Identifier Namespaces in Mathematical Notation

Master Thesis

by

Alexey Grigorev

Submitted to the Faculty IV, Electrical Engineering and Computer Science Database Systems and Information Management Group in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science
as part of the Erasmus Mundus IT4BI programme
at the
Technische Universität Berlin
July 31, 2015

Thesis Advisors:
Moritz Schubotz
Juan Soto

Thesis Supervisor:
Prof. Dr. Volker Markl

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Berlin, July 31, 2015

Alexey Grigorev

Table of Contents

1 Introduction

Some introductory stuff?

1.1 Namespaces in Computer Science

In computer science, a *namespace* refers to a collection of terms that are managed together because they share functionality or purpose, typically for providing modularity and resolving name conflicts [1].

In operation systems, files are organized into directories that are used to group related files together. To refer to a particular file we use the full name, e.g. `/home/user/documents/doc.txt` or `C:\\Users\\Username\\Documents\\doc.txt`. The same directory cannot contain a file with the same name, and we typically store closery related files in the same directory, so we can see file system directories as namespaces.

XML uses namespaces to prefix element names to ensure uniqueness and remove ambiguity between them [2].

Add examples (from the XML course)

Ralf's comment: "namespaces" were deeply studied mainly in the field of distributed systems/ middleware (like DCE, CORBA, etc.) in the middle of the 80s/ 90s ...

In programming languages it is a good design to put related objects into the same namespace, and namespaces give a way to achieve better modularity. There are design principles that tell the best way of grouping objects to achieve the low coupling between packages and high cohesion within packages [3].

Usually we group items

- that share the same functionality (e.g. objects for database access put into one package)
- that are about the same domain (can cite DDD)

In the Java programming language [4] packages are used to organize identifiers into namespaces. Packages solve the problem of ambiguity: for example, we have several classes with name `Date`: `java.util.Date` and `java.sql.Date` or `java.util.List` and `java.awt.List`.

Modularity of packages allows safely plug in libraries and use the code from there.

The programmer, when creating a method in a java class, can use all classes in the package. However, if they need to refer to a class in another

package, they either have to use a fully qualified name, e.g. `java.sql.Date` instead of `Date` or add an import statement `import java.sql.Date` in the header of the java class and use the alias `Date` in the class. (cite JLS [4] 7.5 Import Declarations)

1.2 Namespaces in Mathematical Notation

In this thesis we will extend the notion of namespaces to mathematical formulae.

In logic, a *formula* is defined recursively, and, in essence, it is a collection of variables, functions and other formulas, and formally the symbols for the variables and functions can be chosen arbitrarily [5]. However, in contrast to first order logic, in this work we are interested in the symbols in formulae and in mathematical notations that are used by different research communities. For example, in physics it is common to write the energy-mass relation as $E = mc^2$ rather than $x = yz^2$.

TODO: define what *identifier* is

define *identifier namespaces* as a coherent structure where each identifier is used only once and has a unique definition the process of discovering identifier namespaces is *namespace disambiguation*.

Example: E may refer to "energy", "expected value" or "elimination matrix".

In this thesis we compare different approaches for namespace disambiguation.

How do we construct a namespace?

How to find a namespace? Can be constructed manually using existent category information (refer to that list of categories of scientific articles)

But it's very time consuming. Our approach: use ML techniques to automatically discover the namespaces from corpus with mathematical formulas.

Observations:

In Java or other programming language a class may use objects from other packages via import statement.

Can distinguish between two types of application packages (in domain-driven design)

- domain-specific packages that deal with one particular concept
- packages that use many packages of the first type

For example, `org.company.app.domain.user`, `org.company.app.domain.account` and `org.company.app.tools.auth` are of type 1, while `org.company.app.web.manage` is of type 2 which mostly uses packages of type 1.

So type 1 packages are mostly self-contained and not highly coupled between each other, but type 2 packages mostly use other packages of type 1: they depend on them.

Can extend this intuition to groups of documents. Some groups are of type 1 - they define the namespaces. In some sense they are "pure" - they are all about the same thing

Some documents are not pure: they draw from different concepts, they can be thought as type 2 classes.

Assumptions:

1. documents are "mixtures" of namespaces: they take identifiers from several namespaces
2. there are some documents are more "pure" than others: they either take identifiers exclusively from one namespace or from few very related namespaces
3. there's a correlation between a category of a document and the namespaces the document uses

Formally, A document is a mixture of namespaces: $P(x) = \sum_{i=1}^{K} \pi_i P_i(x)$ where x is a document, $P_i(x)$ is probability that x uses concepts from namespace i, π_i are weights s.t. $\sum_{i=1}^{K} \pi_i = 1$ ...

Can call "pure" groups *namespace defining* groups - they can be thought as type-1 groups/documents.

Under these assumptions: can approximate the process of namespaces discovery by finding groups of "pure" documents. can evaluate purity by using category information and retain only pure ones

(or: that there is a strong correlation between identifiers in a document and the namespace of the document, and this correlation can be exploited to categorize documents and thus discover namespaces)

For example, if we observe a document with two identifiers E, assigned to "energy", and m, assigned to "mass", then it is more likely that the document belongs to the "physics" namespace rather than to "statistics".

Need to rephrase: Then, the process of cauterization may be formalized as follows: Let D denote a document and $Z(D)$ denote the category of the document. A document can be seen as a collection of identifiers, so $D = \{X_1, ..., X_N\}$, where each X_j is a probability distribution over possible definition assignment. Then we can classify a document based on $\arg\max_z P(Z = z \mid X_1 = x_1, \ ... \ , X_N = x_N)$. For example, if in a document we observe E assigned to "energy" and m assigned to "mass", then $P(\text{"physics"} \mid E = \text{"energy"}, m = \text{"mass"}) > P(\text{"statistics"} \mid E =$

"energy", m = "mass") and we may conclude that the identifiers in the document are likely to belong to the "physics" namespace.

To use it, we first need to map identifiers to their definitions, and this can be done by extracting the definitions from the text that surrounds the formula. We explore these methods in section 2.

1.3 Goals

The main objective of this study is to discover identifier namespace in mathematical formulae. We aim to find *meaningful* namespaces, in the sense that they can be related to a real-world area of knowledge, such as physics, linear algebra or statistics.

Once such namespaces are found, they can give good categorization of scientific documents based on formulae and notation used in them.

We believe that this may facilitate better user experience: for instance, it will allow users to navigate easily between documents of the same category and see in which other documents a particular identifier is used, how it is used, how it is derived, etc. Additionally, it may give a way to avoid ambiguity. If we follow the XML approach [2] and prepend namespace to the identifier, e.g. "physics.E", then it will give additional context and make it clear that "physics.E" means "energy" rather than "expected value".

We also expect that using namespaces is beneficial for relating identifiers to definitions. Thus, as an application of namespaces, we would like to be able to use them for better definition extraction. It may help to overcome some of the current problems in this area, for example, the problem of *dangling identifiers* [6] - identifiers that are used in formulae but never defined in the document. Such identifiers may be defined in other documents that share the same namespace, and thus we can take the definition from the namespace and assign it to the dangling identifier.

To accomplish the proposed goal, we plan the following.

First, we would like to study and analyze existing approaches and recognize similarities and differences with identifier namespaces. From the linguistics point of view, the theory of semantic fields [7] and semantic domains [8] are the most relevant areas. Then, namespaces are well studied in computer science, e.g. in programming languages such as Java [4] or markup languages such as XML [2]. XML is an especially interesting in this respect, because it serves as the foundation for knowledge representation languages like OWL (Web Ontology Language) [9] that use the notion of namespaces as well.

The process of manual categorization of mathematical corpus is quite time consuming. What is more, scientific fields are becoming more and more

interconnected, and sometimes it is hard even for human experts to categorize an article. Therefore, we believe that the namespaces should be discovered in an unsupervised manner.

Thus, we would like to try the following methods for finding namespaces: categorization based on the textual data [10], on semantic domains [8], on keywords extracted from the documents [11] or on definitions extracted from the formulae in the documents [6].

The data set that we plan to use is a subset of English wikipedia articles - all those that contain the `<math>` tag. The textual dataset can potentially be quite big: for example, the English wikipedia contains 4.5 million articles, and many thousands of them contain mathematical formulae. This is why it is important to think of ways to parallelize it, and therefore the algorithms will be implemented in Apache Flink [**?**].

2 Mathematical Definition Extraction

Mathematical expressions are hard to understand without the natural language description, so we want to find identifiers in the mathematical expressions and extract their definition from the surrounding text

Example:

The precision P is defined as $P = \frac{w}{w+y}$ where w is the number of correctly labeled pairs and y is the number of incorrectly labeled pairs

We want to extract:

(P, "precision"), (w, "the number of correctly labeled pairs"), (y, "the number of incorrectly labeled pairs")

Another example: "Let e be the base of natural logarithm"

want to extract (e, "the base of natural logarithm")

A phrase that defines a mathematical expression consists of three parts [12]:

- *definiendum*, a mathematical expression or identifier, is the term to be defined;
- *definiens* is the definition itself: it is the word or phrase that defines the definiendum in a definition.
- *definitor* is a relator verb that links definiendum and definiens.

In this work we are interested only in first two parts: definiendum and definiens. Thus we define a *relation* as a pair (definiendum, definiens). Since we are interested in finding definitions only for identifiers, we restrict ourselves only to (identifier, definition) relations.

An *identifier* is a mathematical ...

2.1 Formula Representation: MathML

MathML [13] stands for "Mathematical Markup Language" It is is a standard for mathematical expressions defined by W3C that browsers should support to render math formulas. There are two types of MathML: Presentation MathML, which describes how mathematical expressions should be displayed, and Content MathML, which focuses on the meaning of mathematical expressions. In this section, we will discuss Presentation MathML .

A *token* in MathML is an individual symbol, name or number. Tokens are grouped together to form MathML expressions.

Tokens can be:

– identifier, variable or function names
– numbers
– operators (including brackets - so called “fences”)
– text and whitespaces

A “symbol” is not necessarily one character: it could be a string such as `<mi>sin</mi>` or `<mn>24</mn>`. In MathML they are treated as single tokens.

As in mathematics, MathML expressions are constructed recursively from smaller expressions or single tokens. Complex expressions are created with so-called “layout” constructor elements, while tokens are created with token elements.

