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

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].

XML (eXtensible Markup Language) is a framework for defining markup languages. XML lets users define a set of tags to represent information in some specific domain [2]. For example, XHMTL is an XML language for hypertext markup and MathML is a language for describing mathematical notation.

However, different XML languages may use the same names for elements and attributes. For example, consider two XML languages: XHTML for specifying the layout of web pages, and some XML language for describing furniture. Both these languages have the `<table>` elements there, in XHTML table is used to present some data in a tabular form, while the second one uses it to describe a particular piece of furniture in the database.

The `<table>` elements have very different semantics in these languages and there should be a way to distinguish between these two elements. In XML this problem is solved with XML namespaces [3]: the namespaces are used to ensure the uniqueness of attributes and resolve ambiguity. It is done by binding a short namespace alias with some uniquely defined URI (Unified Resource Identifier), and then appending the alias to all attribute names that come from this namespace. In the example above, we can bind an alias `h` with XHTML's URI `http://www.w3.org/TR/xhtml1` and then use `<h:table>` to refer to XHTML's table. Likewise, in the furniture database language the element names can be prepended with a prefix `d`, where `d` is bound to some URI, e.g. `http://www.furniture.de/2015/db`.

The namespaces are also used in programming languages for organizing variables, procedures and other identifiers into groups and for resolving name collisions. In programming languages without namespaces the programmers have to take special care to avoid naming conflicts. For example, in the PHP programming language prior to version 5.3 [4] there is no notion of namespace, and the namespaces have to be emulated to ensure that the names are unique, and this often results in long names like `Zend_Search_Lucene_Analysis_Analyzer`[1].

[1] http://framework.zend.com/apidoc/1.7/Zend_Search_Lucene/Analysis/Zend_Search_Lucene_Analysis_Analyzer.html

Other programming languages have the notion of namespaces built in from the very first versions. For example, the Java programming language [5] uses packages to organize identifiers into namespaces. In Java, packages solve the problem of ambiguity. For example, in the standard Java API there are two classes with the name `Date`: one in the package `java.util` and another in the package `java.sql`. To be able to distinguish between them, the classes are referred by their *fully qualified name*: an unambiguous name that uniquely specifies the class by combining the package name with the class name. Thus, to refer to a particular `Date` class in Java `java.util.Date` or `java.sql.Date` should be used.

It is not always convenient to use the fully qualified name in the code to refer to some class from another package. Therefore in Java it is possible to *import* the class by using the import statement which associates a short name alias with its fully qualified name. For example, to refer to `java.sql.Date` it is possible to import it by using `import java.sql.Date` and then refer to it by the alias `Date` in the class [5].

Although there is no strict requirement to organize the classes into well defined groups, it is a good software design practice to put related objects into the same namespace and by doing this achieve better modularity. There are design principles that tell software engineers how to best organize the source code: classes in a well designed system should be grouped in such a way that namespaces exhibit low *coupling* and high *cohesion* [6]. Coupling describes the degree of dependence between namespaces, and low coupling means that the interaction between classes of different namespaces should be as low as possible. Cohesion, on the other hand, refers to the dependence within the classes of the same namespace, and the high cohesion principle says that the related classes should all be put together in the same namespace.

1.2 Namespaces in Mathematical Notation

Informally, a mathematical formula is a rule that shows the relationship between different variables. For example, $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ is a formula for solving a quadratic equation $ax^2 + bx + c = 0$.

To give a more formal definition of formula, we first need to define a first-order language that contains primitive symbols such as (1) parentheses, brackets and other boundary symbols (2) *constants* (1, 2, 3, ...) and variables (x, y, ...) (3) *functions* ($+$, $\cdot$, ...) and (4) *predicates*, e.g. binary relation symbols ("$=$", "$<$", "$\geqslant$", ...) [7].

In this language, *constants* are symbols with pre-defined meaning from some alphabet and variables are symbols that can be assigned a value from this alphabet. Any symbol can be a variable, for example, x, y, $\mathbf{w}$, or it can be a symbol with subscripts, for example, $x_1, x_2, \ ...$ or even w_{slope}.

A *well-formed term* t (or just *term*) in this language is defined as

$$t \equiv c \mid x \mid f(t_1, t_2, \ ... \ , t_n) \ ,$$

which means that the term t can be a constant, a variable or an n-ary function $f(t_1, t_2, \ ... \ , t_n)$. An *$n$-ary function* is an function that takes n terms $t_1, t_2, \ ... \ , t_n$ and produces some new term. An *n-ary predicate* (or an *n-ary relation symbol*) is typically a boolean-valued function that can be evaluated to `True` or `False` depending on the values it gets.

Then an *atomic well-formed formula* (or just *formula*) in this language is a n-ary predicate with n terms evaluated to `True` [7].

For example, $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ is a formula, because it represents an equation that always holds true for a quadratic equation $ax^2 + bx + c = 0$. The equality symbol "=" is a predicate that shows the relationship between variables x_1, x_2 and variables a, b, c.

In logic we can use any symbol for variables without changing the meaning of the formula. For example, the energy-mass equivalence relation $E = mc^2$ can be written as $x = yz^2$ and it still will still hold true and remain a valid formula. However, there are research communities in mathematics that have developed a special system of naming these variables, and these naming systems are called *mathematical notations* [8]. For each symbol in a formula, the notation assigns a precise semantic meaning. Therefore, because of the notation, in Physics it is more common to write $E = mc^2$ rather than $x = yz^2$, because the notation assigns unambiguous meaning to the symbols "E", "m" and "c", and the meaning of these symbols is recognized among physicists.

However the notations may conflict. For example, while it is common to use symbol E to denote "Energy" in Physics, it also is used in Probability and Statistics to denote "Expected Value", or in Linear Algebra to denote "Elimination Matrix". We can compare the conflict of notations with the name collision problem in namespaces, and solve it by extending the notion of namespaces to mathematical notation.

Thus, let us define a *notation* $\mathcal{N}$ as a set of pairs $\{(i, s)\}$, where i is a symbol and s is its semantic meaning, such that for any pair $(i, s) \in \mathcal{N}$ there does not exist a pair $(i', s') \in \mathcal{N}$ with $i = i'$. Let us call i *identifier* and s *definition*, and then (i, s) is an *identifier-definition pair*. Two notations $\mathcal{N}_1$

and $\mathcal{N}_2$ *conflict*, if there exists a pair $(i_1, s_1) \in \mathcal{N}_1$ and a pair $(i_2, s_2) \in \mathcal{N}_2$ such that $i_1 = i_2$ and $s_1 \neq s_2$.

Then we can define *namespace* as a named notation, i.e. a pair $(\text{uid}, \mathcal{N})$ where $\mathcal{N}$ is a notation and "uid" is a some string that uniquely identifies the notation. For convenience, in this work we can use the Java syntax to refer to specific entries of a namespace. If $(\text{name}, \mathcal{N})$ is a namespace and i is an identifier such that $(i, s) \in \mathcal{N}$ for some s, then "name.i" is a *fully qualified name* of the identifier i that relates it to the definition s. For example, given a namespace $\big($"Physics", $\{(E,$"energy"$)$, $(m,$"mass"$)$, $(c,$"speed of light"$)\}\big)$, "Physics.E" refers to "energy" – the definition of E in the namespace "Physics".

Analogously to namespaces in Computer Science, formally a mathematical namespace can contain any set of identifier-definition pairs that satisfies the definition of the namespace, but typically namespaces of mathematical notation exhibit the same properties as well-designed Java packages: they have low coupling and high cohesion, meaning that all definitions come from the same area of mathematical knowledge and the definitions from different notations do not intersect heavily.

Additionally, we can introduce a document-centric view on the mathematical namespaces: suppose we have a collection of documents of n documents $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ and a set of K namespaces $\{(n_1, \mathcal{N}_1), (n_2, \mathcal{N}_2), \ldots, (n_K, \mathcal{N}_K)\}$. A document d_j can use a namespace $(n_k, \mathcal{N}_k)$ by *importing* identifiers from it. To import an identifier, the document uses an import statement where the identifier i is referred by its fully qualified name defined by this namespace.

TODO: Continue, re-introduce low coupling and high cohesion in terms of document collection.

However, in real-life scientific documents there are no import statements in the document preamble, and therefore ...

The goal is to automatically discover the set of mathematical namespaces given a collection of documents.

Manual labeling is time consuming

Want: approximation via ML algorithms

To do that need to be able to extract definition from text

1.3 Namespaces in Mathematical Notation

An *identifier namespace* is a coherent structure where each identifier is used only once and has a unique definition.

How to find a namespace? Namespaces can be constructed by manually labeling each identifier/definition pair with appropriate name. But this is very time consuming, and in this work we suggest a different approach: use Machine Learning techniques for discovering namespaces automatically from a collection of scientific documents containing mathematical formulae.

Many modern programming languages use namespaces for modularity. For example, in the Java programming language [5] namespaces are called "packages" and a class may refer to classes from other packages via the `import` statement.

Typically in a well-designed application, we can distinguish between two types of application packages [20]:

- *type 1*: domain-specific packages that deal with one particular concept, and
- *type 2*: packages that use many other packages of the first type

For example, we have an application `org.company.app` with several domain-specific packages: `org.company.app.domain.user` with classes related to users, `org.company.app.domain.account` with classes related to user accounts, and a system-related package `org.company.app.tools.auth` that deals with authentication and authorization. Then we also have a package `org.company.app.web.manage`, which belongs to the type 2: it handles web requests while relying on classes from `user` and `account` to implement the business logic and on `auth` for making sure the requests are authorized.

We can observe that the type 1 packages are mostly self-contained and not highly coupled between each other, but type2 packages mostly use other packages of type 1: they depend on them.

We can extend this idea to scientific documents and identifier namespaces. A document can be seen as a class that uses concepts defined in other documents. Then the documents can be grouped such that some groups are of *type 1*: they define the namespaces. In some sense the documents of type 1 are "pure" - they contain infromation about closely related concepts and not highly coupled with other document groups. But some documents are of *type 2* and they are not pure: they draw from different concepts.

This intuition allows us to have the following 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

With these assumptions we can refer to "pure" groups as *namespace defining* groups. These groups can be seen as "type 1" packages: they define namespaces that are used by other "type 2" document groups.

