Broad Match and Homonym Filtering with Stochastic Singular Value Decomposition (SSVD)

Dmitriy Lyubimov[1]

[1]dlyubimov at apache dot org

San Francisco, 2012

Outline

Quote

"We'll never run out of math teachers because they always ***multiply***."

Outline

1. Motivation
   - Search Ad Network Overview
2. Broad Match
   - Building classification score
   - Introducing LSA as a scoring component
3. LSA
   - Step-by-step LSA math
   - Toy LSA Example in R
   - Training with Mahout
   - Stochastic SVD
4. References

Search ad network overview

What is the search ad network?

Search ad network is an open auction for customer intents:

Request for proposal

- A person comes to an auction and announces intent to a crowd of experts with a short simple request.
  - e.g.: “Blue suede shoes” $\mapsto Q = \{"blue", "suede", "shoe"\}$

Some experts' reply:

- Some experts:
  “I bet X amount that I know what you need.
  I am ready to pay it for you to eyeball my proposal.”

Search ad network overview
What is the search ad network?

Search ad network is an open auction for customer intents:

Request for proposal

- A person comes to an auction and announces intent to a crowd of experts with a short simple request.
  - e.g.: "Blue suede shoes" $\mapsto Q = \{"blue", "suede", "shoe"\}$

Some experts' reply:

- Some experts:
  "I bet X amount that I know what you need.
  I am ready to pay it for you to eyeball my proposal."

Search ad network overview

Automated bidding thru a classifier

- Human experts don't bid directly.
- They build *multi-class classifiers* to automate the process:

$$Q \mapsto b \in \{b_1, b_2, ..., b_n, no\ bid\}$$

    - Training is manual via an editorial effort
- Let's simplify: $Q \mapsto b \in \{0, 1\}$ (no match/match)

Search ad network overview

Automated bidding thru a classifier

- Human experts don't bid directly.
- They build *multi-class classifiers* to automate the process:

$$Q \mapsto b \in \{b_1, b_2, ..., b_n, no\ bid\}$$

  - Training is manual via an editorial effort
- Let's simplify: $Q \mapsto b \in \{0, 1\}$ (no match/match)

Search ad network overview

Automated bidding thru a classifier

- Human experts don't bid directly.
- They build *multi-class classifiers* to automate the process:

$$Q \mapsto b \in \{b_1, b_2, \ldots, b_n, no\ bid\}$$

  - Training is manual via an editorial effort

- Let's simplify: $Q \mapsto b \in \{0, 1\}$ (no match/match)

Search ad network overview
Automated bidding thru a classifier

- Human experts don't bid directly.
- They build *multi-class classifiers* to automate the process:

$$Q \mapsto b \in \{b_1, b_2, ..., b_n, \mathit{no\ bid}\}$$

  - Training is manual via an editorial effort
- Let's simplify: $Q \mapsto b \in \{0, 1\}$ (no match/match)

Search ad network overview

Specify classifier rules: exact terms vs. concept a.k.a Broad Match

Bidding on exact terms

kwords: "java", "classes" -coffee

$Q \mapsto 1$: java classes✓; java course✓; java island×; java job×

$Q \mapsto 0$: java coffee✓; I want to learn java×

Bidding on a concept a.k.a *Broad Match*

concept: java classes

$Q \mapsto 1$: java✓|±; java classes✓; java course✓; I want to learn java✓

$Q \mapsto 0$: java coffee✓; java job✓; java island✓

Search ad network overview

Specify classifier rules: exact terms vs. concept a.k.a Broad Match

Bidding on exact terms

kwords: "java", "classes" -coffee

$Q \mapsto 1$: java classes✓; java course✓; java island×; java job×

$Q \mapsto 0$: java coffee✓; I want to learn java×

Bidding on a concept a.k.a *Broad Match*

concept: java classes

$Q \mapsto 1$: java✓|±; java classes✓; java course✓; I want to learn java✓

$Q \mapsto 0$: java coffee✓; java job✓; java island✓

Problems arising from exact term matching

Resolving synonymy and polisemy

- Defining synonymy is hard but possible:
  - tons of keywords: “java classes”, “java course”, “learning java” -coffee [...]
  - but building synonym keyword dictionaries works ok though
- Polysemy is much harder for a human expert to eliminate
  - “corner case” mentality: pockets of homonym creep go undiagnosed and unnoticed;
  - adding synonyms improves recall but hurts precision and multiplies homonym contexts
    - e.g.: java classes, java courses, study java, -coffee, -golf, -room, -office
- Therefore we need a Machine Learning solution for homonym filtering!

