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

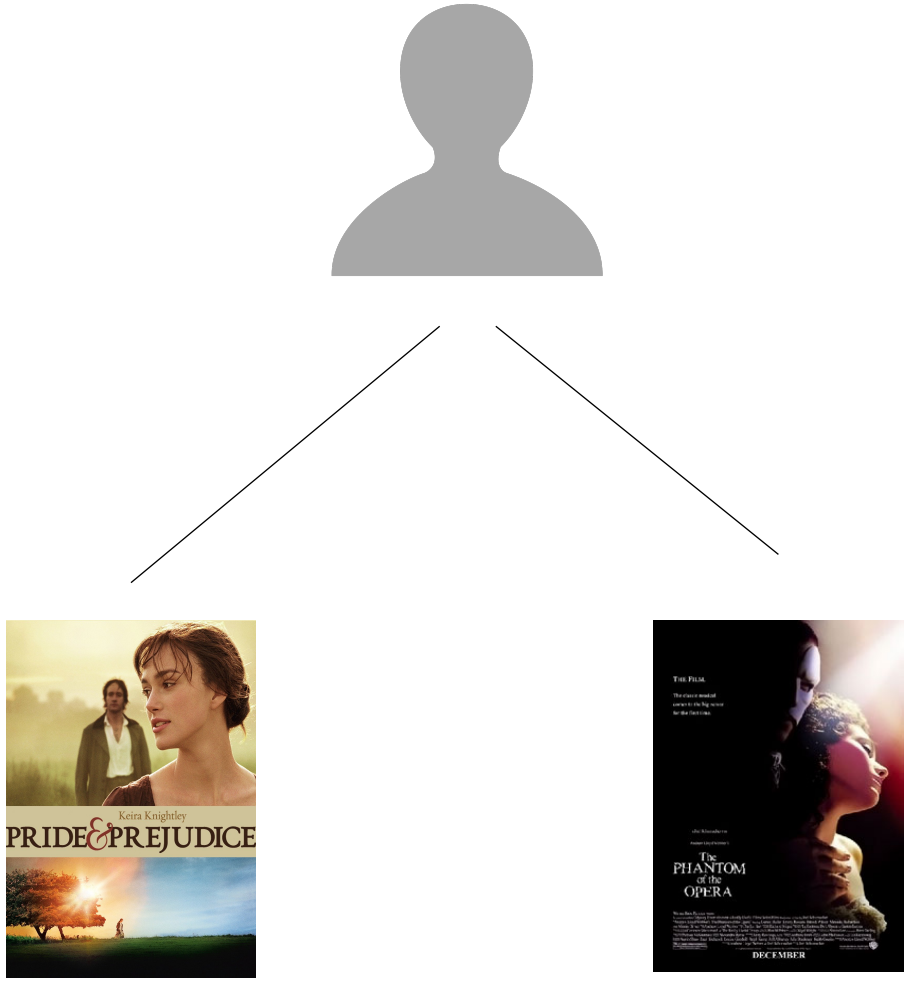
- Learn about distributed computing for machine learning tasks used for making recommendations and topic modelling, in particular:
  - Matrix completion (alternating least squares algorithm)
  - Latent Semantic Analysis (LSA) / Singular Value Decomposition (SVD)
  - Latent Dirichlet Allocation (LDA)

Topics of this lecture

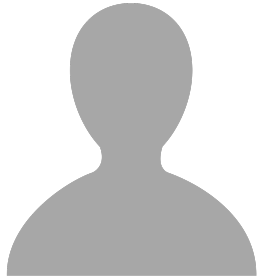
- Matrix completion: introduction
- Matrix completion: alternating least squares algorithm
- Latent Semantic Analysis (LSA)
- Latent Dirichlet Allocation (LDA)

Matrix completion: introduction

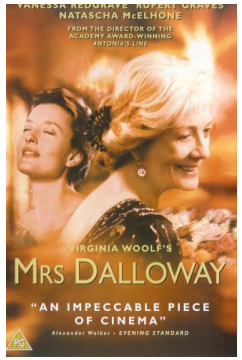
Example: movie recommendations



Example: movie recommendations (cont'd)



You may also like



Example: movie recommendations (cont'd)

Data

Example: movie recommendations (cont'd)

Data: matrix representation

items

users

5 ? 5 ? ? ? ?

? ? 5 ? 3 4 ?