Additionally, we can assume that there is a strong correlation between identifiers in a document and the namespace of the document, and this correlation can be exploit to categorize documents into groups.

Thus by combining these assumptions we can conclude that it should be approximate the process of namespace discovery by discovering groups of namespace defining documents, and this can be done by applying cluster analysis techniques to documents, represented by identifiers they contain.

In the next section we will argue why we can use traditional document clustering techniques and what are the characteristics that texts and identifiers have in common.

1.4 Discovering Namespaces with Document Cluster Analysis

We believe that cluster analysis techniques developed for text documents should work for identifiers.

Let us consider the characteristics of text data:

There are many distinct words in natural language. For example, if $\mathcal{V}$ is a set of all possible words, then usually $|\mathcal{V}| \approx 10^5$, but each individual document may contain only 500 distinct words, or sometimes even less if we consider sentences or small documents (e.g. tweets)

number of words across different document may wary a lot

word distributions follow Power Laws (e.g. Zipf's law)

The identifiers have the same properties!

Natural languages suffer from lexical problems of variability and ambiguity, and the two main problems are synonymy and polysemy [21] [22]:

- two words are *synonymous* if they have the same meaning (for example, "word" and "term" are synonyms),
- a word is *polysemous* is it can have multiple meanings (for example, "trunk" can refer to a part of elephant or a part of a car).

We can note that identifiers have the same problems. For example, in Information Theory, the Shannon Entropy is usually denoted by "H", but sometimes it is also denoted by "I" or by "S", thus these identifiers may be seen as synonyms. Also, "E" can stand both for "Energy" and "Expected value", so "E" is polysemous.

These problems have been studied in Information Retrieval and Natural Language Processing literature. One possible solution for the polysemy problem is *Word Sense Disambiguation* [10]: either replace a word with its sense [23] or append the sense to the word, for example if the polysemous word is "bank" with meaning "financial institution", then we replace it with "bank_finance". The same idea can be used for identifiers, for example "E" can be replaced with "E_energy".

Document clustering techniques usually use Vector Space Models [24] [25] to represent documents. We can define "Identifier Spaces" analogous to Vector Space Models. We assume that Identifier Spaces exhibit the same characteristics as traditional Vector Space Models, and thus

In the next section we review the Vector Space Model, and then introduce the Identifier VSM in the chapter 3.2

Then we can apply cluster analysis techniques to document-identifier matrices.

1.5 Thesis Outline

This work is organized as follows: In chapter 2 we discuss how extract definitions for identifiers in texts with mathematical formulae; in chapter 3 the approaces to namespace extraction and we argue why the cluster analysis can be used for that; in chapter 3.4 we review cluster analysis methods and in chapter **??** we discuss how to extract latent information from data using different matrix factorization techniques. Finally, we describe how the techniques are implemented (chapter 4) and evaluated (chapter 5).

2 Mathematical Definition Extraction

Mathematical expressions are hard to understand without the natural language description, therefore we want to extract identifiers from mathematical expressions and then find their definitions from the surrounding text.

For example, given the sentence "The relation between energy and mass is described by the mass-energy equivalence formula $E = mc^2$, where E is energy, m is mass and c is the speed of light" the goal is to extract the following identifier-definition relations:

- (E, "energy")
- (m, "mass")
- (c, "the speed of light")

Consider another example: "Let e be the base of natural logarithm". We would like to extract (e, "the base of natural logarithm").

Formally, a phrase that defines a mathematical expression consists of three parts [9]:

- *definiendum* is the term to be defined: it is a mathematical expression or an identifier;
- *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 in the first two parts: *definiendum* and *definiens*. Thus we define a *relation* as a pair (definiendum, definiens). For example, (E, "energy") is a relation where E is a definiendum, and "energy" is a definiens.

In this chapter we will discuss how the relations can be discovered automatically. It is organized as follows: then discuss the Math-Aware POS Tagging procedure in section 2.1 and finally review the extraction methods in section 2.2 and briefly discuss how the quality of extracted identifiers is evaluated in section **??**.

2.1 Math-aware POS tagging

Part-of-Speech Tagging (POS Tagging) is a typical Natural Language Processing task which assigns a POS Tag to each word in a given text [10].

While the POS Tagging task is mainly a tool for text processing, it can also be applicable to scientific documents with mathematical expressions, and can be adjusted to dealing with formulae [11] [12].

A *POS tag* is an abbreviation that corresponds to some part of speech. Penn Treebank POS Scheme [13] is a commonly used POS tagging scheme which defines a set of part-of-speech tags for annotating English words. For example, `JJ` is an adjective ("big"), `RB` as in adverb, `DT` is a determiner ("a", "the"), `NN` is a noun ("corpus") and `SYM` is used for symbols (">", "=").

However the Penn Treebank scheme does not have special tags for mathematics, but it is flexible enough and can be extended to include additional tags. For example, we can include a math-related tag `MATH`. Usually it is done by first applying traditional POS taggers (like Stanford CoreNLP [14]), and then refining the results by re-tagging math-related tokens of text as `MATH` [11].

For example, consider the following sentence: "The relation between energy and mass is described by the mass-energy equivalence formula $E = mc^2$, where E is energy, m is mass and c is the speed of light". In this case we will assign the MATH tag to "$E = mc^2$", "E", "m" and "c"

However we can note that for finding identifier-definition relations the `MATH` tag alone is not sufficient: we need to distinguish between complex mathematical expressions and stand-alone identifiers - mathematical expressions that contain only one symbol: the identifier. For the example above we would like to be able to distinguish the expression "$E = mc^2$" from identifier tokens "E", "m" and "c". Thus we extend the Penn Treebank scheme even more and introduce an additional tag `ID` to denote stand-alone identifiers.

Thus, in the example above "$E = mc^2$" will be assigned the `MATH` tag and "E","m" and "c" will be annotated with `ID`.

In the next section we discuss how this can be used to find identifier-definition relations.

2.2 Extraction Methods

There are several ways of extracting the identifier-definition relations. Here we will review the following:

- Nearest Noun
- Pattern Matching
- Machine-Learning based methods
- Probabilistic methods

Nearest Noun Method

The Nearest Noun [15] [16] is the simplest definition extraction method. It assumes that the definition is a combination of ad It finds definitions by looking for combinations of adjectives and nouns (sometimes preceded by determiners) in the text before the identifier.

I.e. if we see a token annotated with `ID`, and then a sequence consisting only of adjectives (`JJ`), nouns (`NN`, `NNS`) and determiners (`DET`), then we say that this sequence is the definition for the identifer.

For example, given the sentence "In other words, the bijection σ normalizes G in ..." we will extract relation (σ, "bijection").

Pattern Matching Methods

The Pattern Matching method [17] is an extension of the Nearest Noun method: In Nearest Noun, we are looking for one specific patten where identifier is followed by the definition, but we can define several such patterns and use them to extract definitions.

For example, we can define the following patterns:

- `IDE DEF`
- `DEF IDE`
- let|set `IDE` denote|denotes|be `DEF`
- `DEF` is|are denoted|`def`ined|given as|by `IDE`
- `IDE` denotes|denote|stand|stands as|by `DEF`
- `IDE` is|are `DEF`
- `DEF` is|are `IDE`
- and many others

In this method `IDE` and `DEF` are placeholders that are assigned a value when the pattern is matched against some subsequence of tokens. `IDE` and DEF need to satisfy certain criteria in order to be successfully matched: like in the Nearest Noun method we assume that `IDE` is some token annotated with `ID` and `DEF` is a phrase containing adjective (`JJ`), nouns (`NN`) and determiners (`DET`). Note that the first patten corresponds to the Nearest Noun pattern.

The patterns above are combined from two sources: one is extracted from a guide to writing mathematical papers in English ([18]) by **TODO**, and

another is extracted from Graphs and Combinatorics papers from Springer by **TODO**.

The pattern matching method is often used as the baseline method for identifier-definition extraction methods [9] [19] [12].

Machine Learning Based Methods

The definition extraction problem can be formulated as a binary classification problem: given a pair (identifier, candidate-definition), does this pair correspond to real identifier-definition relation?

To do this we find all candidate pairs: identifiers are tokens annotated with `ID`, and candidate defections are nouns and noun phrases from the same sentence as the definition.

Once the candidate pairs are found, we extract the following features [19] [16]:

- boolean features for each of the patterns from section 2.2 indicating if the pattern is matched
- indicator if there's a colon or comma between candidate and identifier
- indicator if there's another math expression between candidate and identifier
- indicator if candidate is inside parentheses and identifier is outside
- distance (in words) between the identifier and the candidate
- the position of candidate relative to identifier
- text and POS tag of one/two/three preceding and following tokens around the candidate
- text of the first verb between candidate and identifier
- many others

Once the features are extracted, a binary classifier can be trained to predict if an unseen candidate pair is a relation or not. For this task the popular choices of classifiers are Support Vector Machine classifier with linear kernel [19] [16] and Conditional Random Fields [19], but, in principle, any other binary classifier can be applied as well.

Probabilistic Approaches In the Mathematical Language Processing approach [12] a definition for an identifier is extracted by ranking candidate definitions by the probability of defininig the identifier, and only the most probable candidates are retained.

The main idea of this approach is that the definitions occur very closely to identifiers in sentences, and the closeness can be used to model the probability distribution over candidate definitions.

The candidates are ranked by the following formula:

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

where α, β, γ are weighting parameters.

Finally, for weights α, β, γ the following parameters were chosen in [12]: $\alpha = \beta = 1$ and $\gamma = 0.1$.

3 Namespaces as Document Clusters

In this chapter, we discuss how the process of namespace discovery can be automated.

First, in section 1.3 we describe identifier namespaces and we compare the namespace discovery with cluster analysis techniques applied to textual data and see how clustering algorithms can be useful. Next, in section 3.1 we review the Vector Space Model (VSM): the traditional way of representing a collection of documents as vectors, and then in section 3.2 we introduce the Identifier VSM - which is a way to represent identifier-definition relations in the vector space. Finally we go over common similarity and distance functions that are useful for document clustering in section 3.3 and discuss how similarity search can be made faster by using inverted index **??**.

3.1 Vector Space Model

Vector Space Model is a statistical model for representing documents in some vector space. It is an Information Retrieval model [26], but it is also used for various Text Mining tasks such as Document Classification [27] and Document Clustering [24] [25].

