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

Abstract

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. For example, XML uses namespaces to prefix element names to ensure uniqueness and remove ambiguity between them, and the Java programming language uses packages to organize identifiers into namespaces for modularity.

In this thesis we extend the notion of namespaces to mathematical formulae. In mathematics, the same identifier may be used in different areas but denote different things: For example, "E" may refer to "energy", "expected value" or "elimination matrix", depending on the domain of the article where this identifier is used. By introducing namespaces to formulae, this ambiguity can be resolved. If we follow the XML approach and prepend namespace name to the identifier, e.g. "physics.E", then it will give additional context and make it clear that "physics.E" means "energy" rather than "expected value".

There is correlation between the identifiers used in a document and the namespace of its identifiers, and we exploit it to discover namespaces. We argue that document representation in terms of identifiers is similar to the traditional way of representing documents as a Bag of Words in the Vector Space Model. By comparing these approaches we develop a method for automatic namespace discovery based on techniques from document clustering.

To cluster documents we use established cluster analysis algorithms like K-Means, DBSCAN and Latent Semantic Analysis, and these techniques are evaluated on English Wikipedia. Additionally, we also use Russian Wikipedia to compare the obtained results, and try the same set of techniques to extract namespaces from Java source code.

The obtained results indicate that the identifier namespace discovery is possible, ... (**TODO**: finish when conclusion is finished).

Table of Contents

1 Introduction 5
  1.1 Namespaces in Computer Science 5
  1.2 Namespaces in Mathematical Notation 6
  1.3 Discovery of Identifier Namespaces 9
  1.4 Namespace Discovery by Cluster Analysis 10
  1.5 Thesis Outline 11
2 Mathematical Definition Extraction 13
  2.1 Math-aware POS tagging 13
  2.2 Extraction Methods 14
3 Namespaces as Document Clusters 18
  3.1 Vector Space Model 18
  3.2 Similarity Measures and Distances 21
  3.3 Document Clustering Techniques 23
  3.4 Latent Semantic Analysis 27
4 Implementation 31
  4.1 Data set 31
  4.2 Definition Extraction 32
  4.3 Document Clustering 40
  4.4 Experiments 44
  4.5 Building Hierarchy 63
  4.6 Java Language Processing 65
  4.7 Result Analysis and Experiment Conclusions 68
5 Conclusions 70
  5.1 Future Work 70
6 Bibliography 73

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].

Namespaces are used in XML (eXtensible Markup Language), which 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`.

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 long names like `Zend_Search_Lucene_Analysis_Analyzer`[1]. is the result.

[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, and 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

The idea of namespaces can be extended to identifiers in mathematical formulae.

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* ($+$, $\times$, ...) 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, including symbols with subscripts. For example, x, y, $\mathbf{w}$ are variables, but so are $x_1, x_2,$ ... or even w_{slope}.

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

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

which means that the term t can be a constant, a variable or an n-ary function $f(t_1, t_2, \ \dots \ , t_n)$. An *$n$-ary function* is an function that takes n terms $t_1, t_2, \ \dots \ , t_n$ and produces a new term t. 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 will still hold true and remain a valid formula. Nevertheless, there are research communities in mathematics that have developed a special system of naming these variables, and this naming system is called *mathematical notation* [8]. For each symbol in a formula, the notation assigns it 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, 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 try to address this problem 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 or *identifier* and s is its semantic meaning or *definition*, such that for any pair $(i, s) \in \mathcal{N}$ there does not exist another pair $(i', s') \in \mathcal{N}$ with $i = i'$. 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 $(n, \mathcal{N})$ where $\mathcal{N}$ is a notation and n is its *name*: a string that uniquely identifies the notation. For example, $(\text{“Physics”}, \mathcal{N}_{\text{physics}})$ can refer to the notation used in Physics. For convenience, in this work we can use the Java syntax to refer to specific entries of a namespace. If $(n, \mathcal{N})$ is a namespace and i is an identifier such that $(i, s) \in \mathcal{N}$ for some s, then “$n.i$” is a *fully qualified name* of the identifier i that relates i to the definition s. For example, given a namespace $\big(\text{“Physics”},\ \{(E, \text{“energy”}),\ (m, \text{“mass”}),\ (c, \text{“speed of light”})\}\big)$, “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 software packages: they have low coupling and high cohesion, meaning that all definitions in a namespace come from the same area of mathematical knowledge and the definitions from different namespace do not intersect heavily.

However, mathematical notation does not exist in isolation, and it is usually observed indirectly by its usage in documents. Therefore we need to introduce a document-centric view on mathematical namespaces: suppose we have a collection of n documents $\mathcal{D} = \{d_1, d_2, \ ... \ , d_n\}$ and a set of K namespaces $\big\{(n_1, \mathcal{N}_1), (n_2, \mathcal{N}_2), \ ... \ , (n_K, \mathcal{N}_K)\big\}$. 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. For example, a document “Energy-mass equivalence” would import “Physics.E”, “Physics.m”, and “Physics.c”, and then these identifiers can be used in formulae of this document unambiguously.

Then a namespace exhibits low coupling if it is used only in a small subset of documents, and high cohesion if all the documents in this subset are related to the same domain.

But in real-life scientific document there are no import statements in the document preamble, and they contain only natural language texts along with some mathematical formulae. Yet we may still assume that these import exists, but they are implicit, i.e. they are latent and cannot be observed directly. Additionally, the namespaces themselves are also not observed.

Typically in mathematical texts, when an identifier is first introduced, its definition is given in the natural language description that surrounds the formula. This description can be extracted and used to assign the meaning to the identifiers. Once identifier definitions are extracted, a document can be represented as a set of identifier-definition pairs, and these pairs can be used to discover the namespaces.

In this work we study the problem of namespace discovery and the goal of this work is to **automatically discover a set of identifier namespaces given a collection of documents**.

In the next section we discuss how this problem can be addressed.

1.3 Discovery of Identifier Namespaces

How we can construct a set of namespaces given a collection of documents? It is possible to do manually by assigning each identifier/definition pair to some namespace, but it is expensive and very time consuming. Therefore in this work we suggest a different approach: use Machine Learning techniques for discovering namespaces automatically.

We illustrate our idea by first drawing an analogy between identifier namespaces and namepsaces in programming languages. In a well-designed application, we can distinguish between two types of application packages [9]:

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

For example, for an application `org.company.app` there can be 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 packages `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 type 2 packages mostly use other packages of type 1: they depend on them.

This idea can be extended on the document-centric view on identifier namespaces. Each document can be seen as a class that imports identifiers defined in other documents. Then the documents can be grouped together based on the identifiers and the definitions they have, and then among these groups there are some groups of documents that are of *type 1* and the rest are of *type 2*. The type 1 document groups contain information about closely related concepts, and they are very homogenous (they have high cohesion), and they are also not highly coupled with other document groups. By using the import metaphor, we can say that the type 1 document groups import only from few closely related namespaces. Other documents are of *type 2* and they do not have low coupling: they are not very homogenous and they import from several namespaces

With this intuition we can refer to *type 1* document 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. Once the namespace defining groups are found, we can learn the namespace of these document.

Thus we need to find groups of homogenous documents given a collection, and this is exactly what Cluster Analysis methods do.

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 Namespace Discovery by Cluster Analysis

We believe that cluster analysis techniques developed for text documents should also work for cases when documents are represented by identifers they contain.

First, let us consider the characteristics of text data. A natural language typically contains many different words. If $\mathcal{V}$ is a set of all possible words in a document collection, then usually $|\mathcal{V}| \approx 10^5$, but each individual document may contain only a small portion of these words [10]. Numbers of words across different documents may wary a lot, and the word distribution usually follows some power law distribution, e.g. Zipf's law [10]. Power laws are commonplace [11], so it is safe to assume that number of identifiers across documents are also distributed according to some power law. Intuitively, it is true because there are many identifiers like x or n that are very frequent and used in all mathematical articles, while there are some quite specific identifiers like ℓ_∞ that do not occur very often. We verify this assumption in the implementation chapter (see Definition Extraction, section 4.2).

Additionally, natural languages suffer from lexical problems of variability and ambiguity, and the two main problems are synonymy and polysemy [12] [13]:

- 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* [14]: either replace a word with its sense [15] 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 if we have an identifier “E” which is defined as “energy”, then “E” can be replaced with “E_energy”.

Thus we see that text representation of documents and identifier representation of documents have many similarities and therefore we can apply the set of techniques developed for text representation for clustering documents based on identifiers.

For document clustering, documents are usually represented using Vector Space Models [16] [17]. Likewise, we can introduce “Identifier Space Model” analogously to Vector Space Models, and then we can apply clustering algorithm to documents represented in this space.

1.5 Thesis Outline

As discussed, it is important to disambiguate identifiers in mathematical formulae. One way of doing it is extracting definitions of these identifiers from the document, and we discuss the methods for doing it in chapter 2.

In chapter 3 we review the vector space model of representing documents and cluster analysis methods used for finding groups in documents; and we also study how matrix decomposition techniques can be used to extract semantic information from the corpus.

Finally, we describe how these techniques are implemented and evaluated in chapter 4.

2 Mathematical Definition Extraction

In Natural Language Processing, Word Sense Disambiguation is a problem of identifying in which sense a polysemous word is used [14]. Analogously, the Identifier Disambiguation problem is a problem of determining the meaning of an identifier in a mathematical formula. This problem is typically solved by extracting definitions from the natural language description that surrounds the formula.

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") and (c, "the speed of light").

Formally, a phrase that defines a mathematical expression consists of three parts [18]:

– *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. We refer to definiendum as identifier, and to definiens as definition, so relations are identifier-definition pairs.

In this chapter we will discuss how the relations can be discovered automatically. It is organized as follows: first, we discuss Part-of-Speech Tagging and its application to mathematical texts in section 2.1 and then review the extraction methods in section 2.2.

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 [14]. 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 [19] [20].

A *POS tag* is an abbreviation that corresponds to some part of speech. Penn Treebank POS Scheme [21] 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 [22]), and then refining the results by re-tagging math-related tokens of text as `MATH` [19].

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

2.2.1 Nearest Noun Method

The Nearest Noun [23] [24] 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 a relation (σ, "bijection").

2.2.2 Pattern Matching Methods

The Pattern Matching method [25] 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 lists: one is extracted from a guide to writing mathematical papers in English ([26]), and another is extracted from "Graphs and Combinatorics" papers from Springer [18].

The pattern matching method is often used as the baseline method for identifier-definition extraction methods [18] [27] [20].

2.2.3 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 a 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 [24] [27]:

- boolean features for each of the patterns from section 2.2.2 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 [27] [24] and Conditional Random Fields [27], but, in principle, any other binary classifier can be applied as well.

2.2.4 Probabilistic Approaches

In the Mathematical Language Processing approach [20] a definition for an identifier is extracted by ranking candidate definitions by the probability of definining 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:

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

where $\mathit{\Delta}$ is the number of tokens between the identifier and the definition candidate, and $R_{\sigma_d}(\mathit{\Delta})$ is a Gaussian that models this distance, parametrized with σ_d; n is the number of sentences between the definition candidate and the sentence where the identifier occurs for the first time, and is a Gaussian parameterized with σ_s; finally $\mathrm{tf}(t)$ is a frequency of term t in a sentence. All these quantities are combined together and α, β, γ are weighting parameters.

The following weighting parameters α, β, γ are proposed in [20]: $\alpha = \beta = 1$ 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 3.1 we describe the Vector Space Model (VSM): the traditional way of representing a collection of documents as vectors, and also extend it to identifiers and discuss how definitions can be incorporated in this space. Then, we go over common similarity and distance functions that are useful for document clustering in section 3.2. Next, we review common techniques for document clustering in section 3.3 and the Latent Semantic Analysis method in section 3.4.

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 [10], but it is also used for various Text Mining tasks such as Document Classification [28] and Document Clustering [16] [17].

In Vector Space Model we make two assumptions about the data: (1) *Bag of Words assumption*: the order of words is not important, only word counts; (2) *Independence assumption*: we treat all words as independent. Both assumptions are quite strong, but nonetheless this method often gives good results.

Let $\mathcal{V} = \{t_1, t_2, \ ... \ , t_m\}$ be a set of n terms. Then documents can be represented as m-vectors $\mathbf{d}_i = (w_1, w_2, \ ... \ , w_m)$, where w_j is the weight of term t_j in the document $\mathbf{d}_i$, and the document collection can be represented by a *term-document matrix* D, where columns of D are document vectors $\mathbf{d}_1, \mathbf{d}_2, \ ... \ , \mathbf{d}_n$ and rows of D are indexed by terms $t_1, t_2, \ ... \ , t_m$ (see fig. 1).

