# Julia Tutorial Series - III

## Text Mining with Julia

Abhijith Chandraprabhu

February 1, 2014

## Overview

## Text Mining

- Type of *Information retrieval*, where we try to extract relevant information from huge collection of textual data(documents).
- Documents: web pages, biomedical literature, movie reviews, research articles etc.
- Semantic and Non-Semantic text mining.
- Vector-Space model technique.

[1]Refer JuliaTutorial-I

## Applications

- Web Search Engines : Search phrase is the query and the WWW collection of documents.
- Biomedical Applications : Searching for relevant research articles from huge databases like PubMed.
- Opinion/Sentiment Analysis : To find the general opinion from blogs.
- Customer relationships : Obtain valuable information from E-mails, surveys, etc

## Document1

It's also important to point out that even though these arrays are generic, they're not boxed: an Int8 array will take up much less memory than an Int64 array, and both will be laid out as continuous blocks of memory; Julia can deal seamlessly and generically with these different immediate types as well as pointer types like String.

## Document2

To celebrate some of the amazing work that's already been done to make Julia usable for day-to-day data analysis, I'd like to give a brief overview of the state of statistical programming in Julia. There are now several packages that, taken as a whole, suggest that Julia may really live up to its potential and become the next generation language for data analysis.

## Document3

Only later did I realize what makes Julia different from all the others. Julia breaks down the second wall — the wall between your high-level code and native assembly. Not only can you write code with the performance of C in Julia, you can take a peek behind the curtain of any function into its LLVM Intermediate Representation as well as its generated assembly code — all within the REPL. Check it out.

## Document4

Homoiconicity — the code can be operated on by other parts of the code. Again, R kind of has this too! Kind of, because I'm unaware of a good explanation for how to use it productively, and R's syntax and scoping rules make it tricky to pull off. But I'm still excited to see it in Julia, because I've heard good things about macros and I'd like to appreciate them.

## Document5

Graphics. One of the big advantages of R over similar languages is the sophisticated graphics available in ggplot2 or lattice. Work is underway on graphics models for Julia but, again, it is early days still.

## Term-Document Matrix

| | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 | Query 1 | Query 2 |
|---|---|---|---|---|---|---|---|
| Array | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| continous block | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Julia | 1 | 3 | 3 | 1 | 1 | 0 | 0 |
| types | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| string | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| amazing | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| data analysis | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| statistical computing | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| high-level | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| performance | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| LLVM | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| homoiconicity | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| syntax | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| macros | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| graphics | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| advantages | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

## Terminologies explained:

Corpus : Structured set of text, obtained by preprocessing raw textual data.

Lexicon : Distinctive collection of all the words/terms in the corpus.

Document, Query : Bag of words/terms, represented as vector.

Keyword, term : Elements of Lexicon.

Term Frequency : Frequency of occurence of a term in a Document.

TDM : A matrix, with term frequencies as entries across all Documents.

## Major Steps in Text Mining

1. Preprocess the documents to form a *corpus*.
2. Identify the *lexicon* from the corpus.
3. Form the *Term-Document matrix* (Numericized Text).
4. Apply VSM, LSI, K-Means to measure the proximity between all the documents and the query.

## Preprocessing using TextAnalysis.jl package

- Raw input Text is usually stream of characters. Need to convert this to stream of terms(basic processing units).
- The first step would be to create Documents which is collection of terms.
- Four types of documents can be created in Julia.
- A FileDocument which represents the files on disk
- str="Julia is a high-level, high-performance dynamic programming language for technical computing"

```
sd = StringDocument(str)
td = TokenDocument(str)
nd = NGramDocument(str)
```

## Preprocessing

Most of the times we have huge volumes of unstructured data, with content not carrying any useful information w.r.t Text Mining.

- HTML tags
- Numbers
- Stop words
- Prepositions, Articles, Pronouns,

In Julia, we can remove these uneccessary using the functions,

```
remove_articles!(), remove_indefinite_articles!()
remove_definite_articles!(), remove_pronouns!()
remove_prepositions!(), remove_stop_words!()
```

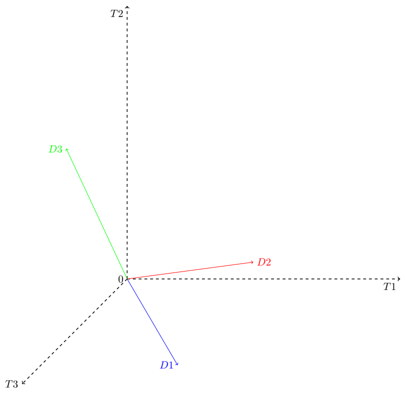
## Stemming

*I have a* **differ***ent opinion*
*I* **differ** *with your opinion*
*I opine* **differ***ently*