Let us consider an example. A mathematical expression $(a+b)^2$ can be represented in MathML as follows:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <msup>
    <mrow>
      <mo>(</mo>
      <mrow>
        <mi>a</mi>
        <mo>+</mo>
        <mi>b</mi>
      <mrow>
      <mo>)</mo>
    </mrow>
    <mn>2</mn>
  </msup>
</math>
```

It has the tree structure and recursive. If we take another mathematical expression $\frac{3}{(a+b)^2}$. It is a fraction and we see that its denominator is the same as the previous expression. This is also true for the MathML representation:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mfrac>
    <mn>3</mn>
    <msup>
```

```
      <mrow>
        <mo>(</mo>
        <mrow>
          <mi>a</mi>
          <mo>+</mo>
          <mi>b</mi>
        <mrow>
        <mo>)</mo>
      </mrow>
      <mn>2</mn>
    </msup>
  </mfrac>
</math>
```

Token Elements

Token elements are needed for representing tokens: the smallest units of mathematical notation that convey some meaning.

There are several token elements:

- `mi` identifier
- `mn` number
- `mo` operator, fence, or separator
- `mtext` text
- `mspace` space
- `ms` string literal

Often tokens are just single characters, like `<mi>E</mi>` or `<mn>5</mn>`, but there are cases when tokes are multi-character, e.g. `<mi>sin</mi>` or `<mi>span</mi>`.

In MathML `mi` elements represent some symbolic name or text that should be rendered as identifiers. Identifiers could be variables, function names, and symbolic constants.

Transitional mathematical notation often involve some special typographical properties of fonts, e.g. using bold symbols e.g. $\mathbf{x}$ to denote vectors or capital script symbols e.g. $\mathcal{G}$ to denote groups and sets. To address this, there

is a special attribute “mathvariant” that can take values such as “bold”, “script” and others.

Numerical literals are represented with `mn` elements. Typically they are sequences of digits, sometimes with a decimal point, representing an unsigned integer or real number, e.g. `<mn>50</mn>` or `<mn>50.00</mn>`.

Finally, operators are represented with `mo` elements. Operators are ...

Layouts

Layout elements are needed to form complex mathematical expressions from simple ones. They group elements in some particular way. For example:

- `mrow` groups any number of sub-expressions horizontally
- `mfrac` form sa fraction from two sub-expressions
- `msqrt` forms a square root (radical without an index)

Some layout elements are used to add subscripts and superscripts:

- `msub` attach a subscript to a base
- `msup` attach a superscript to a base
- `msubsup` attach a subscript-superscript pair to a base

And special kinds of scripts (TODO: describe in more details)

- `munder` attach an underscript to a base
- `mover` attach an overscript to a base
- `munderover` attach an underscript-overscript pair to a base

For example, $\boldsymbol{v}$ will be rendered as

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <mover>
    <mi>v</mi>
    <mo>&rarr;</mo>
  </mover>
</math>
```

This is how we would represent $\hat{\mathbf{x}}$ (a bold x with a hat) in MathML:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
```

```
    <mover>
      <mrow>
        <mi mathvariant="bold">x</mi>
      </mrow>
      <mo>&#x005E;<!-- ^ --></mo>
    </mover>
  </mrow>
</math>
```

There are more complex elements such as `mtable`.

MathML presentation elements only suggest specific ways of rendering Math Entities

Certain characters are used to name identifiers or operators that in traditional notation render the same as other symbols or usually rendered invisibly.

entities `&InvisibleTimes; &rarr;`

The complete list of MathML entities is described in [Entities].

2.2 Automatic Definition Extraction

We have an identifier and want to find what it stands for.

The definitions of mathematical expressions and identifiers can be found from the natural text description around the expression.

Assumption: the definitions of mathematical expressions are always noun phrases

In general, a noun phrase can be

- a simple noun
- a compound noun (e.g. adjective + noun)
- a compound noun with a clause, prepositional phrase, etc

2.3 Preprocessing

The typical (see e.g. [12] [6] (more?)) pipeline is the following:

- Read corpus of documents in some markup language, e.g. Mediawiki or Latex
- Translate all formulas in the documents into MathML
- Process MathML formulas
- Replace formulas with some placeholder
- Annotate text (using Math-Aware POS Tagging, see section 2.4)
- Find relations in the text

For example:

2.4 Math-aware POS tagging

NLP is a tool for text processing, but it's also applicable to scientific documents with math expressions. So we can adjust traditional NLP methods to dealing with formulas.

Penn Treebank POS Scheme [14] doesn't have special classes for mathematics. What we can do is to add other math-related classes:

ID for identifiers (e.g. "... where E stands for energy", E should be tagged as ID)

`MATH` for formulas (e.g. "$E = mc^2$ is the mass-energy equivalence formula", "$E = mc^2$ should be tagged as `MATH`)

Mathematical expressions are usually contained within special tags, e.g. inside tag `<math>...</math>` for wikipedia, or inside `$...$` for latex documents.

We find all such mathematical expressions and replace each with a unique single token `MATH_mathID`

the `mathID` could be a randomly generated string or result of some hash function applied to the content of formula. The latter approach is preferred when we want to have consistent strings across several runs.

Then we apply traditional [[POS Tagging]] (e.g. via Stanford CoreNLP [15]) techniques to the textual data. They typically will annotate such `MATH_mathID` tokens as nouns

after that we may want to re-annotate all math tokens: if it contains only one identifier, we label it as `ID`, if several - as `MATH`. But in some cases we want to keep original annotation

after that we can bring the mathematical content back to the document

2.5 Extraction Methods

Nearest Noun Method Definition is a combination of adjectives and nouns (also sometimes determinants) in the text before the identifier

[16] [17]

This way it only can be compound nouns without additional phrases.

E.g:

- "In other words, the bijection σ normalizes G in ..."
- It will extract relations (σ, "bijection")

Pattern Matching Methods Use patterns to extract definitions
For example,

- DESC IDENTIFIER (this is actually the same as nearest noun method)
- let—set IDENTIFIER denote—denotes—be DESC
- DESC is—are denoted—defined—given as—by IDENTIFIER
- IDENTIFIER denotes—dentore—stand—stands as—by IDENTIFIER
- IDENTIFIER is—are DESC
- DESC is—are IDENTIFIER
- and others

Patterns taken from [18] (**TODO**: mention who exactly did that) and sentence patterns from Graphs and Combinatorics papers from Springer
Used in [19]
Others ([12], [20], [6]) usually use this method as the baseline for comparison.

Machine Learning Based Methods Formulates definition extraction as a binary classification problem

- find candidate pairs (identifier, candidate definition)
- candidates are nouns and noun phrases from the same sentence as the expression

Features:

- sentence patterns (true if one of the patterns can capture this relation - could be separate feature for each pattern)
- if there's a colon/comma between candidate and identifier
- if there's another math expression between
- if candidate is inside parentheses and identifier is outside
- word-distance between candidate and identifier
- position of candidate relative to identifier
- text and POS tag of one/two/three preceding and following tokens around the candidate
- unigram/bigram/trigram of previous features
- text of first verb between candidate and identifier
- others

Classifiers: [[SVM]] (linear kernel) ([20], [17])
[[Conditional Random Fields]] ([20])

Probabilistic Approaches Mathematical Language Processing (MLP) Approach [6]: Statistical definition discovery: rank candidate definitions by probability and design an information extraction system that shots the most relevant (i.e. probable) definition to the reader to facilitate reading scientific texts with mathematics.

The main idea: the definitions occur very closely to identifiers in sentences.

extract identifiers from MathML

Two assumptions

- identifier and its definition occur in the same sentence, so the candidates are taken only from the same sentences (as in the ML approach)
- definitions are likely occur earlier in the document when authors introduce the meaning of an identifier, in subsequent uses the definition is typically not repeated

These assumptions are used in the ranking formula

each candidate is ranked with the following weighed sum:

$$R(n, \Delta, t, d) = \frac{\alpha\, R_{\sigma_d}(\Delta) + \beta\, R_{\sigma_s}(n) + \gamma\, \mathrm{tf}(t, s)}{\alpha + \beta + \gamma}$$

where

t is the candidate term, s set of sentences in the document, Δ is the distance (the amount of word tokens) between identifier and the candidate term t, n n is the number of sentences between the formula where the identifier is used and the place where the definition candidate is found.

The distances modeled with Gaussian (instead of taking the raw ones)

$$R_\sigma(\Delta) = \exp\left(-\frac{1}{2} \cdot \Delta^2 - 1\sigma_2\right)$$

assume that the probability to find a relation at $\Delta = 1$ is maximal

Finding Parameters σ_d and σ_s

σ_d - the standard deviation of Gaussian that models the distance to definition candidate manual evaluation showed that $R_{\sigma_d}(1) \approx R_{\sigma_d}(5)$, i.e. it's two times more likely to find the real definition at distance $\Delta = 1$ than at distance $\Delta = 5$. thus $\sigma_d = \sqrt{\frac{12}{\ln 2}}$

σ_s - the standard deviation of the Gaussian that models the distance from the beginning of document $\sigma_s = 2\sqrt{\frac{1}{\ln 2}}$

weights α, β, γ: $\alpha = \beta = 1$ and $\gamma = 0.1$ because some valid definitions may occur more often than other valid definitions, e.g. "length" vs "Hamiltonian"

Other Ideas Translation of mathematical formulas to English using machine-translation techniques [21]

Expand or delete?

2.6 Performance Measures

TODO: Write more about this

- Precision: no of math expresssion with correctly extracted definitions / no of extracted definitions
- Recall: no of math expresssion with correctly extracted definitions / no of expressions with definitions
- $F_1 = 2PR/(P + R)$: harmonic mean between P and R

3 Namespaces as Document Clusters

3.1 Discovering Namespaces with Cluster Analysis

Why it will work?

List properties of text that identifier-based representation of documents share as well

list characteristics of textual data

- dimensionality is very large, but vectors are very sparse. e.g. vocabulary size $|V| = 10^5$, but documents may contain only 500 distinct words, or even less - when we consider sentences or tweets
- lexicon of document may be large, but words are typically correlated with each other so number of concepts ("principal components") is $\ll |V|$
- number of words across different documents may wary a lot
- word distributions follow Power Laws (Zipf's law etc - cite)

Problems in text:

- problems of polysemy, homonymy and synonymy: semantic relations between words
- Polysemy is the capacity for a sign (such as a word, phrase, or symbol) to have multiple meanings (multiple senses)
- The state of being a homonym is called homonymy. In linguistics, a homonym is, in the strict sense, one of a group of words that share the same spelling and pronunciation but have different meanings.[1] http://dictionary.reference.com/
- Words that are synonyms are said to be synonymous, and the state of being a synonym is called synonymy.
- The analysis of synonymy, polysemy, and hyponymy and hypernymy is vital to taxonomy and ontology in the information-science senses of those terms.
- Homonyms are words that have the same pronunciation and spelling, but have different meanings. For example, rose (a type of flower) and rose (past tense of rise) are homonyms.