Problems arising from exact term matching

Resolving synonymy and polisemy

- Defining synonymy is hard but possible:
  - tons of keywords: "java classes", "java course", "learning java" -coffee [...]
  - but building synonym keyword dictionaries works ok though
- Polysemy is much harder for a human expert to eliminate
  - "corner case" mentality: pockets of homonym creep go undiagnosed and unnoticed;
  - adding synonyms improves recall but hurts precision and multiplies homonym contexts
    - e.g.: java classes, java courses, study java, -coffee, -golf, -room, -office
- Therefore we need a Machine Learning solution for homonym filtering!

Problems arising from exact term matching

Resolving synonymy and polisemy

- Defining synonymy is hard but possible:
  - tons of keywords: "java classes", "java course", "learning java" -coffee [...]
  - but building synonym keyword dictionaries works ok though
- Polysemy is much harder for a human expert to eliminate
  - "corner case" mentality: pockets of homonym creep go undiagnosed and unnoticed;
  - adding synonyms improves recall but hurts precision and multiplies homonym contexts
    - e.g.: java classes, java courses, study java, -coffee, -golf, -room, -office
- Therefore we need a Machine Learning solution for homonym filtering!

Problems arising from exact term matching

Resolving synonymy and polisemy

- Defining synonymy is hard but possible:
  - tons of keywords: "java classes", "java course", "learning java" -coffee [...]
  - but building synonym keyword dictionaries works ok though
- Polysemy is much harder for a human expert to eliminate
  - "corner case" mentality: pockets of homonym creep go undiagnosed and unnoticed;
  - adding synonyms improves recall but hurts precision and multiplies homonym contexts
    - e.g.: java classes, java courses, study java, -coffee, -golf, -room, -office
- Therefore we need a Machine Learning solution for homonym filtering!

Outline

Broad match

classification score: Naive scoring

- initial approach Jaccard coefficient (Q – query tokenset, K – keyword tokenset):

$$J_c = \frac{|Q \cap K|}{|Q \cup K|}$$

  - e.g.:

$$J_c\left(\text{"}java\ class\text{"}, \text{"}java\ class\text{"}\right) = 1.0 \text{ (great!)};$$

$$J_c\left(\text{"}java\ class\text{"}, \text{"}java\ course\text{"}\right) = 0.33;$$

  - but:

$$J_c\left(\text{"}java\ coffee\text{"}, \text{"}java\ course\text{"}\right) = 0.33.$$

- Can we do better??

Broad match
classification score: Naive scoring

- initial approach Jaccard coefficient (Q – query tokenset, K – keyword tokenset):

$$J_c = \frac{|Q \cap K|}{|Q \cup K|}$$

  - e.g.:

$$J_c\left(\text{"}java\ class\text{"}, \text{"}java\ class\text{"}\right) = 1.0\ (\text{great!})\,;$$

$$J_c\left(\text{"}java\ class\text{"}, \text{"}java\ course\text{"}\right) = 0.33;$$

  - but:

$$J_c\left(\text{"}java\ coffee\text{"}, \text{"}java\ course\text{"}\right) = 0.33.$$

- Can we do better??

Outline

Latent Semantic Analysis(LSA) as a Broad Match scoring component

LSA pros

- Fairly well established
  - used in patent searches for a Prior Art
- Solves synonymy to some extent and homonym filtering
- Math is easier than pLSA, LDA, LDA-CVB
  - implementation is easy (on a small scale)
- Manageable editorial effort
  - less tedious than e.g. labeling documents for classification
  - even irrelevant documents are ok as long as they are not a machine-generated ones

Latent Semantic Analysis(LSA) as a Broad Match scoring component

LSA pros

- Fairly well established
  - used in patent searches for a Prior Art
- Solves synonymy to some extent and homonym filtering
- Math is easier than pLSA, LDA, LDA-CVB
  - implementation is easy (on a small scale)
- Manageable editorial effort
  - less tedious than e.g. labeling documents for classification
  - even irrelevant documents are ok as long as they are not a machine-generated ones

Latent Semantic Analysis(LSA) as a Broad Match scoring component

LSA pros

- Fairly well established
  - used in patent searches for a Prior Art
- Solves synonymy to some extent and homonym filtering
- Math is easier than pLSA, LDA, LDA-CVB
  - implementation is easy (on a small scale)
- Manageable editorial effort
  - less tedious than e.g. labeling documents for classification
  - even irrelevant documents are ok as long as they are not a machine-generated ones

Latent Semantic Analysis(LSA) as a Broad Match scoring component

LSA pros

- Fairly well established
    - used in patent searches for a Prior Art
- Solves synonymy to some extent and homonym filtering
- Math is easier than pLSA, LDA, LDA-CVB
    - implementation is easy (on a small scale)
- Manageable editorial effort
    - less tedious than e.g. labeling documents for classification
    - even irrelevant documents are ok as long as they are not a machine-generated ones

Latent Semantic Analysis(LSA) as a Broad Match scoring component

LSA pros

- Fairly well established
  - used in patent searches for a Prior Art
- Solves synonymy to some extent and homonym filtering
- Math is easier than pLSA, LDA, LDA-CVB
  - implementation is easy (on a small scale)
- Manageable editorial effort
  - less tedious than e.g. labeling documents for classification
  - even irrelevant documents are ok as long as they are not a machine-generated ones

Latent Semantic Analysis(LSA)

LSA cons

- Hard to do at scale. But:
  - easier now!
  - Training is a seldom act. Can rent from a cloud.