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 $\boldsymbol{M}$ 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 $\boldsymbol{M}$ by a low-rank matrix $\widehat{\boldsymbol{M}}$ (say of rank k)

$$\widehat{M}_{i,j} = \boldsymbol{u}_i^\top \boldsymbol{v}_j$$

  where $\boldsymbol{u}_i \in \mathbf{R}^k$ is a *user parameter vector* and $\boldsymbol{v}_j \in \mathbf{R}^k$ is an *item parameter vector*

- In matrix notation: $\widehat{\boldsymbol{M}} = \boldsymbol{U}^\top \boldsymbol{V}$ where $\boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]$ and $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m]$

Loss function minimization

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

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

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

- Minimizing f is a *non-convex optimization problem* because of the quadratic terms $\boldsymbol{u}_i^\top \boldsymbol{v}_j$
  - Note however that f is convex in $\boldsymbol{U}$ for every fixed $\boldsymbol{V}$ 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 $\boldsymbol{M}$ is a random matrix with independent Gaussian entries and let $\boldsymbol{M}_\Omega$ be entries of $M_{i,j}$ for $(i,j) \in \Omega$

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

and

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

where is α is a hyper-parameter

- By Bayes' rule, posterior distribution of $\boldsymbol{U}, \boldsymbol{V}$ is given by $p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{M}_\Omega) \propto p(\boldsymbol{M}_\Omega \mid \boldsymbol{U}, \boldsymbol{V}) p(\boldsymbol{U}, \boldsymbol{V})$ where $p(\boldsymbol{U}, \boldsymbol{V})$ is a given prior distribution

- Assume prior distribution is of product form $p(\boldsymbol{U}, \boldsymbol{V}) = \prod_{i=1}^{n} p(\boldsymbol{u}_i) \prod_{j=1}^{m} p(\boldsymbol{v}_j)$ with Gaussian marignals $\boldsymbol{u}_i \sim N(\boldsymbol{0}, \lambda \boldsymbol{I})$ and $\boldsymbol{v}_j \sim N(\boldsymbol{0}, \lambda \boldsymbol{I})$ 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

$$f(\boldsymbol{U},\boldsymbol{V}) \coloneqq -\log\left(p(\boldsymbol{U},\boldsymbol{V} \mid \boldsymbol{M}_{\Omega})\right)$$

$$= \frac{1}{2}\alpha \sum_{(i,j)\in\Omega}\left(M_{i,j} - \boldsymbol{u}_i^{\top}\boldsymbol{v}_j\right)^2 + \frac{1}{2}\lambda\left(\sum_{i=1}^{n}\|\boldsymbol{u}_i\|^2 + \sum_{j=1}^{m}\|\boldsymbol{v}_j\|^2\right)$$

- 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 $\boldsymbol{U}$ for fixed $\boldsymbol{V}$, and
  - Phase V: optimizing over $\boldsymbol{V}$ for fixed $\boldsymbol{U}$

- Gradient vector $\nabla_{\boldsymbol{U}} f(\boldsymbol{U}, \boldsymbol{V})$ entries are given by

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

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

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

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: $\nabla_{\boldsymbol{U}} f(\boldsymbol{U}, \boldsymbol{V}) = \mathbf{0}$
  - For Phase V: $\nabla_{\boldsymbol{V}} f(\boldsymbol{U}, \boldsymbol{V}) = \mathbf{0}$

- Condition $\nabla_{\boldsymbol{U}} f(\boldsymbol{U}, \boldsymbol{V}) = \mathbf{0}$ is equivalent to

$$\left(\lambda \boldsymbol{I} + \sum_{j:(i,j)\in\Omega} \boldsymbol{v}_j \boldsymbol{v}_j^\top\right) \boldsymbol{u}_i = \sum_{j:(i,j)\in\Omega} M_{i,j} \boldsymbol{v}_j \text{ for } i \in [n]$$

- Condition $\nabla_{\boldsymbol{V}} f(\boldsymbol{U}, \boldsymbol{V}) = \mathbf{0}$ is equivalent to

$$\left(\lambda \boldsymbol{I} + \sum_{i:(i,j)\in\Omega} \boldsymbol{u}_i \boldsymbol{u}_i^\top\right) \boldsymbol{v}_j = \sum_{i:(i,j)\in\Omega} M_{i,j} \boldsymbol{u}_i \text{ for } j \in [m]$$

