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 Similarity Measures and Distances

Jaccard similarity, jaccard distance

Dot product, cosine similarity, cosine distance

Euclidean distance, relationship between cosine and Euclidean

3.3 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

3.4 Inverted Index

3.5 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)

4 Document Clustering Techniques

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

Types of clustering techniques:

- hierarchical
- partitioning
- density-based
- etc

4.1 Hierarchical clustering

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

4.2 K-Means

Often K-Means shows the best results [24] [23], but it takes long for large collections.

Solution: minibatch K-means (web-scale k-means clustering paper) [25]

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

4.3 Extensions of K-Means

Scatter/Gather [22]
Bisecting K-Means [23]
Center adjustment (vector average dumping) [26]
Smart seed selection [22] [26]

4.4 DBSCAN

describe the algo [27]
pseudocode
Can be adapted to take similarity measure instead of distance

4.5 Extensions of DBSCAN

SNN Clustering: [28]
Different similarity measure
Applicable to document clustering and discovering topics [29]

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]
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 [31]

- 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 [32] [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 [33]

One of them is NMF [34]

NMF can be directly interpreted as cluster assignment and has been used for document clustering as an alternative to LSA [35]

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 [36]. 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 1, 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 [37].

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 [38].

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 [39] 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" [40]. 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 [41]. 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 filed 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 [42] 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”, “Miscellaneous Symbols”, “Geometric Shapes”, “Arrows”, “Miscellaneous Technical”, “Box Drawing”, “Mathematical Operators” (except “∇” which is

sometimes used as an identifier) and "Supplemental Mathematical Operators" [42]. 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.

6.4 Document Clustering

First step is discarding all definitions (from each id-pair definitions) that occur only once and therefore do not have much discriminating power.

Then we need to build an Identifier Space (see section 3.5). There are three ways of doing it.

- keeping identifiers alone
- "weak identifier-definition association": putting identifiers along with definitions
- identifier-definition pairs

In the first case we use only identifier information - and discard the definitions altogether.

In the second and third cases we keep the definitions. But there is some variability in the definitions. For example, the same identifier can be assigned to "Cauchy stress tensor" and "stress tensor" - which are almost the same thing. To reduce this variability we tokenize the definition and use individual tokens as dimensions of the space. For example, if we have two pairs (σ, "Cauchy stress tensor") and (σ, "stress tensor"), for second case we have dimensions $(\sigma, \text{Cauchy}, \text{stress}, \text{tensor})$, while for the third case we have $(\sigma\_\text{Cauchy}, \sigma\_\text{stress}, \sigma\_\text{tensor})$.

Additionally, to decrease the effect of variability further, a stemming technique is applied to each definition token. We use Snowball stemmer for English [43] implemented in NLTK [44]: a python library for Natural Language Processing.

Each document is vectorized (converted to a vector form) by using `TfidfVectorizer` from scikit-learn [45]. We use the following settings:

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

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.

K-Means and Mini-Batch K-Means implementation from scikit-learn [45]. Classes `KMeans`, `MiniBatchKMeans`

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: each time a data set is clustered with k-means with $k = 2$. if the subset being clustered is big, ($N > 2000$) then mini-batch k-means is used, otherwise usual K-means

Scatter/Gather (see section 4.3) was implemented manually using scipy [46] and numpy [47] 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, but we want to use a similarity matrix, and not a distance matrix because converting would make the matrix dense, and we want to avoid that. Additionally, available implementation do not use the structure of data: in text-like data clustering algorithms can be sped up by using an inverted index (section 3.4)

LSA (section 5.1) is implemented via randomized SVD [48], The implementation is taken from scikit-learn [45] - method `randomized_svd`.

Non-negative Matrix Factorization (section 5.3) is also taken from cikit-learn [45], class `NMF`.

To assess the quality of clustering we used wikipedia categories. It is difficult to extract category information from raw wikipedia text, therefore we used DBPedia [37] for that. Additionally DBPedia provides a SKOS ontology about categories.

6.5 Building Hierarchy

AMS Mathematics Subject Classification (2010) [49] 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) [50]

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

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 [55] - 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 [56].

Describe the dataset

7 Evaluation

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 [57]

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. 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.
24. 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.
25. David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.
26. 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.
27. 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.
28. 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.
29. Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. pages 83–103, 2004.
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. Nicholas Evangelopoulos, Xiaoni Zhang, and Victor R Prybutok. Latent semantic analysis: five methodological recommendations. *European Journal of Information Systems*, 21(1):70–86, 2012.
32. Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.
33. Stanislaw Osinski. Improving quality of search results clustering with approximate matrix factorisations. In *Advances in Information Retrieval*, pages 167–178. Springer, 2006.
34. Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
35. 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.
36. Wikimedia Foundation. English wikipedia XML data dump, 2015. http://dumps.wikimedia.org/enwiki/latest/, accessed: TODO.
37. 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.
38. Apache Software Foundation. Apache Flink 0.8.1. http://flink.apache.org/, accessed: 2015-01-01.
39. Ronald Rivest. The md5 message-digest algorithm. 1992.
40. Eclipse Foundation. Mylyn WikiText 1.3.0, 2015. http://projects.eclipse.org/projects/mylyn.docs, accessed: 2015-01-01.
41. Daniel Sonntag. Assessing the quality of natural language text data. In *GI Jahrestagung (1)*, pages 259–263, 2004.
42. Julie D Allen et al. *The Unicode Standard*. Addison-Wesley, 2007.

43. Martin F Porter. Snowball: A language for stemming algorithms, 2001.
44. 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.
45. 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.
46. Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. http://www.scipy.org/, accessed: 2015-02-01.
47. 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.
48. A Tropp, N Halko, and PG Martinsson. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical report, Technical Report, 2009.
49. American Mathematical Society. AMS mathematics subject classification 2010, 2009. http://msc2010.org/, accessed: 2015-06-01.
50. American Physical Society. PACS 2010 regular edition, 2009. http://www.aip.org/publishing/pacs/pacs-2010-regular-edition/, accessed: 2015-06-01.
51. Bernard Rous. Major update to acm's computing classification system. *Commun. ACM*, 55(11):12–12, November 2012.
52. 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.
53. Association for Computing Machinery. ACM computing classification system., 2012. https://www.acm.org/about/class/2012, accessed: 2015-06-21.
54. Daniel Krech. RDFLib 4.2.0. https://rdflib.readthedocs.org/en/latest/, accessed: 2015-06-01.
55. Sreenivasa Viswanadha, Danny van Bruggen, and Nicholas Smith. JavaParser 2.1.0, 2015. http://javaparser.github.io/javaparser/, accessed: 2015-06-15.
56. Apache Software Foundation. Apache Mahout 0.10.1. http://mahout.apache.org/, accessed: 2015-06-15.
57. Nora Oikonomakou and Michalis Vazirgiannis. A review of web document clustering approaches. In *Data mining and knowledge discovery handbook*, pages 921–943. Springer, 2005.