There are the following term weighting schemes [10]:

- 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 terml
- 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. Sometimes a term is used too often in a document, and we want to reduce its influence, and this is typically done by applying some sublinear transformation to TF, for instance, a square root or a logarithms.

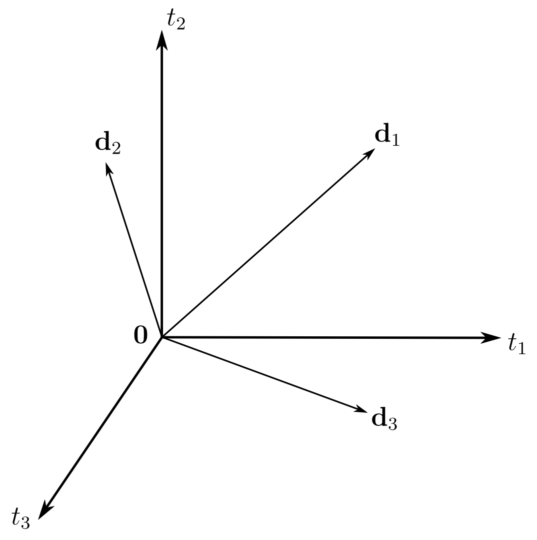


Fig. 1: Documents $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3$ in a document space with dimensions t_1, t_2, t_3.

Document Frequency (DF) weights terms by their global frequency in the collection, which is the number of documents that contain the token. But 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 we use **Inverse Document Frequency (IDF)** to give more weight to rare words rather than to frequent words.

A good weighting system gives the best performance when it assigns more weights to terms with high TF, but low DF [29]. This can be achieved by combining both TF and IDF schemes. 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. The TF and IDF are combined together in **TF-IDF** weighting scheme:

$$\text{tf-idf}(t, \mathbf{d}) = (1 + \log \text{tf}(t, \mathbf{d})) \cdot \log \frac{n}{\text{df}(t)},$$

where $\text{tf}(t, \mathbf{d})$ is term frequency of term t in document $\mathbf{d}$ and $\text{df}(t)$ is the document frequency of term t in the document collection.

The Vector Space Model can be adjusted to represent documents by identifers they contain instead of words. To do that we replace the vocabulary

$\mathcal{V}$ with a set of identifiers $\mathcal{I} = \{i_1, i_2, \ ... \ , i_m\}$, but documents are still represented as m-vectors $\mathbf{d}_j = (w_1, w_2, \ ... \ , w_m)$, where w_k is a weight of identifier i_k in the document $\mathbf{d}_j$. Likewise, we can define an identifier-document matrix D as a matrix where columns are document vectors and rows are indexed by the identifiers.

Identifiers, as terms, suffer from the problems of synonymy and polysemy, and we solve this problem by extracting definitions for all the identifiers. There are several ways of incorporating the extracted definitions into the model:

- do not include definition information at all, use only identifiers;
- 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 three relations (E, "energy"), (m, "mass") and (c, "speed of light"), and three documents $d_1 = \{E, m, c\}, d_2 = \{m, c\}, d_3 = \{E\}$. Then

- no definitions: dimensions are (E, m, c) and the identifier-document matrix is

$$D = \left[\begin{array}{r|ccc} & d_1 & d_2 & d_3 \\ \hline E & 1 & 0 & 1 \\ m & 1 & 1 & 0 \\ c & 1 & 1 & 0 \end{array}\right];$$

- "weak" association: dimensions are (E, m, c, energy, mass, speed of light), and the matrix is

$$D = \left[\begin{array}{r|ccc} & d_1 & d_2 & d_3 \\ \hline E & 1 & 0 & 1 \\ m & 1 & 1 & 0 \\ c & 1 & 1 & 0 \\ \text{energy} & 1 & 0 & 1 \\ \text{mass} & 1 & 1 & 0 \\ \text{speed of light} & 1 & 1 & 0 \end{array}\right];$$

– "strong" association: dimensions are (E_energy, m_mass, c_speed of light), and the matrix is

$$D = \left[\begin{array}{r|ccc} & d_1 & d_2 & d_3 \\ \hline E\_\text{energy} & 1 & 0 & 1 \\ m\_\text{mass} & 1 & 1 & 0 \\ c\_\text{speed of light} & 1 & 1 & 0 \end{array}\right].$$

3.2 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 consider Euclidean distance, inner product, cosine similarity and Jaccard coefficient.

3.2.1 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. The square of the Euclidean distance is defined as

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

This 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 [30].

3.2.2 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 [31].

3.2.3 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 $\cos\theta = \mathbf{x}^T\mathbf{z}\,/\,(\|\mathbf{x}\|\,\|\mathbf{z}\|)$.

We do not need the angle itself and can use the cosine directly [10]. 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 a distance function. 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 [32], but it is nonetheless useful.

The cosine distance and the Euclidean distance are connected [32]. For two unit-normalized vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ the Euclidean distance between them is $\|\mathbf{d}_1 - \mathbf{d}_2\|^2 = 2 - 2\,\mathbf{d}_1^T\mathbf{d}_2 = 2\,d_c(\mathbf{d}_1, \mathbf{d}_2)$. Thus we can use Euclidean distance on unit-normalized vectors and interpret it as cosine distance.

3.2.4 Jaccard Coefficient

Finally, the Jaccard Coefficient is a function that compares how similar two sets are. Given two sets A and B, it is computed as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. It is also applicable to document vectors with binary weights, and it can be defined as $J(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1^T\mathbf{d}_2}{\|\mathbf{d}_1^T\|^2 + \|\mathbf{d}_2^T\|^2 - \mathbf{d}_1^T\mathbf{d}_2}$ [10].

3.3 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 [17]. In Information Retrieval clustering is used to assist the users and group retrieved results into clusters [33].

There are several types of clustering algorithms: hierarchical (agglomerative and divisive), partitioning, density-based, and others.

3.3.1 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**, when the clusters are merged based on the closest pair; **Complete Linkage**, when the clusters are merged based on the worst-case similarity – the similarity between the most distant objects on the clusters; **Group-Average Linkage**, based on the average pair-wise similarity between all objects in the clusters; and **Ward's Method** when the clusters to merge are chosen to minimize the within-cluster error between each object and its centroid is minimized [16].

Among these algorithms only Single Linkage 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 [34].

3.3.2 *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 [35]: given a desired number of clusters K, it iteratively improves the Euclidean distance between each data point and the centroid, closest to it.

Let $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ ... \ , \mathbf{d}_n\}$ be the document collection, where documents $\mathbf{d}_i$ are represented is a 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 $c_i \in \{1, 2, \ ... \ , k\}$, which specifies to what cluster centroid $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ ... \ , \boldsymbol{\mu}_k$ 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 repeated until the algorithm converges. The pseudocode for K-Means is presented in the listing 1.

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
            $\mathcal{C}_j \leftarrow \{ \mathbf{d}_i$ 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 $(c_1, c_2, \ ... \ , c_n)$
```

Usually, K-Means shows very good results for document clustering, and in several studies it (or its variations) shows the best performance [34] [36] .

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 [37] uses Mini-Batch Gradient Descent method, which is a different optimization technique that converges faster.

K-Means uses Euclidean distance, which does not always behave well in high-dimensional sparse vector spaces like document vector spaces. However, as discussed in section 3.2, if document vectors are normalized, the Euclidean distance and cosine distance 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. For example, Bisecting K-Means [34] 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 insight into the clustering process. Additionally, in the results it produces clusters of comparable sizes.

The algorithm is simple: (1) start with a single cluster; (2) choose a cluster to split (for example, the largest one); (3) apply traditional K-Means to this cluster with $K = 2$ to split it; (4) repeat until have desired number of clusters.

However, when there are many documents, the centroids tend to contain a lot of words, which leads to a significant slowdown. To solve this problem, some terms of the centroid can be truncated. There are several possible ways of truncating 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].

3.3.3 DBSCAN

DBSCAN is a density-based 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 (see listing 2).

Algorithm 2 DBSCAN

```
function DBSCAN(database $\mathcal{D}$, radius $\varepsilon$, MinPts)
    result $\leftarrow \varnothing$
    for all $p \in \mathcal{D}$ do
        if $p$ is visited then
            continue
        mark $p$ as visited
        $\mathcal{N} \leftarrow$ Region-Query$(p, \varepsilon)$                ▷ $\mathcal{N}$ is the neighborhood of $p$
        if $\mathcal{N} <$ MinPts then
            mark $p$ as NOISE
        else
            $\mathcal{C} \leftarrow$ Expand-Cluster$(p, \mathcal{N}, \varepsilon, \text{MinPts})$
            result $\leftarrow$ result $\cup \{\mathcal{C}\}$
    return result

function Expand-Cluster(point $p$, neighborhood $\mathcal{N}$, radius $\varepsilon$, MinPts)
    $\mathcal{C} \leftarrow \{p\}$
    for all $x \in \mathcal{N}$ do
        if $x$ is visited then
            continue
        mark $x$ as visited
        $\mathcal{N}_x \leftarrow$ Region-Query$(x, \varepsilon)$                ▷ $\mathcal{N}_x$ is the neighborhood of $x$
        if $|\mathcal{N}_x| \geqslant$ MinPts then
            $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}_x$
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}$
    return $\mathcal{C}$