This is also true for identifiers and they have the same problems (illustrate that) These problems are studied in IR and NLP literature, so we can apply them for identifiers

Identifier spaces (need to define what it is - analogous to documents vector space models) have the same characteristics so we can apply vector space techniques

in VSM for word sense disambiguation often word meaning is attached e.g. "bank_finances", can do the same e.g. "E_energy"

Then we discover namespaces in the identifier namespaces - it's done by clustering document-identifier matrix

How to deal with these problems? Term Extraction techniques: these techniques create "artificial" terms that aren't really terms - they are generated, and not the ones that actually occurred in the text The original terms don't have the optimal dimensionality for document content representation because of the problems of polysemy, homonymy and synonymy so we want to find better representation that doesn't suffer from these issues

Assumptions:

1. documents are "mixtures" of namespaces: they take identifiers from several namespaces
2. there are some documents are more "pure" than others: they either take identifiers exclusively from one namespace or from few very related namespaces
3. there's a correlation between a category of a document and the namespaces the document uses

Under these assumptions: can approximate namespaces discovery by finding groups of "pure" documents. can evaluate purity by using category information and retain only pure ones

(or: that there is a strong correlation between identifiers in a document and the namespace of the document, and this correlation can be exploited to categorize documents and thus discover namespaces)

3.2 Vector Space Model

Vector Space model: term selection, weighting schemes; how to build identifier space and include definition information

[10] - *classification*

TF-IDF

tf is normalized by idf

idf - reduces weights of terms that occur more frequently

When applied to identifiers: some identifiers like x or y occur very frequently and don't have much discriminating power

to ensure that document matching is done with more discriminative words apply sub-linear transformation to to avoid the dominating effect of words that occur very frequently

3.3 Identifier Space Model

There are three ways of incorporating the definition information into the identifier space.

Given (λ, regularization) and (w, weight vector)

- use only identifier information. dimensions are (λ, w)
- use "weak" identifier-definition association: dimensions are (λ, w, regularization, weight vector)
- use "strong" association: dimensions are (λ_regularization, w_weight vector)

3.4 Similarity Measures and Distances

Once the documents are represented in some vector space, we need to define how to compare these documents to each other. There are two ways of doing this: using a similarity function that tells how similar two objects are (the higher values - the more similar the objects), or using a distance function, sometimes called "dissimilarity function", which is the opposite of similarity (the higher the values, the less similar the objects)

The most commonly used distance function in vector spaces is Euclidean distance which corresponds to the geometric distance between two points in the space. For example, if we have two points $\mathbf{x}$ and $\mathbf{z}$ then the Euclidean distance is the length of the line that connects these two points. It is defined as $\|\mathbf{x}-\mathbf{z}\| = \sqrt{(\mathbf{x}-\mathbf{z})^T(\mathbf{x}-\mathbf{z})} = \sqrt{\sum_i (x_i - z_i)^2}$. This distance is also often called L_2 distance.

But Euclidean distance is not always meaningful for high dimensional data.

$\|\mathbf{x}-\mathbf{z}\|^2 = (\mathbf{x}-\mathbf{z})^T(\mathbf{x}-\mathbf{z}) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{z} + \mathbf{z}^T\mathbf{z} = \|\mathbf{x}\|^2 - 2\mathbf{x}^T\mathbf{z} + \|\mathbf{z}\|^2$

We see that it also takes into account the length of each individual vector, and therefore distance between related and not related documents can be the same.

Consider the following example (from [29]): There are 4 data points in 10-dimensional space indexed by identifiers $A_1, \ldots, A_{10}$

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
$\mathbf{p}_1$	3	0	0	0	0	0	0	0	0	0
$\mathbf{p}_2$	0	0	0	0	0	0	0	0	0	4
$\mathbf{p}_3$	3	2	4	0	1	2	3	1	2	0
$\mathbf{p}_4$	0	2	4	0	1	2	3	1	2	4

distance between $\mathbf{p}_1$ and $\mathbf{p}_2$ is $\|\mathbf{p}_1 - \mathbf{p}_2\| = 5$ distance between $\mathbf{p}_3$ and $\mathbf{p}_4$ is $\|\mathbf{p}_3 - \mathbf{p}_4\| = 5$ so Euclidean distance between these two vectors is

the same! but suppose these vectors correspond to documents and words ([[Vector Space Models]]) $\mathbf{p}_3$ and $\mathbf{p}_4$ must be more similar to each other than $\mathbf{p}_1$ and $\mathbf{p}_2$: $\mathbf{p}_3$ and $\mathbf{p}_4$ have 7 words in common whereas $\mathbf{p}_1$ and $\mathbf{p}_2$ have only 2

When the data is sparse it's better to use different measure of distance/similarity we need to ignore records where both vectors have 0 for example: [[Dot Product]] and [[Cosine Similarity]] [[Jaccard Coefficient]]

Jaccard similarity, jaccard distance

Dot product Dot product, or inner product, can also be seen as a similarity measure: the

Geometric Definition Let $\boldsymbol{v} \cdot \boldsymbol{w}$ denote the "dot product" between vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ definition: $\boldsymbol{v} \cdot \boldsymbol{w} = \|\boldsymbol{v}\| \cdot \|\boldsymbol{w}\| \cdot \cos\theta$ where θ is the angle between $\boldsymbol{v}$ and $\boldsymbol{w}$ $\|\boldsymbol{v}\|$ denotes the length of $\boldsymbol{v}$ if two vectors are perpendicular, then $\cos\theta = 0$ and thus $\boldsymbol{v} \cdot \boldsymbol{w} = 0$ if they co-directional, then $\theta = 0$ and $\boldsymbol{v} \cdot \boldsymbol{w} = \|\boldsymbol{v}\| \cdot \|\boldsymbol{w}\|$ consequently, we have $\boldsymbol{v} \cdot \boldsymbol{v} = \|\boldsymbol{v}\|^2$

Why does this geometric definition make sense?

consider vectors $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ let $\boldsymbol{v} + \boldsymbol{u} = \boldsymbol{w}$ or $\boldsymbol{u} = \boldsymbol{w} - \boldsymbol{v}$ (ADD PROPER IMAGE WITH COSINE THEOREM http://habrastorage.org/files/d9f/8b1/073/d9f8b10734864b9
)

$\|\boldsymbol{u}\|^2 = \|\boldsymbol{w} - \boldsymbol{v}\|^2 = (\boldsymbol{w} - \boldsymbol{v}) \cdot (\boldsymbol{w} - \boldsymbol{v}) = \|\boldsymbol{w}\|^2 + \|\boldsymbol{v}\|^2 - 2 \cdot \boldsymbol{w}\boldsymbol{v}$ by the Cosine Theorem we know that $\|\boldsymbol{u}\|^2 = \|\boldsymbol{w} - \boldsymbol{v}\|^2 = \|\boldsymbol{w}\|^2 + \|\boldsymbol{v}\|^2 - 2 \cdot \|\boldsymbol{w}\| \cdot \|\boldsymbol{v}\| \cdot \cos\theta$ so $\|\boldsymbol{w}\| \cdot \|\boldsymbol{v}\| \cdot \cos\theta = \frac{1}{2}(\|\boldsymbol{w}\|^2 + \|\boldsymbol{v}\|^2 - \|\boldsymbol{w} - \boldsymbol{v}\|^2) = \frac{1}{2}(\|\boldsymbol{w}\|^2 + \|\boldsymbol{v}\|^2 - \|\boldsymbol{w}\|^2 - \|\boldsymbol{v}\|^2 + 2 \cdot \boldsymbol{w}\boldsymbol{v}) = \boldsymbol{w} \cdot \boldsymbol{v}$ thus $\boldsymbol{w} \cdot \boldsymbol{v} = \|\boldsymbol{w}\| \cdot \|\boldsymbol{v}\| \cdot \cos\theta$ i.e. the definition makes sense from the The Cosine Theorem point of view

For two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ we define the dot product as $\mathbf{v}^T\mathbf{w} = \sum_{i=1}^{n} v_i w_i$

These definitions are equivalent

Cosine similarity is a [[Similarity Function]] that is often used in [[Information Retrieval]] it measures the angle between two vectors, and in case of IR - the angle between two documents

recall the definition of the [[Dot Product]]: $\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \cdot \|\mathbf{w}\| \cdot \cos\theta$ or, by rearranging get $\cos\theta = \dfrac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \cdot \|\mathbf{w}\|}$ so, let's define the cosine similarity function as $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \dfrac{\mathbf{d}_1^T \mathbf{d}_2}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|}$ cosine is usually $[-1, 1]$, but document vectors (see [[Vector Space Model]]) are usually non-negative, so the angle between two documents can never be greater than 90 degrees, and for document vectors $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) \in [0, 1]$ min cosine is 0 (max angle: the

documents are orthogonal) max cosine is 1 (min angle: the documents are the same)

If documents have unit length, then cosine similarity is the same as [[Dot Product]] $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \dfrac{\mathbf{d}_1^T \mathbf{d}_2}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|} = \mathbf{d}_1^T \mathbf{d}_2$ thus we can "unit-normalize" document vectors $\mathbf{d}' = \dfrac{\mathbf{d}}{\|\mathbf{d}\|}$ and then compute dot product on them and get cosine this "unit-length normalization" is often called "cosine normalization" in IR

Cosine Distance for documents $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) \in [0, 1]$ it is max when two documents are the same how to define a distance? distance function should become larger as elements become less similar since maximal value of cosine is 1, we can define "'cosine distance"' as $d_c(\mathbf{d}_1, \mathbf{d}_2) = 1 - \text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = 1 - \dfrac{\mathbf{d}_1^T \mathbf{d}_2}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|}$

Let's check if cosine distance is a proper metric, i.e. it satisfies all the requirements Let D be the document space and $\mathbf{d}_1, \mathbf{d}_2 \in D$ $d_c(\mathbf{d}_1, \mathbf{d}_2) \geqslant 0$: checks - 0 is minimum $d_c(\mathbf{d}_1, \mathbf{d}_1) = 0$ checks - $1 - \cos 0 = 0$ $d_c(\mathbf{d}_1, \mathbf{d}_2) = d_c(\mathbf{d}_2, \mathbf{d}_1)$: checks - angle is the same

What about the triangle inequality? under certain conditions is doesn't hold (Korenius2007) - so it's not a proper metric

Cosine and Euclidean Distance Euclidean distance $\|\mathbf{d}_1 - \mathbf{d}_2\| = \sqrt{(\mathbf{d}_1 - \mathbf{d}_2)^T (\mathbf{d}_1 - \mathbf{d}_2)}$

There's a connection between Cosine Distance end Euclidean Distance consider two unit-normalized vectors $\mathbf{x}_1 = \mathbf{d}_1 / \|\mathbf{d}_1\|$ and $\mathbf{x}_2 = \mathbf{d}_1 / \|\mathbf{d}_1\|$ $\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = (\mathbf{x}_1 - \mathbf{x}_2)^T (\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{x}_1 - 2\,\mathbf{x}_1^T \mathbf{x}_2 + \mathbf{x}_2^T \mathbf{x}_2 = \|\mathbf{x}_1\|^2 - 2\,\mathbf{x}_1^T \mathbf{x}_2 + \|\mathbf{x}_2\|^2 = 2 - 2\,\mathbf{x}_1^T \mathbf{x}_2$ if vectors are unit-normalized, cosine = dot product, so we have $\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = 2\,(1 - \mathbf{x}_1^T \mathbf{x}_2) = 2\,\big(1 - \text{cosine}(x_1, x_2)\big) = 2\,d_c(x_1, x_2)$

It can also make some sense visually:

https://habrastorage.org/files/f73/289/979/f732899792f246358649e89765cd88da.png recall the [[Cosine Theorem]]: $a^2 = b^2 + c^2 - 2bc\cos\theta$ $b = c = 1$, so we have $a^2 = 2\,(1 - \cos\theta)$

Thus we can use Euclidean distance and interpret it as Cosine distance

Korenius, Tuomo, Jorma Laurikkala, and Martti Juhola. "On principal component analysis, cosine and Euclidean measures in information retrieval." 2007.