ALS pseudo-code

Initialization: U, V

Repeat until a termination criteria is met

For $i = 1$ to n

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

For $j = 1$ to m

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

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

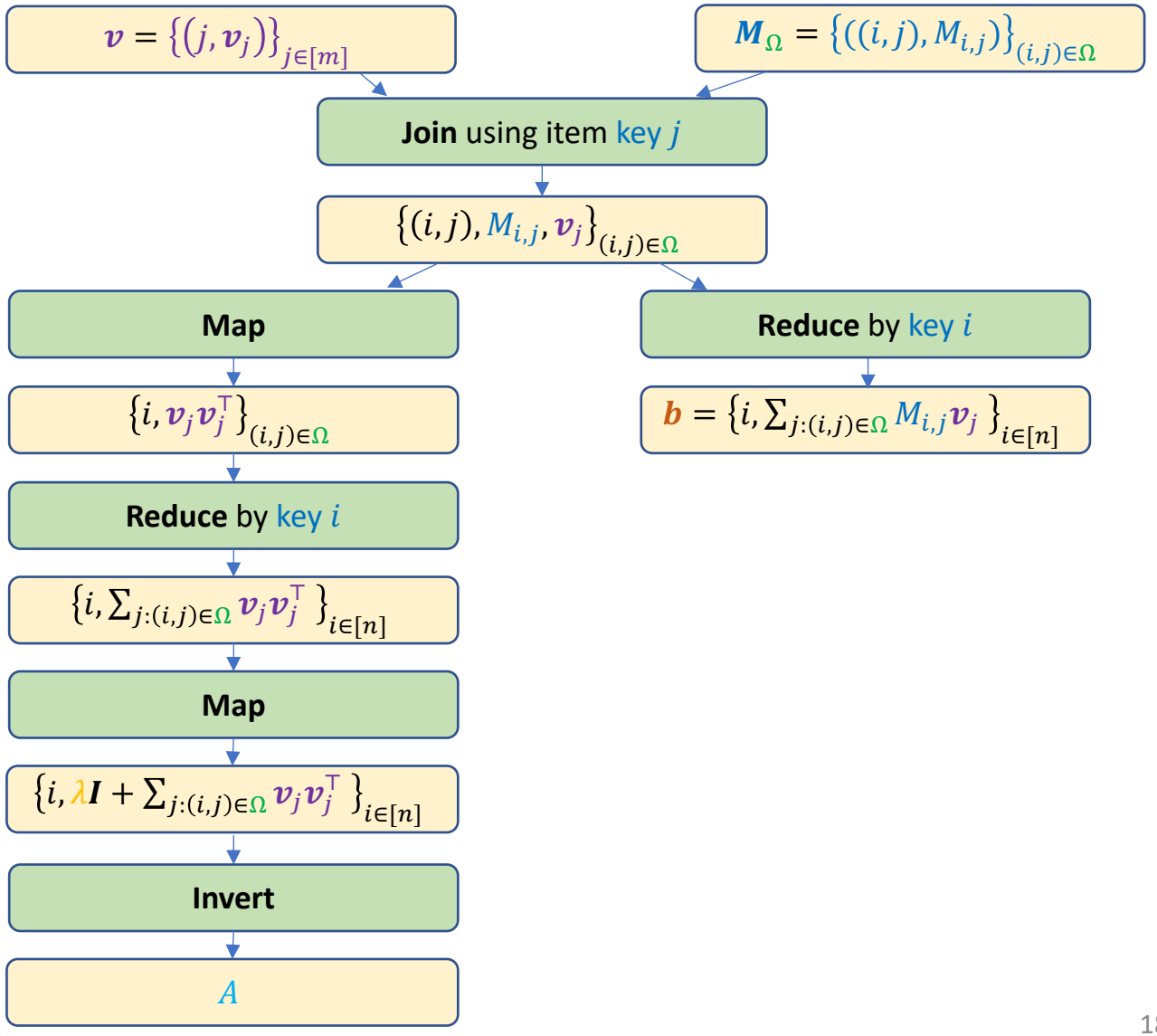
Distributed ALS using join

- Optimization over $\boldsymbol{U}$ for fixed $\boldsymbol{V}$:

$$\boldsymbol{u}_i \leftarrow \boldsymbol{A}_i\, \boldsymbol{b}_i$$

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

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



Distributed ALS using broadcast

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

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

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 $\boldsymbol{U}$ and $\boldsymbol{V}$ and broadcast $\boldsymbol{V}$ to all machines

Repeat until a termination criteria is met:

- Compute $\boldsymbol{u}_i$'s locally on machines using user-key partition $\boldsymbol{M}_{\Omega}^{u}$ and $\boldsymbol{V}$
- Broadcast $\boldsymbol{U}$ to all machines
- Compute $\boldsymbol{v}_j$'s locally on machines using item-key partition $\boldsymbol{M}_{\Omega}^{v}$ and $\boldsymbol{U}$
- Broadcast $\boldsymbol{V}$ to all machines

Some recent theoretical results

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

$$f(\boldsymbol{U}) = \sum_{(i,j)\in\Omega}\left(M_{i,j} - \boldsymbol{u}_i^\top \boldsymbol{u}_j\right)^2$$

  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)