Latent Semantic Analysis(LSA)
LSA cons

- Hard to do at scale. But:
  - easier now!
  - Training is a seldom act. Can rent from a cloud.

Outline

LSA Math

let's take a closer look at LSA Math: bag-of-words

- Bag-of-words model:
  - corpus is a set of documents $D = \{d_i\}$;
  - document is a set of terms: $T_d = \{t_{d,i}\}$

LSA Math

Training Step 1 - Forming input: TF-IDF

- TF (term frequency) for $t, d : t \in T_d$

$$\operatorname{tf}(d, t) = \frac{\operatorname{count}(t, d)}{\max_{w \in T_d}(\operatorname{count}(w, d))}$$

- IDF (inverse document frequency)

$$\operatorname{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

- TF-IDF

$$\operatorname{tfidf}(t, d, D) = \operatorname{tf}(d, t) \cdot \operatorname{idf}(t, D)$$

LSA Math

Training Step 1 - Forming input: TF-IDF

- TF (term frequency) for $t, d : t \in T_d$

$$\mathrm{tf}(d, t) = \frac{\mathrm{count}(t, d)}{\max_{w \in T_d}(\mathrm{count}(w, d))}$$

- IDF (inverse document frequency)

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

- TF-IDF

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(d, t) \cdot \mathrm{idf}(t, D)$$

LSA Math

Training Step 1 - Forming input: TF-IDF

- TF (term frequency) for $t, d : t \in T_d$

$$\mathrm{tf}(d, t) = \frac{\mathrm{count}(t, d)}{\max_{w \in T_d}(\mathrm{count}(w, d))}$$

- IDF (inverse document frequency)

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

- TF-IDF

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(d, t) \cdot \mathrm{idf}(t, D)$$

LSA Math

Training: input matrix

- Input is *sparse num-docs*$\times$ *num-terms* matrix $\mathbf{A}$ such that

$$a_{i,j} = \mathrm{tfidf}\left(j, i, D\right).$$

  - rows correspond to documents and columns correspond to terms and values correspond to tf-idf value of such

LSA Math
Training: SVD

- reduced k-rank SVD

$$\mathbf{A} \approx \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\top},$$

  - rows of **U** correspond to documents
  - rows of **V** correspond to terms

LSA Math

Runtime: folding-in queries (or keywords)

- $\tilde{\mathbf{c}}_q$ is new document observation (query) such that $\forall t_i \in T$:

$$\tilde{c}_{q,i} = \begin{cases} 1, & t_i \in Q; \\ 0, & t_i \notin Q. \end{cases}$$

- query fold-in to document space of $\mathbf{U}$:

$$\tilde{\mathbf{u}}_q = \left(\mathbf{\Sigma}^{-1}\mathbf{V}^\top\right) \cdot \left[\tilde{\mathbf{c}}_q \circ \mathbf{idf}\left(\cdot\right)\right],$$

- cosine similarity of two queries (or a keyword and a query)

$$\mathrm{sim}\left(\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2\right) = \cos\Theta = \frac{\tilde{\mathbf{u}}_1 \cdot \tilde{\mathbf{u}}_2}{\left|\tilde{\mathbf{u}}_1\right| \left|\tilde{\mathbf{u}}_2\right|}$$

- final classification score takes $J_c\left(\mathbf{q}, \mathbf{k}\right)$ and $\mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$ as predictors

LSA Math

Runtime: folding-in queries (or keywords)

- $\tilde{\mathbf{c}}_q$ is new document observation (query) such that $\forall t_i \in T$:

$$\tilde{c}_{q,i} = \begin{cases} 1, & t_i \in Q; \\ 0, & t_i \notin Q. \end{cases}$$

- query fold-in to document space of $\mathbf{U}$:

$$\tilde{\mathbf{u}}_q = \left(\mathbf{\Sigma}^{-1}\mathbf{V}^\top\right) \cdot \left[\tilde{\mathbf{c}}_q \circ \mathbf{idf}\left(\cdot\right)\right],$$

- cosine similarity of two queries (or a keyword and a query)

$$\mathrm{sim}\left(\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2\right) = \cos\Theta = \frac{\tilde{\mathbf{u}}_1 \cdot \tilde{\mathbf{u}}_2}{\left|\tilde{\mathbf{u}}_1\right| \left|\tilde{\mathbf{u}}_2\right|}$$

- final classification score takes $J_c\left(\mathbf{q}, \mathbf{k}\right)$ and $\mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$ as predictors