The process of transforming a text to its vector representation is called "vectorization". But before documents can be vectorized they are preprocessed. The preprocessing usually consists of the following steps:

- tokenization: extracting individual words from the text;
- stop words removal: removes functional words that have no discriminative power;
- word normalization (includes stemming or lemmatization): reduces words to some common form;

There are two assumptions made about the data:

- *Bag of Words assumption*: the order of words is not important, only word counts;
- *Independence assumption*: we treat all words as independent.

Both assumptions are quite strong, but nonetheless this method often gives good results.

Document-Term Matrix - representation of a document for text analysis each row of the matrix - is a "document vector" each component of the

document vectors is a concept, a key word, or a term, but usually it's terms documents don't contain many distinct words, so the matrix is sparse

Notation: let $\mathcal{D} = \{d_1, \ \dots \ , d_n\}$ be a set of m documents and let $\mathcal{V} = \{t_1, \ \dots \ , t_m\}$ be a set of n terms (the vocabulary). Each document is set of weighed terms $d_i = \{w_1, \ \dots \ , w_m\}$ where w_j is the weight of term t_j.

There are following term weighting schemes [26]:

- binary: 1 if a term is present, 0 otherwise
- term frequency (TF): number of occurrences of the term in a document
- document frequency (DF): number of documents containing the term
- TF-IDF: combination of TF and inverse DF

Term Frequency (TF) weights terms by local frequency in the document. That is, the term is weighed by how many times it occurs in the document. We can define TF formally as

$$\mathrm{tf}(t, d) = \big|\{t' \in d \ : \ t' = t\}\big|$$

Sublinear TF: sometimes the term is used too often in a document and we want to reduce its influence compared to other less frequent tokens. This can be done by applying some sublinear transformation to TF, for instance, a squared root $\sqrt{\mathrm{tf}(w, d)}$ or a logarithm $\log \mathrm{tf}(w, d)$.

Document Frequency (DF) weights terms by their global frequency in the collection, which is the number of documents that contain the token. Formally it can be defined as

$$\mathrm{df}(t, \mathcal{D}) = \big|\{d \in \mathcal{D} \ : \ t \in d\}\big|$$

Inverse Document Frequency (IDF): more often we are interested in domain specific words than in neutral words, and these domain specific words tent to occur less frequently and they usually have more discriminative power: that is, they are better in telling one document apart from another. So IDF should give more weights to rare words rather than to frequent words. Typically IDF is defined as follows:

$$\mathrm{idf}(t, \mathcal{D}) = \log \frac{|\mathcal{D}|}{\mathrm{df}(t, \mathcal{D})}$$

A good weighting system gives the best performance when it assigns more weights to terms with high TF, but low DF [28]. This can be achieved by combining both TF and IDF schemes. TF is good for getting high frequency

words, but using just TF is not enough if high frequency words are contained in many documents, thus need a collection dependent factor that favors terms that are contained in fewer documents: IDF.

Usually a sublinear TF is used to avoid the dominating effect of words that occur too frequently. As the result, terms appearing too rarely or too frequently are ranked low.

So, we can combine TF and IDF then my multiplying:

$$\text{tf-idf}(t, d \mid \mathcal{D}) = (1 + \log \text{tf}(t, d)) \cdot \log \frac{|\mathcal{D}|}{\text{df}(t, \mathcal{D})}$$

Once the weighting scheme is established, documents can be represented by vectors $d_i = (w_1, \ \ldots \ w_m)$ where w_j is the weight of term t_j.

Then these vectors can be put together to form a matrix. Let D be a $m \times n$ matrix, where rows of D are indexed by terms t_i, columns of D are indexed by documents d_j, and element $(D)_{ij}$ is a weight w_i of term t_i in document d_j. Then such matrix D is called a *term-document matrix*.

Alternatively, D can be $n \times m$ matrix with rows being indexed by documents d_j and columns - by terms t_i. Then such D is called *document-term matrix*. Note that if D is a term-document matrix, then D^T is a document-term matrix. In Information Retrieval literature term-document matrices are used more often, than document-term matrices, but in some applications, like Clustering, it is more convenient to use document-term matrix.

Let us consider the column space of the term-document matrix D. The column of D are documents from the corpus $\mathcal{D}$, so the column space of D contains document vectors where dimensions are terms $t_1, t_2, \ \ldots \ , t_m$. This vector space is called the *Document VSM* (see fig. 1).

3.2 Identifier Space Model

The Vector Space Model gives a good foundation. In this work documents are not represented by words they contain, but rather by identifiers they have. Because words and identifiers exhibit similar properties (see section 1.4) we can replace the Term VSM by *Identifier VSM*: a vector space where documents are represented as vectors indexed by identifiers they contain. Thus, the "vocabulary" of this space is $\text{id}_1, \ \ldots \ , \text{id}_k$, and documents are represented as $d_j = (w_1, \ \ldots \ , w_k)$ where w_i is the weight of identifier id_i.

Then we can define an identifier-document matrix D as $k \times n$ matrix where columns are documents and rows are identifers.

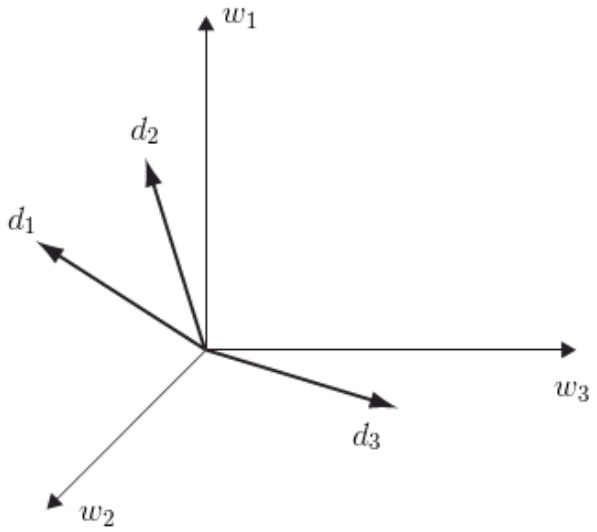


Fig. 1: **TODO redraw in vector**

The Identifer VSM also suffers from the problems of polysemy and synonymy (see section 1.4). They can be solved by extracting definitions for all the identifers and incorporating these definitions into the Identifier VSM.

There are three ways of adding the definition information into Identifier VSM:

- use only identifier information, and no not include the definitions;
- use "weak" identifier-definition association: include identifiers and definitions as separate dimensions;
- use "strong" association: append definition to identifier.

To illustrate how it is done, consider two relations (λ, "regularization") and (w, "weight vector")

- no definitions: dimension of the Identifier VSM are (λ, w)
- "weak" association: dimensions are (λ, w, regularization, weight vector)
- "strong" association: dimensions are (λ_regularization, w_weight vector)

3.3 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).

We will consider the following similarity and distance functions:

- Euclidean distance
- Inner product (or dot product)
- Cosine similarity
- Jaccard coefficient

Euclidean Distance

The Euclidean distance function is the most commonly used distance function in vector spaces. Euclidean distance corresponds to the geometric distance between two data points in the vector 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

$$\|\mathbf{x} - \mathbf{z}\|^2 = (\mathbf{x} - \mathbf{z})^T(\mathbf{x} - \mathbf{z}) = \sum_i (x_i - z_i)^2$$

Euclidean distance is useful for low-dimensional data, but it does not always work well in high dimensions, especially with sparse vector such as document vectors [29].

Inner product

The inner product between two vectors can be used as a similarity function: the more similar two vectors are, the larger is their inner product.

Geometrically the inner product between two vectors $\mathbf{x}$ and $\mathbf{z}$ is defined as

$$\mathbf{x}^T\mathbf{z} = \|\mathbf{x}\| \, \|\mathbf{z}\| \, \cos\theta$$

where θ is the angle between vectors $\mathbf{x}$ and $\mathbf{z}$. In Linear Algebra, however, the inner product is defined as a sum of element-wise products of two vectors: given two vectors $\mathbf{x}$ and $\mathbf{z}$, the inner product is $\mathbf{x}^T\mathbf{z} = \sum_{i=1}^{n} x_i \, z_i$ where x_i and z_i are ith elements of $\mathbf{x}$ and $\mathbf{z}$, respectively. The geometric and algebraic definitions are equivalent [30].

Cosine Similarity

Inner product is sensitive to the length of vectors, and thus it may make sense to consider only the angle between them: the angle does not depend on the magnitude, but it is still a very good indicator of vectors being similar or not.

The angle between two vectors can be calculated from the geometric definition of inner product: $\mathbf{x}^T\mathbf{z} = \|\mathbf{x}\|\,\|\mathbf{z}\|\cos\theta$. By rearranging the terms we get

$$\cos\theta = \frac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|\,\|\mathbf{z}\|}\ .$$

We do not need to the angle itself and can use the cosine directly [26]. Thus can define *cosine similarity* between two documents $\mathbf{d}_1$ and $\mathbf{d}_2$ as

$$\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1^T\mathbf{d}_2}{\|\mathbf{d}_1\|\,\|\mathbf{d}_2\|}\ .$$

If the documents have unit lengths, then cosine similarity is the same as dot product: $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \mathbf{d}_1^T\mathbf{d}_2$.

The cosine similarity can be converted to cosine distance. The maximal possible cosine is 1 for two identical documents. Therefore we can define *cosine distance* between two vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ as $d_c(\mathbf{d}_1, \mathbf{d}_2) = 1 - \text{cosine}(\mathbf{d}_1, \mathbf{d}_2)$. The cosine distance is not a proper metric [31], but it is nonetheless useful.

The cosine distance and the Euclidean distance are connected [31]. For two unit-normalized vectors $\mathbf{d}_1 = \mathbf{d}'_1/\|\mathbf{d}'_1\|$ and $\mathbf{d}_2 = \mathbf{d}'_1/\|\mathbf{d}'_1\|$ the Euclidean distance between them is

$$\|\mathbf{d}_1 - \mathbf{d}_2\|^2 = \|\mathbf{d}_1\|^2 - 2\,\mathbf{d}_1^T\mathbf{d}_2 + \|\mathbf{d}_2\|^2 = 2 - 2\,\mathbf{d}_1^T\mathbf{d}_2.$$

Since the vectors are unit-normalized, we know that $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \mathbf{d}_1^T\mathbf{d}_2$, so we have $\|\mathbf{d}_1 - \mathbf{d}_2\|^2 = 2\,\big(1 - \text{cosine}(\mathbf{d}_1, \mathbf{d}_2)\big) = 2\,d_c(\mathbf{d}_1, \mathbf{d}_2)$.

