, \eta) = \sum_{d=1}^D \left\{ \begin{array}{l} E_q[\log(p(w_d | \theta_d, z_d, \beta))] \\ + E_q[\log(p(z_d | \theta_d))] \\ - E_q[\log(q(z_d))] \\ + E_q[\log(p(\theta_d | \alpha))] \\ - E_q[\log(q(\theta_d))] \\ + \frac{1}{D} (E_q[p(\beta | \eta)] - E_q[\log(q(\beta))]) \end{array} \right\}$$

Variational Bayes inference (cont'd)

- Assume symmetric prior: $\alpha = (\alpha, \alpha, \dots, \alpha)$ and $\eta = (\eta, \eta, \dots, \eta)$
- Expanding $L(\mathbf{w}, \boldsymbol{\phi}, \boldsymbol{\alpha}, \boldsymbol{\eta})$ using variational parameters $\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}$, we have

$$L(\mathbf{n}, \boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) = \sum_{d=1}^D l(\mathbf{n}_d, \boldsymbol{\phi}_d, \boldsymbol{\gamma}_d, \boldsymbol{\lambda})$$

where $n_{d,w}$ is the number of occurrences of word w in document d and

$$\begin{aligned} l(\mathbf{n}_d, \boldsymbol{\phi}_d, \boldsymbol{\gamma}_d, \boldsymbol{\lambda}) &= \sum_w n_{d,w} \left(\sum_z \phi_{d,z,w} (\mathbf{E}_q[\log(\theta_{d,z})] + \mathbf{E}_q[\log(\beta_{z,w})] - \log(\phi_{d,w,z})) \right. \\ &\quad \left. - \log(\Gamma(\sum_z \gamma_{d,z})) \right) \\ &\quad + \sum_z \left((\alpha - \gamma_{d,z}) \mathbf{E}_q[\log(\theta_{d,z})] + \log(\Gamma(\gamma_{d,z})) \right) \\ &\quad + \frac{1}{D} \left(\sum_z (-\log(\Gamma(\sum_w \lambda_{z,w}))) \right) + \sum_w (\eta - \lambda_{z,w}) \mathbf{E}_q[\log(\beta_{z,w})] + \log(\Gamma(\lambda_{z,w})) \\ &\quad + \log(K\alpha) - K \log(\Gamma(\alpha)) \\ &\quad + \frac{1}{D} (\log(W\eta) - W \log(\Gamma(\eta))) \end{aligned}$$

Variational Bayes inference (cont'd)

- Setting gradients with respect to variational parameters ϕ, γ, λ to zero, we obtain
 - $\phi_{d,w,z} \propto \exp(\mathbf{E}_q[\log(\theta_{d,z})] + \mathbf{E}_q[\log(\beta_{z,w})])$
 - $\gamma_{d,z} = \alpha + \sum_w n_{d,w} \phi_{d,w,z}$
 - $\lambda_{z,w} = \eta + \sum_d n_{d,w} \phi_{d,w,z}$
- The following equations hold
 - $\mathbf{E}_q[\log(\theta_{d,z})] = \Psi(\gamma_{d,z}) - \Psi(\sum_z \gamma_{d,z})$
 - $\mathbf{E}_q[\log(\beta_{z,w})] = \Psi(\lambda_{z,w}) - \Psi(\sum_z \lambda_{z,w})$

where Ψ is the digamma function $\Psi(x) := \Gamma'(x)/\Gamma(x)$

Batch variational Bayes inference

Initialization: λ randomly initialized, ϵ is a small positive constant

While relative improvement of $L(\mathbf{n}, \boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda})$ is larger than ϵ :

// E step

For $d = 1$ to D

For $z = 1$ to K : $\gamma_{d,z} = 1$

Repeat

For $z = 1$ to K :