LSA Math

Runtime: folding-in queries (or keywords)

- $\tilde{\mathbf{c}}_q$ is new document observation (query) such that $\forall t_i \in T$:

$$\tilde{c}_{q,i} = \begin{cases} 1, & t_i \in Q; \\ 0, & t_i \notin Q. \end{cases}$$

- query fold-in to document space of $\mathbf{U}$:

$$\tilde{\mathbf{u}}_q = \left(\mathbf{\Sigma}^{-1}\mathbf{V}^{\top}\right) \cdot \left[\tilde{\mathbf{c}}_q \circ \mathbf{idf}\left(\cdot\right)\right],$$

- cosine similarity of two queries (or a keyword and a query)

$$\mathrm{sim}\left(\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2\right) = \cos\Theta = \frac{\tilde{\mathbf{u}}_1 \cdot \tilde{\mathbf{u}}_2}{\left|\tilde{\mathbf{u}}_1\right| \left|\tilde{\mathbf{u}}_2\right|}$$

- final classification score takes $J_c\left(\mathbf{q}, \mathbf{k}\right)$ and $\mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$ as predictors

LSA Math

Runtime: folding-in queries (or keywords)

- $\tilde{\mathbf{c}}_q$ is new document observation (query) such that $\forall t_i \in T$:

$$\tilde{c}_{q,i} = \begin{cases} 1, & t_i \in Q; \\ 0, & t_i \notin Q. \end{cases}$$

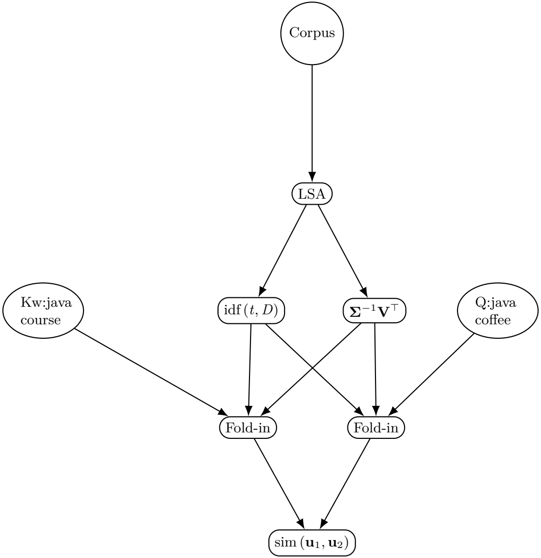
- query fold-in to document space of $\mathbf{U}$:

$$\tilde{\mathbf{u}}_q = \left(\mathbf{\Sigma}^{-1}\mathbf{V}^\top\right) \cdot \left[\tilde{\mathbf{c}}_q \circ \mathbf{idf}\left(\cdot\right)\right],$$

- cosine similarity of two queries (or a keyword and a query)

$$\mathrm{sim}\left(\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2\right) = \cos\Theta = \frac{\tilde{\mathbf{u}}_1 \cdot \tilde{\mathbf{u}}_2}{\left|\tilde{\mathbf{u}}_1\right| \left|\tilde{\mathbf{u}}_2\right|}$$

- final classification score takes $J_c\left(\mathbf{q}, \mathbf{k}\right)$ and $\mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$ as predictors


Corpus
LSA
Kw:java course
idf (t, D)
$\mathbf{\Sigma}^{-1}\mathbf{V}^{\top}$
Q:java coffee
Fold-in
Fold-in
sim $(\mathbf{u}_1, \mathbf{u}_2)$

Outline

1. Motivation
   - Search Ad Network Overview
2. Broad Match
   - Building classification score
   - Introducing LSA as a scoring component
3. LSA
   - Step-by-step LSA math
   - Toy LSA Example in R
   - Training with Mahout
   - Stochastic SVD
4. References

Toy LSA example in R

text1.txt - java programming course

This course is an introduction to software engineering, using the Java™ programming language.
The class cover concepts useful to 6.005. Students will learn the fundamentals of Java programming.
The focus is on developing high quality, working software that solves real problems.
The course is designed for students with some programming experience, but
if you have none and are motivated you will do fine. Students who have taken 6.005 should
not take this course. Each class is composed of one hour of lecture and one hour of assisted lab work.
This course is offered during the Independent Activities ...

LSA example in R (contd.)

text2.txt - java coffee

Java coffee refers to coffee beans produced in the Indonesian island of Java. In some countries, including the United States, "Java" can refer to coffee. The Indonesian phrase Kopi Jawa refers not only to the origin of the coffee, but is used to distinguish a style of strong, black, very sweet coffee.