3.5 Inverted Index

Inverted Index is a IR technique for being able to retrieve results faster. It also can be used for making clustering faster

if D is a document-term matrix, then the inverted index is build by considering D^T and

In [[Databases]], [[Indexing (databases)—indexing]] is needed to speed up queries want to avoid full table scan same is true for [[Information Retrieval]] and other [[Text Mining]]/[[NLP]] tasks Inverted index is a way of achieving this, and it can be generalized to other forms of input, not just text

For IR index is a partial representation of a document that contains the most important information about the document usually want to find terms to index automatically

Inverted Index for Similarity Search

Idea: usually a document contains only a small portion of terms so document vectors are very sparse typical distance is cosine similarity - it ignores zeros. for cosine to be non-zero, two docs need to share at least one term D^T is the inverted index of the term-document matrix D

This, to find docs similar to d: for each $w_i \in d$ let $D_i = \text{index}[w_i] - d$ be a set of documents that contain w_i (except for d itself) then take the union of all D_i calculate similarity only with documents from this union

Can be used in [[Document Clustering]] to speed up similarity computation

Posting List Build a dictionary: a "posting" list for each word we store ids of documents that have this word document are sorted by ids http://slidewiki.org/upload/media/im width-550 source of picture: http://slidewiki.org/print/deck/339 sorting - because it's easier to take union: just merge the posting list

4 Document Clustering Techniques

Scatter/Gather - first successful document clustering for information retrieval [22]

Types of clustering techniques:

- hierarchical
- partitioning
- density-based
- etc

Cluster analysis: organizing collection of items into coherent groups

IR: need to assist user and group retrieved results into clusters

Finding topics in a collection of document is an important task in text mining. Usually done by applying a clustering technique and hope it will result if groups that represent documents in some common theme or topic [23] The goal is to find clusters with strong coherent themes (even if we omit some documents) in reality it's true: we don't expect all documents to belong to some strong category.

A group of documents with a strong topic is characterized by how it uses words from some specialized vocabulary in additional to general vocabulary.

A doc about olympics would contain general terms (the, a) and specific terms (country names)

Data model [23] 1: there are concepts from which documents pick their words. DESCRIBE it.

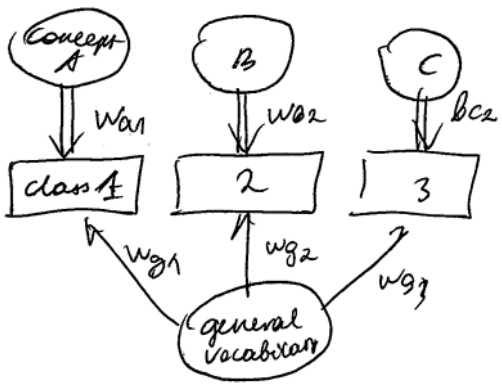

Fig. 1. TODO: Abstract data model

4.1 Hierarchical clustering

Agglomerative Clustering General concept: merge items into clusters based on distance/similarity usually based on best pairwise similarity

Typical steps:

- at the beginning each document is a cluster on its own - then we compute similarity between all pairs of clusters and store the results in a similarity matrix - merge two most similar clusters - update the similarity matrix - repeat until everything belongs to the same cluster

Linkage Types

How to join two clusters? - Single Linkage (SLINK) - Complete Linkage (CLINK) - Group-Average Linkage - Ward

Single Linkage Merge two groups A and B based on their closest pair

Implementation: compute all similarity pairs sort them in order of decrease process pairs in this order

advantage: efficient to implement equivalent to a Spanning Tree algo on the complete graph of pair-wise distances can use Prim's Spanning Tree algo

Drawbacks encourages chaining similarity is usually not transitive: i.e. if A is similar to B, and B is similar to C, it doesn't mean that A must be similar to C but single linkage encourages grouping through transitivity chains

References: Sibson, Robin. "SLINK: an optimally efficient algorithm for the single-link cluster method." 1973.

Complete Linkage Worst-case similarity: avoids chaining altogether but it's very expensive computationally

References: Defays, Daniel. "An efficient algorithm for a complete link method." 1977.

Group-Average Linkage similarity between groups A and B are calculated as average similarity between each $a \in A$ and $b \in B$

It's way slower than single linkage, but it's more robust: it doesn't show the chaining behavior

Speeding up: can approximate it by using the distance between centroids: mean doc in A and mean doc in B

Ward's Method Merge the pair of clusters that minimizes the total within-group error (sum of squares) between each document and centroid

Result: spherical tightly bound clusters less sensitive to outliers

References: El-Hamdouchi, Abdelmoula, and Peter Willett. "Hierarchic document classification using Ward's clustering method." 1986.

Pros and Cons Single-link algorithms are best for capturing clusters of different sizes and shapes but it's also sensitive to noise complete link and

group average are not affected by noise, but have a bias towards finding global patterns

Computational complexity: only Single-Link is computationally possible for large datasets, but it doesn't give good results because uses too little information

[24] - not always good for document clustering because they make mistakes at early iterations that are impossible to correct afterwards

4.2 K-Means

Lloyd algorithm is the most popular way of implementing k-means

Algorithm First we choose k - the number of clusters we want to get Then we randomly initialize k cluster centers (cluster centroids)

This is an iterative algorithm, and on each iteration it does 2 things cluster assignment step move centroids step

Cluster Assignment Step: go through each example and choose the closest centroids and assign the example to it

Move Centroids Step: Calculate the average for each group and move the centroids there

Repeat this until converges

Pseudo Code

Let's have the following notation $c_i \in \{1, 2, \ ... \ , k\}$ - index of cluster to which example $\mathbf{x}_i$ is assigned $\boldsymbol{\mu}_k$ - cluster centroid k ($\boldsymbol{\mu}_k \in \mathbb{R}^n$) μ_{c_i} - cluster centroid of example $\mathbf{x}_i$

k-means(k, $\{\mathbf{x}_i, y_i\}$):

randomly initialize k cluster centroids $\boldsymbol{\mu} = \Big(\mu_1, \mu_2, \ ... \ , \mu_k\Big) \in \mathbb{R}^{k+1}$ repeat: cluster assignment step: for $i = 1$ to m: $c_i \leftarrow$ closest to $\mathbf{x}_i$ centroid using [[Euclidean Distance]] dist $= \|\mathbf{x}_i - \boldsymbol{\mu}_i\|^2$ move centroids step: for $i = 1$ to k: $\boldsymbol{\mu}_k \leftarrow$ average of all points assigned to c_k

Often K-Means shows the best results [25] [24], but it takes long for large collections.

Solution: minibatch K-means (web-scale k-means clustering paper) [26]

Lloyd's classical algorithm is slow for large datasets (Sculley2010) Use [[Mini-Batch Gradient Descent]] for optimizing K-Means reduces complexity while achieving better solution than [[Stochastic Gradient Descent]]

Notation: $f(C, \mathbf{x})$ returns the nearest centroid for $\mathbf{x}$

Algorithm: given k, batch size b, max. number of iterations t and dataset X initialize each $\boldsymbol{\mu}$ with randomly selected elements from X repeat t times: $M \leftarrow b$ random examples from X for $\mathbf{x} \in M$: $d[\mathbf{x}] = f(C, \mathbf{x})$ // cache the

centroid nearest to $\mathbf{x}$ for $\mathbf{x} \in M$: $\boldsymbol{\mu} \leftarrow d[\mathbf{x}]$ $v[\boldsymbol{\mu}] = v[\boldsymbol{\mu}] + 1$ // counts per centroid $\eta = 1/v[\boldsymbol{\mu}]$ // per-centroid learning rate $\boldsymbol{\mu} \leftarrow (1 - \eta) \cdot \boldsymbol{\mu} + \eta \cdot \mathbf{x}$ //gradient step

If rows are unit-normalized, then Euclidean K-means is the same as Cosine Distance K-Means, but with convergence guarantees. TODO: ADD REFERENCE

4.3 Extensions of K-Means

Bisecting K-Means [24]

This is a variant of K-Means it's a [[Hierarchical Clustering]] method, and it's useful for [[Document Clustering]]

Algorithm: start with a single cluster repeat until have desired number of clusters choose a cluster to split (e.g. the largest one) find two subclusters using K-means with $k = 2$ and split may repeat this procedure several times and take the clusters with highest overall similarity

Scatter/Gather [22]

This is use [[Hierarchical Clustering]] for seed selection, k-means for clustering

Scatter/Gather is a variation of [[K-Means]] used for [[Document Clustering]] with special seed selection usual k-means then several cluster refinement operations

Idea: use some hierarchical clustering algorithm on a sample to find good initial seeds use K-Means afterwards

After selecting seeds just do usual K-Means but do some additional operations to improve quality for document clustering

Cluster refinement hypothesis: documents that belong to the same cluster in finer granularity will also occur in the same cluster in coarser granularity

Split Operations The main idea is to continue splitting after K-Means has finished sometimes clusters as not as granular as we'd like continue splitting them to refine clusters don't need to apply it to all clusters - only to non-coherent ones it will help create more coherent clusters

(so it's just applying k-means once again on the cluster with $k = 2$

Algo: select a cluster to split apply buckshot with $k = 2$ on this cluster re-cluster around these centers

How to measure "coherence"? compute self-similarity of a cluster: similarity of documents to the centroid cluster or average pairwise similarity within the cluster apply split only to clusters with low self-similarity