function Region-Query(point $p$, radius $\varepsilon$)
    return $\{x \ : \ \|x - p\| \leqslant \varepsilon\}$                ▷ all points within distance $\varepsilon$ from $p$
```

The details of implementation of Region-Query are not specified, and it can be implemented differently. For example, it can use Inverse Index to make the similarity search faster [10] [30].

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 $\{x \ : \ \text{similarity}(x, p) \geqslant \varepsilon\}$.

Shared Nearest Neighbors Similarity (SNN Similarity) [30] is a special similarity function that is particularity useful for high-dimensional spaces, it works well with DBSCAN, and it is applicable to document clustering and topic discovery [40].

SNN Similarity is specified in terms of the K nearest neighbors. Let $\text{NN}_{K,\,\text{sim}}(p)$ 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

$$\text{snn}(p, q) = \big|\text{NN}_{K,\,\text{sim}}(p) \cup \text{NN}_{K,\,\text{sim}}(q)\big|.$$

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 $\{q \ : \ \text{snn}(p, q) \geqslant \varepsilon\}$. For pseudocode, see the listing 3.

Algorithm 3 SNN Clustering Algorithm

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

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 [30].

3.4 Latent Semantic Analysis

In section 1.4 we have discussed the lexical variability and ambiguity problems in natural language: synonymy and polysemy. We can treat these problems as "statistical noise" and apply dimensionality reduction techniques to find the optimal dimensionality for the data and thus reduce the amount of noise there. This technique is called Latent Semantic Analysis (LSA) [41] or Latent Semantic Indexing [12], and it is often used for document clustering [17] [42].

There are three major steps in Latent Semantic Analysis [43]: (1) preprocess documents; (2) construct a term-document matrix D using the Vector Space Model; (3) de-noise D by reducing its dimensionality with Singular Value Decomposition (SVD).

The first two steps are the same as for traditional Vector Space Models and in the result we obtain a term-document matrix D. If D has rank r, then the SVD of D is $D = U \Sigma V^T$, where U is an $m \times r$ orthogonal matrix; Σ is a diagonal $r \times r$ matrix with singular values ordered by their magnitude; and V is an $n \times r$ orthogonal matrix.

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 [12] 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]. Once D is decomposed as $D \approx U_k \Sigma_k V_k^T$ it is enough to keep only the low dimensional representation $\hat{D} = V_k \Sigma_k$: the calculation of inner product between two documents i and j in the reduced semantic space corresponds to computing the inner product between ith and jth rows of $\hat{D}$ [12]. Since the Euclidean distance is defined in terms of inner product, it can also be used directly on the rows of $\hat{D}$.

Therefore, a generic LSA-based clustering algorithm consists of the following steps:

1. Build a term-document matrix D from the document collection;
2. Select number of latent concepts k and apply rank-reduced SVD on D to get $\hat{D} = V_k \Sigma_k$;
3. Apply the cluster algorithm on the rows of $V_k \Sigma_k$.

LSA has some drawbacks. Because SVD looks for an orthogonal basis for the new reduced document space, there could be negative values that are harder to interpret, and what is more, the cosine similarity can become negative as well. However, it does not significantly affect the cosine distance: it still will always give non-negative results.

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]. Using NMF solves the problem of negative coefficients: when it is applied to non-negative data such as term-document matrices, NMF produces non-negative rank-reduced approximations.

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 [46] (see fig. 2).

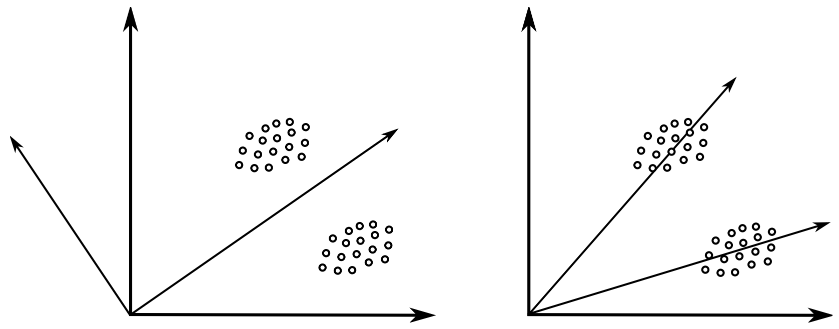


Fig. 2: Directions found by SVD (on the left) vs directions by NMF (on the right)

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]. When D is a term-document matrix and $D \approx UV^T$, then 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]:

1. Construct the term-document matrix D and perform NMF on D to get U and V;

2. Normalize rows $\mathbf{v}_i$ of V by using the rule $\mathbf{v}_i \leftarrow \mathbf{v}_i \, \|\mathbf{u}_i\|$;
3. Assign document $\mathbf{d}_i$ to cluster x if $x = \arg\max_j (V)_{ij}$.

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 4.2.1). Next, the implementation of clustering algorithms is described in the section 4.2.1. 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 24, 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 2.2.4. The extraction process is implemented using Apache Flink [49] and it is based on the open source implementation provided by Pagael and Schubotz in [20][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 [20] 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) [22]. 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 and we were not able to find a suitable implementation of any other POS taggers in Java. 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 integrated into the StanfordNLP pipeline. The English tagger uses the Penn Treebank POS Scheme [21], 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 2.2.4). 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 [20] with very minor modifications.

4.2.1 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.

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. "$\vec{x}$") 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. "$\mathbb{R}$"), and so on.

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 "$\vec{x}$", "$\boldsymbol{x}$" or "$\mathbf{x}$", and a real line by "$\mathbb{R}$", "$\mathbf{R}$" or "$\mathfrak{R}$". In natural languages there are related problems of lexical ambiguity such as synonymy, when different words refer to the same concept, and it can be solved by replacing the ambiguous words with some token, representative of the concept. Therefore this problem with identifiers can be solved similarly by reducing identifiers to their "root" form. This can be done by discarding all additional visual information, such that "$\bar{X}$" becomes "X", "$\mathbf{w}$" becomes "w" and "$\mathfrak{R}$" becomes "R". The disadvantage of this approach is that we lose the additional semantic information about the identifier that overwise could be useful.

The diacritic marks can easily be discarded because they are represented by special MathML instructions that can be ignored when the identifiers are retrieved. 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). Some identifiers (such as "$\hbar$" or "ℓ") are expressed using characters from a special "Letterlike Symbols" table, and these characters are normalized as well.

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

To filter out these one-symbol false identifiers we fully exclude all characters from the following unicode blocks: "Spacing Modifier Letters", "Miscellaneous Symbols", "Geometric Shapes", "Arrows", "Miscellaneous Technical", "Box Drawing", "Mathematical Operators" (except "∇" which is sometimes used as an identifier) and "Supplemental Mathematical Operators" [56]. Some symbols (like "$=$", "$+$", "$\tilde{}$", "%", "?", "!") 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: operator and function names like "`sin`", "`cos`", "`exp`", "`max`", "`trace`"; words commonly used in formulas like "`const`", "`true`", "`false`", "`vs`", "`iff`"; auxiliary 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. The stop list of false positives is quite similar for both English and Russian: for the Russian wikipedia we only need to handle the auxiliary words such as "где" ("where"), "иначе" ("else") and so on. The names for operators and functions are more or less consistent across both data sources.

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" ("элемент"), "number" ("число") or "variable" ("переменная") 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 elements of the stop list are also consistent across both data data sets, in the sense that the false definition candidates are same but expressed in different languages.

The Russian language is highly inflected, and because of this extracted definitions have many different forms, depending on grammatical gender, form (singular or plural) and declensions. This highly increases the variability of the definitions, and to reduce it lemmatize the definitions: they are reduced to the same common form (nominative, singular and masculinum). This is done using Pymorphy2: a python library for Russian and Ukrainian morphology [57].

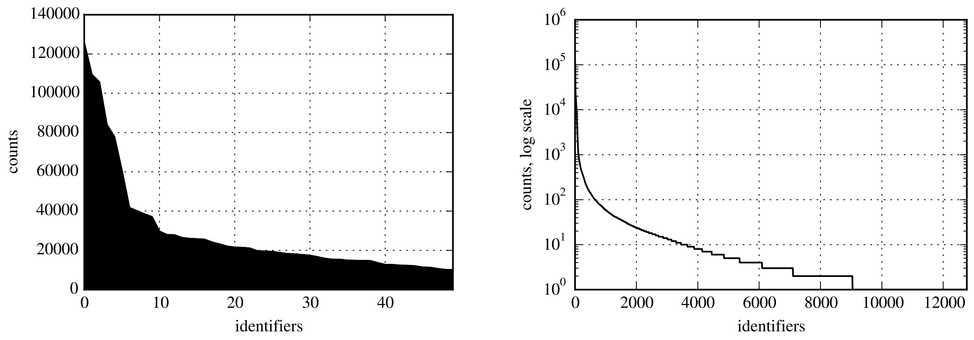
At the next stage the retrieved identifier/definition pairs are used for document clustering. Some definitions are used only once and we can note that they are not very useful because they do not have any discriminative power. Therefore all such definitions are excluded.

4.2.2 Dataset Statistics

During the data cleaning at the identifier extraction step some false identifiers are discarded, and after that some documents become empty: they contain no identifiers at all, and these documents are no longer considered for the analysis. Additionally, we discard all the documents that have only one identifier. This leaves only 22 515 documents out of 30 000, and they contain 12 771 distinct identifiers, which occur about 2 million times.

The most frequent identifiers are x (125 500 times), p (110 000), m (105 000 times) and n (83 000 times), but about 3 700 identifiers occur only once and

1 950 just twice. Clearly, the distribution of identifiers follows some power law distribution (see fig. 3).



(a) Frequencies of the first 50 identifiers

(b) Frequencies, log scale

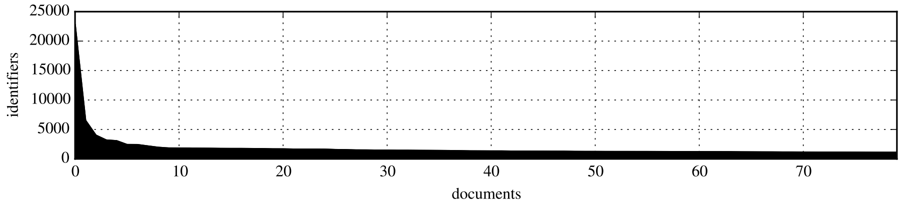
Fig. 3: Distribution of frequencies of identifiers

The distribution of counts for identifiers inside the documents also appears to follow a long tail power law distribution: there are few articles that contain many identifiers, while most of the articles do not (see fig. 4a). The biggest article ("Euclidean algorithm") has 22 766 identifiers, a and the second largest ("Lambda lifting") has only 6 500 identifiers. The mean number of identifiers per document is 33. The distribution for number of distinct identifiers per document is less skewed (see fig. 4b). The largest number of distinct identifiers is 287 (in the article "Hooke's law"), and it is followed by 194 (in "Dimensionless quantity"). The median number of identifiers per document is 10.

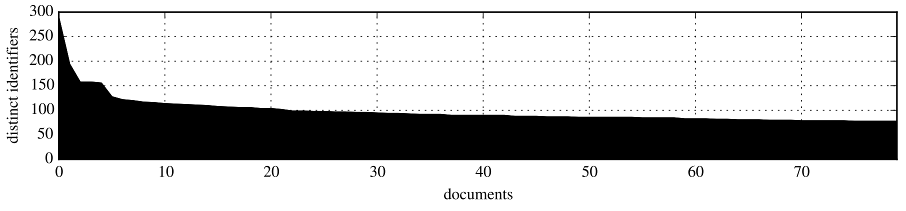
For 12 771 identifiers the algorithm extracted 115 300 definitions, and the number of found definitions follows a long tail distribution as well (see fig. 4c), with the median number of definitions per page being 4.

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

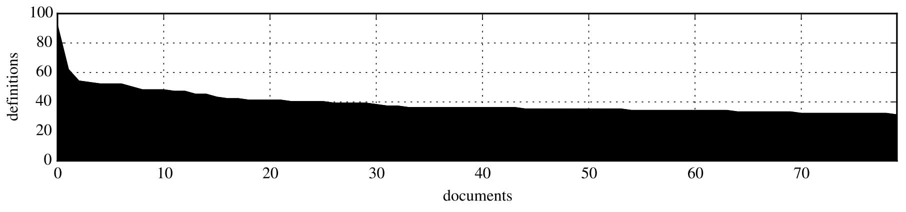
- t: "time" (1086)
- m: "mass" (424)
- θ: "angle" (421)
- T: "temperature" (400)
- r: "radius" (395)
- v: "velocity" (292)



(a) Identifier frequencies per document for first 80 most largest documents



(b) No. of distinct identifiers per document for first 80 most largest documents



(c) Definitions per document

Fig. 4: Frequencies per documents

- 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)

In Russian Wikipedia only 5 300 articles contain enough identifiers, and the remaining 9 500 are discarded.

The identifiers and definitions extracted from the Russain version of Wikipedia exhibit the similar properties. The most frequently occurring identifier is x with 13 248 occurrences, but the median frequency of an identifer is only 3 times. The article with the largest number of identifiers is “Уравнения Максвелла” (“Maxwell’s equations”) which contains 1 831 identifiers, while the median number of identifiers is just 3; the article with the largest number of distinct identifiers is also “Уравнения Максвелла” with 112 unique identifiers, and the median number of distinct identifiers in the data set is 5. Finally, the largest number of extracted definitions is 44 (again, for “Уравнения Максвелла”) with 2 being the median number of definitions per page.

The following is the list most frequent relations extracted from Russian wikipedia:

- 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 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 3.1) 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 [58] implemented in NLTK [59]: a python library for Natural Language Processing. For Russian we use Pymorphy2 [57].

Using `TfidfVectorizer` from scikit-learn [60] we vectorize each document.

We use the following settings:

A `use_idf=True, min_df=2`
B `use_idf=False, min_df=2`
C `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 (see section 3.3.2) implemented as a class `KMeans` in scikit-learn and Mini-Batch K-Means (class `MiniBatchKMeans`) [60]. Note that if rows are unit-normalized, then running K-Means with Euclidean distance is equivalent to cosine distance (see section 3.2.3).

Bisecting K-Means (see section 3.3.2) 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 3.3.2), was implemented manually using scipy [61] and numpy [62] because scikit-learn's implementation of K-Means does not allow using user-defined distances.

DBSCAN (section 3.3.3) and SNN Clustering (also section 3.3.3) 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.