LSA example in R (contd.)

text3.txt - another topic

The easiest way is to use your phone as just a phone. There is no shortage of old-fashioned, flip-phone plans that can keep your bill south of $50, provided you don't end up receiving a bunch of unexpected text messages. If you want a phone-only phone, you might want to look away from the major carriers, however, which are now focused on lucrative data-hogging customers. If you wander into a local Verizon store, for example, you are likely to find only one or two basic phone options. Smaller carriers and pre-paid services are the right choice here. Those who want cellphones only for emergencies and pay for only the minutes they use can keep their bills down to $20 or even $10 per month. Ditto for those who just don't want to have their face buried in a smartphone for hours per day.

```r
# parse file and return vector of term frequences
computeTF <- function(fname) {
    f <- file(fname, "r")
    on.exit(close(f), T)

    # parse into character vector and cleanup
    words <- tolower(unlist(strsplit(readLines(f), "[^[:alnum:]]+")))
    words <- grep("^[^[:digit:]]+$", words, value = T)

    # word count
    wc <- tapply(words, words, length)

    # term frequency
    wc/max(wc)
}
```

```r
# get named list of term frequences and convert it into a
# tf-idf matrix
computeTFIDF <- function(tflist) {

    # compile idf af all terms: terms
    terms <- unlist(lapply(tflist, function(x) names(x)))
    # doc count
    docCount <- length(tflist)

    idf <- sapply(terms, function(term) {
        dfreq <- sum(sapply(tflist, function(x) if (is.na(x[term])) 0 e
        log(docCount/dfreq)
    })
    names(idf) <- terms

    m <- matrix(0, nrow = docCount, ncol = length(terms), dimnames = li
        terms))

    sapply(names(tflist), function(dn) {
        d <- tflist[[dn]]
        m[dn, names(d)] <<- d * idf[names(d)]
```

```
foldin <- function(query) {
    # parse query
    words <- unlist(tolower(unlist(strsplit(query, "[^[:alnum:]]+"))))
    words <- grep("^[^[:digit:]]+$", words, value = T)

    words <- levels(as.factor(words[words %in% rownames(vsigma)]))

    # for simplicity, we assume term frequency = 1 in queries
    termv <- idf[words]
    names(termv) <- words

    # fold-in of the new observation
    t(termv) %*% vsigma[words, , drop = F]
}


cosineSim <- function(vec1, vec2) (sum(vec1 * vec2))/sqrt(sum(vec1^2) *
    sum(vec2^2))
```

Compute TF-IDF input matrix

```
files <- c("text1.txt", "text2.txt", "text3.txt")

# compute all term frequences for all documents
a <- lapply(files, function(x) computeTF(x))
names(a) <- files

# compute tfidf matrix
a <- computeTFIDF(a)
names(a)

## [1] "idf" "m"
```

TF-IDFs of some terms? (a vertical block of the input)

```
a$m[, c("java", "class", "programming", "coffee", "the")]

##                java  class programming coffee the
## text1.txt 0.04505 0.2441      0.3662  0.000   0
## text2.txt 0.24328 0.0000      0.0000  1.099   0
## text3.txt 0.00000 0.0000      0.0000  0.000   0
```

Actually compute SVD, fold-in components and save them

```
# LSA
s <- svd(a$m)
vsigma <- s$v %*% diag(1/s$d)
rownames(vsigma) <- colnames(a$m)
u <- s$u
rownames(u) <- rownames(a$m)
rm(s)

# save idf as well for fold-in
idf <- a$idf
rm(a)
```

fold-in some queries or keywords

```
jclasses <- foldin("java class")
jclasses

##           [,1]    [,2]   [,3]
## [1,] 0.00164 0.03998 -0.164


jprog <- foldin("java programming")
jcourse <- foldin("java course")

jcoffee <- foldin("java coffee")
jcoffee

##            [,1]   [,2]     [,3]
## [1,] 0.003309 0.4966 0.006854
```

nice synonymy and polisemy at work!

```
cosineSim(jclasses, jprog)

## [1] 0.9975

cosineSim(jclasses, jcourse)

## [1] 0.9943

cosineSim(jclasses, jcoffee)

## [1] 0.2235
```

we can even measure relevance to the original documents

```
cosineSim(u["text1.txt", ], jcourse)

## [1] 0.9939

cosineSim(u["text2.txt", ], jcourse)

## [1] 0.1088

cosineSim(u["text3.txt", ], jcourse)

## [1] -0.02029
```

Outline

Training with Mahout

- `seq2sparse`: bigram/trigram analysis to improve assumptions of bag-of-words model
- `ssvd`: stochastic SVD

Outline

1. Motivation
   - Search Ad Network Overview
2. Broad Match
   - Building classification score
   - Introducing LSA as a scoring component
3. LSA
   - Step-by-step LSA math
   - Toy LSA Example in R
   - Training with Mahout
   - Stochastic SVD
4. References

Base modified algorithm with power iterations

Single-threaded version, step-by-step

Given an $m \times n$ matrix $\mathbf{A}$, a target rank $k \in \mathbb{N}_1$, an oversampling parameter $p \in \mathbb{N}_1$, and the number of additional power iterations $q \in \mathbb{N}_0$, this procedure computes an k-rank reduced SVD $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$(some notations are adjusted):