- In *Information Retrieval*, morphological variants of words/terms carrying the same semantic information adds to redundancy.
- **Stemming** linguistically normalizes the lexicon of a corpus.

```
stem!(Corpus)
stem!(Document) #sd, td or nd
```

## Vector-Space Model



- Documents are vectors in the *Term-Document Space*
- The elements of the vector are the weights[1], $w_{ij}$, corresponding to Document $i$ and term $j$
- The weights are the frequencies of the *terms* in the *documents*.
- Proximity of documents calculated by the *cosine* of the angle between them.

[a]Refer Weighting Schemes

## Term-Document Space

- Let $D_j$, be a collection of $i$ documents.
- $T = t_1, t_2, t_3, ..., t_i$, is the Lexicon set.
- $w_{ji}$, is the frequency of occurence of the term $j$ in document $i$.
- Document, $d_j = [w_{1j}, w_{2j}, w_{3j}, ..., w_{ij}]$.
- **TDM** $= [d_1 \;\; d_2 \;\; d_3 \;\; ... \;\; d_j]_{ij}$
- $d_j \in \mathcal{R}^{i \times j}$

## Weighting Schemes

- Binary Scheme : $w_{ij} = 1$, if $t_i$ occurs in $d_j$, 0 else not.
- Term Frequency (TF) Scheme : $w_{ij} = f_{ij}$, i.e. the number of times $t_i$ occurs in document $d_j$.
- Term Frequency - Inverse Document Frequency (TF-IDF) Scheme : Term Frequency $tf_{ij} = \frac{f_{ij}}{max[f_{1j}, f_{2j}, f_{3j} \dots, f_{ij}]}$
  Inverse-Document frequency, $idf_i = log \frac{N}{df_i}$,
  $N$: number of documents and $df_i$: Number of documents in which the term $t_i$ occurs.
  $w_{ij} = tf_{ij} \times idf_i$.

```
m=DocumentTermMatrix(crps)
tf_idf(m)
```

## Query Matching

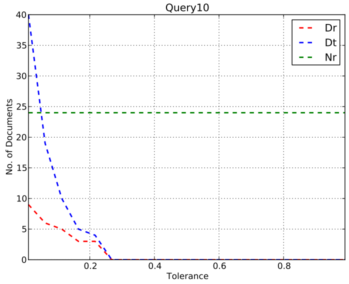
- Finding the relevant documents for a query, $q$.
- *Cosine Distance Measure* is used.
- $cos(\theta) = \frac{q^T d_j}{\|q\|_2 \|d_j\|_2}$, where $\theta$ is the angle between the query $q$ and document $d_j$.
- The documents for which $cos(\theta) > tol$, are considered relevant, where $tol$, is the predefined tolerance.

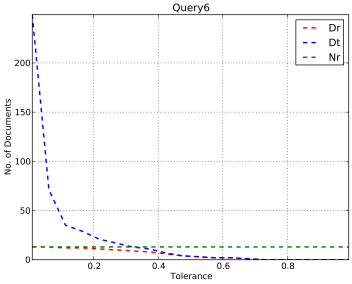
## Performance Modeling

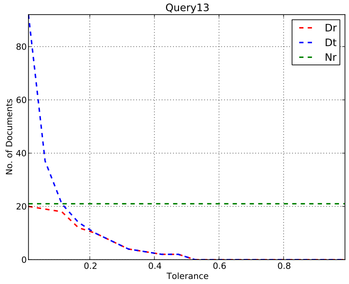
- The *tol*, decides the number of documents returned.
- With low *tol*, more documents are returned.
- But chances of the documents being irrelevant increases.
- Ideally we need higher number of documents returned and majority of the returned documents to be relevant.
- Precision, $P = \frac{D_r}{D_t}$, where $D_r$ is the number of relevant documents retrieved, and $D_t$ is the total number of documents retrieved.
- Recall, $R = \frac{D_r}{N_r}$, where $N_r$ is the total number of relevant documents in the database.

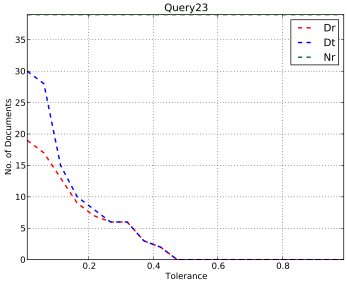
```
VecSpaceModelQueries(qNum,A)
```

This function is used to obtain $D_r, D_t N_r$ for any specific query using Vector Space model.