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 3.4) which is implemented using randomized Singular Value Decomposition (SVD) [63], The implementation of randomized SVD is taken from scikit-learn [60] – method `randomized_svd`. Non-negative Matrix Factorization is an alternative technique for dimensionality reduction (section 3.4). Its implementation is also taken from scikit-learn [60], 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.

Unfortunately, there is no information about articles from Russian Wikipedia on DBPedia. However the number of documents is not very large, and therefore this information can be retrieved via MediaWiki API[6] individually for each document.

4.3.1 Building Namespaces

Once a cluster analysis algorithms assigns documents in our collection to some clusters, we need to find namespaces among these clusters. We assume that some clusters are namespace-defining: they are not only homogenous in the cluster analysis sense (for example, in case of K-Means it means that within-cluster sum of squares is minimal), but also "pure": they are about the same topic.

A cluster is *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

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

where C is a cluster, and c_i is some category. Thus we can select all clusters with purity above some pre-defined threshold and refer to them as namespace-defining clusters.

Then we convert these clusters into namespaces by collecting all the identifiers and their definitions in the documents of each cluster. To do this, we first collect all the identifier-definition pairs, and then group them by identifier. When extracting, each definition candidate is scored, and this score is used to determine, which definition an identifier will be assigned in the namespace.

For example, consider three documents with the following extracted relations:

- Document A:
  - n: (predictions, 0.95), (size, 0.92), (random sample, 0.82), (population, 0.82)

[6] http://ru.wikipedia.org/w/api.php

  - θ: (estimator, 0.98), (unknown parameter, 0.98), (unknown parameter, 0.94)
  - μ: (true mean, 0.96), (population, 0.89)
  - μ_4: (central moment, 0.83)
  - σ: (population variance, 0.86), (square error, 0.83), (estimators, 0.82)
- Document B:
  - P_θ: (family, 0.87)
  - X: (measurable space, 0.95)
  - θ: (sufficient statistic, 0.93)
  - μ: (mean, 0.99), (variance, 0.95), (random variables, 0.89), (normal, 0.83)
  - σ: (variance, 0.99), (mean, 0.83)
- Document C:
  - n: (tickets, 0.96), (maximum-likelihood estimator, 0.89)
  - x: (data, 0.99), (observations, 0.93)
  - θ: (statistic, 0.95), (estimator, 0.93), (estimator, 0.93), (rise, 0.91), (statistical model, 0.85), (fixed constant, 0.82)
  - μ: (expectation, 0.96), (variance, 0.93), (population, 0.89)
  - σ: (variance, 0.94), (population variance, 0.91), (estimator, 0.87)

We take all these relations, and combine together. If an identifer have two or more definitions that are exactly the same, them we merge them into one and its score is the sum of scores:

- P_θ: (family, 0.87)
- X: (measurable space, 0.95), (Poisson, 0.82)
- n: (tickets, 0.96), (predictions, 0.95), (size, 0.92), (maximum-likelihood estimator, 0.89), (random sample, 0.82), (population, 0.82)
- x: (data, 0.99), (observations, 0.93)
- θ: (estimator, 0.98+0.93+0.93), (unknown parameter, 0.98+0.94), (statistic, 0.95), (sufficient statistic, 0.93), (rise, 0.91), (statistical model, 0.85), (fixed constant, 0.82)
- μ: (random variables, 0.89+0.89+0.89), (variance, 0.95+0.93), (mean, 0.99), (true mean, 0.96), (expectation, 0.96), (normal, 0.83)
- μ_4: (central moment, 0.83)
- σ: (variance, 0.99+0.94), (population variance, 0.91+0.86), (estimator, 0.87), (square error, 0.83), (mean, 0.83), (estimators, 0.82)

There is some lexical variance in the definitions. For example, “variance” and “population variance” or “mean” and “true mean” are very related definitions, and it makes sense to group them together to form one definition.

It can be done by fuzzy string matching (or approximate matching) [64]. To implement it, we use a python library FuzzyWuzzy [65], and using fuzzy matching we group related identifiers and then sum over their scores.

Then we have the following:

– P_θ: (family, 0.87)
– X: (measurable space, 0.95), (Poisson, 0.82)
– n: (tickets, 0.96), (predictions, 0.95), (size, 0.92), (maximum-likelihood estimator, 0.89), (random sample, 0.82), (population, 0.82)
– x: (data, 0.99), (observations, 0.93)
– θ: (estimator, 2.84), (unknown parameter, 1.92), ({statistic, sufficient statistic}, 1.88), (rise, 0.91), (statistical model, 0.85), (fixed constant, 0.82)
– μ: (random variables, 2.67), ({mean, true mean}, 1.95), (variance, 1.88), (expectation, 0.96), (normal, 0.83)
– μ_4: (central moment, 0.83)
– σ: ({variance, population variance}, 3.7), ({estimator, estimators}, 1.69), (square error, 0.83), (mean, 0.83)

In a namespace an identifier can have at most one definition, and therefore the next step is selecting the definition with the highest score. This gives us the following namespace:

– (P_θ, family, 0.87)
– (X, measurable space, 0.95)
– (n, tickets, 0.96)
– (x, data, 0.99
– (θ, estimator, 2.84)
– (μ, random variables, 2.67)
– (μ_4, central moment, 0.83)
– (σ: variance, 3.7)

The category of this namespace is selected as the category that the majority of the documents in the namespace-defining cluster share.

4.4 Experiments

There are many different clustering algorithms, each with its own set of parameter. In this section we describe how we find the settings that find the best namespaces.

The following things can be changed:

- Ways to incorporate definition information (no definitions, soft association, hard association);
- Weighting schemes for the identifier-document matrix D: TF, sublinear TF, TF-IDF;
- There are different clustering algorithms: agglomerative clustering, DB-SCAN, SNN clustering, K-Means, Bisecting K-Means, each algorithm has its own set of parameters;
- Dimensionality of D can be reduced via SVD or NMF, parameter k controls the rank of output.

To find the best parameters set we use the grid search approach: we try different combinations of parameters and keep track on the number of pure clusters and the purity.

The overall purity of cluster assignment is calculated as a weighed sum of individual cluster purities, where the weight is chosen proportionally to the size of a cluster.

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. 5), but at some point the number of pure clusters will start decreasing (see fig. 6).

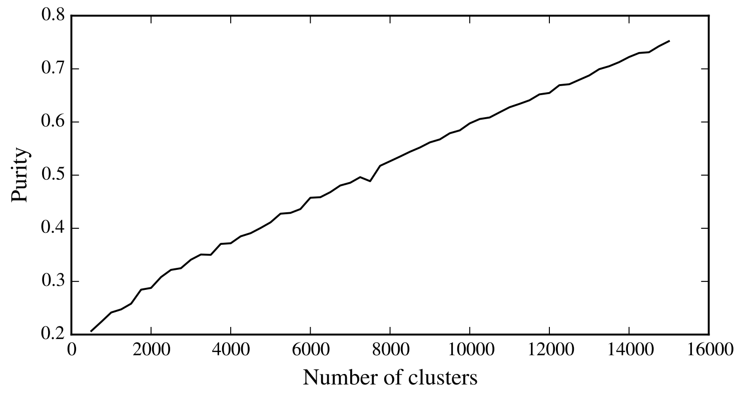


Fig. 5: K in K-Means vs overall purity of clustering: the purity increases linearly with K ($R^2 = 0.99$).

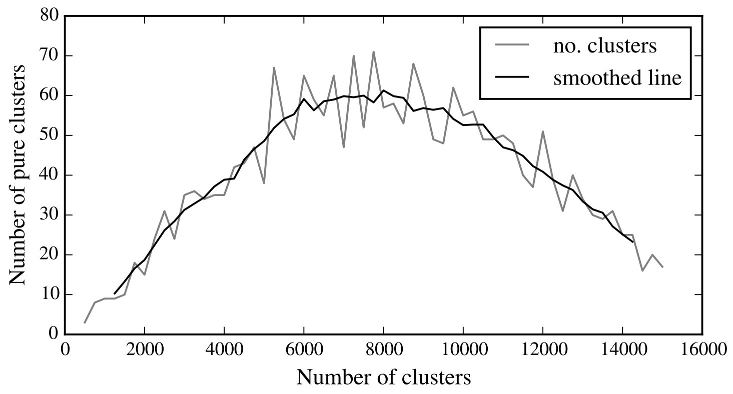


Fig. 6: K in K-Means vs the number of pure clusters: it grows initially, but after $K \approx 8\,000$ starts to decrease.

4.4.1 Baseline

We compare the performance of clustering algorithms against a random categorizer. The simplest version of such a categorizer is the random cluster assignment categorizer, which assigns each document to some random cluster. In this case, we constrain the categorizer to include 3 documents in each cluster, and once a document belongs to some cluster, it cannot be re-assigned. It is done by first creating a vector of assignments and shuffling it.

Then we record how many pure clusters (at least 80% pure) are in the cluster assignment.

We repeated this experiment for 200 times (see fig. 7): the maximal achieved value is 39, while the mean value is 23.85.

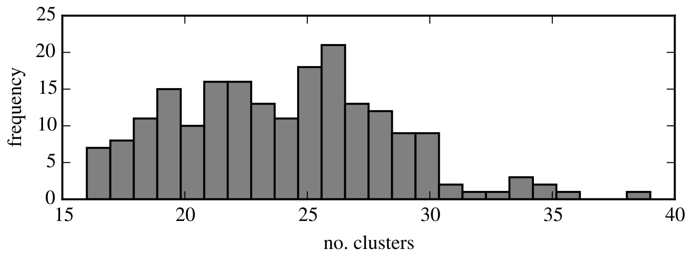


Fig. 7: Distribution of the number of pure clusters across 200 trials.

4.4.2 Only Identifiers

The first way of building the identifier space is to use only identifiers and do not use definitions at all.

If we do this, the identifier-document matrix is 6075×22512 (we keep only identifiers that occur at least twice), and it contains 302 541 records, so the density of this matrix is just 0.002.

First, we try to apply agglomerative clustering, then DBSCAN with SNN similarity based on Jaccard coefficient and cosine similarity, then we do K-Means and finally we apply LSA using SVD and NMF and apply K-Means on the reduced space.

Agglomerative clustering take too long to compute, they were not able to finish clustering in several hours, so we stopped the computation (**add graphs**). Additionally, the implementation we use from scikit-learn require a dense matrix, when densified, the identifeir-document matrix occupies a lot of space. Therefore we exclude these clustering algorithms from further analysis.

The second method is **DBSCAN** with **SNN Similarity**. To compute SSN similarity we need to use some other base similarity measure. We start with Jaccard coefficient, and use a binarized identifier-document matrix: a matrix with only ones and zeros. For example, the closest article to “Linear Regression” is “Linear predictor function” with Jaccard coefficient of 0.59 and “Low-rank approximation” is the closest to “Singular value decomposition” with coefficient of 0.25. With Jaccard, we were able to discover 87 clusters, which is two times better than the baseline (see fig. 8) and the best parameters are 10 nearest neighbors, $\varepsilon = 3$ and `MinPts` $= 4$ (see fig. 8b).

For the best result with 87 namespace defining clusters, in total 5 580 clusters were found and 4 219 documents are considered as “noise”. Let us consider top largest found clusters (see table 1). One of them, “Statistics”, is quite homogenous with purity of 0.84 (see table 2). Although some of the articles in this cluster are not related to statistics, all of them articles share the same identifier “β”. We can also check how β is defined across all namespace-defining clusters (see table 3). There are not many instance of β, and not all of them correct or meaningful: for example “i/ab” in the “Physics” cluster is a non-definition that was not successfully filtered out during the data cleaning phase.

Then we run the same algorithm, but with cosine similarity, using an identifier-document matrix with $\log \text{TF} \times \text{IDF}$ weights, and calculate pairwise similarity between each document. For example, let us take an article

Name	Size	Purity
Continuum mechanics	40	0.8000
Constellations listed by Ptolemy	17	1.0000
Statistics	13	0.8462
Knot theory	13	0.8462
Electromagnetism	12	0.8333
Bayer objects	11	1.0000
Analytic number theory	10	0.9000
Acoustics	9	0.8889
Chemical properties	9	0.8889
Set theory	9	0.8889
Combinatorics	8	0.8750
Theory of computation	8	1.0000
Partial differential equations	8	0.8750
Mathematical logic	8	0.8750

Table 1: Top namespace-defining clusters discovered by SNN DBSCAN with Jaccard coefficient.

Article Name	Identifiers
Partition of sums of squares	$y, \beta_0, \beta_1, \varepsilon_i, \beta_p, S, \varepsilon, y_i, \beta, ...$
Deming regression	$\sigma, \eta, \eta_i, \beta, \beta_0, \varepsilon_i, \varepsilon, \delta, y_i, x, ...$
Instrumental variable	$\beta, \varepsilon, \delta, x_1, Y, X, g, f, \varepsilon_i, \beta_k, ...$
Ordinary least squares	$\mathrm{w}_i, \alpha, \beta, \varepsilon, \sigma, E, \eta, \chi, \varepsilon_i, \beta_j, ...$
Endogeneity	$\gamma, \varepsilon_i, \beta_1, x_i, \alpha, \beta, \varepsilon, ...$
Linear predictor function	$x_i, \varepsilon_2, \beta_0, \varepsilon_1, \beta, \varepsilon, \phi_p, \varepsilon_n, ...$
Stochastic frontier analysis	$\beta_n, f, \beta_0, u_i, x_i, \beta, T, y_i, ...$
Proofs involving ordinary least squares	$x_i, \beta_0, \beta_1, \beta, \varepsilon, \pi, \sigma, \chi, \varepsilon_i, ...$
Projection pursuit regression	$\beta, \beta_0, j, S, \varepsilon, y_i, X, Y, x, x_i, ...$
De Casteljau's algorithm	$w_i, \beta_0, \beta_1, \beta, \beta_i, t, \beta_n, t_0, P_0, ...$
Regression analysis	$x_i, \beta_0, \beta, \varepsilon, y_i, \sigma, f, \beta_p, \varepsilon_i, ...$
Linear regression	$x_i, \varepsilon_2, \mathit{\Omega}, \beta_1, \beta_2, y_n, \varepsilon_1, \beta, \varepsilon, ...$
Least squares	$\nabla, \beta_j, \beta, \varepsilon, w_{ii}, Wy, \sigma, \varepsilon_i, ...$

(a) Identifiers and articles of the "Statistics" cluster (some identifiers are removed)

ID	Definition	Score
X_1	regressors	0.92
Y	regression model	0.95
f	linear predictor function	0.99
p	regressors	1.77
w_i	control points	1.66
y_0	mean response	0.99
y_i	dependent variable	3.71
z	causal relationship	0.90
α	confidence level	1.94
β	estimator	2.82
β_1	slope	1.82
ε	errors	2.90
ε_i	error term	2.71
σ	residual variance	2.68

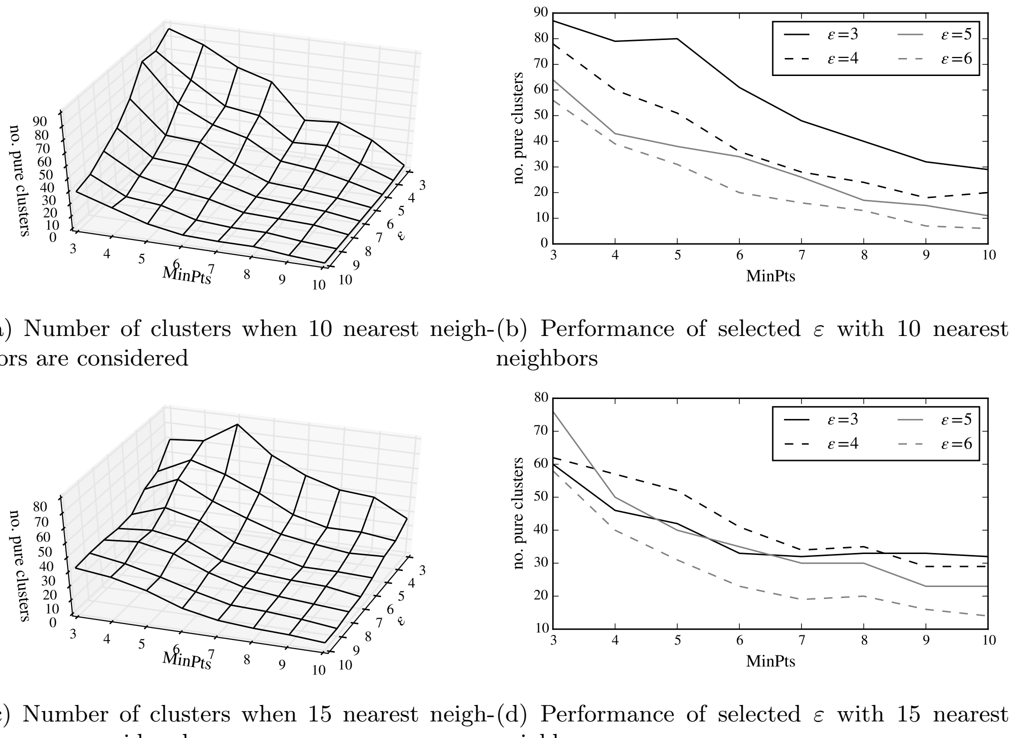
(b) Definitions in "Statistics" (some definitions are removed)

Table 2: A namespace-defining cluster about Statistics dicovered by SNN DBSCAN with Jaccard coefficient.

"Linear regression" and calculate the cosine with the rest of the corpus. The closest document is "Linear predictor function". They have 23 identifiers in common, and they indeed look related. However cosine is not always giving the best closets neighbors. For example, the nearest neighbor of "Singular

β			
Size	Namespace Name	Definition	Score
3	Complex analysis	absolute value	0.95
40	Continuum mechanics	bias	1.75
5	Fluid dynamics	momentum factor	0.95
5	Physics	i/ab	0.84
3	Quantum mechanics	voltage-to-barrier-field conversion factor	1.87
5	Radio frequency propagation	clutter factor	0.95
9	Set theory	successor ordinal	0.89
13	Statistics	estimator	2.82
6	Theoretical physics	quantity	0.96

Table 3: Definitions of β in the namespaces discovered by SNN DBSCAN with Jaccard coefficient.



(a) Number of clusters when 10 nearest neighbors are considered

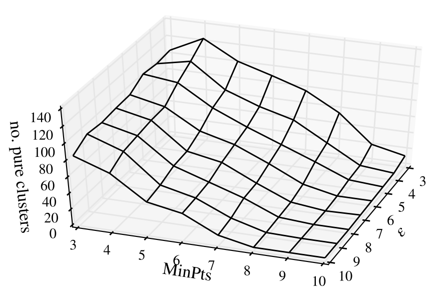
(b) Performance of selected ε with 10 nearest neighbors

