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

Dmitriy Lyubimov[1]

[1]dlyubimov at apache dot org

San Francisco, 2012

# Outline

# Outline

1. Motivation
   - Broad match problems
2. Latent Semantic Analysis
   - Introducing LSA
   - Toy LSA Example in R
   - Stochastic SVD in Mahout
3. References

# Search advertisments

- Ad campaign keyword matching types: exact, phrase, broad, negative.
  - broad: *java programming classes*
  - phrase: *[java programming classes]*
  - exact: *"java programming classes"*
  - negative: *-coffee*
- User query
  - *Java coffee beans*
- Matching problem: which keyword - ? Matching quality - ?

## Search advertisments

- Ad campaign keyword matching types: exact, phrase, broad, negative.
  - broad: *java programming classes*
  - phrase: *[java programming classes]*
  - exact: *"java programming classes"*
  - negative: *-coffee*
- User query
  - *Java coffee beans*
- Matching problem: which keyword - ? Matching quality - ?

# Search advertisments

- Ad campaign keyword matching types: exact, phrase, broad, negative.
    - broad: *java programming classes*
    - phrase: *[java programming classes]*
    - exact: *"java programming classes"*
    - negative: *-coffee*
- User query
    - *Java coffee beans*
- Matching problem: which keyword - ? Matching quality - ?

# Search advertisments

- Ad campaign keyword matching types: exact, phrase, broad, negative.
  - broad: *java programming classes*
  - phrase: *[java programming classes]*
  - exact: *"java programming classes"*
  - negative: *-coffee*
- User query
  - *Java coffee beans*
- Matching problem: which keyword - ? Matching quality - ?

# Broad match
## Importance

- Promote coverage
  - Even large network struggles with coverage around 15%
- Convenient if it “does the right thing”
  - reduces editorial effort of exact and phrase keywords

# Broad match
## Importance

- Promote coverage
  - Even large network struggles with coverage around 15%
- Convenient if it "does the right thing"
  - reduces editorial effort of exact and phrase keywords

# Broad match

## Naive score

- Positive broad match candidate: all normalized tokens present in the query;
- initial match quality with 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:

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

# Broad match

## Naive score

- Positive broad match candidate: all normalized tokens present in the query;
- initial match quality with 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:

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

# Broad match
## Naive score

- Positive broad match candidate: all normalized tokens present in the query;
- initial match quality with Jaccard coefficient ($Q$ – query tokenset, $K$ – keyword tokenset):

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

  - e.g.:

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

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

  - but:

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

# Broad match

## Synonymy

- We can do better if we recognize synonyms vs. everything else.
- ML can handle synonymy, but
  - Exploration: keyword editorial effort is more precise than massive semantic analysis given same editorial effort
  - synonymous keywords ("java class", "java course", "java study" etc.), but omit non-targeting words
    - works ok for a startup and sufficient editorial effort.

# Broad match
## Synonymy

- We can do better if we recognize synonyms vs. everything else.
- ML can handle synonymy, but
  - Exploration: keyword editorial effort is more precise than massive semantic analysis given same editorial effort
  - synonymous keywords ("java class", "java course", "java study" etc.), but omit non-targeting words
    - works ok for a startup and sufficient editorial effort.

# Broad match
## Synonymy

- We can do better if we recognize synonyms vs. everything else.
- ML can handle synonymy, but
  - Exploration: keyword editorial effort is more precise than massive semantic analysis given same editorial effort
  - synonymous keywords ("java class", "java course", "java study" etc.), but omit non-targeting words
    - works ok for a startup and sufficient editorial effort.

# Broad match
## Synonymy

- We can do better if we recognize synonyms vs. everything else.
- ML can handle synonymy, but
  - Exploration: keyword editorial effort is more precise than massive semantic analysis given same editorial effort
  - synonymous keywords (“java class”, “java course”, “java study” etc.), but omit non-targeting words
    - works ok for a startup and sufficient editorial effort.