Thus we can use Euclidean distance on unit-normalized vectors and interpret it as cosine distance.

Jaccard Coefficient

Jaccard similarity, jaccard distance

3.4 Document Clustering Techniques

Cluster analysis is a set of techniques for organizing collection of items into coherent groups. In Text Mining clustering is often used for finding topics in a collection of document. In Information Retrieval clustering is used to assist the users and group retrieved results into clusters.

There are several types of clustering algorithms:

- Hierarchical (Agglomerative and Divisive)
- Partitioning
- Density-based
- and others

In this chapter we will review some of the clustering techniques: in section 3.5 we will discuss Agglomerative clustering. We discuss K-Means, a partitioning algorithms, and its extensions in section **??**. Finally, a density-based algorithm DBSCAN is explained in section 3.7 along with its extensions.

3.5 Agglomerative clustering

The general idea of agglomerative clustering algorithms is to start with each document being its own cluster and iteratively merge clusters based on best pair-wise cluster similarity.

Thus, a typical agglomerative clustering algorithms consists of the following steps:

1. let each document be a cluster on its own
2. compute similarity between all pairs of clusters an store the results in a similarity matrix
3. merge two most similar clusters
4. update the similarity matrix
5. repeat until everything belongs to the same cluster

These algorithms differ only in the way they calculate similarity between clusters.

It can be:

- **Single Linkage (SLINK)**: The clusters are merged based on the closest pair. It can be very efficient, but it encourages chaining behavior.

– **Complete Linkage (CLINK)**: The clusters are merged based on the worst-case similarity - the similarity between the most distant objects on the clusters. It's very expensive computationally, but it avoids chaining altogether.
– **Group-Average Linkage**: Similarity between clusters is calculated as average pair-wise similarity between all objects in the clusters, and the most similar clusters are merged.
– **Ward's Method**: The clusters to merge are chosen such that the within-cluster error (e.g. sum of squares) between each object and its centroid is minimized.

Among these algorithms only SLINK is computationally feasible for large data sets, but it doesn't give good results compared to other agglomerative clustering algorithms. Additionally, these algorithms are not always good for document clustering because they tend to make mistakes at early iterations that are impossible to correct afterwards [32].

3.6 *K*-Means

Unlike agglomerative clustering algorithms, K-Means is an iterative algorithm, which means that it can correct the mistakes made at earlier iterations.

Lloyd's algorithm is the most popular way of implementing K-Means [33]: given a desired number of clusters k, it iteratively improves the Euclidean distance between each data point and the centroid, closest to it.

Let $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ \dots\ , \mathbf{d}_n\}$ be the document collection, where documents $\mathbf{d}_i$ are represented is some document vector space $\mathbb{R}^m$ and k is the desired number of clusters. Then we define k cluster centroids $\boldsymbol{\mu}_j$ that are also in the same document vector space $\mathbb{R}^m$. Additionally for each document $\mathbf{d}_i$ we maintain the assignment variable $c_i \in \{1, 2, \ \dots\ , k\}$, which specifies to what cluster centroid $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ \dots\ , \boldsymbol{\mu}_k$ the document $\mathbf{d}_i$ belongs.

The algorithms consists of three steps: (1) seed selection step, where each $\boldsymbol{\mu}_j$ is randomly assigned some value, (2) cluster assignment step, where we iterate over all document vectors $\mathbf{d}_i$ and find its closest centroid, and (3) move centroids step, where the centroids are re-calculated. Steps (2) and (3) are repreated until the algorithm converges. The pseudocode for K-Means is presented in the listing 1.

Usually, K-Means shows very good results for document clustering, and in several studies it (or its variations) shows the best performance [34] [32].

Algorithm 1 Lloyd's algorithm for K-Means

function K-MEANS(no. clusters k, documents $\mathcal{D}$)
  for $j \leftarrow 1$ .. k **do** ▷ random seed selection
    $\boldsymbol{\mu}_j \leftarrow$ random $\mathbf{d} \in \mathcal{D}$
  while not converged **do**
    for each $\mathbf{d}_i \in \mathcal{D}$ **do** ▷ cluster assignment step
      $c_i \leftarrow \arg\min_j \|\mathbf{d}_i - \boldsymbol{\mu}_j\|^2$
    for $j \leftarrow 1$ .. k **do** ▷ move centroids step
      $\mathcal{C}_j \leftarrow \{\, \mathbf{d}_i \text{ s.t. } c_i = j \,\}$
      $\boldsymbol{\mu}_j \leftarrow \frac{1}{|\mathcal{C}_j|} \sum_{\mathbf{d}_i \in \mathcal{C}_j} \mathbf{d}_i$
  return $(c_1, c_2, \ \ldots\ , c_n)$

However for large document collections Lloyd's classical K-Means takes a lot of time to converge. The problem is caused by the fact that it goes through the entire collection many times. Mini-Batch K-Means [35] uses Mini-Batch Gradient Descent method, which is a different optimization technique that converges faster. The pseudocode for Mini-Batch K-Means is presented in listing 2.

Algorithm 2 MiniBatch K-Means

function MINIBATCH-K-MEANS(no. clusters k, no. iterations t, batch size b, documents $\mathcal{D}$)
  for $j \leftarrow 1$ .. k **do** ▷ random initialization
    $\boldsymbol{\mu}_j \leftarrow$ random $\mathbf{d} \in \mathcal{D}$
  repeat t times
    $\mathcal{M} \leftarrow b$ random examples from $\mathcal{D}$
    for each $\mathbf{d}_i \in \mathcal{M}$ **do**
      centroids$[\mathbf{d}_i] \leftarrow \arg\min_j \|\mathbf{d}_i - \boldsymbol{\mu}_j\|^2$ ▷ cache the centroid nearest to $\mathbf{d}_i$
    for each $\mathbf{d}_i \in \mathcal{M}$ **do**
      $c_i \leftarrow$ centroids$[\mathbf{d}_i]$ ▷ the centroid index of document $\mathbf{d}_i$
      $v[c_i] \leftarrow v[c_i] + 1$ ▷ counts per centroid c_i
      $\eta \leftarrow 1/v[c_i]$ ▷ per-centroid learning rate
      $\boldsymbol{\mu}_{c_i} \leftarrow (1 - \eta) \cdot \boldsymbol{\mu}_{c_i} + \eta \cdot \mathbf{d}_i$ ▷ gradient step
  until converged
  return $(c_1, c_2, \ \ldots\ , c_n)$

Note that K means uses Euclidean distance, and Euclidean distance does not always behave well in high-dimensional sparse vector spaces like Document VSMs (see section 3.3). However, as discussed in section 3.3, if document vectors are normalized, the Euclidean distance and Cosine dis-

tance are related, and therefore Euclidean K-means is the same as "Cosine Distance" K-Means.

K-Means is the most popular clustering algorithms and there are many extensions to this algorithm. In the next section we will discuss some of the extensions related to document clustering.

There are several extensions of K-Means.

For example, Bisecting K-Means [32] is a combination of partitioning and hierarchical (divisive) algorithms. It's a variant of K-Means that gradually splits the document space in halves until the desired number of clusters is obtained. Bisecting K-Means can achieve good performance while giving the user additional information about ... ?

Algorithm:

- start with a single cluster
- choose a cluster to split (for example, the largest one)
- apply K-means to this cluster with $K = 2$ to split it
- repeat until have desired number of clusters

TODO: pseudocode

Scatter/Gather is another popular variation of K-means, but initially used for clustering retrieved documents for Information Retrieval systems [36]. This variation includes: special smart seed selection procedure (applying hierarchical cluster on a subset of document vectors to initialize the centroids at the initialization step) and several cluster refinement operations. Additionally, in Scatter/Gather cluster centroids are concatenations of all terms in the cluster documents, not a mean value; and the cosine similarity is used instead of Euclidean distance.

There are two cluster refinement operations: split and join.

The split operation is used to continue splitting clusters, and it's applied only to the clusters that are not coherent enough. Essentially, the split operation splits the non-coherent clusters in the same way as Bisecting K-Means. The coherence is measured via *self-similarity* of a cluster, which is the mean similarity of all documents in the cluster to its centroid, or the mean pair-wise similarity between all documents of the cluster.

The join operation merges the clusters that are very similar to each other. The similarity is measured by computing "typical" terms for each

cluster (usually the most frequent terms of the centroid) and examining which clusters have significant overlaps between their typical terms.

However, when there are many documents, the centroids tend to contain a lot of words, which leads to a significant slowdown. A solution to this problem is a center adjustment method, called vector average dumping **TODO** [37]. Alternatively, some terms of the centroid can be truncated. There are several possible ways of trucating the terms: for example, we can keep only the top c terms, or remove the least frequent words such that at least 90% (or 95%) of the original vector norm is retained [38].

3.7 DBSCAN

DBSCAN is a clustering algorithm that can discover clusters of complex shapes based on the density of data points [39].

The *density* associated with a data point is obtained by counting the number of points in a region of radius ε around the point, where ε is defined by the user. If a point has a density of at least some user defined threshold MinPts, then it is considered a *core point*. The clusters are formed around these core points, and if two core points are within the radius ε, then they belong to the same cluster. If a point is not a core point itself, but it belong to the neighborhood of some core point, then it is a *border point*. But if a point is not a core point and it is not in the neighborhood of any other core point, then it does not belong to any cluster and it is considered *noise*.

DBSCAN works as follows: it selects an arbitrary data point p, and then finds all other points in ε-neighborhood of p. If there are more than MinPts points around p, then it is a core point, and it is considered a cluster. Then the process is repeated for all points in the neighborhood, and they all are assigned to the same cluster, as p. If p is not a core point, but it has a core point in its neighborhood, then it's a border point and it is assigned to the same cluster and the core point. But if it is a noise point, then it is marked as noise or discarded.

The DBSCAN algorithm uses the Euclidean distance, but can be adapted to use any other distance or similarity function. For example, to modify the algorithm to use the Cosine similarity (or any other similarity function) the Region-Query has to be modified to return $\{x \ : \ \text{similarity}(x, p) \geqslant \varepsilon\}$.