(c) Number of clusters when 15 nearest neighbors are considered

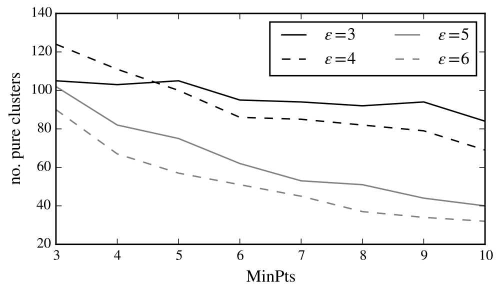
(d) Performance of selected ε with 15 nearest neighbors

Fig. 8: Effect of parameters ε, `MinPts` and number of nearest neighbors on performance of SNN DBSCAN when Jaccard coefficient is used.

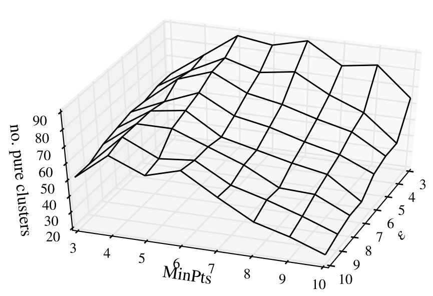
value decomposition" is "Rule of Sarrus", and although their cosine score is 0.92, they have only 3 identifiers in common.



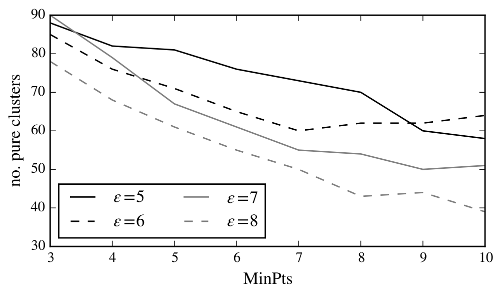
(a) Number of clusters when 10 nearest neighbors are considered



(b) Performance of selected ε with 10 nearest neighbors



(c) Number of clusters when 15 nearest neighbors are considered



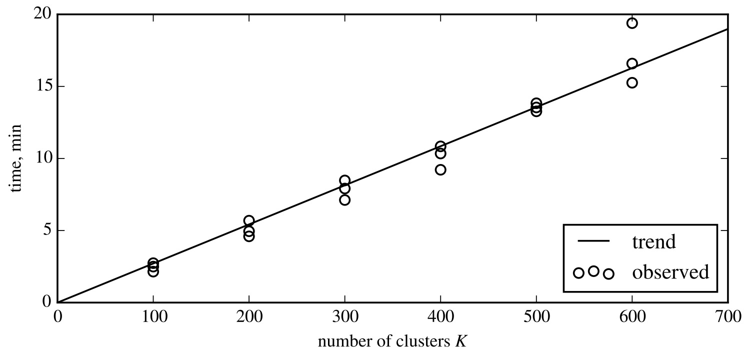
(d) Performance of selected ε with 15 nearest neighbors

Fig. 9: Effect of parameters ε, `MinPts` and number of nearest neighbors on performance of SNN DBSCAN when cosine is used.

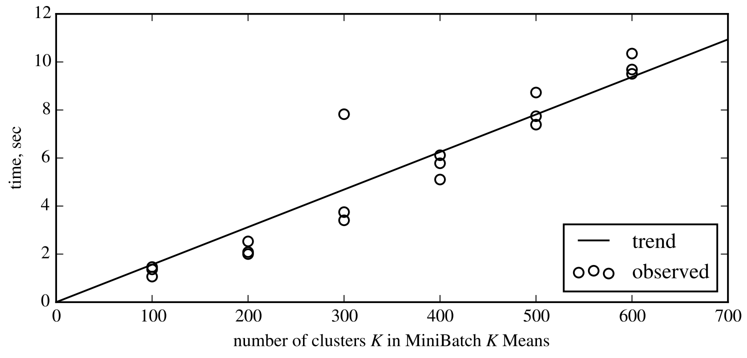
With cosine as the base similarity function for SNN DBSCAN we were able to discover 124 namespace-defining clusters (see fig. 9), which is significantly better than the baseline. The best parameters are 10 nearest neighbors and $\varepsilon = 4$, $\texttt{MinPts} = 3$ (see fig. 9b).

Next, we apply **K-Means**. We observe that increasing K leads to linear increase in time (see fig. 10a), which means that for bigger values of K, it takes longer, so it is not feasible to run: for example, $K = 10\,000$ we estimate the runtime to be about 4.5 hours. As **MiniBatch K-Means** is expected to be significantly faster than usual K-Means, we use it as well. Although we observe that the run time of MiniBatch K-Means also increases linearly with K (see fig. 10b), it indeed runs considerably faster. For example, MiniBatch K-Means takes 15 seconds with $K = 700$ while usual K-Means takes about 15 minutes (see fig. 10a and fig. 10b).

Usual K-Means with small K does not find many good and pure clusters, and MiniBatch K-Means does even worse: no matter what K is selected, the



(a) K in K-Means vs time in minutes ($R^2 = 0.99$).



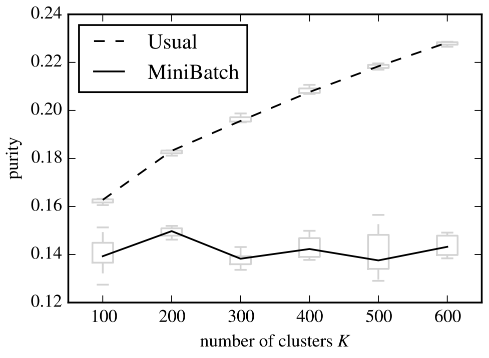
(b) K in MiniBatch K-Means vs time in seconds ($R^2 = 0.97$).

Fig. 10: Runtime of K-Means and MiniBatch K-Means

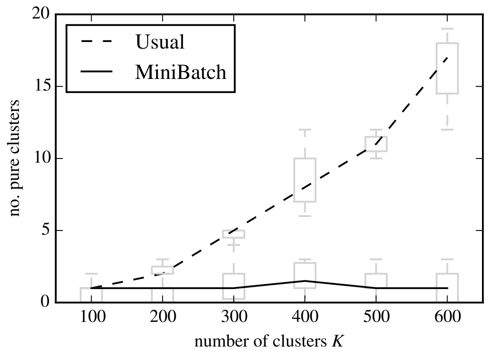
number of pure clusters and purity does not change much (see fig. 11a and fig. 11b). This is also true for larger values of K (see fig. 11c).

The best result was found by usual K-Means with $K = 600$: it was able to discover 19 clusters with purity at least 0.8 (note that this is worse than the baseline of 39 pure clusters). Although it is not many, the found clusters are very tight and quite big. Among others, it discovered the following clusters: “Astronomical catalogues” (53 documents) “Stochastic processes” (39 documents), “Theory of computation” (17 documents), “Thermodynamics” (16 documents), “Statistics” (13 documents). For example, the cluster “Astronomical catalogues” (see table 4) primarily constants documents about stars, and all these documents share the same set of identifers ($p, R_*, m, R_\odot$).

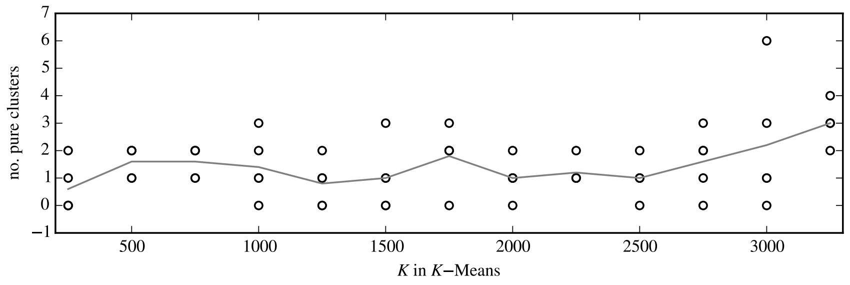
Next, we use **Latent Semantic Analysis** with **SVD** to reduce the dimensionality of the identifier-document matrix D, and then apply K-Means



(a) Purity vs number of clusters K in K-Means and MiniBatch K-Means



(b) Number of pure clusters vs K in K-Means and MiniBatch K-Means



(c) Number of pure clusters vs K in MiniBatch K-Means for larger K

Fig. 11: Effect of K on performance in K-Means

Article	Identifiers
Beta Columbae	$p, R_*, m, R_\odot$
Eta Pegasi	$p, R_*, m, R_\odot$
Sigma Puppis	$p, R_*, m, R_\odot$
Gamma Ceti	$p, R_*, m, R_\odot$
Epsilon Corvi	$p, R_*, m, R_\odot$
Beta Corvi	$p, R_*, m, R_\odot$
Gamma Herculis	$p, R_*, m, R_\odot$
Psi2 Aurigae	$p, R_*, m, R_\odot$
Psi7 Aurigae	$p, R_*, m, R_\odot$
Epsilon Sagittarii	$p, R_*, m, R_\odot$
Alpha Gruis	$p, R_*, m, R_\odot$
Theta Centauri	$p, R_*, m, R_\odot$
...	...

Table 4: Stars cluster discovered by K-Means

on the reduced space. As discussed in the LSA section (see section ref), it should reveal the latent structure of data. We expect that it should improve the results achieved by usual K-Means.

Randomized SVD is very fast, but the runtime does not grow linearly with k, it looks quadratic (see fig. 12). However, the typical values of k for SVD used in latent semantic analysis is 150-250 [17] [43], therefore the run time is not prohibitive, and we do not need to rut it with very large k.

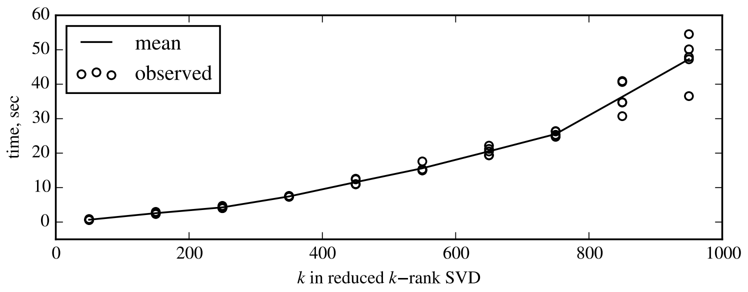
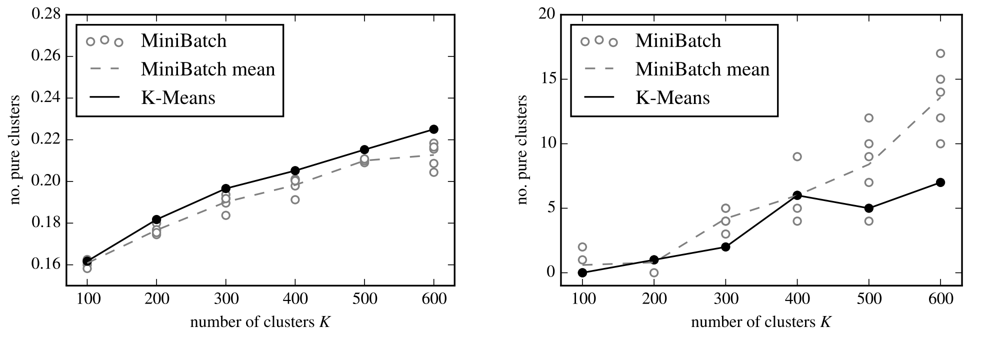


Fig. 12: k in k-rank-reduced randomized SVD vs time in seconds.

When the dimensionality is reduced, the performance of K-Means and MiniBatch K-Means is similar (see fig. 13a), but with MiniBatch K-Means we were able to discover more interesting pure clusters (see fig. 13b). The reason for this may be the fact that in the reduced space there is less noise and both methods find equally good clusters, but because MiniBatch K-Means works faster, we are able to run it multiple times thus increasing its chances to find a good local optimum where there are many pure document clusters. Note that the obtained result is below the baseline.

We can observe that as K increases, the number of interesting clusters increases (see fig. 13b). Therefore, we try a wide range of larger K for different $k \in \{150, 250, 350, 500\}$. The performance in terms of discovered pure clusters does not depend much on the rank k of the reduced space (see fig. 14). In fact, it is very hard to distinguish different lines because they are quite perplexed. The maximum for is achieved at $K \approx 10000$ for all k.

A clustering obtained with $K = 9500$ and $k = 250$ discovered 213 namespace-defining clusters (see table 5), which is significantly better than the baseline. Let us consider a namespace defined by the Measure Theory cluster with 8 documents, where 7 articles in this cluster are from "Measure theory" category, and all the documents share the μ identifier (see table 6).



(a) Purity vs number of clusters K in K-Means and MiniBatch K-Means

(b) Number of pure clusters vs K in K-Means and MiniBatch K-Means

Fig. 13: The performance of K-Means and MiniBatch K-Means on the reduced document space with $k = 150$

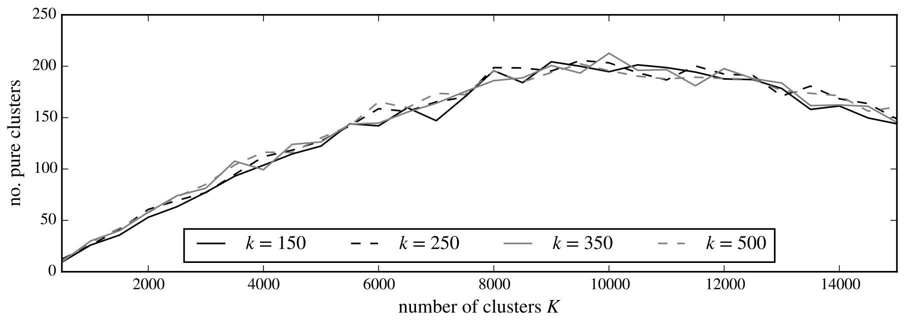


Fig. 14: Number of discovered pure clusters in K-Means for different number of clusters K and rank k.

It is also interesting to look at all the occurrences of identifier μ in the results, and see how it is defined across different namespace-defining cluster (see table 7). We can note several problems. First, sometimes there are namespaces with the same name, but μ stands for different things there, for example, for namespaces “Force” probably either “entropy” or “permeability” is not assigned correctly. Second, there are clearly incorrect definitions such as “assumptions” for the “Measure theory” namespace. Also, there are non-definitions such as “expression a_μb” or “time t” assigned to μ. And finally, some namespaces are questionable, e.g. “Bletchley Park”. But overall it does discover many definitions correctly, for example, “measure”, “mean”, “expectation” and others.

Name	Size	Purity
Astronomical catalogues	53	0.9811
Quantum mechanics	14	0.9286
Electrochemistry	12	0.8333
Particle physics	10	0.8000
Partial differential equations	10	0.8000
Electromagnetism	10	1.0000
Topology	10	0.9000
Measure theory	8	0.8750
Propositional calculus	8	0.8750
Baseball statistics	7	0.8571
Group theory	7	0.8571
Abstract algebra	6	0.8333

Table 5: Top namespace-defining clusters discovered by K-Means with $K = 9500$ on LSA-space with $k = 250$

Article Name	Identifiers
Signed measure	$\Sigma, E, d, f, l, n, P, R, t, x, N, \nu, \mu$
Ergodicity	$\Sigma, E, g, H, n, T, X, \mu, t$
Support (measure theory)	$\Sigma, D, g, f, C, T, L, R, U, t, \lambda, \mu, d$
Convergence of measures	$d, g, f, \mu_n, l, n, P, S, R, \eta, t, \nu, \mu$
Riesz–Markov–Kakutani representation theorem	$E, d, f, \psi, K, \alpha, U, X, x, \mu$
Invariant measure	$\Sigma, B, \phi, f, \mu, S, R, t, x, b, T$
Fernique's theorem	$E, l_*, G, k, l, \alpha, t, x, \mu, d$
Gauge symmetry	$E, d_\nu, J, L, U, W, \nu, \mu$

(a) Identifiers and articles of the "Measure Theory" cluster (some identifiers are removed)

ID	Definition	Score
C	closed set	0.95
f	measurable function	3.50
t	measurable	1.64
Σ	sigma algebra	1.86
α	form	0.87
λ	measure	0.96
μ	measure	11.16
μ_n	measure	0.97
ν	nonnegative measure	0.96
ϕ	monoid	0.91
ψ	positive linear functional	1.89

(b) Definitions in "Measure Theory" (some definitions are removed)

Table 6: A namespace-defining cluster about Measure Theory discovered by K-Means with $K = 9500$ and $k = 250$

When **Non-Negative Matrix Factorization** is used for LSA, the results are similar to the results, obtained by SVD (see fig. 15), however for NMF the results obtained by different k-rank approximations are not as perplexed as for SVD and $k = 250$ does better on $K = [8000; 12000]$ than $k = 150$ and $k = 350$. For example, K-Means with $K = 9500$ and $k = 250$ discovered a clustering with 200 namespace-defining clusters. The found clusters are slightly different. For example, there is no "Measure Theory" cluster of size 8 as in the results, found by SVD (see table 5). Let us instead con-

μ			
Size	Namespace Name	Definition	Score
7	Baseball statistics	dissociation constant	0.89
5	Bletchley Park	motor wheels	4.51
3	Continuous distributions	mean	4.52
5	Fluid dynamics	dynamic viscosity	0.99
3	Force	entropy	0.93
3	Force	permeability	3.63
5	General relativity	vierbein field	0.95,
3	General relativity	reduced mass	1.79
3	Machine learning	hyperparameter	0.86
5	Mathematical analysis	measure	3.69
5	Mathematical finance	brownian motion	0.84
8	Measure theory	measure	11.16
6	Measure theory	assumptions	0.93
3	Mechanics	modulus	0.89
3	Molecular physics	absorption coefficient	0.95
5	Orbits	semi-major axis	0.89
4	Physics	numerator	0.95
4	Probability distributions	ratio	0.96
4	Probability theory	mean	0.95
6	Quantum field theory	gamma matrices	1.73
14	Quantum mechanics	molecular dipole operator	0.96
4	Quantum mechanics	magnetic moment	0.97
3	Quantum mechanics	magnetic dipole moment	0.9
5	Quantum mechanics	chemical potential	0.96
3	Quantum mechanics	formal limit	0.95
4	Set theory	order type	0.97
3	Special functions	partitions	0.99
3	Stars	angular rate	0.96
6	Statistical theory	true mean	1.79
3	Statistical theory	standard deviation	1.90
4	Stochastic processes	expectation	0.96
3	Stochastic processes	time t	0.99
4	Structural engineering	kinetic energy	0.93
6	Theoretical physics	covariant derivative	0.89
4	Theory of relativity	expression a_μb	0.93
3	Theory of relativity	four-velocity	0.87
3	Thermodynamics	chemical potential	1.88

Table 7: Definitions of μ in the namespaces discovered by K-Means with $K = 9500$ and SVD with $k = 250$ (some identifier/definition pairs are removed).

sider a namespace defined by the cluster "Statistics" (see table 9): there are 6 documents in the cluster (see table 9a) and about 43 identifier-definition in the namespace defined by the cluster (see table 9b). Note that in this cluster μ is not assigned correctly: it is defined as "variance" instead of "mean". If

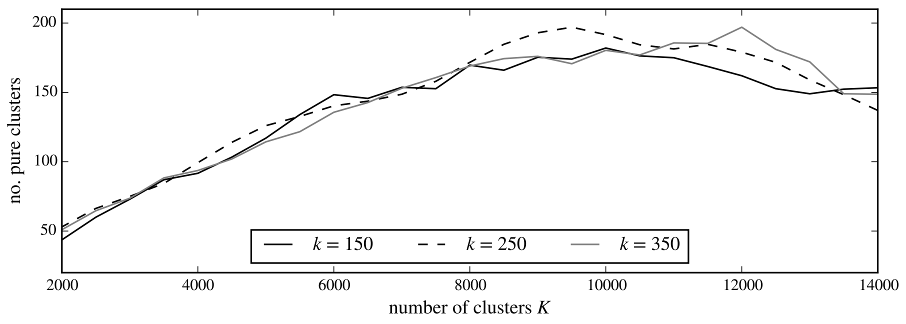


Fig. 15: Number of discovered pure clusters in K-Means and NMF for different number of clusters K and rank k.

Name	Size	Purity
Astronomical catalogues	53	0.9811
Complex analysis	18	0.8333
Quantum mechanics	12	0.8333
Mathematical analysis	10	0.8000
Physics	9	0.8889
Thermodynamics	6	0.8333
Astronomical catalogues of stars	6	1.0000
Knot theory	6	1.0000
Stochastic processes	6	1.0000
Physics	6	0.8333
Stochastic processes	6	1.0000
Statistics	6	0.8333

Table 8: Top namespace-defining clusters discovered by K-Means with $K = 9500$ on NMF-space with $k = 250$

we look at all definitions of μ in this clustering (see table 10) we see that, although the problems are the same as with decomposition by SVD, the found definitions are different. However the difference is likely to be because of non-convexity of K-Means rather than some crucial difference in the way the matrix are factorized.

We see that generally clustering works better on reduced spaces.

In the experiments above we used $\log \text{TF} \times \text{IDF}$. Let us compare the effect of different weighting on the resulting clusters. We will use SVD with $k = 150$ and smaller K because it is computationally faster to compute. We can observe that performance of K-Means does not depend significantly on the weighting system when no definitions are used (see fig. 16).

Article Name	Identifiers
Pivotal quantity	$\zeta, \theta, \nu, \mu, \rho, \sigma, N, X_2, X_1, x, z, ...$
Errors and residuals	$\sigma, \chi, X_n, \epsilon_i, X_i, n, X_1, X, N, \mu, S_n, ...$
Prediction interval	$\epsilon_i, X_i, \alpha, \gamma, \beta, \mu, s_n, \sigma, E, \varPhi, ...$
Cochran's theorem	$\mu, \sigma, B, \chi, R, U, T, Y, X, k, j, Q_2, Q_1, ...$
Skewness	$\gamma, m_3, \kappa, \nu, \mu, \sigma, E, F, s, u, t, ...$
Variance	$\varSigma, \mathrm{SS}_{\mathrm{within}}, \theta, \lambda, \mu, \rho, \sigma, E, f, ...$

(a) Identifiers and articles of the "Measure Theory" cluster (some identifiers are removed)

ID	Definition	Score
E	non-central moment	0.92
F	cumulative distribution function	0.89
f	probability density function	0.99
s	sample variance	3.66
λ	exponential distribution	1.79
μ	variance	3.52
ρ	correlation	0.99
σ	standard deviation	4.61

(b) Definitions in "Measure Theory" (some definitions are removed)

Table 9: A namespace-defining cluster about Measure Theory discovered by K-Means with $K = 9500$ and $k = 250$