- Create seed for random $n \times (k+p)$ matrix $\boldsymbol{\Omega}$.
- $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, $\mathbf{Y} \in \mathbb{R}^{m \times (k+p)}$.
- Column-orthonormalize $\mathbf{Y} \to \mathbf{Q}$ by computing thin decomposition $\mathbf{Y} = \mathbf{QR}$. Also, $\mathbf{Q} \in \mathbb{R}^{m \times (k+p)}$, $\mathbf{R} \in \mathbb{R}^{(k+p) \times (k+p)}$. I denote this as $\mathbf{Q} = \mathrm{qr}\left(\mathbf{Y}\right).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^\top \mathbf{A} : \ \mathbf{B} \in \mathbb{R}^{(k+p) \times n}$.
- If $q > 0$ repeat: for $i = 1..q$: $\mathbf{B}_i^\top = \mathbf{A}^\top \mathrm{qr}\left(\mathbf{A}\mathbf{B}_{i-1}^\top\right).\mathbf{Q}$ (power iterations step)

Base modified algorithm with power iterations

Single-threaded version, step-by-step

Given an $m \times n$ matrix $\mathbf{A}$, a target rank $k \in \mathbb{N}_1$, an oversampling parameter $p \in \mathbb{N}_1$, and the number of additional power iterations $q \in \mathbb{N}_0$, this procedure computes an k-rank reduced SVD $\mathbf{A} = \mathbf{U\Sigma V}^\top$(some notations are adjusted):

- Create seed for random $n \times (k+p)$ matrix $\mathbf{\Omega}$.
- $\mathbf{Y} = \mathbf{A\Omega}$, $\mathbf{Y} \in \mathbb{R}^{m\times(k+p)}$.
- Column-orthonormalize $\mathbf{Y} \to \mathbf{Q}$ by computing thin decomposition $\mathbf{Y} = \mathbf{QR}$. Also, $\mathbf{Q} \in \mathbb{R}^{m\times(k+p)}$, $\mathbf{R} \in \mathbb{R}^{(k+p)\times(k+p)}$. I denote this as $\mathbf{Q} = \mathrm{qr}\left(\mathbf{Y}\right).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^\top\mathbf{A}: \ \mathbf{B} \in \mathbb{R}^{(k+p)\times n}$.
- If $q > 0$ repeat: for $i = 1..q$: $\mathbf{B}_i^\top = \mathbf{A}^\top \mathrm{qr}\left(\mathbf{AB}_{i-1}^\top\right).\mathbf{Q}$ (power iterations step)

Base modified algorithm with power iterations

Single-threaded version, step-by-step

Given an $m \times n$ matrix $\mathbf{A}$, a target rank $k \in \mathbb{N}_1$, an oversampling parameter $p \in \mathbb{N}_1$, and the number of additional power iterations $q \in \mathbb{N}_0$, this procedure computes an k-rank reduced SVD $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\top}$(some notations are adjusted):

- Create seed for random $n \times (k + p)$ matrix $\boldsymbol{\Omega}$.
- $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, $\mathbf{Y} \in \mathbb{R}^{m \times (k+p)}$.
- Column-orthonormalize $\mathbf{Y} \to \mathbf{Q}$ by computing thin decomposition $\mathbf{Y} = \mathbf{QR}$. Also, $\mathbf{Q} \in \mathbb{R}^{m \times (k+p)}$, $\mathbf{R} \in \mathbb{R}^{(k+p) \times (k+p)}$. I denote this as $\mathbf{Q} = \mathrm{qr}\left(\mathbf{Y}\right).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^{\top}\mathbf{A} : \ \mathbf{B} \in \mathbb{R}^{(k+p) \times n}$.
- If $q > 0$ repeat: for $i = 1..q$: $\mathbf{B}_i^{\top} = \mathbf{A}^{\top}\mathrm{qr}\left(\mathbf{A}\mathbf{B}_{i-1}^{\top}\right).\mathbf{Q}$ (power iterations step)

Base modified algorithm with power iterations

Single-threaded version, step-by-step

Given an $m \times n$ matrix $\mathbf{A}$, a target rank $k \in \mathbb{N}_1$, an oversampling parameter $p \in \mathbb{N}_1$, and the number of additional power iterations $q \in \mathbb{N}_0$, this procedure computes an k-rank reduced SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$(some notations are adjusted):