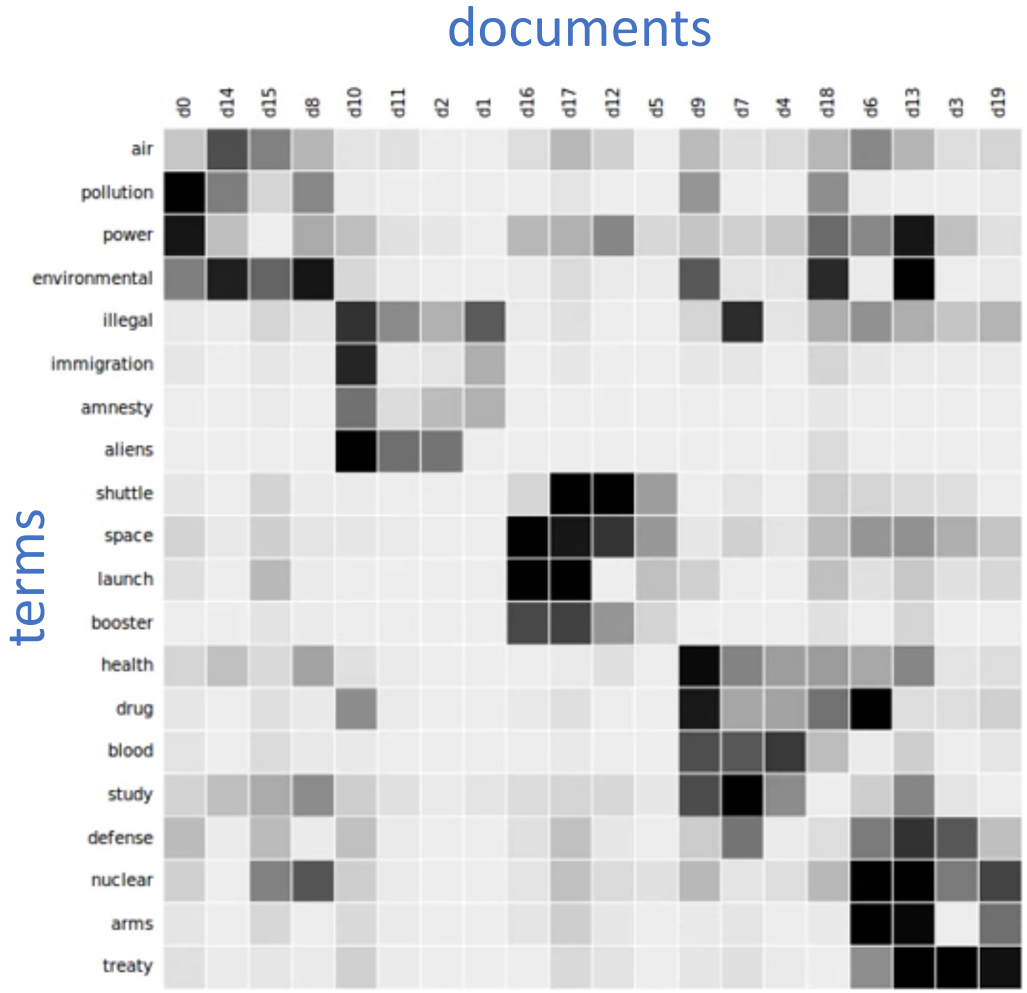
Latent Semantic Analysis

- 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* $\boldsymbol{M}$ 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 $\boldsymbol{M}$ of rank k is given by

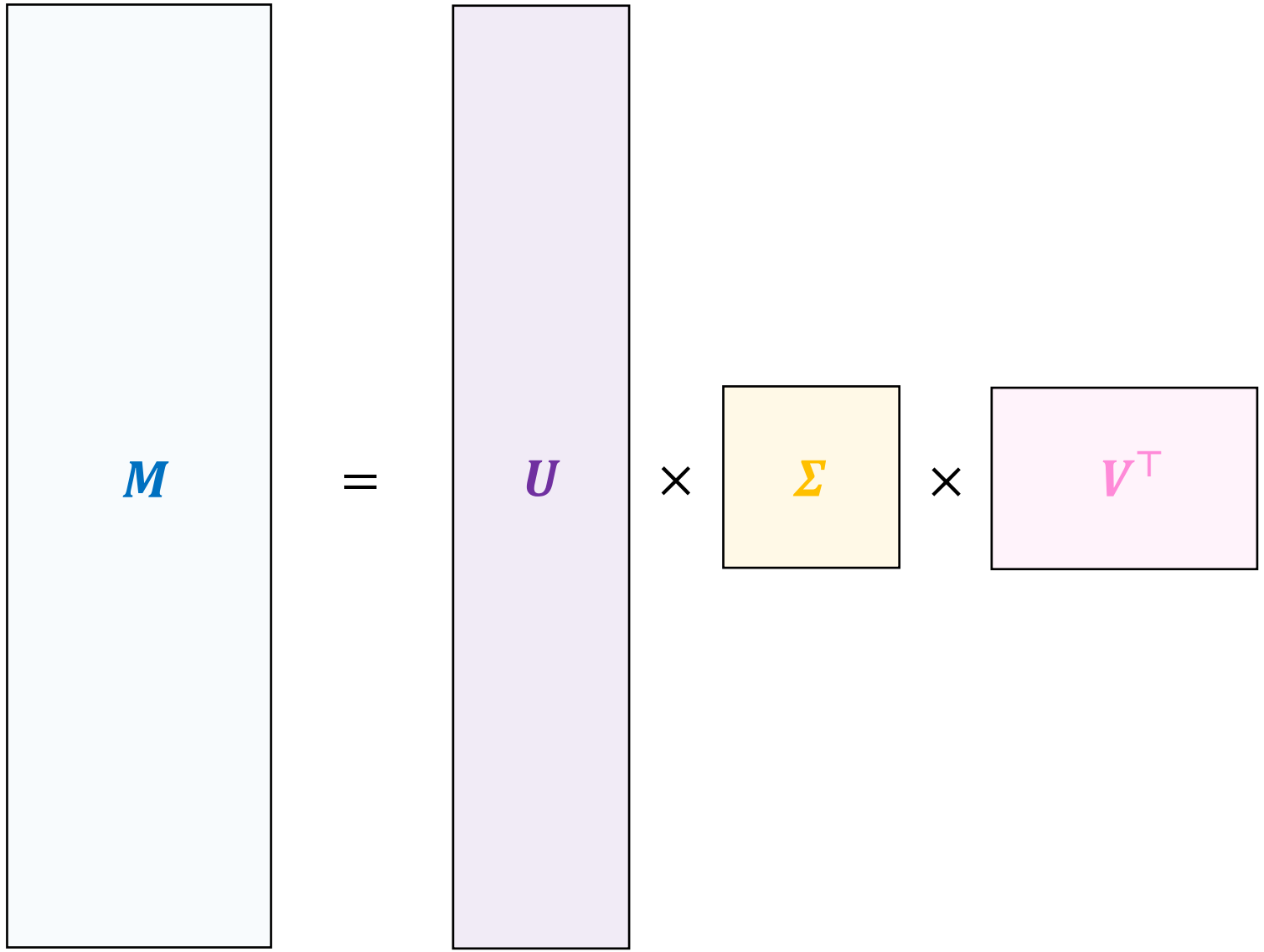
$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$$