Join Operation The idea: after K-means has finished, join very similar clusters into one

How? compute typical words of each cluster: i.e. examine most frequent words on the centroid two clusters are similar if there's significant overlap between words of these clusters

Challenges if there are many documents, the centroid will contain many word =¿ leads to slowdown solution: use [[Projection onto Subspaces—projection]] techniques like [[PCA]]

Center adjustment (vector average dumping) [27]

Smart seed selection [22] [27]

4.4 DBSCAN

describe the algo [28]

It's a density-based clustering algorithm

Density associated with a point is obtained by counting the number of points in a region of specified radius ϵ around each point points with density $\geqslant$ min_pts are considered as "core points" noise and non-core points are discarded clusters are formed around the core points if two core points are within a radius ϵ, then they belong to the same cluster

Disadvantages can find clusters of different shapes, but can't find clusters of different densities

Pseudocode (taken from Wiki)

```
DBSCAN(D, eps, MinPts) {
   C = 0
   for each point P in dataset D {
      if P is visited
         continue next point
      mark P as visited
      NeighborPts = regionQuery(P, eps)
      if sizeof(NeighborPts) < MinPts
         mark P as NOISE
      else {
         C = next cluster
         expandCluster(P, NeighborPts, C, eps, MinPts)
      }
   }
}

expandCluster(P, NeighborPts, C, eps, MinPts) {
   add P to cluster C
```

```
   for each point P' in NeighborPts {
      if P' is not visited {
         mark P' as visited
         NeighborPts' = regionQuery(P', eps)
         if sizeof(NeighborPts') >= MinPts
            NeighborPts = NeighborPts joined with NeighborPts'
      }
      if P' is not yet member of any cluster
         add P' to cluster C
   }
}

regionQuery(P, eps)
   return all points within P's eps-neighborhood (including P)
```

Can be adapted to take similarity measure instead of distance

4.5 Extensions of DBSCAN

SNN Clustering: [29]

Different similarity measure

Applicable to document clustering and discovering topics [23]

The goal: find clusters of different shapes, sizes and densities in high-dimensional data [[DBSCAN]] is good for finding clusters of different shapes and sizes, but it fails to find clusters with different densities it will find only one cluster: http://habrastorage.org/files/ff4/b40/6fc/ff4b406fc5d948d7bf3b2d4e3c18a71d.png (figure source: Ertoz2003)

Distance: [[Euclidean Distance]] is not good for high-dimensional data use different similarity measure in terms of [[KNN]]s - "Shared Nearest Neighbors" then define density in terms of this similarity

"Jarvis-Patrick" algorithm, as in Jarvis1973

Step 1: SNN sparsification: construct an SSN [[Graph]] from data matrix as follows if p and q have each others in the KNN list then create a link between them

Step 2: Weighting weight the links with $\text{sim}(p, q) = \left| \text{NN}(p) \cup \text{NN}(q) \right|$ where NN(p) and NN(q) are k neighbors of p and q resp.

Step 3: Filtering then filter the edges: remove all edges with weight less than some threshold

Step 4: Clusters let all connected components be clusters

Illustration http://habrastorage.org/files/b2b/174/cd8/b2b174cd84e3488a8d1dad51687bf194.p (figure source: Ertoz2003) note that this procedure removed the noise and clusters are of uniform density: it breaks the links in the transition regions

Usual density is not good: In the Euclidean space, the density is the number of points per unit volume but as dimensionality increases, the volume increases rapidly so unless the number of points increases exponentially with dimensionality, the density tends to 0 Density-based algorithms (e.g. [[DBSCAN]]) will not work properly

Need different intuition of density can use a related concept from if kth nearest neighbor is close, then the region is most likely of high density so the distance to kth neighbor gives a measure of density of a point because of the [[Curse of Dimensionality]], the approach is not good for [[Euclidean Distance]], [[Cosine Similarity]] or others but we can use the SNN-Similarity to define density

SSN-based measures of density: sum of SSN similarities over all KNNs why sum an not just kth? to reduce random variation - which happens when we look only at one point to be consistent with the graph-based view of the problem of it can be the number of points within some radius - specified in terms of SNN distance like in [[DBSCAN]], but with SSN distance

SNN Clustering algorithm is a combination of Jarvis-Patrick algorithm and DBSCAN with SSN Similarity and SSN Density

Parameters k ϵ min_pts $< k$

Steps: compute the similarity matrix sparsify the matrix by keeping only k most similar neighbors for each data point construct the SSN graph (use the Jarvis-Patrick algo) find SSN density of each point p: in the KNN list of p count q s.t. $\text{sim}(p, q) \geqslant \epsilon$ find the core points all points with SSN density greater than min_pts are the core ones form clusters from the core points all non-core points not within ϵ from the core ones are discarded as noise align non-noise non-core points to clusters

Parameter tuning: k controls granularity of clusters if k is small, then it will find small and very tight clusters if k is large, it'll find big and well-separated clusters

The algorithm runs in $O(n^2)$ time can speed up with [[Kd-Trees]] or [[R-Tree]]s alternatively, can use [[Canopy Clustering—canopies]]

4.6 Scaling?

LSH etc

5 Discovering Latent Semantics

How to deal with these problems? Term Extraction techniques: these techniques create "artificial" terms that aren't really terms - they are generated, and not the ones that actually occurred in the text The original terms don't have the optimal dimensionality for document content representation because of the problems of polysemy, homonymy and synonymy so we want to find better representation that doesn't suffer from these issues

5.1 Latent Semantic Analysis

LSI paper [30]
Using SVD for clustering IR results [31]
Latent Semantic Analysis (LSA) is an NLP method:

- mathematical/statistical method for modeling the meaning of words/passages by analysis of text via extracting and inferring relations of expected contextual usage of words in texts
- idea: words that are used in the same contexts tend to have the same meaning
- it extracts and represents "usage-in-context" meaning of words and it gives a characterization of words meaning

LSA
brings up the latent structure of the vocabulary

- LSA closely approximates how humans learn and understand meaning of words and similarity between words
- it applies Factor Analysis to texts to extract concepts and then clusters documents into similar categories based on factor scores
- produces measures of word-word, word-passage, passage-passage relations via dimensionality reduction technique SVD
- Similarity estimates derived by LSA are not just frequencies or co-occurrences counts: it can infer deeper relations: hence "Latent" and "Semantic"

Limitations

- makes no use of words order, punctuation

LSA Steps
3 major steps in LSA [32]

- Prepare documents
- Construct Term-Document matrix D
- Reduce dimensionality of D via SVD

Representation: Term-Document Matrix
Construct a Term-Document Matrix D using the Vector Space Model

- typically rows of D - terms, columns of D - documents/passages,
  - or sometimes, rows of D are documents, columns of D - terms
- not necessarily documents, it can have passages: paragraphs or entire texts
- each cell - typically a frequency with which a word occurs in a doc
- also apply weighting: TF or TF-IDF

SVD and Dimensionality Reduction
Let D be an $t \times p$ Term-Passage matrix

- t rows are terms, p columns are passages, rank $D = r$
- then SVD decomposition is $D = T \cdot \Sigma \cdot P^T$
- T is $t \times r$ Orthogonal Matrix, contains left singular vectors, corresponds to term vectors
- Σ is $r \times r$ a diagonal matrix of singular values
- P is $r \times p$ Orthogonal Matrix, contains right singular vectors, corresponds to passage vectors
- and then $T\sqrt{\Sigma}$ are loadings for terms and $P\sqrt{\Sigma}$ - for passages

Now reduce the dimensionality:

- want to combine the surface text information into some deeper abstraction
- finding the optimal dimensionality for final representation in the Semantic Space is important to properly capture mutual usage of words
- the "True Semantic Space" should address the problems of ambiguity

So, Apply reduced-rank SVD

- $D \approx T_k \cdot \Sigma_k \cdot P_k^T$
- keep only k largest singular values
- the result: best k-dim approximation of the original matrix D
- for NLP $k = 300 \pm 50$ usually works the best

– but it should be tuned because it heavily depends on the domain

Semantic Space
LSA constructs a semantic space via SVD:

– T is $t \times r$ Orthogonal Matrix, contains left singular vectors, corresponds to term vectors
– Σ is $r \times r$ a diagonal matrix of singular values
– P is $r \times p$ Orthogonal Matrix, contains right singular vectors, corresponds to passage vectors
– and then $T\sqrt{\Sigma}$ are loadings for terms and $P\sqrt{\Sigma}$ - for passages

Language-theoretic interpretation:

– LSA vectors approximate:
– the meaning of a word as its average effect of the meaning of passages in which they occur
– the meaning of a passage as meaning of its words

After doing the SVD, we get the reduced space - this is the semantic space
Examples
Let's consider titles of some articles example from [33] [30]
TODO: change it to identifiers!

– c_1: "Human machine interface for ABC computer applications"
– c_2: "A survey of user opinion of computer system response time"
– c_3: "The EPS user interface management system"
– c_4: "System and human system engineering testing of EPS"
– c_5: "Relation of user perceived response time to error measurement"
– m_1: "The generation of random, binary, ordered trees"
– m_2: "The intersection graph of paths in trees"
– m_3: "Graph minors IV: Widths of trees and well-quasi-ordering"
– m_4: "Graph minors: A survey"

Matrix:

$D =$

	c_1	c_2	c_3	c_4	c_5	m_1	m_2	m_3	m_4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Note:

- row vectors for "human" and "user" are orthogonal: their dot product is zero, but they are supposed to be similar, so it must be positive
- also, "human" and "minors" are orthogonal, but they are not similar, so it must be negative

Let's apply SVD:

- $D = W \Sigma P$
- 2-dim approximation: $D_2 = W_2 \Sigma_2 P_2$

$D_2 =$

	c_1	c_2	c_3	c_4	c_5	m_1	m_2	m_3	m_4
human	0.16	0.4	0.38	0.47	0.18	−0.05	−0.12	−0.16	−0.09
interface	0.14	0.37	0.33	0.4	0.16	−0.03	−0.07	−0.1	−0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.7	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	−0.07	−0.15	−0.21	−0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	−0.07	−0.14	−0.2	−0.11
survey	0.1	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	−0.06	0.23	−0.14	−0.27	0.14	0.24	0.55	0.77	0.66
graph	−0.06	0.34	−0.15	−0.3	0.2	0.31	0.69	0.98	0.85
minors	−0.04	0.25	−0.1	−0.21	0.15	0.22	0.5	0.71	0.62

What's the effect of dimensionality reduction here?