- Create seed for random $n \times (k+p)$ matrix $\mathbf{\Omega}$.
- $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$, $\mathbf{Y} \in \mathbb{R}^{m\times(k+p)}$.
- Column-orthonormalize $\mathbf{Y} \to \mathbf{Q}$ by computing thin decomposition $\mathbf{Y} = \mathbf{QR}$. Also, $\mathbf{Q} \in \mathbb{R}^{m\times(k+p)}$, $\mathbf{R} \in \mathbb{R}^{(k+p)\times(k+p)}$. I denote this as $\mathbf{Q} = \mathrm{qr}\left(\mathbf{Y}\right).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^\top\mathbf{A}: \ \mathbf{B} \in \mathbb{R}^{(k+p)\times n}$.
- If $q > 0$ repeat: for $i = 1..q$: $\mathbf{B}_i^\top = \mathbf{A}^\top\mathrm{qr}\left(\mathbf{A}\mathbf{B}_{i-1}^\top\right).\mathbf{Q}$ (power iterations step)

Base modified algorithm with power iterations

Single-threaded version, step-by-step

Given an $m \times n$ matrix $\mathbf{A}$, a target rank $k \in \mathbb{N}_1$, an oversampling parameter $p \in \mathbb{N}_1$, and the number of additional power iterations $q \in \mathbb{N}_0$, this procedure computes an k-rank reduced SVD $\mathbf{A} = \mathbf{U\Sigma V}^\top$(some notations are adjusted):

- Create seed for random $n \times (k+p)$ matrix $\boldsymbol{\Omega}$.
- $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, $\mathbf{Y} \in \mathbb{R}^{m\times(k+p)}$.
- Column-orthonormalize $\mathbf{Y} \rightarrow \mathbf{Q}$ by computing thin decomposition $\mathbf{Y} = \mathbf{QR}$. Also, $\mathbf{Q} \in \mathbb{R}^{m\times(k+p)}$, $\mathbf{R} \in \mathbb{R}^{(k+p)\times(k+p)}$. I denote this as $\mathbf{Q} = \mathrm{qr}\left(\mathbf{Y}\right).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^\top\mathbf{A}: \; \mathbf{B} \in \mathbb{R}^{(k+p)\times n}$.
- If $q > 0$ repeat: for $i = 1..q$: $\mathbf{B}_i^\top = \mathbf{A}^\top \mathrm{qr}\left(\mathbf{AB}_{i-1}^\top\right).\mathbf{Q}$ (power iterations step)

Base modified algorithm with power iterations

Single-threaded version, step-by-step(contd)

- Compute Eigensolution of a small Hermitian $\mathbf{B}_q\mathbf{B}_q^\top = \widehat{\mathbf{U}}\mathbf{\Lambda}\widehat{\mathbf{U}}^\top$. $\mathbf{B}_q\mathbf{B}_q^\top \in \mathbb{R}^{(k+p)\times(k+p)}$.
- Singular values $\mathbf{\Sigma} = \mathbf{\Lambda}^{0.5}$, or, in other words, $s_i = \sqrt{\sigma_i}$.
- If needed, compute $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
- If needed, compute $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}$.
  - and bunch of other options for the output (see doc)

Base modified algorithm with power iterations

Single-threaded version, step-by-step(contd)

- Compute Eigensolution of a small Hermitian $\mathbf{B}_q\mathbf{B}_q^\top = \widehat{\mathbf{U}}\mathbf{\Lambda}\widehat{\mathbf{U}}^\top$. $\mathbf{B}_q\mathbf{B}_q^\top \in \mathbb{R}^{(k+p)\times(k+p)}$.
- Singular values $\mathbf{\Sigma} = \mathbf{\Lambda}^{0.5}$, or, in other words, $s_i = \sqrt{\sigma_i}$.
- If needed, compute $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
- If needed, compute $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}$.
  - and bunch of other options for the output (see doc)

Base modified algorithm with power iterations

Single-threaded version, step-by-step(contd)

- Compute Eigensolution of a small Hermitian $\mathbf{B}_q\mathbf{B}_q^\top = \widehat{\mathbf{U}}\mathbf{\Lambda}\widehat{\mathbf{U}}^\top$. $\mathbf{B}_q\mathbf{B}_q^\top \in \mathbb{R}^{(k+p)\times(k+p)}$.
- Singular values $\mathbf{\Sigma} = \mathbf{\Lambda}^{0.5}$, or, in other words, $s_i = \sqrt{\sigma_i}$.
- If needed, compute $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
- If needed, compute $\mathbf{V} = \mathbf{B}_q^\top \hat{\mathbf{U}} \mathbf{\Sigma}^{-1}$.
  - and bunch of other options for the output (see doc)

Base modified algorithm with power iterations

Single-threaded version, step-by-step(contd)

- Compute Eigensolution of a small Hermitian $\mathbf{B}_q\mathbf{B}_q^\top = \widehat{\mathbf{U}}\boldsymbol{\Lambda}\widehat{\mathbf{U}}^\top$. $\mathbf{B}_q\mathbf{B}_q^\top \in \mathbb{R}^{(k+p)\times(k+p)}$.
- Singular values $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}^{0.5}$, or, in other words, $s_i = \sqrt{\sigma_i}$.
- If needed, compute $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
- If needed, compute $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\boldsymbol{\Sigma}^{-1}$.
    - and bunch of other options for the output (see doc)