# Homonym filtering

## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
  - more readily name what they want vs. what they don't want?
  - corner case mentality?
  - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
  - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Homonym filtering

## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
  - more readily name what they want vs. what they don't want?
  - corner case mentality?
  - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
  - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Homonym filtering
## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
  - more readily name what they want vs. what they don't want?
  - corner case mentality?
  - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
  - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Homonym filtering

## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
    - more readily name what they want vs. what they don't want?
    - corner case mentality?
    - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
    - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Homonym filtering

## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
  - more readily name what they want vs. what they don't want?
  - corner case mentality?
  - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
  - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Homonym filtering

## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
  - more readily name what they want vs. what they don't want?
  - corner case mentality?
  - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
  - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Homonym filtering

## why it is a problem?

- Exploration: human actors are worse at filtering all possible homonyms than expanding synonyms
  - more readily name what they want vs. what they don't want?
  - corner case mentality?
  - not readily obvious, so smaller pockets of performance are inevitably going unnoticed
- Problem exacerbates as more synonyms introduced: more contexts to be filtered out
  - example: java class, java course, java study, -coffee, -golf, -room
- Bottom line: ML help is needed, minimum editorial efforts (3 engineers, 0 editors!)

# Outline

# Latent Semantic Analysis(LSA)

## pros

- Fairly well established
- Great subjective precision and recall out of the door
- Used in patent searches for a Prior Art
- Math is easier than pLSA, LDA, LDA-CVB (at least for me)
  - Implementation is easy (on a small scale)
- Manageable editorial effort
  - less than classification labeling
  - ok as long as not machine-generated

# Latent Semantic Analysis(LSA)

## pros

- Fairly well established
- Great subjective precision and recall out of the door
- Used in patent searches for a Prior Art
- Math is easier than pLSA, LDA, LDA-CVB (at least for me)
  - Implementation is easy (on a small scale)
- Manageable editorial effort
  - less than classification labeling
  - ok as long as not machine-generated

# Latent Semantic Analysis(LSA)

## pros

- Fairly well established
- Great subjective precision and recall out of the door
- Used in patent searches for a Prior Art
- Math is easier than pLSA, LDA, LDA-CVB (at least for me)
  - Implementation is easy (on a small scale)
- Manageable editorial effort
  - less than classification labeling
  - ok as long as not machine-generated

# Latent Semantic Analysis(LSA)

## pros

- Fairly well established
- Great subjective precision and recall out of the door
- Used in patent searches for a Prior Art
- Math is easier than pLSA, LDA, LDA-CVB (at least for me)
  - Implementation is easy (on a small scale)
- Manageable editorial effort
  - less than classification labeling
  - ok as long as not machine-generated

# Latent Semantic Analysis(LSA)

## pros

- Fairly well established
- Great subjective precision and recall out of the door
- Used in patent searches for a Prior Art
- Math is easier than pLSA, LDA, LDA-CVB (at least for me)
  - Implementation is easy (on a small scale)
- Manageable editorial effort
  - less than classification labeling
  - ok as long as not machine-generated

# Latent Semantic Analysis(LSA)

## pros

- Fairly well established
- Great subjective precision and recall out of the door
- Used in patent searches for a Prior Art
- Math is easier than pLSA, LDA, LDA-CVB (at least for me)
  - Implementation is easy (on a small scale)
- Manageable editorial effort
  - less than classification labeling
  - ok as long as not machine-generated

# Latent Semantic Analysis(LSA)

## cons

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

# Latent Semantic Analysis(LSA)

## cons

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

# Latent Semantic Analysis(LSA)

## cons

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

# LSA Math

- document set $D = \{d_i\}$
- bag-of-words: term set in a document $d$: $T_d = \{t_{d,i}\}$
- 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

- document set $D = \{d_i\}$
- bag-of-words: term set in a document $d$: $T_d = \{t_{d,i}\}$
- 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

- document set $D = \{d_i\}$
- bag-of-words: term set in a document $d$: $T_d = \{t_{d,i}\}$
- 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

- document set $D = \{d_i\}$
- bag-of-words: term set in a document $d$: $T_d = \{t_{d,i}\}$
- 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

- document set $D = \{d_i\}$
- bag-of-words: term set in a document $d$: $T_d = \{t_{d,i}\}$
- TF (term frequency) for $t, d : t \in T_d$

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

- IDF (inverse document frequency)

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

- TF-IDF

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

# LSA Math (contd.)

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

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

- (so rows correspond to documents and columns correspond to terms).