where

$\boldsymbol{U}$ is a $m \times k$ unitary matrix
$\boldsymbol{\Sigma}$ is a $k \times k$ diagonal matrix with real-valued non-negative diagonal elements
$\boldsymbol{V}$ is a $n \times k$ unitary matrix

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

Tall and skinny case



Distributed computing: tall and skinny case

- Key idea: separation in two steps
  - Step 1: computation of $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$
  - Step 2: computation of $\boldsymbol{U}$

- Step 1: compute $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$

  - Compute $\boldsymbol{M}^\top \boldsymbol{M}$ using all-to-one communication to driver node
    - $\boldsymbol{M}^\top \boldsymbol{M}$ is a $n \times n$ matrix that can fit in memory of a single node

  - Compute $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$ by eigen-decomposition $\boldsymbol{M}^\top \boldsymbol{M} = \boldsymbol{V} \boldsymbol{\Sigma}^2 \boldsymbol{V}^\top$ locally on the driver node
    - $\boldsymbol{M}^\top \boldsymbol{M}$ is a positive semidefinite matrix with eigenvalues equal to squares of singular values of $\boldsymbol{M}$

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

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

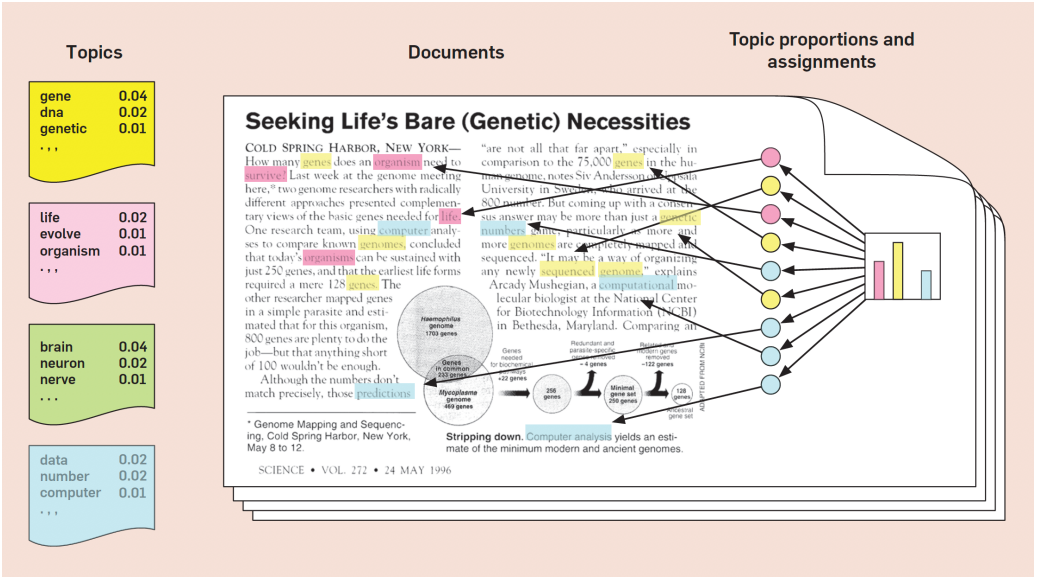
- Example: https://spark.apache.org/docs/2.2.0/mllib-dimensionality-reduction.html
- Full example code: https://github.com/apache/spark/blob/master/examples/src/main/python/mllib/svd_example.py