The details of implementation of Region-Query are not specified, and it can be implemented differently. For example, it can use Inverse Index (see section **??**, and listing **??** for the pseudocode) to make the similarity search faster.

Algorithm 3 DBSCAN

```
function DBSCAN(database 𝒟, radius ε, MinPts)
    result ← ∅
    for all p ∈ 𝒟 do
        if p is visited then
            continue
        mark p as visited
        𝒩 ← Region-Query(p, ε)                    ▷ 𝒩 is the neighborhood of p
        if 𝒩 < MinPts then
            mark p as NOISE
        else
            𝒞 ← Expand-Cluster(p, 𝒩, ε, MinPts)
            result ← result ∪ {𝒞}
    return result

function Expand-Cluster(point p, neighborhood 𝒩, radius ε, MinPts)
    𝒞 ← {p}
    for all x ∈ 𝒩 do
        if x is visited then
            continue
        mark x as visited
        𝒩_x ← Region-Query(x, ε)                  ▷ 𝒩_x is the neighborhood of x
        if |𝒩_x| ⩾ MinPts then
            𝒩 ← 𝒩 ∪ 𝒩_x
        𝒞 ← 𝒞 ∪ {x}
    return 𝒞

function Region-Query(point p, radius ε)
    return {x : ‖x − p‖ ⩽ ε}                      ▷ all points within distance ε from p
```

As discussed, DBSCAN can be extended to use any distance or similarity function. Shared Nearest Neighbors Similarity (SNN Similarity) [29] is a special similarity function that is particularity useful for high-dimensional spaces. And this similarity function is applicable to document clustering and topic discovery [40].

SNN Similarity is specified in terms of the K nearest neighbors. Let $\mathrm{NN}_{K,\,\mathrm{sim}}(p)$ be a function that returns top K closest points of p according to some similarity function `sim`. Then the SNN similarity function is defined as

$$\mathrm{snn}(p, q) = \left|\mathrm{NN}_{K,\,\mathrm{sim}}(p) \cup \mathrm{NN}_{K,\,\mathrm{sim}}(q)\right|$$

The extension of DBSCAN that uses the SNN Similarity is called SSN Clustering algorithm. The user needs to specify the SSN similarity function by setting parameter K and choosing the base similarity function $\texttt{sim}(\cdot, \cdot)$ (typically Cosine, Jaccard or Euclidean). The algorithm itself has the same parameters as DBSCAN: radius ε (such that $\varepsilon < K$) and the core points density threshold MinPts. The Region-Query function is modified to return $\{q \ : \ \mathrm{snn}(p, q) \geqslant \varepsilon\}$. For pseudocode, see the listing 4.

Algorithm 4 SNN Clustering Algorithm

function SNN-Cluster(database $\mathcal{D}$, K, similarity function `sim`, radius ε, MinPts)
  for all $p \in \mathcal{D}$ **do** ▷ Pre-compute the KNN lists
    $\mathrm{NN}[p] \leftarrow \mathrm{NN}_{K,\,\mathrm{sim}}(p)$
  for all $(p, q) \in (\mathcal{D} \times \mathcal{D})$ **do** ▷ Pre-compute the SNN similarity matrix
    $A[p, q] \leftarrow \left|\,\mathrm{NN}[p] \ \cup \ \mathrm{NN}[q]\,\right|$
  return DBSCAN$(A, \varepsilon, \mathrm{MinPts})$

The algorithm's running time complexity is $O(n^2)$ time, where $n = |\mathcal{D}|$, but it can be sped up by using the Inverted Index.

3.8 Latent Semantic Analysis

The Vector Space Model discussed in section 3.1. The solution to this problem is to assume that there exists some optimal document vector space where the document vectors do not suffer from the ... This vector space can be found by finding the best k-rank approximation to the Term-Document matrix using Singular Value Decomposition (SVD). This technique is called

Latent Semantic Analysis [41] or Latent Semantic Indexing in the context of Information Retrieval [21]. It is also a popular Text Mining technique for reducing the dimensionality of text data and it is often used for document clustering [25] [42].

There are three major steps in Latent Semantic Analysis [43]:

- Preprocess documents
- Construct a Term-Document matrix D using the Vector Space Model
- Reduce dimensionality of D by using SVD

The first two steps are the same as for traditional Vector Space Models: consider a set of document $\mathcal{D} = \{d_1, d_2, \ \dots \ , d_n\}$ with the vocabulary $\mathcal{V} = \{t_1, t_2, \ \dots \ , t_m\}$, then D is a $m \times n$ Term-Document Matrix. If the matrix D has rank r, then the SVD of D is $D = U\Sigma V^T$, where:

- U is an $m \times r$ orthogonal matrix, i.e. $UU^T = I$;
- Σ is a diagonal $r \times r$ matrix with singular values ordered by their magnitude;
- V is an $n \times r$ orthogonal matrix, $VV^T = I$.

The dimensionality reduction is done by finding the best k-rank approximation of D, which is obtained by keeping only the first k singular values of Σ and setting the rest to 0. Typically, not only Σ is truncated, but also U and V, and therefore, the k-rank approximation of D using SVD is written as $D \approx D_k = U_k \Sigma_k V_k^T$ where U_k is an $m \times k$ matrix with first k columns of U, Σ_k is an $k \times k$ diagonal matrix with singular values, and V_k is an $n \times k$ matrix with first k columns of V. This decomposition is called *rank-reduced* SVD and when applied to text data it reveals the "true" latent semantic space. The parameter k corresponds to the number of "latent concepts" in the data. The idea of LSA is very nicely illustrated by examples in [21] and [41].

LSA can be used for clustering as well, and this is usually done by first transforming the document space to the LSA space and then doing applying transitional cluster analysis techniques there [38].

However these is not need to reconstruct the rank-reduced matrix to apply clustering, and in many cases it is not possible: the original input space is very sparse, but the rank-reduced reconstructed matrix becomes very dense. Therefore we do not reconstruct the entire matrix, but instead keep only the low dimensional representation $V_k \Sigma_k$, which is enough for many clustering algorithms.

For example, consider the inner product. Document-document similarity in the original space is calculated as DD^T (the columns of D are the document $\mathbf{d}_1, \ldots, \mathbf{d}_n$), and by applying SVD we have $D^TD = V\Sigma^TU^TU\Sigma V^T = V\Sigma^T\Sigma V^T = (V\Sigma)(V\Sigma)^T$. Thus, to compute the similarity between document $\mathbf{d}_i$ and $\mathbf{d}_j$ we compute the inner product between ith and jth rows of $V\Sigma$. Likewise, to compute the similarity between documents i and j in the reduced representation, we compute the inner product between the respective rows of $V_k\Sigma_k$.

Additionally, the Euclidean distance in the reduced space can also be computed directly on the rows of $V_k\Sigma_k$. Recall that the Euclidean distance can be expressed as an inner product $\|\mathbf{d}_i - \mathbf{d}_j\|^2 = \mathbf{d}_i^T\mathbf{d}_i - 2\,\mathbf{d}_i^T\mathbf{d}_j + \mathbf{d}_j^T\mathbf{d}_j$, and since we know how to compute the inner product in the semantic space, we can compute the distance in this space as well by using the rows of $V_k\Sigma_k$.

This means that we can apply any clustering algorithm, including K-means, on the rows of $V_k\Sigma_k$ and without having to reconstruct the entire term-document matrix.

A generic LSA-based clustering algorithm therefore consists of the following steps:

- Build a term-document matrix D from the document collection
- Select number of latent concepts k and apply rank-reduced SVD on D to get $V_k\Sigma_k$
- Apply the cluster algorithm on the rows of $V_k\Sigma_k$

LSA has some drawbacks. Because SVD looks for orthogonal basis for the new document space, there are negative values that are harder to interpret. Additionally, with negative values in the reconstructed space can cause the cosine to take negative values as well. However, it does not affect significantly the properties of the cosine distance: it still will always give the results larger than 0. This problem can be solved by using a different matrix decomposition technique instead of SVD, and we discuss one of them in the next section.

Apart from SVD there are many other different matrix decomposition techniques that can be applied for document clustering and for discovering the latent structure of the term-document matrix [44], and one of them in Non-Negative Matrix Factorization (NMF) [45].

NMF is a matrix decomposition technique. When it is applied to non-negative data, NMF produces non-negative rank-reduced approximations. Since term-document matrices do not have negative values, it makes NMF

a good candidate to replace SVD in LSA. The main conceptual difference between SVD and NMF is that SVD looks for orthogonal directions to represent document space, while NMF does not require orthogonality. As the result, SVD often produces semantic spaces with negative values, but NMF does not [46] (see fig. 2).

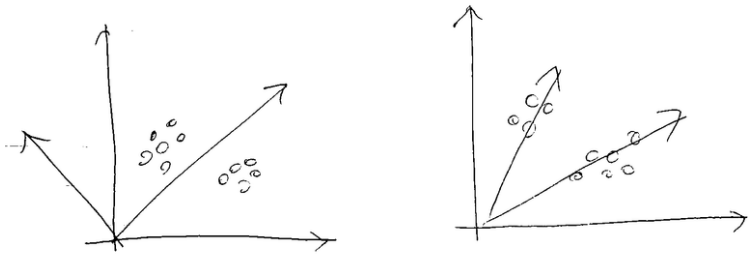


Fig. 2: **TODO redraw** Directions found by SVD (on the left) vs directions by NMF (on the right)

The NMF of an $m \times n$ term-document matrix D is $D \approx D_k = UV^T$ where U is an $m \times k$ matrix, V is an $n \times k$ matrix and k is the number of semantic concepts in D. Non-negativity of elements in D_k is very good for interpretability: it ensures that documents can be seen as a non-negative combination of the key concepts.

Additionally, NMF is useful for clustering: the results of NMF can be directly interpreted as cluster assignment and there is no need to use separate clustering algorithms [46].

What is more, NMF is a co-clustering algorithms: it clusters both rows of D and columns of D at the same time. For a term-document matrix $D \approx UV^T$, where U defines the reduced vector space for terms and V defines the reduced vector space for documents. Since all elements are non-negative, it can have the following interpretation: elements $(U)_{ij}$ of U represent the degree to which terms i belongs to cluster j, and elements $(V)_{ij}$ represent the degree to which document i belongs to cluster j.