# LSA Math (contd.)

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

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

  - (so rows correspond to documents and columns correspond to terms).

# LSA Math (contd.)

- reduced k-rank SVD

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

  - rows of $\mathbf{U}$ correspond to documents
  - rows of $\mathbf{V}$ correspond to terms

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

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

where $\tilde{\mathbf{c}}_q$ is new observation (query or keyword)

$$\tilde{\mathbf{c}}_q = \left\{\mathrm{idf}\left(t_i\right) : t_i \in Q \text{ or } K\right\}$$

  - for simplicity, we assume $\mathrm{tf}\left(t_i : t_i \in Q \text{ or } K\right) = 1$

# LSA Math (contd.)

- reduced k-rank SVD

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

  - rows of $\mathbf{U}$ correspond to documents
  - rows of $\mathbf{V}$ correspond to terms

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

$$\tilde{\mathbf{u}}_q = \left(\boldsymbol{\Sigma}^{-1}\mathbf{V}^{\top}\right)\tilde{\mathbf{c}}_q,$$

where $\tilde{\mathbf{c}}_q$ is new observation (query or keyword)

$$\tilde{\mathbf{c}}_q = \left\{\mathrm{idf}\left(t_i\right) : t_i \in Q \text{ or } K\right\}$$

  - for simplicity, we assume $\mathrm{tf}\left(t_i : t_i \in Q \text{ or } K\right) = 1$

# LSA Math (contd.)

- reduced k-rank SVD

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

  - rows of $\mathbf{U}$ correspond to documents
  - rows of $\mathbf{V}$ correspond to terms

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

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

  where $\tilde{\mathbf{c}}_q$ is new observation (query or keyword)

$$\tilde{\mathbf{c}}_q = \{\mathrm{idf}\,(t_i) : t_i \in Q \text{ or } K\}$$

  - for simplicity, we assume $\mathrm{tf}\,(t_i : t_i \in Q \text{ or } K) = 1$

# LSA Math (contd.)

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

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

- we will use $\mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$ of folded keywords and queries and combine heuristically with $J_c$
  - final score along the lines of $J_c\left(\mathbf{q}, \mathbf{k}\right) \cdot \mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$
  - throw away candidates below cosine similarity threshold, get N-best of the rest

# LSA Math (contd.)

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

$$\operatorname{sim}(\mathbf{u}_1, \mathbf{u}_2) = \cos\Theta = \frac{\mathbf{u}_1 \cdot \mathbf{u}_2}{|\mathbf{u}_1|\,|\mathbf{u}_2|}$$

- we will use $\operatorname{sim}(\mathbf{q}, \mathbf{k})$ of folded keywords and queries and combine heuristically with $J_c$
  - final score along the lines of $J_c(\mathbf{q}, \mathbf{k}) \cdot \operatorname{sim}(\mathbf{q}, \mathbf{k})$
  - throw away candidates below cosine similarity threshold, get N-best of the rest

# LSA Math (contd.)

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

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

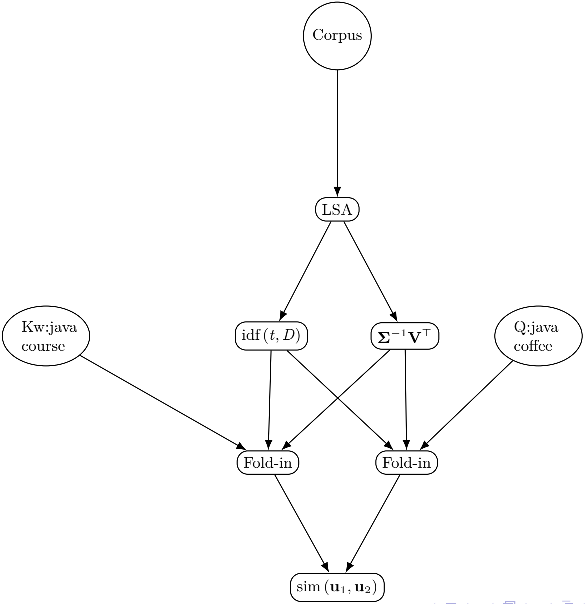
- we will use $\mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$ of folded keywords and queries and combine heuristically with $J_c$
  - final score along the lines of $J_c\left(\mathbf{q}, \mathbf{k}\right) \cdot \mathrm{sim}\left(\mathbf{q}, \mathbf{k}\right)$
  - throw away candidates below cosine similarity threshold, get N-best of the rest

