Exploring Wikipedia with LSA

BDA - Master MSE - june 2017

Lucy Linder, Kewin Dousse, Davide Mazzoleni, Christophe Blanquet

Plan

- Concepts
- Dataset and preprocessing
- Book’s implementation
  - SVD
  - Improvements
- Our implementation
  - LDA
- Results

1.

Concepts

Latent Semantic Analysis (LSA)

aims to discover underlying “topics” / determine the relationship between terms and concepts in a collection of documents.

5,414,583

Is the number of articles in the English Wikipedia
(as of 29 May, 2017)

$$svd(A) = [U, s, V]$$

Singular Value Decomposition

SVD

Matrix of document / term frequency (TF-IDF)

Matrix of concept / term (V^T)

concepts (s)

$$\begin{bmatrix} M \times N \end{bmatrix} = \begin{bmatrix} M \times k \end{bmatrix} \times \begin{bmatrix} k \times 1 \end{bmatrix} \times \begin{bmatrix} k \times N \end{bmatrix}$$

Matrix of document / concept (U)

M = documents
N = terms
K = concepts

Latent Dirichlet Allocation

LDA

Clustering algorithm:

- cluster centers ⇒ topics
- Examples ⇒ documents
- Features ⇒ word counts
- Distance ⇒ ~~Euclidean~~ based on a statistical model (Bayesian inference rules / Dirichlet distribution)

Parameters (priors): k, α, β

2.