Latent Dirichlet Allocation

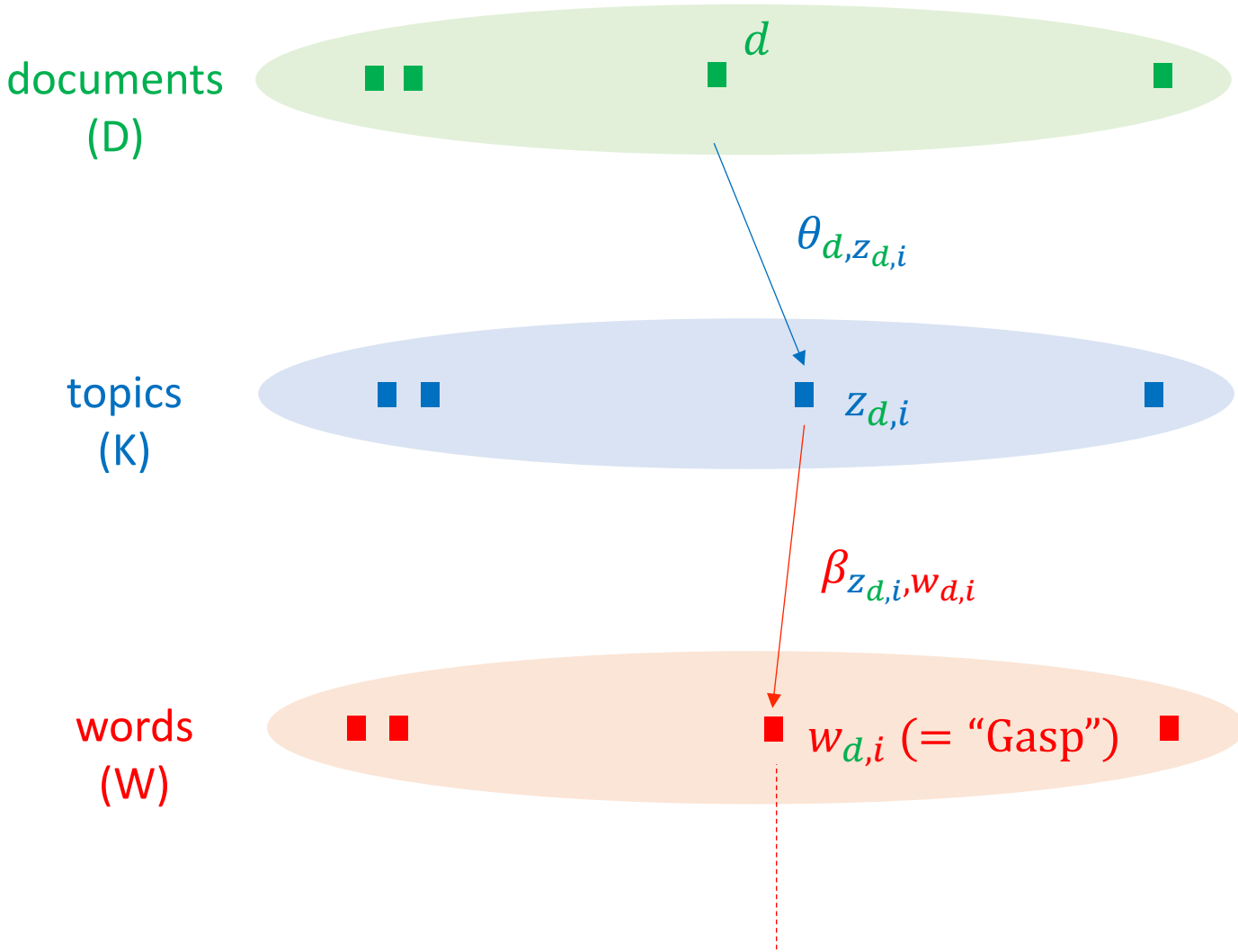
Probabilistic topic models

- 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)

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

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

Matrix factorization interpretation:

$$\mathbf{P}\left[w_{d,i} = w \mid \boldsymbol{\theta}_d, \boldsymbol{\beta} \right] = \sum_z \theta_{d,z} \beta_{z,w}$$

Bayesian models and inference

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

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

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

- Parameter vector $\boldsymbol{\theta}$ interpreted as a random variable with given prior distribution $p_0(\boldsymbol{x})$

- Bayesian inference: posterior distribution of $\boldsymbol{\theta}$ conditional on observed data $\boldsymbol{Y} = \boldsymbol{y}$

- By Bayes' rule: posterior distribution $p_1(\boldsymbol{x})$ of $\boldsymbol{\theta}$ is given by

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

Bayesian inference for LDA

- Prior distributions:
  - $\theta_d \sim \mathrm{Dir}(\boldsymbol{\alpha})$
  - $\beta_z \sim \mathrm{Dir}(\boldsymbol{\eta})$

  where $\mathrm{Dir}(\boldsymbol{a})$ is the *Dirichlet distribution* with parameter $\boldsymbol{a}$