Base modified algorithm with power iterations

Single-threaded version, step-by-step(contd)

- Compute Eigensolution of a small Hermitian $\mathbf{B}_q\mathbf{B}_q^\top = \widehat{\mathbf{U}}\mathbf{\Lambda}\widehat{\mathbf{U}}^\top$. $\mathbf{B}_q\mathbf{B}_q^\top \in \mathbb{R}^{(k+p)\times(k+p)}$.
- Singular values $\mathbf{\Sigma} = \mathbf{\Lambda}^{0.5}$, or, in other words, $s_i = \sqrt{\sigma_i}$.
- If needed, compute $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
- If needed, compute $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}$.
    - and bunch of other options for the output (see doc)

Characteristics of SSVD

Precision

- Needs a good decay of the spectrum
  - White noise precision is poor compared to optimal solution
  - Not a tremendous problem with real life problems such as LSA
- Easier estimate of error bound
  - expressed in terms of $\frac{\sigma_{k+p}}{\sigma_1}$
- precision with q=1 is very good,
  - with q=2 visually indistinguishable from that of Lanczos solver (at the scale where Lanczos still works)

Characteristics of SSVD

Precision

- Needs a good decay of the spectrum
  - White noise precision is poor compared to optimal solution
  - Not a tremendous problem with real life problems such as LSA
- Easier estimate of error bound
  - expressed in terms of $\frac{\sigma_{k+p}}{\sigma_1}$
- precision with q=1 is very good,
  - with q=2 visually indistinguishable from that of Lanczos solver (at the scale where Lanczos still works)

Characteristics of SSVD

Precision

- Needs a good decay of the spectrum
  - White noise precision is poor compared to optimal solution
  - Not a tremendous problem with real life problems such as LSA
- Easier estimate of error bound
  - expressed in terms of $\frac{\sigma_{k+p}}{\sigma_1}$
- precision with q=1 is very good,
  - with q=2 visually indistinguishable from that of Lanczos solver (at the scale where Lanczos still works)

Characteristics of SSVD (contd.)

Performance

- Fixed amount of MR jobs : $N = 3 + 2q$
  - Some of the jobs are map-only
- options allow to adjust trade-off between speed vs. precision
- running time is not absolutely linearly scalable to the input size:
  - task time $\propto\sim (k+p)^{1.5}$
- Takes on bigger datasets than Lanczos without giving up
  - largest experiment known to me on an input of a little under 1Tb
  - see experiments staged by Nathan in his dissertation

Characteristics of SSVD (contd.)

Performance

- Fixed amount of MR jobs : $N = 3 + 2q$
  - Some of the jobs are map-only
- options allow to adjust trade-off between speed vs. precision
- running time is not absolutely linearly scalable to the input size:
  - task time $\propto\sim (k+p)^{1.5}$
- Takes on bigger datasets than Lanczos without giving up
  - largest experiment known to me on an input of a little under 1Tb
  - see experiments staged by Nathan in his dissertation

Characteristics of SSVD (contd.)

Performance

- Fixed amount of MR jobs : $N = 3 + 2q$
  - Some of the jobs are map-only
- options allow to adjust trade-off between speed vs. precision
- running time is not absolutely linearly scalable to the input size:
  - task time $\propto\sim (k+p)^{1.5}$
- Takes on bigger datasets than Lanczos without giving up
  - largest experiment known to me on an input of a little under 1Tb
  - see experiments staged by Nathan in his dissertation

Characteristics of SSVD (contd.)

Performance

- Fixed amount of MR jobs : $N = 3 + 2q$
  - Some of the jobs are map-only
- options allow to adjust trade-off between speed vs. precision
- running time is not absolutely linearly scalable to the input size:
  - task time $\propto\sim (k+p)^{1.5}$
- Takes on bigger datasets than Lanczos without giving up
  - largest experiment known to me on an input of a little under 1Tb
  - see experiments staged by Nathan in his dissertation

Characteristics of SSVD (contd.)

miscellaneous notes

- Known bottleneck is matrix multiplication of $\mathbf{A}\mathbf{B}^\top$ step,
  - further remedies could be applicable
- QR of $\mathbf{B}_0$ pipeline could be replaced with a Cholesky decomposition trick
- Presents interesting additional optimization opportunities for massive scale PCA
- Applicable to LSA and PCA, not really applicable to recommenders

References

- usage information :

https://cwiki.apache.org/confluence/display/MAHOUT/Stochastic+Singular+Value+Decomposition

- N. Halko, et.al. "Funding structure with randomness..."
- Working notes

https://github.com/dlyubimov/mahout-commits/tree/ssvd-docs

- N. Halko's dissertation
- Blog discusses parallelization of some components

Thank you!