The document clustering using NMF consists of the following steps [46]:

- Construct the term-document matrix D;
- Perform NMF on D to get U and V;

- Normalize rows $\mathbf{v}_i$ of V by using $\mathbf{v}'_i = \mathbf{v}_i \cdot \|\mathbf{u}_i\|$ and rows $\mathbf{u}_i$ of U with $\mathbf{u}'_i = \mathbf{u}_i / \|\mathbf{u}_i\|$;
- To determine the cluster assignment for document $\mathbf{d}_i$, examine $\mathbf{v}'_i$ (the ith row of V) and find the largest component of this vector. That is, ith document belongs to cluster x if $x = \arg\max_j v_{ij}$ where v_{ij} are components of $\mathbf{v}_i$;
- If the desired number of clusters K is larger than the rank k of the reduced matrix D_k, the clustering can be performed directly on the rows of V, for example, by using K-Means.

4 Implementation

In section 4.1 we describe the data set that we use, then we describe how we extract identifiers from this dataset (section 4.2) and how this dataset is cleaned (section 4.3). Next, the implementation of clustering algorithms is described in the section 4.3. After the clusters are found, we combine them into a hierarchy in the section 4.5.

Finally, in the section 4.6 we explore how the same set of techniques can be applied to source code in Java.

4.1 Data set

Wikipedia is a big online encyclopedia where the content are written and edited by the community. It contains a large amount of articles on a variety of topics, including articles about Mathematics and Mathematics-related fields such as Physics. It is multilingual and available in several languages, including English, German, French, Russian and others. The content of wikipedia pages are authored in a special markup language and the content of the entire encyclopedia is freely available for download.

The techniques discussed in this work are mainly applied to the English version of Wikipedia. At the moment of writing (July 15, 2015) English Wikipedia contains about 4.9 million articles[2]. However, just a small portion of these articles are math related: there are only 30.000 pages that contain at least one `<math>` tag.

Apart from the text data and formulas Wikipedia articles have information about categories, and we can exploit this information as well. The category information is encoded directly into each Wikipedia page with a special markup tag. For example, the article "Linear Regression"[3] belongs to the category "Regression analysis" and `[[Category:Regression analysis]]` tag encodes this information. However there are other indirect ways to associate a page with some category, for example, by using Wiki templates. A template is a user-defined macro that is executed by the Wikipedia engine, and the content of a template can include category association tags. It is hard to extract the content information from the template tag and therefore we use category information available in a structured machine-processable form in DBPedia [47]. Additionally, DBPedia provides extra information such as parent categories (categories of categories) that is very easy to process and incorporate into analysis.

[2] https://en.wikipedia.org/wiki/Wikipedia:Statistics
[3] https://en.wikipedia.org/wiki/Linear_regression

Wikipedia is available in other languages, not only English. While the most of the analysis is performed on the English Wikipedia, we also apply some of the techniques to the Russian version [48] to compare it with the results obtained on the English Wikipedia. Russian Wikipedia is smaller that the English Wikipedia and contains 1.9 million articles[4], among which only 15.000 pages are math-related (i.e. contain at least one `<math>` tag).

4.2 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 2.2. The extraction process is implemented using Apache Flink [49] and it is based on the open source implementation provided by Pagael and Schubotz in [12][5].

The first step is to keep only mathematical articles and discard the rest. This is done by retaining only those articles that contain at least one `<math>` tag. Once the data set is filtered, then all the LaTeX formulas form the `<math>` tags are converted to MathML, an XML-based representation of mathematical formulae [50].

The dataset is stored in a big XML file in the Wiki XML format. It makes it easy to extract the title and the content of each document, and then process the documents separately. The formulas are extracted by looking for the `<math>` tags. However some formulas for some reasons are typed without the tags using the unicode symbols, and such formulas are very hard to detect and therefore 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 [51] of the tag's content represented as a hexadecimal string. After that the content of the tags is kept separately from the document content.

The next step is to find the definitions for identifiers in formulas. We are not interested in the semantics of a formula, only in the identifiers it contains. In MathML `<ci>` corresponds to identifiers, and hence extracting identifiers from MathML formulas amounts to finding all `<ci>` tags and retrieving their content. It is enough to extract simple identifiers such as "t", "C", "μ", but there also are complex identifiers with subscripts, such as "x_1", "ξ_i" or even "β_{slope}". To extract them we need to look for tags `<msub>`. We do not process superscripts because they are usually powers (for example, "x^2"),

[4] https://en.wikipedia.org/wiki/Russian_Wikipedia
[5] https://github.com/rbzn/project-mlp

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 [2]:

- `//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 Formulae": analogously to the Bag of Words approach (see section 3.1) we keep only the counts of occurrences of different identifiers and we do not preserve any other structure.

The content of Wikipedia document is authored with Wiki markup – a special markup language for specifying document layout elements such as headers, lists, text formatting and tables. 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" [52]. Almost all annotations are discarded at this stage, and only inner-wiki links are kept: they can be useful as candidate definitions. The implementation of this step is taken entirely from [12] with only a few minor changes.

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 the Stanford Core NLP library (StanfordNLP) [14]. The first part of 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 2.1). For English documents from the English Wikipedia we use StanfordNLP's Maximal Entropy POS Tagger [53]. Unfortunately, there are no trained models available for POS tagging the Russian language for the StanfordNLP library. Additionally, we were not able to find a suitable implementation of any other POS taggers in Java, and therefore we implemented a simple rule-based POS tagger ourselves. The implementation is based on a PHP function from [54]: it is translated into Java and seamlessly incorporated into the StanfordNLP pipeline. The English tagger uses the Penn Treebank POS Scheme [13], and hence we follow the same convention for the Russian tagger.

For handling mathematics we introduce two new POS classes: "`ID`" for identifiers and "`MATH`" for formulas. These classes are not a part of the Penn Treebank POS Scheme, and 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". We also 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 the "ID" tag.

At the Wikipedia markup processing step we discard almost all markup annotations, but we do keep inner Wikipedia 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]], and then these subsequences are concatenated and tagged as "LINK".

Successive nouns (both singular and plurals), possible modified by an adjective, are also candidates for definitions. Therefore we find all such sequences on the text and then concatenate each into one single token tagged with "NOUN_PHRASE".

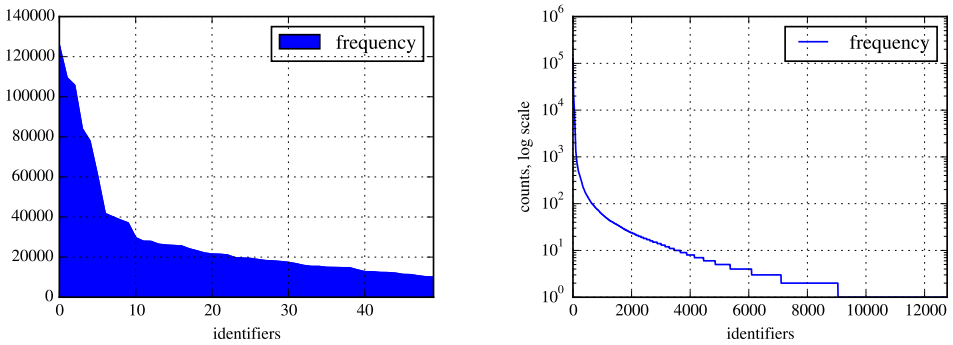
The next stage is selecting the most probable identifier-definition pairs, and this is done by ranking definition candidates. The definition candidates are tokens annotated with "NN" (noun singular), "NNS" (noun plural), "LINK" and "NOUN_PHRASE". We rank these tokens by a score that depends how far it is from the identifer of interest and how far is the closest formula that contains this identifier (see section 2.2). The output of this step is a list of identifier-definition pairs along with the score, and only the pairs with scores above the user specified threshold are retained. The implementation of this step is also taken entirely from [12] with very minor modifications.

There are about 2 million identifiers in total extracted from the English Wikipedia, and there are 12.771 distinct identifers, 3731 of which occur only once. and 1941 occur just twice. The most frequent identifiers are x (125.500 times), p (110.000), m (105.000) and n (83.000).

Only 115297 definitions are found

The following is the list of the most common identifier-definition pairs extracted from the English Wikipedia:

- t: "time" (1086)
- m: "mass" (424)
- θ: "angle" (421)
- T: "temperature" (400)
- r: "radius" (395)



(a) Frequencies of the first 50 identifiers

(b) Frequencies, log scale

Fig. 3: Distribution of frequencies of identifiers

– 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)
– d: “distance” (163)
– X: “topological space” (159)

And this is the list of relations extracted from the Russian one:

– t: “функция” (“function”) (215)
– t: “время” (“time”) (130)
– X: “множество” (“set”) (113)
– m: “масса” (“mass”) (103)
– c: “скорость свет” (“speed of light”) (89)
– G: “группа” (“group”) (87)

- T: “температура” (“temperature”) (69)
- h: “постоянный планка” (“Plank constant”) (68)
- ρ: “плотность” (“density”) (57)
- M: “многообразие” (“manifold”) (53)
- K: “поль” (“field”) (53)
- X: “пространство” (“space”) (50)
- v: “скорость” (“speed”) (50)
- X: “топологический пространство” (“topological space”) (46)
- G: “граф” (“graph”) (44)
- R: “радиус” (“radius”) (38)
- R: “кольцо” (“ring”) (36)
- G: “гравитационный постоянный” (“gravitational constant”) (34)
- E: “энергия” (“energy”) (34)
- m: “модуль” (“modulo”) (33)
- S: “площадь” (“area”) (32)
- k: “постоянный больцмана” (“Boltzmann constant”) (30)

4.3 Data Cleaning

The Natural Language data is famous for being noisy and hard to clean [55]. 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 4.2).

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 **??** 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 [56] 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 "^", "#","∀", "∫"; miscellaneous symbols like "⋄" or "∘", arrows like "→" and "⇒", and special characters like "⌈".

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" [56]. 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.

4.4 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 3.2) 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 $(\sigma, \text{Cauchy}, \text{stress}, \text{tensor})$, while for the “strong” association case we will have $(\sigma\_\text{Cauchy}, \sigma\_\text{stress}, \sigma\_\text{tensor})$.

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 [57] implemented in NLTK [58]: a python library for Natural Language Processing.