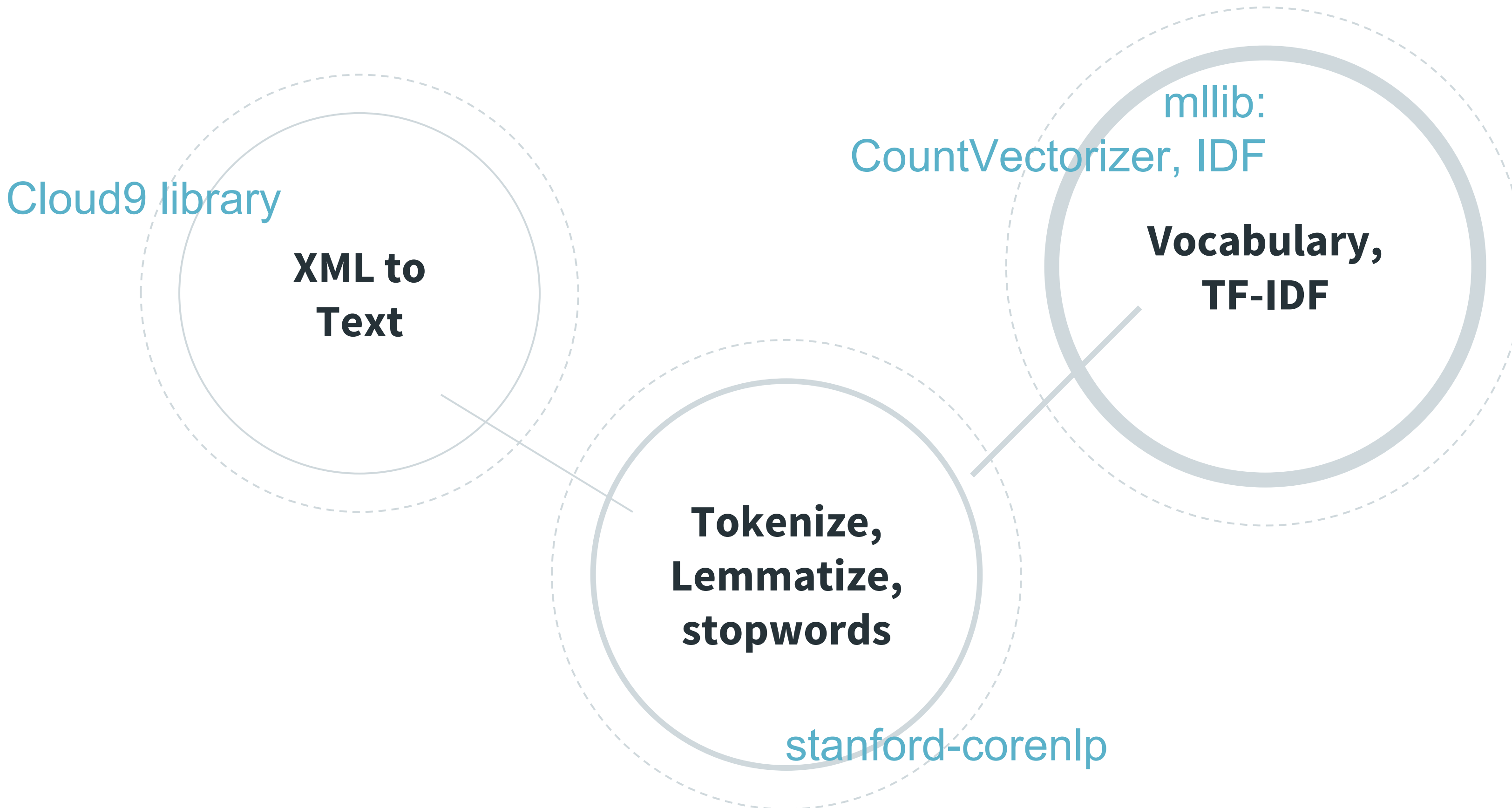
Dataset and preprocessing

datasets

- Wikidump 2017 : 57.5 GB
- Samples : ?
- Custom dump
  - https://en.wikipedia.org/wiki/Special:Export

All in XML format.

Preprocessing steps



Wikidump ⇒ (docTermMatrix, termIds, tfidf)

3.

Book's implementation

SVD

Three steps

Preparation	Topic modeling	Query
Whole wikipedia set.	Using SVD.	LSAQueryEngine.
wikiXmlToPlainText	RunLSA	topDocsForTerm
contentToTerms	computeSVD	topTermsForTerm
documentTermMatrix		topDocsForDoc
		topDocsForTermQuery
		...