For $w = 1$ to W :

$$\begin{aligned}\phi_{d,w,z} &\propto \exp(E_q[\log(\theta_{d,z})] + E_q[\log(\beta_{z,w})]) \\ \gamma_{d,z} &\leftarrow \alpha + \sum_w n_{d,w} \phi_{d,w,z}\end{aligned}$$

until $\|\Delta\gamma_d\|_1 \leq K\epsilon$ // $\Delta\gamma_d$ is difference of γ_d in two successive iterations

// M step

For $z = 1$ to K :

For $w = 1$ to W :

$$\lambda_{z,w} \leftarrow \eta + \sum_d n_{d,w} \phi_{d,w,z}$$

- Computation complexity: requires one pass through all documents in each iteration

Online variational Bayes inference

Initialization: λ randomly initialized, ϵ is a small positive constant, τ_0 and κ to positive numbers

For $d = 0, 1, \dots$

$$\rho \leftarrow 1/(\tau_0 + d)^\kappa$$

// E step

For $z = 1$ to K :

$$\gamma_{d,z} = 1$$

Repeat

For $z = 1$ to K :

For $w = 1$ to W :

$$\phi_{d,w,z} \propto \exp(E_q[\log(\theta_{d,z})] + E_q[\log(\beta_{z,w})])$$

$$\gamma_{d,z} \leftarrow \alpha + \sum_w n_{d,w} \phi_{d,w,z}$$

until $\|\Delta\gamma_d\|_1 \leq K\epsilon$ // $\Delta\gamma_d$ is difference of γ_d in two successive iterations

// M step

For $z = 1$ to K :

For $w = 1$ to W :

$$\tilde{\lambda}_{z,w} \leftarrow \eta + D n_{d,w} \phi_{d,w,z}$$

$$\lambda \leftarrow (1 - \rho)\lambda + \rho \tilde{\lambda}$$

Spark LDA API

```
from pyspark.mllib.clustering import LDA, LDAModel
from pyspark.mllib.linalg import Vectors

# Load and parse the data
data = sc.textFile("data/mllib/sample_lda_data.txt")
parsedData = data.map(lambda line: Vectors.dense([float(x) for x in line.strip().split(' ')]))
# Index documents with unique IDs
corpus = parsedData.zipWithIndex().map(lambda x: [x[1], x[0]]).cache()

# Cluster the documents into three topics using LDA
ldaModel = LDA.train(corpus, k=3)

# Output topics. Each is a distribution over words (matching word count vectors)
print("Learned topics (as distributions over vocab of " + str(ldaModel.vocabSize())
     + " words):")
topics = ldaModel.topicsMatrix()
for topic in range(3):
    print("Topic " + str(topic) + ":")
    for word in range(0, ldaModel.vocabSize()):
        print(" " + str(topics[word][topic]))

# Save and load model
ldaModel.save(sc, "target/org/apache/spark/PythonLatentDirichletAllocationExample/LDAModel")
sameModel = LDAModel\
    .load(sc, "target/org/apache/spark/PythonLatentDirichletAllocationExample/LDAModel")
```

- Source: <https://spark.apache.org/docs/2.2.0/mllib-clustering.html#latent-dirichlet-allocation-lda>
- Full code example: [Spark GitHub repo](#)

Spark LDA implementation

```
156     /** Supported values for Param [[optimizer]]. */
157     @Since("1.6.0")
158     final val supportedOptimizers: Array[String] = Array("online", "em")
159
160     /**
161      * Optimizer or inference algorithm used to estimate the LDA model.
162      * Currently supported (case-insensitive):
163      * - "online": Online Variational Bayes (default)
164      * - "em": Expectation-Maximization
165      *
166      * For details, see the following papers:
167      * - Online LDA:
168      *   Hoffman, Blei and Bach. "Online Learning for Latent Dirichlet Allocation."
169      *   Neural Information Processing Systems, 2010.
170      *   See <a href="http://www.cs.columbia.edu/~blei/papers/HoffmanBleiBach2010b.pdf">here</a>
171      * - EM:
172      *   Asuncion et al. "On Smoothing and Inference for Topic Models."
173      *   Uncertainty in Artificial Intelligence, 2009.
174      *   See <a href="http://arxiv.org/pdf/1205.2662.pdf">here</a>
175      *
176      * @group param
177     */
```