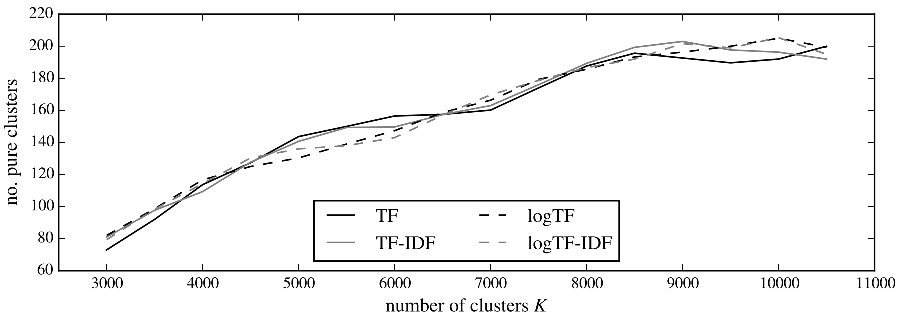


Fig. 16: The effect of using different weighting systems on K-Means with SVD.

4.4.3 Weak Association

The identifier-document matrix has the dimensionality 10419×22512, and there are 485 337 elements in the matrix, so the density is about 0.002.

We do not attempt to use hierarchical methods and start with DBSCAN.

DBSCAN SNN

$k = 10$ dist = jaccard

ε=7 points MinPts=5 points

In general doesn't give clusters

TODO add some numbers and graphs

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

μ			
Size	Namespace Name	Definition	Score
4	Abstract algebra	lebesgue measure	0.83
4	Algebraic curves	first	0.93
3	Condensed matter physics	magnetic permeability	0.99
3	Coordinate systems	form	1.81
3	Differential equations	invariant measure	2.73
3	Doppler effects	spatial resolution	0.95
6	Dynamical systems	tent map	1.71
5	Dynamical systems	entropy	0.93
6	Electromagnetism	permeability	1.90
4	Electromagnetism	gauge condition	0.99
4	Financial markets	drift rate	0.89
3	Force	th-component	0.95
3	Functions and mappings	lebesgue measure	0.91
3	Gravitation	semi-major axis	0.89
3	Group theory	tate module	1.76
3	Magnetism	magnetic moment	1.90
5	Measure theory	measure	8.15
6	Multivariable calculus	measure	1.82
5	Numerical analysis	step size	0.93
6	Physical chemistry	chemical potential	2.57
3	Probability distributions	confidence interval	2.77
3	Probability distributions	laplace	3.42
3	Probability distributions	scale factor	1.84
4	Probability theory	mean	0.95
5	Quantum field theory	field	0.81
12	Quantum mechanics	gamma matrices	1.78
6	Quantum mechanics	magnetic moment	0.87
3	Quantum mechanics	mass	0.89
3	Solid state engineering	charge	0.93
6	Statistical theory	known mean	0.99
6	Statistics	variance	3.52
3	Statistics	mean	1.85
6	Stochastic processes	time t	0.99
6	Stochastic processes	free rate	0.95
3	Stochastic processes	expectation	0.96
5	Systems theory	earths gravity	0.93
3	Telecommunications engineering	yield	0.93
4	Theories of gravitation	indices	1.94
3	Theory of relativity	velocity component	0.93

Table 10: Definitions of μ in the namespaces discovered by K-Means with $K = 9500$ and NMF with $k = 250$ (some identifier/definition pairs are removed).

With **MiniBatch K-Means** applied on the plain untransformed document space we are able to find some interesting clusters, but it general,

Article	Identifiers
Codex Ephraemi Rescriptus	P, papyrus
Categories of New Testament manuscripts	P, papyrus
Papyrus 4	P, papyrus
Uncial 0308	P, M, papyrus, 47
Codex Athous Lavrensis	P, papyrus
Papyrus 92	P, papyrus
Papyrus 90	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
...	...

Table 11: Papyrus cluster by DBSCAN

similarity to the no-definition case, the does not show good results overall. Therefore we apply it to the LSA space reduced by **SVD**, when identifier-document matrix is reduced to rank k. We search for the best combination trying $K = [500; 15000]$ and $k \in \{150, 250, 350\}$. Unlike the case where no definitions are used, the space produced by the soft definition association is affected by k (see fig. 17) and the results produced by $k = 350$ are almost always better. The weighing scheme used for this experiment is $(\log \text{TF}) \times \text{IDF}$.

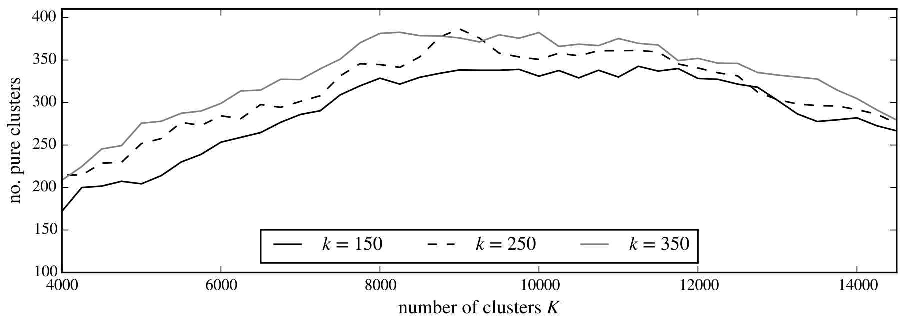


Fig. 17: Number of discovered pure clusters in K-Means and SVD for different number of clusters K and rank k.

Name	Size	Purity
Astronomical catalogues	53	0.9811
Statistics	20	0.8500
Category theory	16	0.8125
Electromagnetism	12	0.8333
Thermodynamics	11	0.8182
Mathematical analysis	11	0.8182
Graph theory	10	0.9000
Graph algorithms	10	0.8000
Fluid dynamics	10	1.0000
Numerical analysis	9	0.8889
Group theory	9	1.0000
Stochastic processes	9	1.0000
Measure theory	8	1.0000

Table 12: Top namespace-defining clusters discovered by K-Means with $K = 9750$ on LSA-space with $k = 350$

Article	Identifiers
Diagonalizable matrix	$v_1, \lambda_1, v_k, \lambda_3, \lambda_2, \lambda_i, \lambda_k, \lambda_j, \lambda_n, ...$
Eigenvalues and eigenvectors	$v_i, \mu_A, \lambda_i, d, \lambda_n, ...$
Principal axis theorem	$v_1, u_1, \lambda_1, \lambda_2, D, S, u, ...$
Eigendecomposition of a matrix	$\lambda, \lambda_1, \lambda, \lambda_2, \lambda_k, R, U, T, ...$
Min-max theorem	$\sigma, u_n, u_k, u_i, u_1, \alpha, \lambda_1, \lambda, \lambda_i, ...$
Linear independence	$\Lambda, v_j, u_2, v_3, u_n, \lambda_1, \lambda_3, \lambda_2, ...$
Symmetric matrix	$\Lambda, \lambda_1, \lambda_2, D, Q, P, \lambda_i, ...$

(a) Wiki Articles in the cluster "Linear Algebra"

ID	Definition	Score
D	diagonal matrix	1.81
t	real argument	0.99
u	eigenvalues	0.89
u_i	eigenvector	0.89
v_1	eigenvectors	1.86
Λ	diagonal matrix	2.65
λ	eigenvalue	0.85
λ_1	eigenvalues	3.66
λ_2	eigenvalues	1.76
λ_3	eigenvalues	0.83
λ_i	eigenvalue	4.40

(b) Definitions in "Linear Algebra"

Table 13: A "Linear Algebra" cluster found by K-Means with $K = 9000$ and $k = 250$.

The best result in this scheme is 414 namespace-defining clusters (ten times better than the baseline), and it is achieved by K-Means with $K = 9750$ and $k = 350$. The purity of this clustering is 0.63. Let us consider a "Linear Algebra" cluster (table 13) with 6 documents and some of extracted definitions in documents of this cluster, and all these articles share identifers λ_1, m and n. Let us consider all definitions of identifier "λ". In total, there are 93 clusters where λ is used (see table 14), and in many cases it is possible to determine that the assignment is correct (e.g. "eigenvalue", "wavelength", "regularization parameter"). Some cases are not correct, for example, when

σ			
Size	Namespace Name	Definition	Score
3	Algebra	multiplicity	0.93
4	Analysis of variance	marquardt	1.71
3	Applied and interdisciplinary physics	wavelength	4.52
6	Cartographic projections	longitude	10.02
3	Cartography	longitude	7.24
3	Category theory	natural isomorphisms	0.84
4	Condensed matter physics	penetration depth	0.95
5	Continuous distributions	affine parameter	0.99
3	Coordinate systems	longitude	2.74
3	Differential equations	differential operator	0.89
8	Differential geometry	vector fields	1.82
7	Electronic amplifiers	typical value	0.93
3	Electrostatics	unit length	0.93
10	Fluid dynamics	wavelength	6.44
6	Fluid dynamics	free path	0.93
3	Infinity	limit ordinals	2.68
7	Linear algebra	eigenvalue	0.85
5	Linear algebra	matrix	0.87
3	Linear algebra	eigenvalue	2.53
3	Liquids	relaxation time	0.88
3	Materials science	rate	0.95
3	Mathematical analysis	eigenvalue	0.86
3	Mathematical theorems	poisson distribution	0.87
4	Measure theory	lebesgue measure	0.95
3	Measurement	order	0.89
8	Mechanics	previous expression	0.95
4	Mechanics	power series	0.88
3	Metalogic	empty word	0.96
7	Number theory	partition	1.90
4	Number theory	modular lambda function	0.99
3	Operator theory	algebraic multiplicity	0.95
5	Optics	wavelength	1.76
5	Partial differential equations	constants	0.87
4	Physical optics	wavelength	3.59
5	Physics	exciton state	2.76
6	Probability distributions	references	0.89
4	Quantum field theory	coupling constant	1.94
5	Quantum mechanics	wavelength	6.47
5	Quantum mechanics	state	2.66
3	Radioactivity	decay	1.82
4	Representation theory of Lie groups	weight	7.11
3	Riemannian geometry	contravariant vector field	0.96
4	Rubber properties	engineering strain	8.96
3	Statistical data types	regularization parameter	0.96
20	Statistics	words	0.93
3	Statistics	expectation	0.99
3	Stellar astronomy	mean free path	0.92
3	Surface chemistry	ideal gas	0.83
3	Theoretical physics	eigenvalue	2.71
5	Theories of gravitation	dicke	0.95
3	Wave mechanics	wavelength	2.65

Table 14: Some of definitions of σ in the clustering obtained with K-Means.

we have clusters with the same name where λ denotes different things (e.g. in two "Quantum Mechanics" clusters), or in the case of "Linear Algebra" cluster where it denotes a matrix.

4.4.4 Strong Association

37879×22512 sparse identifier-document matrix with 499070 entries, so the density of this matrix is just 0.00058.

Best result found with Mini-Batch K-Means is 350 - slightly worse than in the Soft Association.

Add graphs

4.4.5 Russian Wikipedia

Apply the best method for Russian Wiki

4.5 Building Hierarchy

After the namespaces are found, we need to organize them into a hierarchical structure. It is hard to do automatically, and we choose to use existing hierarchies for mathematical knowledge, and then map the found namespaces to these hierarchies.

The first hierarchy that we use is "Mathematics Subject Classification" (MSC) hierarchy [66] by the American Mathematical Society, and it is used for categorizing mathematical articles. In this scheme there are 64 top-level categories such as "Mathematical logic", "Number theory", or "Fourier analysis". It also includes some physics categories such as "Fluid mechanics" or "Quantum Theory". The following top level categories are excluded: "General", "History and biography" and "Mathematics education".

Each top-level category contains second-level categories and third-level categories. In this work we exclude all subcategories those code ends with 99: they are usually "Miscellaneous topics" or "None of the above, but in this section".

Additionally, we excluded the following second level categories because they interfere with PACS, a hierarchy for Physics:

- Quantum theory $\to$ Axiomatics, foundations, philosophy
- Quantum theory $\to$ Applications to specific physical systems
- Quantum theory $\to$ Groups and algebras in quantum theory
- Partial differential equations $\to$ Equations of mathematical physics and other areas of application

The second hierarchy is “Physics and Astronomy Classification Scheme” (PACS) [67], which is a scheme for categorizing articles about Physics. Like in MSC, we remove the top-level category “GENERAL”.

Finally, we also use the ACM Classification Scheme [68] available as a SKOS [69] ontology at their website [70]. The SKOS ontology graph was processed with RDFLib [71]. We use 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 and processing the data, the three hierarchies are merged into one.

However these categories are only good for English articles and a different hierarchy is needed for Russian. One of such hierarchies is “Государственный рубрикатор научно-технической информации” (ГРНТИ) – “State categorizator of scientific and technical information”, which is a state-recommended scheme for categorizing scientific articles published in Russian [72]. The hierarchy is extracted from the official website[7]. It provides a very general categorization and therefore we keep only the following math-related categories: “Астрономия” (“Astronomy”), “Биология” (“Biology”), “Информатика” (“Informatics”), “Математика” (“Mathematics”), “Механика” (“Mechanics”), “Статистика” (“Statistics”), “Физика” (“Physics”), “Химия” (“Chemistry”), “Экономика. Экономические Науки” (“Economics”) and others.

One we a hierarchy is established, each found namespace is mapped to the most suitable second-level category. This is done by keywords matching. First, we extract all key words from the category, which includes top level category name, subcategory name and all third level categories. Then we also extract the category information from the namespace, but we also use the names of the articles that form the namespace. Finally, the keyword matching is done by using the cosine similarity between the cluster and each category. The namespace is assigned to the category with the best (largest) cosine score.

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.

For example, consider a namespace derived from the cluster consisting of “Tautology (logic)”, “List of logic systems”, “Regular modal logic” “Combinational logic” documents. Among others, these articles belong to categories

[7] http://grnti.ru/

“Mathematical logic” and “Logic”. Then the following is the list of keywords extracted from the cluster: “tautology”, “logic”, “list”, “systems”, “regular”, “modal”, “combinational”, “logical”, “expressions”, “formal”, “propositional”, “calculus” and so on. Apparently, this namespace is about mathematical logic.

Then consider a list of keywords for “’General logic”, a subcategory of “Mathematical logic and foundations” from MSC: “mathematical”, “logic”, “foundations”, “general”, “classical”, “propositional”, “type”, “subsystems” and others.

These keywords are represented as vectors in some Vector Space and the cosine score between these vectors is calculated. For this example, the cosine is approximately 0.75, and this is the largest similarity, and therefore this namepsace is mapped to the “General logic” subcategory.

Unfortunately the mapping is not always correct (TODO: add examples)

4.6 Java Language Processing

Previously we have illustrated the idea of identifier namespaces by comparing it with namespaces in Computer Science, and it allowed us to develop an intuition behind the namespaces in mathematics and also propose a method to discover them: we motivated the assumption that there exist “namespace defining” groups of documents by arguing that these groups also exist in programming languages. In this section we will try to do the reserve: apply the methods developed for identifier namespace discovery to the source code.

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 [73]. 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.

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 start with `java` are discarded, at the end the class `WordProcesser` is represented with only one relation ("in", `domain.Word`).

Listing 1: A Java class