- *Dirichlet distribution* of order $r \geq 2$ with parameter $\boldsymbol{a} = (a_1, \ldots, a_r)^\top \in \mathbf{R}_+^r$ is a distribution over the simplex $S_r = \{x \in \mathbf{R}_+^r : \sum_{i=1}^r x_i = 1\}$ with the density function

$$p(x_1, \ldots x_r; a_1, \ldots, a_r) = \frac{1}{B(a_1, \ldots, a_r)} x_1^{a_1 - 1} \cdots x_r^{a_r - 1}$$

  where $B(a_1, \ldots, a_r)$ is the normalizing constant (multivariate Beta function)

- Remarks:
  - $\boldsymbol{a}$ 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 $n, p_1, \dots, p_r$ is defined as follows

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

  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 $\mathrm{Dir}(a_1, \dots, a_r)$ prior distribution

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

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

Posterior distribution for LDA

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

$$p(\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\theta} \mid \boldsymbol{\alpha}, \boldsymbol{\eta}) = \prod_{z\prime} p(\beta_{z\prime} \mid \boldsymbol{\eta}) \prod_d p(\boldsymbol{\theta}_d \mid \boldsymbol{\alpha}) \prod_i p\big(z_{d,i} \mid \boldsymbol{\theta}_d\big) p(w_{d,i} \mid \boldsymbol{\beta}, z_{d,i})$$

- Computing posterior distribution of $\boldsymbol{\alpha}$ and $\boldsymbol{\eta}$ requires marginalizing out the joint distribution of $(\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\theta})$ over $(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\theta})$ 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 $(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\theta})$, the following lower bound holds:

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

- Proof of ELBO is given in the next slide (ELBO: Evidence Lower Bound)

- Maximizing ELBO is equivalent to minimizing the Kullback-Leibler divergence between $q(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\theta})$ and $p(\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\theta} \mid \boldsymbol{\alpha}, \boldsymbol{\eta})$

- *Kullback-Leibler divergence*: a commonly used measure of "distance" between two probability distributions
  - To probe further: see wikipedia

Proof of ELBO

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

$$\log\big(p(\boldsymbol{x})\big) = \log\big(\textstyle\sum_{\boldsymbol{y}\in Y} p(\boldsymbol{x}, \boldsymbol{y})\big)$$

$$= \log\left(\textstyle\sum_{\boldsymbol{y}\in Y} \frac{p(\boldsymbol{x},\boldsymbol{y})}{q(\boldsymbol{y})} q(\boldsymbol{y})\right)$$

$$\geq \textstyle\sum_{y\in Y} \log\left(\frac{p(\boldsymbol{x},\boldsymbol{y})}{q(\boldsymbol{y})}\right) q(\boldsymbol{y}) \qquad \text{(Jensen's inequality)}$$

$$= \mathbf{E}_{\boldsymbol{Y}\sim q}\left[\log\left(\frac{p(\boldsymbol{x},\boldsymbol{Y})}{q(\boldsymbol{Y})}\right)\right]$$

Variational Bayes inference for LDA (cont'd)

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

  $q(z_{d,i} = z) = \phi_{d,w_{d,i},z}$, $q(\boldsymbol{\theta}_d) = \mathrm{Dir}(\boldsymbol{\theta}_d; \boldsymbol{\gamma}_d)$ and $q(\boldsymbol{\beta}_z) = \mathrm{Dir}(\boldsymbol{\beta}_z; \boldsymbol{\lambda}_z)$

  where $\boldsymbol{\phi}$, $\boldsymbol{\gamma}$, $\boldsymbol{\lambda}$ are variational parameters

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

$$L(\boldsymbol{w}, \boldsymbol{\phi}, \boldsymbol{\alpha}, \boldsymbol{\eta}) = \sum_{d=1}^{D} \left\{ \begin{array}{c} \mathbf{E}_q[\log(p(\boldsymbol{w}_d \mid \boldsymbol{\theta}_d, \boldsymbol{z}_d, \boldsymbol{\beta}))] \\ +\mathbf{E}_q[\log(p(\boldsymbol{z}_d \mid \boldsymbol{\theta}_d)] \\ -\mathbf{E}_q[\log(q(\boldsymbol{z}_d))] \\ +\mathbf{E}_q[\log(p(\boldsymbol{\theta}_d \mid \boldsymbol{\alpha}))] \\ -\mathbf{E}_q[\log(q(\boldsymbol{\theta}_d)] \\ +\frac{1}{D}(\mathbf{E}_q[p(\boldsymbol{\beta} \mid \boldsymbol{\eta})] - \mathbf{E}_q[\log(q(\boldsymbol{\beta}))]) \end{array} \right\}$$

Variational Bayes inference (cont'd)

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

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

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

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

Variational Bayes inference (cont'd)

- Setting gradients with respect to variational parameters $\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}$ to zero, we obtain
  - $\phi_{d,w,z} \propto \exp\left(\mathbf{E}_q\left[\log\left(\theta_{d,z}\right)\right] + \mathbf{E}_q\left[\log\left(\beta_{z,w}\right)\right]\right)$
  - $\gamma_{d,z} = \alpha + \sum_w n_{d,w}\phi_{d,w,z}$
  - $\lambda_{z,w} = \eta + \sum_d n_{d,w}\phi_{d,w,z}$