- Source code: <https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/clustering/LDA.scala>

Spark LDA implementation (cont'd)

```
64  /**
65  * :: DeveloperApi ::
66  *
67  * Optimizer for EM algorithm which stores data + parameter graph, plus algorithm parameters.
68  *
69  * Currently, the underlying implementation uses Expectation-Maximization (EM), implemented
70  * according to the Asuncion et al. (2009) paper referenced below.
71  *
72  * References:
73  * - Original LDA paper (journal version):
74  *   Blei, Ng, and Jordan. "Latent Dirichlet Allocation." JMLR, 2003.
75  *   - This class implements their "smoothed" LDA model.
76  * - Paper which clearly explains several algorithms, including EM:
77  *   Asuncion, Welling, Smyth, and Teh.
78  *   "On Smoothing and Inference for Topic Models." UAI, 2009.
79  */
80 @Since("1.4.0")
81 @DeveloperApi
82 final class EMLDAOptimizer extends LDAOptimizer {
83
84     import LDA._
85
86     // Adjustable parameters
87     private var keepLastCheckpoint: Boolean = true
88
89     /**
90      * If using checkpointing, this indicates whether to keep the last checkpoint (vs clean up).
91      */
92     @Since("2.0.0")
93     def getKeepLastCheckpoint: Boolean = this.keepLastCheckpoint
94 }
```

- Source:
<https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/clustering/LDAOptimizer.scala#L82>

Spark LDA implementation (cont'd)

```
255  /**
256  * :: DeveloperApi ::
257  *
258  * An online optimizer for LDA. The Optimizer implements the Online variational Bayes LDA
259  * algorithm, which processes a subset of the corpus on each iteration, and updates the term-topic
260  * distribution adaptively.
261  *
262  * Original Online LDA paper:
263  * Hoffmann, Blei and Bach, "Online Learning for Latent Dirichlet Allocation." NIPS, 2010.
264  */
265  @Since("1.4.0")
266  @DeveloperApi
] 267  final class OnlineLDAOptimizer extends LDAOptimizer with Logging {
268
269  // LDA common parameters
270  private var k: Int = 0
271  private var corpusSize: Long = 0
272  private var vocabSize: Int = 0
273
274  /** alias for docConcentration */
275  private var alpha: Vector = Vectors.dense(0)
276
277  /** (for debugging) Get docConcentration */
278  private[clustering] def getAlpha: Vector = alpha
279
280  /** alias for topicConcentration */
281  private var eta: Double = 0
282
283  /** (for debugging) Get topicConcentration */
284  private[clustering] def getEta: Double = eta
285
286  private var randomGenerator: java.util.Random = null
287
288  /** (for debugging) Whether to sample mini-batches with replacement. (default = true) */
289  private var sampleWithReplacement: Boolean = true
290
291  // Online LDA specific parameters
292  // Learning rate is: (tau0 + t)^{-kappa}
293  private var tau0: Double = 1024
294  private var kappa: Double = 0.51
295  private var miniBatchFraction: Double = 0.05
296  private var optimizeDocConcentration: Boolean = false
297
```

- Source:
<https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/clustering/LDAOptimizer.scala#L267>

References

Collaborative filtering

- Y. Koren et al, [Matrix Factorization Techniques for Recommender Systems](#), Computer, Vol 42, No 8, 2009
- Y. Hu et al, [Collaborative Filtering for Implicit Feedback Datasets](#), ICDM 2008
- Y. Zhou et al, [Large-Scale Parallel Collaborative Filtering for the Netflix Prize](#), AAIM 2008
- X. Meng, [A More Scalable Way of Making Recommendations with Mllib](#), Spark Summit 2015
- B. Yavuz et al, [Scalable Collaborative Filtering with Apache Spark Mllib](#), Databricks, blog, 2014