# LSA Math (contd.)

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

$$\operatorname{sim}(\mathbf{u}_1, \mathbf{u}_2) = \cos\Theta = \frac{\mathbf{u}_1 \cdot \mathbf{u}_2}{|\mathbf{u}_1|\,|\mathbf{u}_2|}$$

- we will use $\operatorname{sim}(\mathbf{q}, \mathbf{k})$ of folded keywords and queries and combine heuristically with $J_c$
  - final score along the lines of $J_c(\mathbf{q}, \mathbf{k}) \cdot \operatorname{sim}(\mathbf{q}, \mathbf{k})$
  - throw away candidates below cosine similarity threshold, get N-best of the rest


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

# 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.

```
# 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)
}
```

```
# 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")]

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

## 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

1. Motivation
   - Broad match problems
2. Latent Semantic Analysis
   - Introducing LSA
   - Toy LSA Example in R
   - Stochastic SVD in Mahout
3. References

# Base modified algorithm with power iterations

## Single-threaded version

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 $m \times (k+p)$ 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 $\mathbf{Q} = \text{qr}(\mathbf{Y}).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^\top \mathbf{A} : \ \mathbf{B} \in \mathbb{R}^{(k+p) \times n}$.
- If $q > 0$ repeat: for $i = 1..q$: $\mathbf{B}_i^\top = \mathbf{A}^\top \text{qr}\left(\mathbf{A}\mathbf{B}_{i-1}^\top\right).\mathbf{Q}$ (power iterations step)

# Base modified algorithm with power iterations

## Single-threaded version

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 $m \times (k+p)$ 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 $\mathbf{Q} = \mathrm{qr}(\mathbf{Y}).\mathbf{Q}$.
- $\mathbf{B}_0 = \mathbf{Q}^\top\mathbf{A}: \ \mathbf{B} \in \mathbb{R}^{(k+p)\times n}$.
- 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

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 $m \times (k+p)$ 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

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 $m \times (k+p)$ 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

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 $m \times (k+p)$ 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 $i = 1..q$: $\mathbf{B}_i^\top = \mathbf{A}^\top \mathrm{qr}\left(\mathbf{A}\mathbf{B}_{i-1}^\top\right).\mathbf{Q}$ (power iterations step)

# Base modified algorithm with power iterations (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 $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\boldsymbol{\Sigma}^{-1}$. Another way is $\mathbf{V} = \mathbf{A}^\top\mathbf{U}\boldsymbol{\Sigma}^{-1}$
  - and bunch of other options for the output (see doc)

# Base modified algorithm with power iterations (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 $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\boldsymbol{\Sigma}^{-1}$. Another way is $\mathbf{V} = \mathbf{A}^\top\mathbf{U}\boldsymbol{\Sigma}^{-1}$
  - and bunch of other options for the output (see doc)

# Base modified algorithm with power iterations (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 $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}$. Another way is $\mathbf{V} = \mathbf{A}^\top\mathbf{U}\mathbf{\Sigma}^{-1}$
  - and bunch of other options for the output (see doc)

# Base modified algorithm with power iterations (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 $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}$. Another way is $\mathbf{V} = \mathbf{A}^\top\mathbf{U}\mathbf{\Sigma}^{-1}$
    - and bunch of other options for the output (see doc)

# Base modified algorithm with power iterations (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 $\mathbf{V} = \mathbf{B}_q^\top\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}$. Another way is $\mathbf{V} = \mathbf{A}^\top\mathbf{U}\mathbf{\Sigma}^{-1}$
  - and bunch of other options for the output (see doc)

# Characteristics of single-threaded version

- Needs a good decay