- words appear less or more frequent than originally
- consider two cells: ("survey", m_4) and ("trees", m_4)
- original document: 1 and 0
- reduced document: 0.42 and 0.66
- because m_4 contains "graph" and "minor", the 0 for "trees" was replaced by 0.42 - they are related terms
- so it can be seen as estimate of how many times word "trees" would occur in other samples that contain "graph" and "minor"
- the count for "survey" went down - it's not expected in this context

So in the reconstructed space:

- dot product between "user" and "human" is positive
- dot product between "human" and "minors" is negative
- it tells us way better whether terms are similar or not even when they never co-occur together

Taking 2 principal components is the same as taking only 2 abstract concepts

- each word in the vocabulary has some amount of these 2 concepts (we see how much by looking at 1st and 2nd column of W)

The idea:

- we don't want to reconstruct the underlying data perfectly, but instead we hope to find the correlation and the abstract concepts

Latent Semantic Analysis (LSA) $\approx$ Latent Semantic Indexing (LSI)

- LSI is the alias of LSA for Information Retrieval
- indexing and retrieval method that uses SVD to identify patterns in relations between terms and concepts
- instead of literal match between query and documents (e.g. using cosine in the traditional vector space morels), convert both into the Semantic Space and calculate the cosine there

After doing dimenstionality reduction with LSA/SVD apply usual k-means (cite papers where this approach is used)

5.2 Semantic Domains

The theory that connects linguistics and LSA [8]

Formal definition of Semantic Domain, why useful (why Semantic Domain may be better concept for namespaces than Clusters), Domain Spaces (sort of Semantic Spaces), how can be discovered with LSA

5.3 Non-Negative Matrix Factorization

There are many different factorization techniques for document clustering [34]

One of them is NMF [35]

NMF can be directly interpreted as cluster assignment and has been used for document clustering as an alternative to LSA [36]

6 Implementation

In section 6.1 we describe the data set that we use, then we describe how we extract identifiers from this dataset (section 6.2) and how this dataset is cleaned (section 6.3). Next, the implementation of clustering algorithms is described in the section 6.3. After the clusters are found, we combine them into a hierarchy in the section 6.5.

Finally, in the section 6.6 we explore how the same set of techniques can be applied to source code in Java.

6.1 Data set

In this work we apply the discussed techniques to the English version of Wikipedia [37]. It's a big web encyclopedia where articles are written and edited by the community. For our work wikipedia is interesting because there are many math pages.

At present (July 6, 2015) English wikipedia contains about 4.9 mln articles[1] and it is 1.5 Gb in the compressed form. However, only a small portion of these articles are math related: only about 30.000 pages contain at least one `<math>` tag.

The math information is enriched with semantic information by MediaWiki and we use this augmented data representation. 30.000 math pages with augmented math tags occupy around 1.5 Gb in uncompressed form.

Apart from the text data and formulas wikipedia articles have information about categories, which can also be exploited. It is hard to extract category information from the raw wikipedia mark up, but this information is available in a structured form in DBPedia [38].

6.2 Definition Extraction

Before we can proceed to discovering identifier namespaces, we need to extract identifier-definition relations. For this we use the probabilistic approach, discussed in the section 2.5. The extraction process in implemented using Apache Flink [39].

But before the original dataset can be preprocessed, it is enriched with augmented MathML (see section 2.1), and then the dataset is filtered such that only articles with the math tag are retained.

In the wikipedia dataset each document is represented using wiki XML. It makes it easy to extract title and content, and then, the all the formulas

[1] https://en.wikipedia.org/wiki/Wikipedia:Statistics

are extracted from the content. The formulas are extracted by looking for `<math>` tags. However some formulas are typed without the tag, but only with unicode characters. Such formulas are not easy to detect and therefore in this word we choose not to process them. Once all `<math>` tags are found, they (along with the content) are replaced with a special placeholder `FORMULA_%HASH%`, where `%HASH%` is MD5 hash [40] of the tag's content represented as a hexadecimal string. After that the content of the tags is kept separately from the document content.

Once formulas are retrieved, we extract the definitions from them. We are not interested in the semantics of a formula, only in the identifiers it contains. Hence we need only to look for all `<ci>` tags. There are two types of identifiers: simple identifiers such as "t", "C", "μ"; and complex identifiers with subscripts such as "x_1", "ξ_i" or even "β_{slope}". We do not process superscripts because they are usually powers (for example, x^2), and therefore they are not interesting for this work. There are exceptions to this, for example, "σ^2" is an identifier, but these cases are rare and can be ignored.

Since MathML is XML, the identifiers are extracted with XPath queries:

- `//m:mi[not(ancestor::m:msub)]/text()` for all `<ci>` tags that are not subscript identifers
- `//m:msub` for subscript identifiers

Once the identifiers are extracted, the rest of the formula is discarded. As the result, we have a "Bag of Formulas".

The content of a wiki document is structured and authored with a special markup language for specifying document layout elements such as headers, lists, text formatting and tables: Wiki markup. Thus the next step is to process the Wiki markup and extract the textual content of an article, and this is done using a Java library "Mylyn Wikitext" [41]. Almost all annotations are discarded at this stage, and only inner-wiki links are kept: they can be useful as candidate definitions.

Once the markup annotations are removed and the text content of an article is extracted, we then apply Natural Language Processing (NLP) techniques. Thus, the next step is the NLP step, and for NLP we use StanfordNLP [15]. The first part at this stage is to tokenize the text and also split it by sentences. Once it is done, we then apply Math-aware POS tagging (see section 2.4). For identifiers and math formulas we introduce two new POS classes: "`ID`" and "`MATH`", respectively. These classes are not a part of the standard Penn Treebank POS Scheme [14] used by StanfordNLP, therefore we need to label all the instances of these tags ourselves during the

additional post-processing step. If a token starts with "FORMULA_", then we recognize that it is a placeholder for a math formula, and therefore we annotate it with the "MATH" tag. Additionally, if this formula contains only one identifier, this placeholder token is replaced by the identifier and it is tagged with "ID". Additionally, we keep track of all identifiers found in the document and then for each token we check if this token is in the list. If it is, then it is re-annotated with "ID" as well.

At the Wikipedia markup processing step we discard almost all markup annotations, but keep only inter-wiki links, because these links are good definition candidates. To use them, we introduce another POS Tag: "LINK". To detect all inner-wiki links, we first find all token subsequences that start with [[and end with]]. Then these subsequences are concatenated and tagged as "LINK".

Also we are interested in all sequences of successive nouns (both singular and plural) possibly modified by an adjective. We concatenate all such sequences into one token tagged with "NOUN_PHRASE".

Next we select the most probably identifier-definition pairs. At this stage we are interested only in tokens annotated with "LINK" and "NOUN_PHRASE": these tokens are definition candidates, and we rank each token by a score that depends how far it is from the identifer of interest and how far is the closest formula that contains this identifier (see section 2.5). The output of this step is a list of identifier-definition pairs along with the score. Only pairs with scores above the user specified threshold are retained.

The following is the list of the most common identifier-definition pairs:

- t: "time" (1086)
- m: "mass" (424)
- θ: "angle" (421)
- T: "temperature" (400)
- r: "radius" (395)
- v: "velocity" (292)
- ρ: "density" (290)
- G: "group" (287)
- V: "volume" (284)
- λ: "wavelength" (263)
- R: "radius" (257)
- n: "degree" (233)
- r: "distance" (220)
- c: "speed of light" (219)
- L: "length" (216)

– n: “length” (189)
– n: “order” (188)
– n: “dimension” (185)
– n: “size” (178)
– M: “mass” (171)

6.3 Data Cleaning

The Natural Language data is famous for being noisy and hard to clean [42]. The same is true for mathematical identifiers and scientific texts with formulas. In this section we describe how the data was preprocessed and cleaned at different stages of Definition Extraction (section 6.2).

Often identifiers contain additional semantic information visually conveyed by special diacritical marks or font features. For example, the diacritics can be hats to denote “estimates” (e.g. “$\hat{w}$”), bars to denote the expected value (e.g. “$\bar{X}$”), arrows to denote vectors (e.g. “$\overrightarrow{x}$”) and others. As for the font features, boldness is often used to denote vectors (e.g. “$\mathbf{w}$”) or matrices (e.g. “$\mathbf{X}$”), calligraphic fonts are used for sets (e.g. “$\mathcal{H}$”), double-struck fonts often denote spaces (e.g. “$\mathbb{R}$”), etc. Unfortunately there is no common notation established across all fields of mathematics and there is a lot of variance. For example, a vector can be denoted by “$\overrightarrow{x}$”, “$\boldsymbol{x}$” or “$\mathbf{x}$”, and a real line by “$\mathbb{R}$”, “$\mathbf{R}$” or “$\mathfrak{R}$”. Therefore we discard all this additional information, such that “$\bar{X}$” becomes “X”, “$\mathbf{w}$” becomes “w” and “$\mathfrak{R}$” becomes “R”.

The diacritic marks can easily be discarded because they are represented by special MathML instructions that easily can be ignored (see the section 2.1 for details). But, on the other hand, the visual features are encoded directly on the character level: the identifiers use special unicode symbols to convey font features such as boldness or Fraktur, so it needs to be normalized by converting characters from special “Mathematical Alphanumeric Symbols” unicode block [43] back to the standard ASCII positions (“Basic Latin” block).

Additionally, the is a lot of noise on the annotation level in MathML formulas: many non-identifiers are captures as identifiers inside `<ci>` tags. Among them there are many mathematic-related symbols like “$\hat{}$”, “#”,“$\forall$”, “$\int$”; miscellaneous symbols like “$\diamond$” or “$\circ$”, arrows like “$\rightarrow$” and “$\Rightarrow$”, and special characters like “$\lceil$”.

To filter out these one-symbol false identifiers we fully exclude all characters from the following unicode blocks: “Spacing Modifier Letters”, “Mis-

cellaneous Symbols", "Geometric Shapes", "Arrows", "Miscellaneous Technical", "Box Drawing", "Mathematical Operators" (except "∇" which is sometimes used as an identifier) and "Supplemental Mathematical Operators" [43]. Some symbols (like "=", "+", "~", "%", "?", "!") belong to commonly used unicode blocks which we cannot exclude altogether. For these symbols we manually prepare a stop list for filtering them.

It also captures multiple-symbol false positives: operators and functions like "`sin`", "`cos`", "`exp`", "`max`", "`trace`"; words commonly used in formulas like "`const`", "`true`", "`false`", "`vs`", "`iff`"; English stop words like "`where`", "`else`", "`on`", "`of`", "`as`", "`is`"; units like "`mol`", "`dB`", "`mm`". These false identifiers are excluded by a stop list as well: if a candidate identifier is in the list, it is filtered out.

Then, at the next stage, the definitions are extracted. However many shortlisted definitions are either not valid definitions or too general. For example, some identifiers become associated with "`if and only if`", "`alpha`", "`beta`", "`gamma`", which are not valid definitions. Other definitions like "`element`", "`number`" or "`variable`" are valid, but they are too general and not descriptive. We maintain a stop list of such false definitions and filter them out from the result.