References (cont'd)

LSA / SVD

- R. Zadeh et al, [Matrix Computations and Optimizations in Apache Spark](#), KDD 2016
 - Section 3.1: Singular Value Decomposition
- S. Deerwester et al, [Indexing by Latent Semantic Analysis](#), Journal of the American Society for Information Science, Vol 41, No 6, 391-407, 1990
 - An early proposal to use SVD for information retrieval
- R. Zadeh, [Dimension Independent Matrix Square using Mapreduce](#), STOC 2013 sides
 - For $\mathbf{M}^T \mathbf{M}$ computation
- R. Zadeh and A. Goel, [Dimension Independent Similarity Computation](#), JMLR 2012

References (cont'd)

LDA

- D. M. Blei, [Probabilistic Topic Models](#), Communications of the ACM, 2012
 - Review article
- D. M. Blei et al, [Latent Dirichlet Allocation](#), JMLR 2003
 - First proposal of LDA method
- M. Hoffman et al, [Online Learning for Latent Dirichlet Allocation](#), NIPS 2010
 - Variational Bayes inference used in Spark class [OnlineLDAOptimizer](#)
- A. Asuncion et al, [On Smoothing and Inference for Topic Models](#), UAI 2009
 - EM algorithm, used in Spark class [EMLDAOptimizer](#)
- C. E. Rasmussen, [Latent Dirichlet Allocation for Topic Modelling](#), lecture, 2016

References (cont'd)

Spark MLLib

- Apache Spark [Machine Learning Library \(MLlib\) Guide](#)
- Apache Spark [Mllib](#)
- X. Meng et al, [MLib: Machine Learning in Apache Spark](#), Vol 17, JMLR 2016
- R. Zadeh, [Distributed Machine Learning on Spark](#), Strata 2015
- X. Meng, [MLlib: Scalable Machine Learning on Spark](#)
- Apache Spark [Optimization - RDD based API](#)
- Spark summer school 2013 [here](#)

Seminar class

- Movielens recommending movies using ALS

```
$ head ratings.csv
userId,movieId,rating,timestamp
1,307,3.5,1256677221
1,481,3.5,1256677456
1,1091,1.5,1256677471
1,1257,4.5,1256677460
1,1449,4.5,1256677264
1,1590,2.5,1256677236
1,1591,1.5,1256677475
1,2134,4.5,1256677464
1,2478,4.0,1256677239
```

```
$ head movies.csv
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
5,Father of the Bride Part II (1995),Comedy
6,Heat (1995),Action|Crime|Thriller
7,Sabrina (1995),Comedy|Romance
8,Tom and Huck (1995),Adventure|Children
9,Sudden Death (1995),Action
```

- SVD and LDA on news text data

```
[ 'I was wondering if anyone out there could enlighten me on this car I sa
w\nthe other day. It was a 2-door sports car, looked to be from the late
60s/\nearly 70s. It was called a Bricklin. The doors were really small. I
n addition,\nthe front bumper was separate from the rest of the body. Thi
s is \nall I know. If anyone can tellme a model name, engine specs, years
\nof production, where this car is made, history, or whatever info you\nh
ave on this funky looking car, please e-mail.', "A fair number of brave s
ouls who upgraded their SI clock oscillator have\nshared their experience
s for this poll. Please send a brief message detailing\nyour experiences
with the procedure. Top speed attained, CPU rated speed,\nadd on cards an
d adapters, heat sinks, hour of usage per day, floppy disk\nfunctionalit
y with 800 and 1.4 m floppies are especially requested.\n\nI will be summar
izing in the next two days, so please add to the network\nknowledge base
```

<https://github.com/lse-st446/lectures2021/blob/master/Week09/class/README.md>