Each document is vectorized (converted to a vector form) by using `TfidfVectorizer` from scikit-learn [59]. 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 3.1), 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`) [59]. Note that if rows are unit-normalized, then running K-Means with Euclidean distance is equivalent to cosine distance (see section 3.6).

Bisecting K-Means (see section **??**) 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, an extension to K-means (see section **??**), was implemented manually using scipy [60] and numpy [61] because scikit-learn's implementation of K-Means does not allow using user-defined distances.

DBScan (section 3.7) and SNN Clustering (section **??**) 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 **??**)

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 3.8) which is implemented using randomized Singular Value Decomposition (SVD) [62], The implementation of randomized SVD is taken from scikit-learn [59] - method `randomized_svd`. Non-negative Matrix Factorization is an alternative technique for dimensionality reduction (section **??**). Its implementation is also taken from scikit-learn [59], 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 [47] 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 (see fig. 3), but at some point the number of pure clusters will start decreasing (see fig. 4).

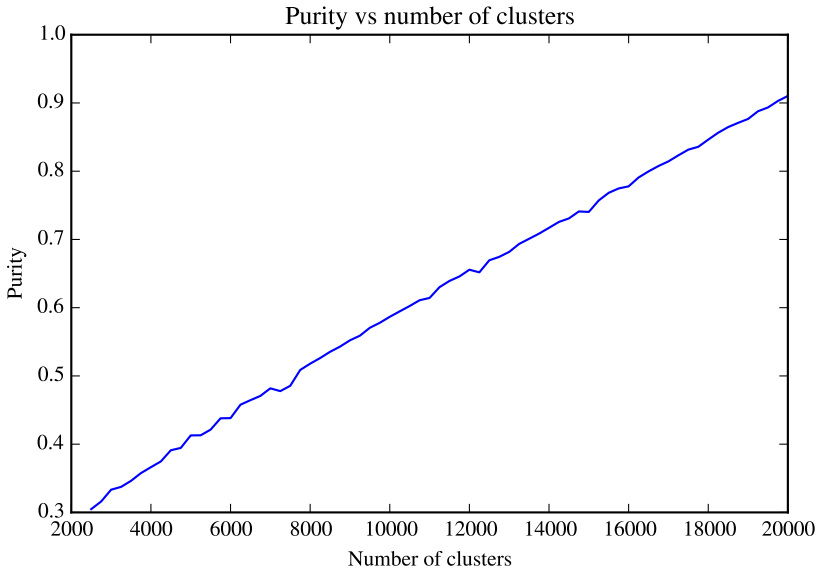


Fig. 4: K in K-Means vs overall purity of clustering: the purity increases linearly with K

4.5 Building Hierarchy

Once

AMS Mathematics Subject Classification (2010) [63] Excluded all subcategories those code end with '99': they are usually 'Miscellaneous topics'

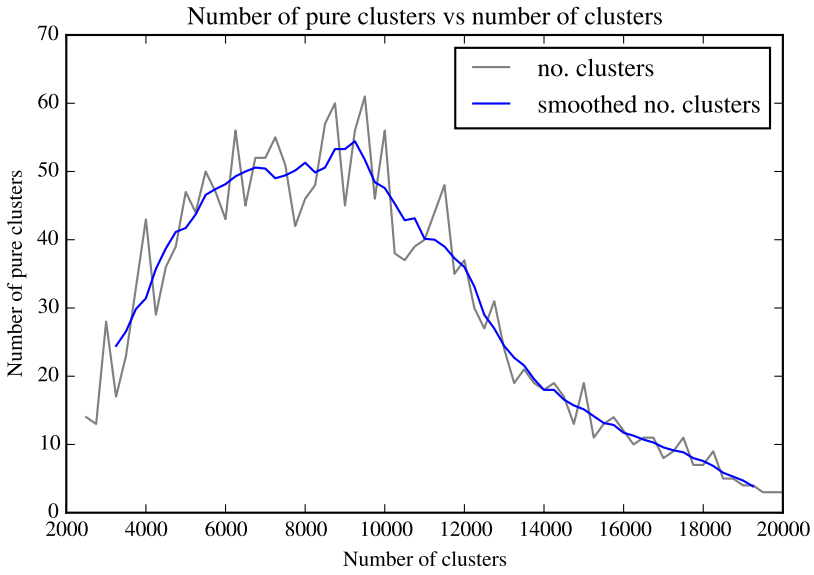


Fig. 5: K in K-Means vs the number of pure clusters: it grows initially, but after $K \approx 1000$ starts to decrease

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) [64]

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 [65] available as a SKOS [66] ontology at their website [67]. The SKOS ontology graph was processed with RDFLib [68]

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.

4.6 Java Language Processing

TODO: also refer back to the introduction

we have compared the identifier namespaces with namespaces in programming languages and with packages in the Java programming language in particular. We motivated the assumption that there exist “namespace defining” groups of documents by arguing that these groups also exist in programming languages. Thus, the same set of techniques for namespace discovery should be applicable to source code as well.

If a programming language is statically typed, like Java or Pascal, usually it is possible to know the type of a variable from the declaration of this variable. Therefore we can see variable names as “identifiers” and variable types as “definitions”. Clearly, there is a difference between variable types and identifier definitions, but we believe that this comparison is valid because the type carries additional semantic information about the variable and in what context it can be used – like the definition of an identifier.

The information about variables and their types can be extracted from a source code repository, and each source file can be processed to obtain its Abstract Syntax Tree (AST). By processing the ASTs, we can extract the variable declaration information. Thus, each source file can be seen as a document, which is represented by all its variable declarations.

In this work we process Java source code, and for parsing it and building ASTs we use a library JavaParser [69]. The Java programming language

was chosen because it requires the programmer to always specify the type information when declaring a variable. It is different for other languages when the type information is usually inferred by the compilers at compilation time.

In Java a variable can be declared in three places: as an inner class variable (or a "field"), as a method (constructor) parameter or as a local variable inside a method or a constructor. We need to process all three types of variable declarations and then apply additional preprocessing, such as converting the name of the type from short to fully qualified using the information from the import statements. For example, `String` is converted to `java.lang.String` and `List<Integer>` to `java.util.List<Integer>`, but primitive types like `byte` or `int` are left unchanged. Secondly,

Consider an example in the listing 1. There is a class variable `threshold`, a method parameter `in` and two local variables `word` and `posTag`. The following relations will be extracted from this class: ("threshold", `double`), ("in", `domain.Word`), ("word", `java.lang.String`), ("posTag", `java.lang.String`). Since all primitives and classes from packages that star with `java` are discarded, at the end the class `WordProcesser` is represented with only one relation ("in", `domain.Word`).

Listing 1: A java class

```
package process;

import domain.Word;

public class WordProcesser  {

    private double threshold;

    public boolean isGood(Word in) {
        String word = in.getWord();
        String posTag = in.getPosTag();
        return isWordGood(word) && isPosTagGood(posTag);
    }

    // ...

}
```

In the experiments we applied this source code analysis to the source code of Apache Mahout 0.10 [70], which is an open-source library for scalable Machine Learning and Data Mining.

As on 2015-07-15, this dataset consists of 1560 java classes with 45878 variable declarations. After discarding declarations from the standard Java API, primitives and types with generic parameters, only 15869 declarations were retained.

The following is top-15 variable/type declarations:

– ("conf", `org.apache.hadoop.conf.Configuration`), 491 times
– ("v", `org.apache.mahout.math.Vector`), 224 times
– ("dataModel", `org.apache.mahout.cf.taste.model.DataModel`), 207 times
– ("fs", `org.apache.hadoop.fs.FileSystem`), 207 times
– ("log", `org.slf4j.Logger`), 171 times
– ("output", `org.apache.hadoop.fs.Path`), 152 times
– ("vector", `org.apache.mahout.math.Vector`), 145 times
– ("x", `org.apache.mahout.math.Vector`), 120 times
– ("path", `org.apache.hadoop.fs.Path`), 113 times
– ("measure", `org.apache.mahout.common.distance.DistanceMeasure`), 102 times
– ("input", `org.apache.hadoop.fs.Path`), 101 times
– ("y", `org.apache.mahout.math.Vector`), 87 times
– ("comp", `org.apache.mahout.math.function.IntComparator`), 74 times
– ("job", `org.apache.hadoop.mapreduce.Job`), 71 times
– ("m", `org.apache.mahout.math.Matrix`), 70 times

5 Evaluation

In section 5.1 we describe how we select the best clustering algorithm.

5.1 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 $D \approx D_k = U_k \Sigma_k V_k^T$, param: k what's the rank of the approximation matrix NMF $D \approx D_k = U_k V_k^T$ 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

Article	Identifiers
APL (programming language)	n, O, R
Binary search tree	O, n
Boolean satisfiability problem	O, n
Complexity	O, n
Earley parser	O, n
Heapsort	O, n, Ω
Lisp (programming language)	O, n
Priority queue	O, n
Sieve of Eratosthenes	O, n
Smoothsort	O, n
Comb sort	Ω, n, p, O
Divide and conquer algorithm	O, n, Ω
Stack (abstract data type)	O, n, t
Skip list	n, p, O
Graph minor	O, n, h
Flex lexical analyser	O, n
Gift wrapping algorithm	O, n, h
Perlin noise	n, O
Pseudo-polynomial time	n, O, m
Hirzebruch surface	O, n, m, p
Beap	n, O
Pairing heap	O, n, Ω
Cost efficiency	O, n, p

(54 documents in total) These articles appear to relate to

DBSCAN SNN

$k = 10$ dist = jaccard

ε=7 points MinPts=5 points

Article	Identifiers
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 J1614–2230	$M_\odot$
Kennicutt–Schmidt 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	$M_\odot$

In general doesn't give clusters

TODO add some numbers and graphs

WEAK ASSOCIATION

K-Means weak association

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

Article	Identifiers
Papyrus 66	Ppapyrus
Alexandrian text-type	P, papyrus
Western text-type	P, papyrus
Codex Ephraemi Rescriptus	P, papyrus
Bodmer Papyri	P, papyrus
Categories of New Testament manuscripts	P, papyrus
Papyrus 4	P, papyrus
Papyrus 75	P, papyrus
Uncial 0308	P, M, papyrus, 47
Codex Athous Lavrensis	P, papyrus
Papyrus 92	P, papyrus
Papyrus 90	P, papyrus
Papyrus 9	P, papyrus
Papyrus 15	P, papyrus
Papyrus 16	P, papyrus
Papyrus 20	P, papyrus
Papyrus 39	P, papyrus
Papyrus 49	P, papyrus
Papyrus 65	P, papyrus
Papyrus 111	P, papyrus
Uncial 0243	P, papyrus
Minuscule 1739	P, papyrus
Minuscule 88	P, papyrus
Authorship of the Epistle to the Hebrews	P, papyrus
Egerton Gospel	P, papyrus
Rylands Library Papyrus P52	P, papyrus
Codex Vaticanus	P, papyrus

K-Means

TODO: think of different ways to represent it - maybe truncate some definitions?