```java
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 [74], which is an open-source library for scalable Machine Learning and Data Mining. As on July 24, 2015, this dataset consists of 1 560 java classes with 45 878 variable declarations. After discarding declarations from the standard Java API, primitives and types with generic parameters, only 15 869 declarations were retained.

The following is top-15 variable/type declarations extracted from the Mahout source code:

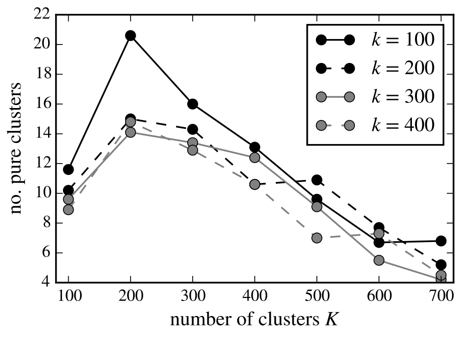
- "conf", `org.apache.hadoop.conf.Configuration` (491 times)
- "v", `org.apache.mahout.math.Vector` (224 times)
- "dataModel", `org.apache.mahout.cf.taste.model.DataModel` (207 times)
- "fs", `org.apache.hadoop.fs.FileSystem` (207 times)
- "log", `org.slf4j.Logger` (171 times)
- "output", `org.apache.hadoop.fs.Path` (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)