Sources on Github

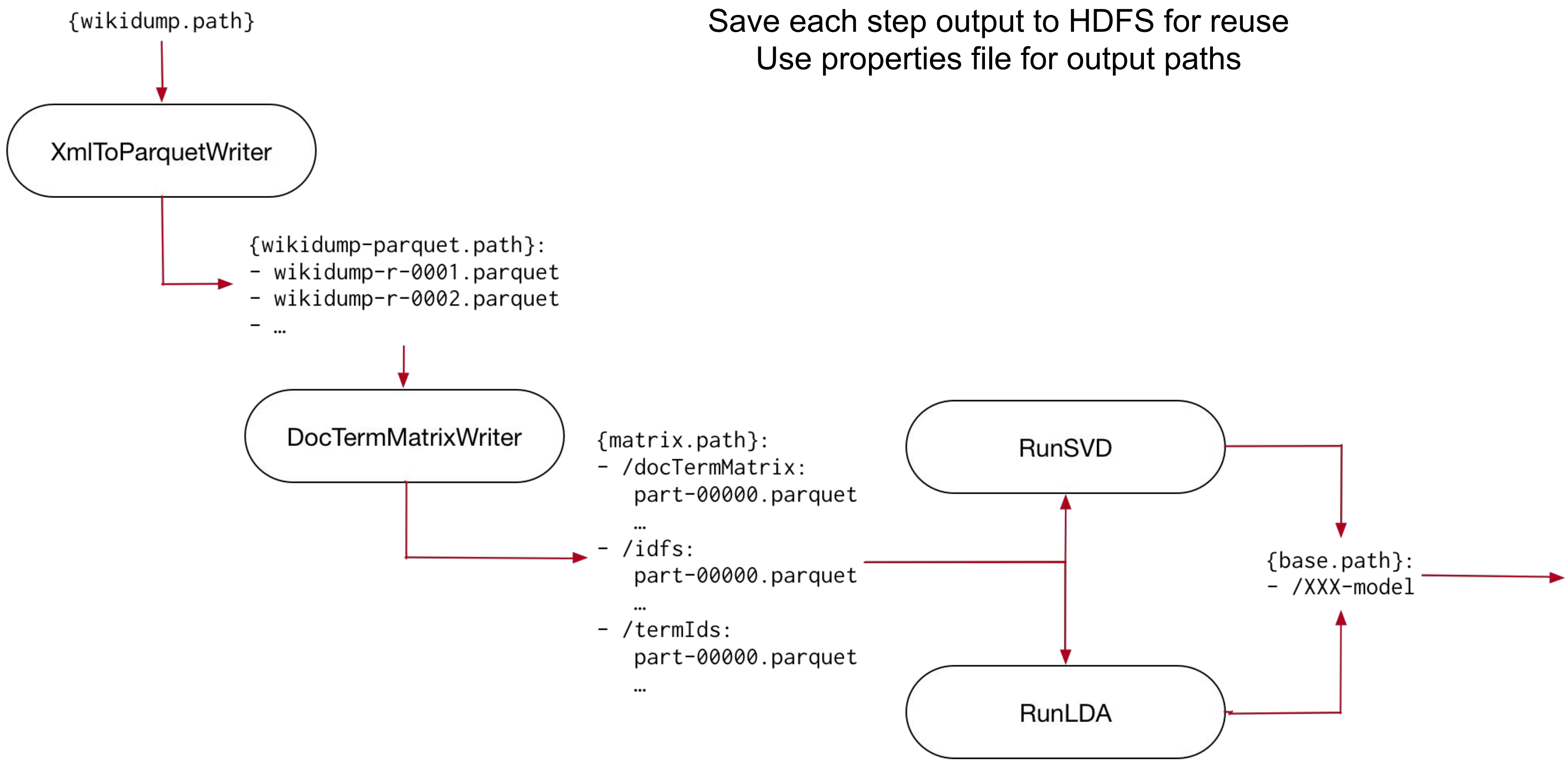
Recreated in a scala worksheet with comments (spark-shell)

Changes from the Book's implementation

Book's	**Changes**
1rst Edition: spark 1.X + RDD	2nd Edition: spark 2.X + SparkQL
Maven with submodules	SBT
Written as one spark program	Broken into steps: (1) XML2Text, (2) TF-IDF, (3) svd, (4) query
Stanford NLP library	spark-corenlp library
Sluggish document IDs tracking	Use `IndexedRowMatrix` to track IDs
Queries in the main program	External class usable in the spark shell
	SVD model serialization

Pipeline / Steps (1)

Save each step output to HDFS for reuse
Use properties file for output paths



Pipeline / Steps (2)

Creating a model

```
spark-submit --class bda.lsa.<package>.Run<model>
   bda-project-lsa-assembly-1.0.jar <args>
```

Using a model

```
spark-shell --jars bda-project-lsa-assembly-1.0.jar
> import bda.lsa
> val data = getData(spark)
> val model = <package>.Run<model>.loadModel(spark)
> val q = <package>.<SVD|<LDA>QueryEngine(model, data)
```

> *Creating a mapping of row IDs to document titles is a little more difficult.*
>
> *We rely on the fact that, if we call [zipWithUniqueId], it will assign the same unique IDs to the transformed rows, as long as the transformations* ***don't change the number of rows or their partitioning****.*

⇒ "hack" working only when all is done in the same spark session

```
SingularValueDecomposition[RowMatrix, Matrix] ⇒
SingularValueDecomposition[IndexedRowMatrix, Matrix] !
```

4.