  - 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 $\frac{\sigma_{k+p}}{\sigma_1}$
- Presents interesting additional optimization opportunities for massive scale PCA

# Characteristics of single-threaded version

- Needs a good decay
  - 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 $\frac{\sigma_{k+p}}{\sigma_1}$
- Presents interesting additional optimization opportunities for massive scale PCA

# Characteristics of single-threaded version

- Needs a good decay
  - 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 $\frac{\sigma_{k+p}}{\sigma_1}$
- Presents interesting additional optimization opportunities for massive scale PCA

# MR characteristics of parallel version

- Fixed amount of MR jobs : $N = 3 + 2q$
  - Some of the jobs are map-only
- Experiments show really superior performance qualities over distributed Lanczos
  - see experiments staged by Nathan in his dissertation

# MR characteristics of parallel version

- Fixed amount of MR jobs : $N = 3 + 2q$
  - Some of the jobs are map-only
- Experiments show really superior performance qualities over distributed Lanczos
  - see experiments staged by Nathan in his dissertation

# MR characteristics of parallel version

- precision with q=1 is very good, with q=2 visually indistinguishable from that of Lanczos solver
  - options allow to adjust trade-off between speed vs. precision
  - running time is not absolutely linearly scalable to the input size:
    - task time $\propto\sim (k+p)^{1.5}$

# MR characteristics of parallel version

- precision with q=1 is very good, with q=2 visually indistinguishable from that of Lanczos solver
  - options allow to adjust trade-off between speed vs. precision
  - running time is not absolutely linearly scalable to the input size:
    - task time $\propto\sim (k+p)^{1.5}$

# MR characteristics of parallel version

- precision with q=1 is very good, with q=2 visually indistinguishable from that of Lanczos solver
  - options allow to adjust trade-off between speed vs. precision
  - running time is not absolutely linearly scalable to the input size:
    - task time $\propto\sim (k+p)^{1.5}$

# MR characteristics of parallel version

- precision with q=1 is very good, with q=2 visually indistinguishable from that of Lanczos solver
  - options allow to adjust trade-off between speed vs. precision
  - running time is not absolutely linearly scalable to the input size:
    - task time $\propto\sim (k+p)^{1.5}$

# MR characteristics of parallel version

- takes bigger datasets than Lanczos without giving up
  - max experiment known to me on an input of a little under 1Tb
- Known bottleneck is matrix multiplication of $\mathbf{AB}^\top$ step,
  - further remedies could be applicable
- QR of $\mathbf{B}_0$ pipeline could be replaced with a Cholesky decomposition trick
- Applicable to LSA and PCA, not really applicable to recommenders

# MR characteristics of parallel version

- takes bigger datasets than Lanczos without giving up
  - max experiment known to me on an input of a little under 1Tb
- Known bottleneck is matrix multiplication of $\mathbf{AB}^{\top}$ step,
  - further remedies could be applicable
- QR of $\mathbf{B}_0$ pipeline could be replaced with a Cholesky decomposition trick
- Applicable to LSA and PCA, not really applicable to recommenders

# MR characteristics of parallel version

- takes bigger datasets than Lanczos without giving up
  - max experiment known to me on an input of a little under 1Tb
- Known bottleneck is matrix multiplication of $\mathbf{AB}^\top$ step,
  - further remedies could be applicable
- QR of $\mathbf{B}_0$ pipeline could be replaced with a Cholesky decomposition trick
- Applicable to LSA and PCA, not really applicable to recommenders

# MR characteristics of parallel version

- takes bigger datasets than Lanczos without giving up
  - max experiment known to me on an input of a little under 1Tb
- Known bottleneck is matrix multiplication of $\mathbf{AB}^\top$ step,
  - further remedies could be applicable
- QR of $\mathbf{B}_0$ pipeline could be replaced with a Cholesky decomposition trick
- 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!