The next stage is using identifier/defintion pairs for document clustering We can note that if some definition is used only once throughout the entire data set, it is not useful because it does not have any discriminative power. Therefore all such definitions are excluded.

6.4 Document Clustering

At the Document Clustering stage we want to find cluster of documents that are good namespace candidates.

Before we can do this, we need to vectorize our dataset: i.e. build the Identifier Space (see section 3.3) and represent each document in this space.

There are three choices for dimensions of the Identifier space:

- identifiers alone
- "weak" identifier-definition association
- "strong" association: use identifier-definition pairs

In the first case we are only interested in identifier information and discard the definitions altogether.

In the second and third cases we keep the definitions and use them to index the dimensions of the Identifier Space. Bur there is some variability in

the definitions: for example, the same identifier "σ" in one document can be assigned to "Cauchy stress tensor" and in other it can be assigned to "stress tensor", which are almost the same thing. To reduce this variability we perform some preprocessing: we tokenize the definitions and use individual tokens to index dimensions of the space. For example, suppose we have two pairs (σ, "Cauchy stress tensor") and (σ, "stress tensor"). In the "weak" association case we have will dimensions (σ, Cauchy, stress, tensor), while for the "strong" association case we will have (σ_Cauchy, σ_stress, σ_tensor).

Additionally, the effect of variability can be decreased further by applying a stemming technique for each definition token. In this work we use Snowball stemmer for English [44] implemented in NLTK [45]: a python library for Natural Language Processing.

Each document is vectorized (converted to a vector form) by using `TfidfVectorizer` from scikit-learn [46]. We use the following settings:

- `use_idf=True, min_df=2`
- `use_idf=False, min_df=2`
- `use_idf=False, sublinear_tf=True, min_df=2`

In the first case we use inverse document frequency (IDF) to assign additional collection weight for "terms" (see section 3.2), while in second and in third we use only term frequency (TF). In the second case we apply a sublinear transformation to the TF component to reduce the influence of frequently occurring words. In all three cases we keep only "terms" that are used in at least two documents.

The output is a document-identifier matrix (analogous to "document-term"): documents are rows and identifiers/definitions are columns. The output of `TfidfVectorizer` is row-normalized, i.e. all rows has unit length.

Once we the documents are vectorized, we can apply clustering techniques to them. We use K-Means (class `KMeans` in scikit-learn) and Mini-Batch K-Means (class `MiniBatchKMeans`) [46]. Note that if rows are unit-normalized, then running k-means with Euclidean distance is equivalent to cosine distance (see section 4.2).

Bisecting K-Means (see section 4.3) was implemented on top of scikit-learn: at each step we take a subset of the dataset and apply K-means with $k = 2$ to this subset. If the subset is big (with number of documents $N > 2000$), then we use Mini-Batch K-means with $k = 2$ because it converges much faster.

Scatter/Gather extensions to K-means (see section 4.3) was implemented manually using scipy [47] and numpy [48] because scikit-learn's implementation of K-Means does not allow using user-defined distances.

DBScan (section 4.4) and SNN Clustering (section 4.5) algorithms were also implemented manually: available DBScan implementations usually take distance measure rather than a similarity measure. The similarity matrix cleated by similarity measures are typically very sparse, because usually only a small fraction of the documents are similar to some given document. Similarity measures can be converted to distance measures, but in this case the matrix will no longer be sparse, and we would like to avoid that. Additionally, available implementations are usually general purpose implementations and do not take advantage of the structure of the data: in text-like data clustering algorithms can be sped up significantly by using an inverted index (section 3.5)

Dimensionality reduction techniques are also important: they not only reduce the dimensionality, but also help reveal the latent structure of data. In this work we use Latent Semantic Analysis (LSA) (section 5.1) which is implemented using randomized Singular Value Decomposition (SVD) [49], The implementation of randomized SVD is taken from scikit-learn [46] - method `randomized_svd`. Non-negative Matrix Factorization is an alternative technique for dimensionality reduction (section 5.3). Its implementation is also taken from scikit-learn [46], class `NMF`.

To assess the quality of produced clusters we use wikipedia categories. It is quite difficult to extract category information from raw wikipedia text, therefore we use DBPedia [38] for that: it provides machine-readable information about categories for each wikipedia article. Additionally, categories in wikipedia form a hierarchy, and this hierarchy is available as a SKOS ontology.

A cluster is said to be "pure" if all documents have the same category. Using categories information we can find the most frequent category of the cluster, and then we can define purity as

$$\text{purity}(C) = \frac{\max_i \text{count}(c_i)}{|C|}$$

(**TODO**: Add backlink to purity definition).

Then we can calculate the overall purity of a cluster assignment and use this to compare results of different clustering algorithms. However it is not enough just to find the most pure cluster assignment: because as the number

of clusters increases the overall purity also grows. Thus we can also optimize for the number of clusters with purity p of size at least n.

When the number of clusters increase, the purity always grows, but at some point the number of pure clusters will start decreasing.

(**TODO**: Add a graph to show the tradeoff between purity and the number of clusters)

6.5 Building Hierarchy

Once

AMS Mathematics Subject Classification (2010) [50] Excluded all subcategories those code end with '99': they are usually 'Miscellaneous topics' or 'None of the above, but in this section'. top level categories 'General', 'History and biography', and 'Mathematics education' were also excluded. Additionally we exclude the following:

- Quantum theory → Axiomatics, foundations, philosophy
- Quantum theory → Applications to specific physical systems
- Quantum theory → Groups and algebras in quantum theory
- Partial differential equations → Equations of mathematical physics and other areas of application
- Statistics → Sufficiency and information
- Functional analysis → Other (nonclassical) types of functional analysis
- Functional analysis → Miscellaneous applications of functional analysis

So these categories do not interfere with PACS.

APS Physics and Astronomy Classification Scheme (2010) [51]

We remove the "GENERAL" top-level category. In PACS there are 3 levels of categories, but we merge all 3-rd level categories into 2nd level.

ACM Classification Scheme [52] available as a SKOS [53] ontology at their website [54]. The SKOS ontology graph was processed with RDFLib [55]

We keep the following top level categories: "Hardware", "Computer systems organization", "Networks", "Software and its engineering", "Theory of computation", "Information systems", "Security and privacy", "Human-centered computing", "Computing methodologies".

After obtaining the data and parsing, all categories, the hierarchies are merged into one and then we try to match the found namespaces with second-level category in the hierarchy.

This is done by keywords matching: we extract all words from the category (this includes top level category name, subcategory name and all subsub categories concatenated). From the cluster we also extract the category information. Then we try to do keyword matching using cosine similarity between the cluster and each category. The cluster is assigned to the category with the best cosine.

If the cosine score is low (below 0.2) or there is only one keyword matched, then the cluster is assigned to the "OTHERS" category.

6.6 Java Language Processing

The same set of techniques can be applied to source code. (todo: why!) If a language is statically typed, like Java or Pascal, usually it is possible to know the type of a variable from its declaration. Therefore we can see variables as identifiers and types as "definitions" (TODO: clearly state the difference between types and definitions).

To extract this information from some source code repository each file source file can be processed to obtain its Abstract Syntax Tree, and then declaration information can be extracted from it.

In this word we process Java source code using JavaParser [56] - a library for parsing java source code. Java was chosen because the variable types always have to be declared, unlike other languages where the type can be inferred by compilers.

The following declarations are processed: fields of a class, method and constructor parameters, inner variable declarations inside methods and constructors. It processes both usual classes and inner classes.

Add example and the results

In the experiments we process Apache Mahout source code [57].

Describe the dataset

7 Evaluation

In section 7.1 we describe how we select the best clustering algorithm.

7.1 Parameter Tuning

We have the following parameters

Way to incorporate definition information (3 ways): no identifier, soft association, hard association

Weighting schemes: TF-IDF, TF, log TF

Clustering algorithm

DBSCAN, SNN Clustering: params: min_pts, epsilon K-Means, Bisecting K-Means: params: k

Distance and similarity measures used Euclidean distance, cosine similarity, jaccard similarity, SNN Similarity

Ways to reduce dimensionality SVD $D \approx D_k = U\Sigma_k V$, param: k what's the rank of the approximation matrix NNMF $D \approx D_k = U_k V_k^T$ param: k number of columns in U and V - also the rank of D_k

The approach for finding the best parameter set is a grid search: different combination are tries and the best result is kept.

Agglomerative: Wald linkage: takes forever never finished

Only identifiers

Usual K-Means

some clusters are useful, but most of them aren't

For example

APL (programming language) nOR Binary search tree On Boolean satisfiability problem On Complexity On Earley parser On Heapsort $On\Omega$ Lisp (programming language) On Priority queue On Sieve of Eratosthenes On Smoothsort On Comb sort ΩnpO Divide and conquer algorithm $On\Omega$ Stack (abstract data type) Ont Skip list npO Graph minor Onh Flex lexical analyser On Gift wrapping algorithm Onh Perlin noise nO Pseudo-polynomial time nOm Hirzebruch surface $Onmp$ Beap nO Pairing heap $On\Omega$ Cost efficiency Onp Marcus Hutter nO Simplex noise On Vizing's theorem Omn Graph traversal nO AC (complexity) On AdlemanPomeranceRumely primality test Onc O (disambiguation) O Warnock algorithm Onp Oddeven sort On Database storage structures On Introselect On Burstsort On Asymptotic computational complexity On Indirect DNA damage $ChromophoreOintactDNAdamaged$ Frer's algorithm $On\Omega$ Even-hole-free graph On Range Minimum Query njO Control table O Zoltn Fredi On

Fenwick tree Ond Constraint graph (layout) On Bubble sort On TC (complexity) On Primitive abundant number On Algorithmic complexity attack On Symbols for zero O Proxmap sort Onc TheilSen estimator On Brodal queue On Alias method On

DBSCAN SNN

k = 10 dist = jaccard

eps=7 points min_point=5 points

Epsilon Eridani in fiction $M_\odot$ Solar mass $M_\odot$ Orders of magnitude (mass) $M_\odot$ Carbon-burning process $M_\odot$ Baryonic dark matter $M_\odot$ PSR J16142230 $M_\odot$ KennicuttSchmidt law $M_\odot$ Portal:Star/Selected article/19 $M_\odot$ NGC 6166 $M_\odot$ Celestial Snow Angel $M_\odot$ Huge-LQG $M_\odot$ High-velocity cloud $M_\odot$ NGC 4845 $M_\odot$ Pulsating white dwarf $M_\odot$ Robust associations of massive baryonic objects $M_\odot$ Black Widow Pulsar $M_\odot$ Betelgeuse $M_\odot$ Andromeda Galaxy $M_\odot$