Article	Identifiers
Direct shear test	$angle, \varphi, friction$
Truncated dodecadodecahedron	$golden, ratio, \phi$
Golden triangle (mathematics)	$golden, section, \theta, \phi$
Petrophysics	$percentage\varphi, symbol, S_w$
Greedy algorithm for Egyptian fractions	$golden, terms, d, \phi, denominator, possible, expa$
Pi Josephson junction	$\pi, junction, \varphi$
Snub dodecahedron	$golden, ratio, \tau, 2\xi, -, V, \xi$
Lucas number	$golden, terms, \phi, m, L, number, values, ratio, I$
54 (number)	$golden, \varphi, ratio$
Special right triangles	$\pi, c, b, ratio, c., \phi, m, golden, radians, n, line, in$
Universal code (data compression)	$golden, code, ratio, power, \phi, l, n, q, p$
Existential instantiation	$csymbol\varphi variable$
Perles configuration	$goldenratio\phi$
Wythoff array	$goldenratio\phi columnnmnumbern\varphi fib$
Golomb sequence	$a_n, n, golden, ratio, \phi$
Almost integer	$constant\pi, gelfonds, \varphi, exampl$
16:10	$golden, ratio, \phi$
Leonardo number	$golden, ratio, \phi, computations, \psi, L, n$
Rogers–Ramanujan continued fraction	$goldenfunctionsratioG\phi Hqmodu$
Great rhombic triacontahedron	$goldenratio\phi$
108 (number)	$golden\varphi ratio$
Random Fibonacci sequence	$goldenBratioM_nprobability\phi f_nsequencenrandominc$
Rhombic triacontahedron	$goldenratio\phi r_iSedger_mV$
Exact trigonometric constants	$function\pi ratio\phi imagegoldenvaluesxV$
Bilunabirotunda	$goldenratio\phi$
Feigenbaum constants	$mapzratioplacesf\phi goldenc_n\alpha\delta varia$

Found some interesting clusters but in general doesn't show good results. Need to use semantic menthols

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

HEre results

5.2 Result analysis

Hierarcical methods are too slow, and SLINK is not good. Bisecting K-Means is good for explaining steps but not very practical

MiniBatch K means is preferred to usual KMeans: fast but same results

NMF takes a lot of time to decompose a matrix with large k

$k = 100$ 30 min, but with results inferior to SVD $k = 250$ 2 hours, with results comparable to SVD

The complexity of NMF is $O(kn)$

The best definition embedding technique is soft association The best clsutering algorithm is K-Means with $K = 10000$ on the semantic space produced by rank-reduced SVD with $k = 200$

5.3 Building Hierarchy

How to evaluate???

5.4 Experiment Conclusions

6 Conclusions

The results are super.

6.1 Future Work

the work is done assuming that document imports only from one namespace but it can import from several. can solve that by dividing the document in parts (e.g. by paragraphs) and then applying the same analysis independently to each paragraph - instead of each document.

Can use additional information from wiki articles. For example, can extract some keywords from the article and use it in clustering

Or interwiki pages.

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 [24]

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

Other dim red techniques for LSA, e.g. Local NMF [71] There should also be randomized NMF that works faster.

Try other clustering techniques: spectral clustering [72] other ways to embed identifiers like word2vec [73] or GloVe [74]

How to extend this method to situations when no additional information about document category is known. I.e. need to replace the notion of purity with some other objective for discovering namespaces and namespace-defining clusters

Micro-clustering (http://arxiv.org/abs/1507.03067) Micro-Clustering: Finding Small Clusters in Large Diversity

7 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. Anders Møller and Michael I Schwartzbach. *An introduction to XML and Web Technologies*. Pearson Education, 2006.
3. 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/.
4. Kevin McArthur. What's new in PHP 6. *Pro PHP: Patterns, Frameworks, Testing and More*, pages 41–52, 2008.
5. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java 8® Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, 2014.
6. Craig Larman. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Pearson Education India, 2005.
7. Jon Barwise, John Etchemendy, Gerard Allwein, Dave Barker-Plummer, and Albert Liu. *Language, proof and logic*. CSLI publications, 2000.
8. Wikipedia. Mathematical notation — Wikipedia, the free encyclopedia, 2015. https://en.wikipedia.org/w/index.php?title=Mathematical_notation&oldid=646730035, accessed: 2015-07-01.
9. 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.
10. Dan Jurafsky and James H Martin. *Speech & language processing*. Pearson Education India, 2000.
11. Ulf Schöneberg and Wolfram Sperber. POS tagging and its applications for mathematics. In *Intelligent Computer Mathematics*, pages 213–223. Springer, 2014.
12. Robert Pagael and Moritz Schubotz. Mathematical language processing project. *arXiv preprint arXiv:1407.0167*, 2014.
13. Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project (3rd revision). 1990.
14. 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.
15. Mihai Grigore, Magdalena Wolska, and Michael Kohlhase. Towards context-based disambiguation of mathematical expressions. In *The Joint Conference of ASCM*, pages 262–271, 2009.
16. 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.
17. 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.
18. Jerzy Trzeciak. *Writing mathematical papers in English: a practical guide*. European Mathematical Society, 1995.
19. Giovanni Yoko Kristianto, Akiko Aizawa, et al. Extracting textual descriptions of mathematical expressions in scientific papers. *D-Lib Magazine*, 20(11):9, 2014.

20. Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
21. 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.
22. Alfio Gliozzo and Carlo Strapparava. *Semantic domains in computational linguistics*. Springer Science & Business Media, 2009.
23. Christopher Stokoe, Michael P Oakes, and John Tait. Word sense disambiguation in information retrieval revisited. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 159–166. ACM, 2003.
24. Nora Oikonomakou and Michalis Vazirgiannis. A review of web document clustering approaches. In *Data mining and knowledge discovery handbook*, pages 921–943. Springer, 2005.
25. Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining Text Data*, pages 77–128. Springer, 2012.
26. Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.
27. Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.
28. Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
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. Deborah Hughes-Hallett, William G. McCallum, Andrew M. Gleason, et al. *Calculus: Single and Multivariable, 6th Edition*. Wiley, 2013.
31. Tuomo Korenius, Jorma Laurikkala, and Martti Juhola. On principal component analysis, cosine and euclidean measures in information retrieval. *Information Sciences*, 177(22):4893–4905, 2007.
32. 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.
33. Rui Xu, Donald Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.
34. 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.
35. David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.
36. 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.
37. 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.
38. Hinrich Schütze and Craig Silverstein. Projections for efficient document clustering. In *ACM SIGIR Forum*, volume 31, pages 74–81. ACM, 1997.
39. 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.
40. Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. pages 83–103, 2004.
41. Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

42. 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.
43. Nicholas Evangelopoulos, Xiaoni Zhang, and Victor R Prybutok. Latent semantic analysis: five methodological recommendations. *European Journal of Information Systems*, 21(1):70–86, 2012.
44. Stanisław Osiński. Improving quality of search results clustering with approximate matrix factorisations. In *Advances in Information Retrieval*, pages 167–178. Springer, 2006.
45. Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
46. 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.
47. 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.
48. Wikimedia Foundation. Russian wikipedia XML data dump, 2015. http://dumps.wikimedia.org/ruwiki/latest/, downloaded from http://math-ru.wmflabs.org/wiki/, accessed: 2015-07-12.
49. Apache Software Foundation. Apache Flink 0.8.1. http://flink.apache.org/, accessed: 2015-01-01.
50. 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/.
51. Ronald Rivest. The MD5 message-digest algorithm. 1992.
52. Eclipse Foundation. Mylyn WikiText 1.3.0, 2015. http://projects.eclipse.org/projects/mylyn.docs, accessed: 2015-01-01.
53. Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.
54. Определение части речи слова на PHP одной функцией (part of speech tagging in PHP in one function), 2012. http://habrahabr.ru/post/152389/, accessed: 2015-07-13.
55. Daniel Sonntag. Assessing the quality of natural language text data. In *GI Jahrestagung (1)*, pages 259–263, 2004.
56. Julie D Allen et al. *The Unicode Standard*. Addison-Wesley, 2007.
57. Martin F Porter. Snowball: A language for stemming algorithms, 2001.
58. 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.
59. 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.
60. Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. http://www.scipy.org/, accessed: 2015-02-01.
61. 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.
62. A Tropp, N Halko, and PG Martinsson. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical report, 2009.
63. American Mathematical Society. AMS mathematics subject classification 2010, 2009. http://msc2010.org/, accessed: 2015-06-01.

64. American Physical Society. PACS 2010 regular edition, 2009. http://www.aip.org/publishing/pacs/pacs-2010-regular-edition/, accessed: 2015-06-01.
65. Bernard Rous. Major update to ACM's computing classification system. *Commun. ACM*, 55(11):12–12, November 2012.
66. 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.
67. Association for Computing Machinery. ACM computing classification system, 2012. https://www.acm.org/about/class/2012, accessed: 2015-06-21.
68. Daniel Krech. RDFLib 4.2.0. https://rdflib.readthedocs.org/en/latest/, accessed: 2015-06-01.
69. Sreenivasa Viswanadha, Danny van Bruggen, and Nicholas Smith. JavaParser 2.1.0, 2015. http://javaparser.github.io/javaparser/, accessed: 2015-06-15.
70. Apache Software Foundation. Apache Mahout 0.10.1. http://mahout.apache.org/, accessed: 2015-06-15.
71. Stan Z Li, Xin Wen Hou, HongJiang Zhang, and QianSheng Cheng. Learning spatially localized, parts-based representation. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–207. IEEE, 2001.
72. Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.
73. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013.
74. Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.