Query10
No. of Documents
Tolerance
Dr
Dt
Nr
0
5
10
15
20
25
30
35
40
0.2
0.4
0.6
0.8


Query6
No. of Documents
Tolerance
Dr
Dt
Nr
0
50
100
150
200
0.2
0.4
0.6
0.8


Query13
No. of Documents
Tolerance
Dr
Dt
Nr
0
20
40
60
80
0.2
0.4
0.6
0.8


Query23
No. of Documents
Tolerance
Dr
Dt
Nr
0
5
10
15
20
25
30
35
0.2
0.4
0.6
0.8

## Latent Semantic Indexing

- There exists underlying latent semantic structure in the data.
- We can identify this structure through SVD.
- Project the data onto two lower dimensional spaces.
- These are the *term space* and the *document space*.
- Dimension reduction is achieved through truncated SVD.

## Singular Value Decomposition

### Theorem (SVD)

*For any matrix $A \in \mathcal{R}^{m\times n}$, with $m > n$, there exists two orthogonal matrices $U = (u_1, \ldots, u_m) \in \mathcal{R}^{m\times m}$ & $V = (v_1, \ldots, v_n) \in \mathcal{R}^{n\times n}$ such that*

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T,$$

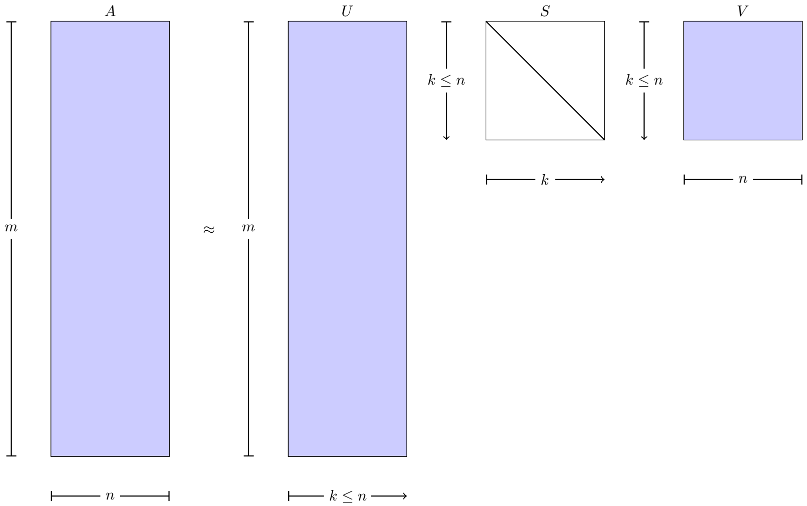
*where $\Sigma \in \mathcal{R}^{n\times n}$ is diagonal matrix, i.e., $\Sigma = (\sigma_1, ...., \sigma_n)$, with $\sigma_1 \geq \sigma_2 \geq .... \geq \sigma_n \geq 0$. $\sigma_1, \ldots, \sigma_n$ are called the singular values of $A$. Columns of $U$ & $V$ are called the right and left singular vectors of $A$ respectively.*

## Low Rank Matrix Approximation using SVD

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T \approx \sum_{i=1}^{k} \sigma_i u_i v_i^T =: A_k \qquad k < r$$

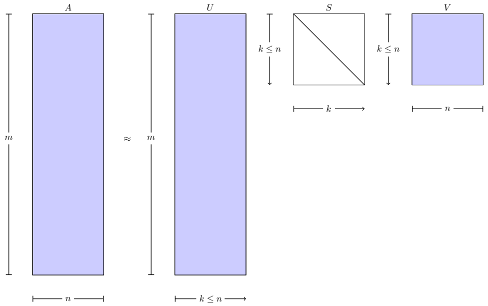
- The above approximation is based on the Eckart-Young theorem.
- It helps in removal of noise, solving ill-conditioned problems, and mainly in dimension reduction of data.
- Using the below function examine the effect of rank reduction.

```
svdRedRank(A,k)
```


$A$
$m$
$n$
$\approx$
$U$
$m$
$k \leq n$
$S$
$k \leq n$
$k$
$V$
$k \leq n$
$n$

```
SVDModelTDM(A::Array{Float64,2},qNum::Int64)
VecSpaceModelTDM(A::Array{Float64,2},qNum::Int64)
```

The above functions returns the Recall and Precision using the LSI model and the Vector space model. The first *qNum* vectors are the queries and the rest are the document vectors, of matrix $A$.



The End