Our implementation

LDA

Goals

- Run both mllib.LDA and ml.LDA
  - ✓ using the same pipeline and query engine
- Compare:
  - ✓ SVD □ LDA
  - ✓ MLLib □ ML

MLLIB	ML
RDD[(Long, mllib_Vector)]	DataFrame[(.., ml_Vector, ...)]
model.run	model.fit.transform
Default optimizer: EM	Default optimizer: Online
Alpha, Beta, ...	optimiseDocConcentration

- ML much faster at model creation
- ML much easier at querying (but slower?)

LDAQueryEngine

- Describe topics with words
- Describe topics with words and stats
- Top topics for term
- Top topics for documents
- Top documents for topic

5.
Results

Execution times - 4'703'192 docs, 20’000 terms, 1’000 topics

	Jobs	Time
DocTermMatrixWriter	**8**	**3h08**
SVD	**2511**	**48 min**
LDA (mllib) -- maxIter 50	**61**	**5h**
LDA (ml) -- maxIter 50	**?**	**1h27**

Daplab

--master yarn --deploy-mode client --num-executors 10

--driver-memory 20G --executor-memory 15G

SVD with 20'000 words and 1'000 topics

Some relevant topics :

- album, band, bar, song, party
- war, force, use, army, government

And some irrelevant :

- mathbf, film, station, party, frac
- kelly, snake, ira, hat, teen

Top documents for “computer” (score):

- History of IBM (972)
- Computer (688)
- IBM Personal Computer (653)
- Personal computer (630)

LDA (mllib) with 20’000 words and 1’000 topics

Some relevant topics :

- route, expressway, bridge, highway, parkway
- nba, basketball, season, team, coach

And some irrelevant :

- stan, capitol, kenton, melrose, shearer
- turtle, denton, hartley, butte, window

Top topics for “mustard” (score):

- asia, asian, southeast, laos, lao (32’504)
- cook, eat, dish, sweet, cuisine (6’478)
- animal, nature, wild, wildlife, sanctuary (0.6)
- nuclear, accident, missile, weapon, viaduct (0.15)

LDA (mllib) : number of topics

Top topics for batman…

… With k=1’000:

- batman, woodward, général, comics, joker (155’246)
- suit, batman, anime, gundam, cyclone (123’800)
- superman, chinese, qing, lois, ethnic (4’644)

  ...

… With k=200:

- green, big, clark, baker, wayne, batman, kent, lock, guy, superman, bennett, flash, giant, wonder, blake, martha, quinn, lantern, slalom, hal, lois, pipe, comics, brave, justice, joker, atkinson, save, gotham, titans, wally, metropolis, lex, alec, watt, liz, lindsey, robin, shelton, bruce, …

⇒ marvel/superheroes/comics all in the same topic !

DEMO

6.

Conclusion and perspectives

Conclusion

Preprocessing is a super important step

Everything takes a lot of time

SVD and LDA both give interesting results, but have different functionalities and applications

Tuning parameters makes the difference between interesting and completely useless results

Perspectives

Try bigger **vocabulary size**, different number of topics

Try to classify **new documents** with LDA

More **LDA tuning** : parameters *alpha*, *beta* and *k*

More complete and efficient **QueryEngines**

Usage of **Pipelines**

Resources

Our repo

https://github.com/derlin/bda-lsa-project

Our wiki

https://github.com/derlin/bda-lsa-project/wiki

Our website

https://derlin.github.io/bda-lsa-project

Thanks!

Any questions?

Model

```scala
val model =
  new mllib_LDA().
    setK(k).
    setOptimizer("em").
    setMaxIterations(maxIterations).
    setAlpha(alpha).
    setBeta(beta).
    run(corpus.mapValues(_._1)).
   asInstanceOf[mllib_DistributedLDAModel]
```

MLLIB: works with

RDDs[(Long, Vector)]

```scala
val model =
      new ml_LDA().
        setK(k).
        setOptimizer("em").
        setMaxIter(maxIterations).
        setOptimizeDocConcentration(true).
        setFeaturesCol("tfidfVec").
        fit(data.dtm).
       asInstanceOf[ml_DistributedLDAModel]
```

ML: works with

DataFrames[(Long, Vector)]