- The following equations hold
  - $\mathbf{E}_q\left[\log\left(\theta_{d,z}\right)\right] = \Psi\left(\gamma_{d,z}\right) - \Psi\left(\sum_{z'} \gamma_{d,z'}\right)$
  - $\mathbf{E}_q\left[\log\left(\beta_{z,w}\right)\right] = \Psi\left(\lambda_{z,w}\right) - \Psi\left(\sum_{z'} \lambda_{z,w'}\right)$

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

Batch variational Bayes inference

Initialization: $\boldsymbol{\lambda}$ randomly initialized, ϵ is a small positive constant

While relative improvement of $L(\boldsymbol{n}, \boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda})$ 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:

$$\phi_{d,w,z} \propto \exp\left(\mathbf{E}_q\left[\log\left(\theta_{d,z}\right)\right] + \mathbf{E}_q\left[\log\left(\beta_{z,w}\right)\right]\right)$$

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

until $\|\Delta\boldsymbol{\gamma}_d\|_1 \leq K\epsilon$ // $\Delta\boldsymbol{\gamma}_d$ is difference of $\boldsymbol{\gamma}_d$ 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: $\boldsymbol{\lambda}$ randomly initialized, ϵ is a small positive constant, τ_0 and κ to positive numbers

For $d = 0, 1, \ldots$

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

// E step

For $z = 1$ to K:

$\gamma_{d,z} = 1$

Repeat

For $z = 1$ to K:

For $w = 1$ to W:

$\phi_{d,w,z} \propto \exp\left(\mathbf{E}_q[\log(\theta_{d,z})] + \mathbf{E}_q[\log(\beta_{z,w})]\right)$

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

until $\|\Delta\boldsymbol{\gamma}_d\|_1 \leq K\epsilon$ // $\Delta\boldsymbol{\gamma}_d$ is difference of $\boldsymbol{\gamma}_d$ in two successive iterations

// M step

For $z = 1$ to K:

For $w = 1$ to W:

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

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

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 GithHub 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)

```scala
/**
 * :: DeveloperApi ::
 *
 * Optimizer for EM algorithm which stores data + parameter graph, plus algorithm parameters.
 *
 * Currently, the underlying implementation uses Expectation-Maximization (EM), implemented
 * according to the Asuncion et al. (2009) paper referenced below.
 *
 * References:
 *  - Original LDA paper (journal version):
 *    Blei, Ng, and Jordan.  "Latent Dirichlet Allocation."  JMLR, 2003.
 *     - This class implements their "smoothed" LDA model.
 *  - Paper which clearly explains several algorithms, including EM:
 *    Asuncion, Welling, Smyth, and Teh.
 *    "On Smoothing and Inference for Topic Models."  UAI, 2009.
 */
@Since("1.4.0")
@DeveloperApi
final class EMLDAOptimizer extends LDAOptimizer {

  import LDA._

  // Adjustable parameters
  private var keepLastCheckpoint: Boolean = true

  /**
   * If using checkpointing, this indicates whether to keep the last checkpoint (vs clean up).
   */
  @Since("2.0.0")
  def getKeepLastCheckpoint: Boolean = this.keepLastCheckpoint

```

- 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)

```scala
/**
 * :: DeveloperApi ::
 *
 * An online optimizer for LDA. The Optimizer implements the Online variational Bayes LDA
 * algorithm, which processes a subset of the corpus on each iteration, and updates the term-topic
 * distribution adaptively.
 *
 * Original Online LDA paper:
 *   Hoffman, Blei and Bach, "Online Learning for Latent Dirichlet Allocation." NIPS, 2010.
 */
@Since("1.4.0")
@DeveloperApi
final class OnlineLDAOptimizer extends LDAOptimizer with Logging {

  // LDA common parameters
  private var k: Int = 0
  private var corpusSize: Long = 0
  private var vocabSize: Int = 0

  /** alias for docConcentration */
  private var alpha: Vector = Vectors.dense(0)

  /** (for debugging)  Get docConcentration */
  private[clustering] def getAlpha: Vector = alpha

  /** alias for topicConcentration */
  private var eta: Double = 0

  /** (for debugging)  Get topicConcentration */
  private[clustering] def getEta: Double = eta

  private var randomGenerator: java.util.Random = null

  /** (for debugging) Whether to sample mini-batches with replacement. (default = true) */
  private var sampleWithReplacement: Boolean = true

  // Online LDA specific parameters
  // Learning rate is: (tau0 + t)^{-kappa}
  private var tau0: Double = 1024
  private var kappa: Double = 0.51
  private var miniBatchFraction: Double = 0.05
  private var optimizeDocConcentration: Boolean = false

```

- 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 $\boldsymbol{M}^\top \boldsymbol{M}$ 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\nfunctionality
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