WEAK ASSOCIATION

K-Means weak association

DBSCAN k=15, eps=8, min_pts=5

Papyrus 66 Ppapyrus Alexandrian text-type Ppapyrus Western text-type Ppapyrus Codex Ephraemi Rescriptus Ppapyrus Bodmer Papyri Ppapyrus Categories of New Testament manuscripts Ppapyrus Papyrus 4 Ppapyrus Papyrus 75 Ppapyrus Uncial 0308 $PMpapyrus47$ Codex Athous Lavrensis Ppapyrus Papyrus 92 Ppapyrus Papyrus 90 Ppapyrus Papyrus 9 Ppapyrus Papyrus 15 Ppapyrus Papyrus 16 Ppapyrus Papyrus 20 Ppapyrus Papyrus 39 Ppapyrus Papyrus 49 Ppapyrus Papyrus 65 Ppapyrus Papyrus 111 Ppapyrus Uncial 0243 Ppapyrus Minuscule 1739 Ppapyrus Minuscule 88 Ppapyrus Authorship of the Epistle to the Hebrews Ppapyrus Egerton Gospel Ppapyrus Rylands Library Papyrus P52 Ppapyrus Codex Vaticanus Ppapyrus

K-Means

Direct shear test $angle\varphi friction$ Truncated dodecadodecahedron $goldenratio\phi$ Golden triangle (mathematics) $goldensection\theta\phi$ Petrophysics $percentage\varphi symbol S_w$ Greedy algorithm for Egyptian fractions $goldentermsd\phi denominatorpossibleexpansionyxratio$ Pi Josephson junction $\pi junction\varphi$ Snub dodecahedron $goldenratio\tau 2\xi - V\xi$ Lucas number $goldentermsd\phi mLnumbervaluesratioL_knL_nF_nL_m$ 54 (number) $golden\varphi ratio$ Special right triangles $\pi cbratioc.\phi mgoldenradiansnlineintegerssegment$ Universal code (data compression) $goldencoderatiopower\phi lnqplaw$ Existential instantiation $csymbol\varphi variable$ Perles configuration $goldenratio\phi$ Wythoff array $goldenratio\phi columnmnumbern\varphi fibonacci$ Golomb sequence $a_n ngoldenratio\phi$ Almost integer $constant\pi gelfondsexample$ 16:10 $goldenratio\phi$ Leonardo number $goldenratio\phi computationsnLnroots$ RogersRamanujan continued

fraction $golden functions ratio G \phi H q modular$ Great rhombic triacontahedron $golden ratio \phi$ 108 (number) $golden \varphi ratio$ Random Fibonacci sequence $golden B ratio M_n probability$ Rhombic triacontahedron $golden ratio \phi r_i Sedger_m V$ Exact trigonometric constants $function \pi ratio \phi image golden values x V \theta example$ Bilunabirotunda $golden ratio \phi$ Feigenbaum constants $map z ratio places f \phi golden c_n \alpha \delta variable x x_i$

Batch, k = 2500 .. 10000 with step 50 SVD with n=600

7.2 Building Hierarchy

How to evaluate???

7.3 Result analysis

7.4 experiment conclusions

8 Conclusions

8.1 Future Work

While I was reading, it occurred to me that one document may have several namespaces, but I'm not yet sure how to model it. Maybe by taking a small context around each formula and treating it as a small document? Or I should hold this thought and wait till we maybe obtain some results from other simpler methods?

Use semi-supervised techniques for evaluation

Pages that describe certain namespaces may be quite interconnected. There are link-based clustering methods e.g. Botafogo and Schneiderman 1991

Can extract wiki graph and use this for clustering . There are hybrid approaches that use both usual textual representation + links [58]

It can be interesting to apply these tecnhiques to a larger dataset, for example, arXiv.

9 Bibliography

References

1. Erik Duval, Wayne Hodgins, Stuart Sutton, and Stuart L Weibel. Metadata principles and practicalities. *D-lib Magazine*, 8(4):16, 2002.
2. Henry Thompson, Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (third edition). W3C recommendation, W3C, December 2009. http://www.w3.org/TR/2009/REC-xml-names-20091208/.
3. Craig Larman. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Pearson Education India, 2005.
4. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java 8 Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, 2014.
5. Jon Barwise, John Etchemendy, Gerard Allwein, Dave Barker-Plummer, and Albert Liu. *Language, proof and logic*. CSLI publications, 2000.
6. Rober Pagael and Moritz Schubotz. Mathematical language processing project. *arXiv preprint arXiv:1407.0167*, 2014.
7. LM Vassilyev. The theory of semantic fields: A survey. *Linguistics*, 12(137):79–94, 1974.
8. Alfio Gliozzo and Carlo Strapparava. *Semantic domains in computational linguistics*. Springer Science & Business Media, 2009.
9. Deborah L McGuinness, Frank Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.
10. Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.
11. Ulf Schöneberg and Wolfram Sperber. POS tagging and its applications for mathematics. In *Intelligent Computer Mathematics*, pages 213–223. Springer, 2014.
12. Giovanni Yoko Kristianto, MQ Ngiem, Yuichiroh Matsubayashi, and Akiko Aizawa. Extracting definitions of mathematical expressions in scientific papers. In *Proc. of the 26th Annual Conference of JSAI*, 2012.
13. David Carlisle, Robert R Miner, and Patrick D F Ion. Mathematical markup language (MathML) version 3.0 2nd edition. W3C recommendation, W3C, April 2014. http://www.w3.org/TR/2014/REC-MathML3-20140410/.
14. Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project (3rd revision). 1990.
15. Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.
16. Mihai Grigore, Magdalena Wolska, and Michael Kohlhase. Towards context-based disambiguation of mathematical expressions. In *The Joint Conference of ASCM*, pages 262–271, 2009.
17. Keisuke Yokoi, Minh-Quoc Nghiem, Yuichiroh Matsubayashi, and Akiko Aizawa. Contextual analysis of mathematical expressions for advanced mathematical search. In *Prof. of 12th International Conference on Intelligent Text Processing and Comptational Linguistics (CICLing 2011), Tokyo, Japan, February*, pages 20–26, 2011.
18. Jerzy Trzeciak. *Writing mathematical papers in English: a practical guide*. European Mathematical Society, 1995.
19. Minh Nghiem Quoc, Keisuke Yokoi, Yuichiroh Matsubayashi, and Akiko Aizawa. Mining coreference relations between formulas and text using wikipedia. In *23rd International Conference on Computational Linguistics*, page 69, 2010.

20. Giovanni Yoko Kristianto, Akiko Aizawa, et al. Extracting textual descriptions of mathematical expressions in scientific papers. *D-Lib Magazine*, 20(11):9, 2014.
21. Minh-Quoc Nghiem, Giovanni Yoko, Yuichiroh Matsubayashi, and Akiko Aizawa. Towards mathematical expression understanding. 2012.
22. Douglass R Cutting, David R Karger, Jan O Pedersen, and John W Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329. ACM, 1992.
23. Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. pages 83–103, 2004.
24. Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.
25. Mark Hall, Paul Clough, and Mark Stevenson. Evaluating the use of clustering for automatically organising digital library collections. In *Theory and Practice of Digital Libraries*, pages 323–334. Springer, 2012.
26. David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.
27. Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–22. ACM, 1999.
28. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.
29. Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *SDM*, pages 47–58. SIAM, 2003.
30. Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.
31. Stanisław Osiński, Jerzy Stefanowski, and Dawid Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent information processing and web mining*, pages 359–368. Springer, 2004.
32. Nicholas Evangelopoulos, Xiaoni Zhang, and Victor R Prybutok. Latent semantic analysis: five methodological recommendations. *European Journal of Information Systems*, 21(1):70–86, 2012.
33. Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.
34. Stanisław Osiński. Improving quality of search results clustering with approximate matrix factorisations. In *Advances in Information Retrieval*, pages 167–178. Springer, 2006.
35. Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
36. Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.
37. Wikimedia Foundation. English wikipedia XML data dump, 2015. http://dumps.wikimedia.org/enwiki/latest/, accessed: TODO.
38. Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, and et al. DBpedia - a crystallization point for the web of data. *Web Semant.*, 7(3):154–165, September 2009.
39. Apache Software Foundation. Apache Flink 0.8.1. http://flink.apache.org/, accessed: 2015-01-01.
40. Ronald Rivest. The MD5 message-digest algorithm. 1992.
41. Eclipse Foundation. Mylyn WikiText 1.3.0, 2015. http://projects.eclipse.org/projects/mylyn.docs, accessed: 2015-01-01.

42. Daniel Sonntag. Assessing the quality of natural language text data. In *GI Jahrestagung (1)*, pages 259–263, 2004.
43. Julie D Allen et al. *The Unicode Standard*. Addison-Wesley, 2007.
44. Martin F Porter. Snowball: A language for stemming algorithms, 2001.
45. Steven Bird. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.
46. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
47. Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. http://www.scipy.org/, accessed: 2015-02-01.
48. S. van der Walt, S.C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.
49. A Tropp, N Halko, and PG Martinsson. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical report, Technical Report, 2009.
50. American Mathematical Society. AMS mathematics subject classification 2010, 2009. http://msc2010.org/, accessed: 2015-06-01.
51. American Physical Society. PACS 2010 regular edition, 2009. http://www.aip.org/publishing/pacs/pacs-2010-regular-edition/, accessed: 2015-06-01.
52. Bernard Rous. Major update to acm's computing classification system. *Commun. ACM*, 55(11):12–12, November 2012.
53. Alistair Miles, Brian Matthews, Michael Wilson, and Dan Brickley. SKOS Core: Simple knowledge organisation for the web. In *Proceedings of the 2005 International Conference on Dublin Core and Metadata Applications: Vocabularies in Practice*, DCMI '05, pages 1:1–1:9. Dublin Core Metadata Initiative, 2005.
54. Association for Computing Machinery. ACM computing classification system., 2012. https://www.acm.org/about/class/2012, accessed: 2015-06-21.
55. Daniel Krech. RDFLib 4.2.0. https://rdflib.readthedocs.org/en/latest/, accessed: 2015-06-01.
56. Sreenivasa Viswanadha, Danny van Bruggen, and Nicholas Smith. JavaParser 2.1.0, 2015. http://javaparser.github.io/javaparser/, accessed: 2015-06-15.
57. Apache Software Foundation. Apache Mahout 0.10.1. http://mahout.apache.org/, accessed: 2015-06-15.
58. Nora Oikonomakou and Michalis Vazirgiannis. A review of web document clustering approaches. In *Data mining and knowledge discovery handbook*, pages 921–943. Springer, 2005.