We used the "soft" association method to incorporate "definition" (i.e. types), and considering each source code file as a document, we build an identifier-document matrix of dimensionality 1436×1560. Only identifiers that occur at least twice are used to build the matrix.

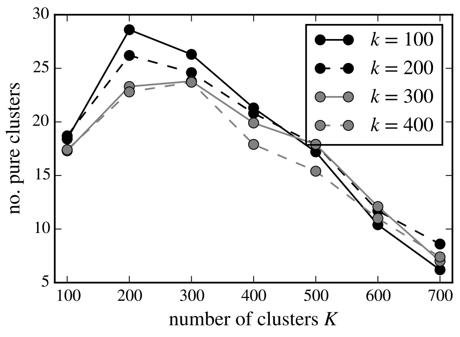
Previously the best performance was achieved by using LSA and Mini-Batch K-Means, and therefore we apply the same algorithms here. However there are a lot of instances of the types `Vector` and `Matrix`, and to mediate its influence, we use two variants of TF-IDF: with usual TF component and with sublinear TF. The best performance is achieved with the rank $k = 200$ of SVD and number of clusters $K = 200$ using sublinear weighting (see fig. 18).

With these parameters the best result is 33 clusters. One of such clusters is a cluster about SVD: there are 5 classes from the `recommender.svd`[8] package (`Factorization`, `FilePersistenceStrategy`, `NoPersistenceStrategy`, `PersistenceStrategy`, `FilePersistenceStrategyTest`) and one from `kddcup.track1.svd`[9] package (`Track1SVDRunner`) Although this cluster is not 100% pure, in the

[8] full name: `org.apache.mahout.cf.taste.impl.recommender.svd`
[9] full name: `org.apache.mahout.cf.taste.example.kddcup.track1.svd`



(a) Usual TF × IDF weighting



(b) Sublinear TF × IDF weighting

Fig. 18: The performance MiniBatch K-Means on the Mahout dataset.

sense that not all of these classes belong to the same package, these classes are clearly related: they are all about SVD. The top dimensions with the most influence in this cluster are `svd.Factorization`[10] and `factorization`.

Note that there is a difference between mathematical namespaces and namespaces discovered in the source code. The found "namespaces" do not necessarily correspond to the real packages in the source code. The reason for this is the document-centric view on the namespaces: for documents we assume that the namespaces are not directly observed, and instead we can see only the documents where these namespaces are used. It is not the case for real namespaces in software: we do observe namespaces directly, and documents (that is, classes) are also the elements of the namespaces.

4.7 Result Analysis and Experiment Conclusions

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

The baseline of random cluster assignment is beaten.

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 $K = 9500$ on the semantic space

[10] full name: `org.apache.mahout.cf.taste.impl.recommender.svd.Factorization`

produced by rank-reduced SVD with $k = 250$ with TF-IDF weight where TF is sublinear.

5 Conclusions

The obtained results indicate that there are namespaces in mathematical notation and it is possible to discover them automatically from a collection of documents. However, there are still ...

Minimize the number of false definitions, and improve the results of clustering. In the next section we discuss how this can be approached.

5.1 Future Work

Clustering ensembles - cluster results are quite different, so need a way of combining several results.

5.1.1 Implementation and Other Algorithms

We observe that cluster algorithms that produce many clusters tend to have good performance. However they also tend to create related clusters from the same category and with same or similar identifiers and definitions. Therefore such results can be refined further and merged. This can be done, for example, by using the join operation from the Scatter/Gather algorithm [33].

The hierarchical agglomerative clustering algorithms may produce good clusters, but their time complexity is prohibitive. For these algorithms we are usually interested in the nearest neighbors of a given data point, and therefore we can use approximation algorithms for computing nearest neighbors such as Locality-Sensitive Hashing (LSH) [75]. The LSH algorithms can be used for text clustering [76], and therefore they should work well for identifiers. Additionally, LSH is also a dimensionality reduction technique, and we have observed that generally reducing dimensionality helps to obtain better clusters.

In this work we use hard assignment clustering algorithms, which means, that a document can import only from one namespace. This assumption does not necessarily always hold true and we may model the fact that documents may import from several namespaces by using Fuzzy Clustering (or Soft Clustering) algorithms [77].

In Latent Semantic Analysis other dimensionality reduction techniques can be used, for example, Local Non-Negative Matrix Factorization [78]. There are also randomized Non-Negative Matrix Factorization that use random projections [79] [80] that potentially can give a speed up while not significantly losing in performance. Another dimensionality reduction technique useful for discovering semantics is Dynamic Auto-Encoders [81].

Topic modeling techniques such as Latent Dirichlet Allocation [82] can be quite useful for modeling namespaces. It can be seen as a "soft clustering" technique and it can naturally model the fact that a document may import from several namespaces.

Finally, we can try different approaches to clustering such as Spectral Clustering [83] or Micro-Clustering [84].

5.1.2 Other Concepts

In this work we assume that document can import only from one namespace, but in reality is should be able to import from several namespaces. As discussed, it can be modeled by Fuzzy Clustering. But it also can be achieved by dividing the document in parts (for example, by paragraphs) and then treating each part as an independent document.

In this work we only use identifiers, extracted definitions and categories. It is possible to take advantage of additional information from Wikipedia articles. For example, extract some keywords from the articles and use them to get a better cluster assignment.

The Wikipedia data set can be seen as a graph, where two articles have an edge if there is an interwiki link between them. Pages that describe certain namespaces may be quite interconnected, and using this idea it is possible to apply link-based clustering methods (such as ones described in [85] and [86]) to find namespace candidates. There are also hybrid approaches that can use both textual representation and links [16].

Vector Space Model is not the only possible model to represent textual information as vectors. There are other ways to embed textual information into vector spaces like word2vec [87] or GloVe [88], and these methods may be useful for representing identifers and definitions as well.

Additionally, tensors may be a better way of representing identifier-definition pairs. For example, we can represent the data set as a 3-dimensional tensor indexed by documents, identifiers and definition. Tensor Factorization methods for revealing semantic information are an active area of research in NLP and linguistics [89], and we can also apply these methods to the namespace discovery problem.

5.1.3 Datasets

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

There are Q&A websites on the stack exchange network with formulae, such as mathematics, mathoverflow, cross validated, data science, theoretical computer science, physics, astronomy, economics and many others that have math, and there are data dumps available from these sites with all the questions and answers.

5.1.4 Unsolved Questions

There are questions that are still not answered and we are not certain how they can be approached.

The biggest question is how to extend this method to situations when no additional information about document category is known. To solve it, we need to replace the notion of purity with some other objective for discovering namespace-defining clusters.

We depend on existing hierarchies, and they are not always complete and there are mismatches. And when this technique is applied to some other language, a different hierarchy is needed for this language. When we applied it to Russian, we needed to find a different hierarchy. There should be a way of building these hierarchies automatically, without the need of external dataset. Potentially it should be possible to use hierarchical clustering algorithms, but it may result in very deep and unnatural hierarchies.

Also, a metric for evaluating the quality of a namespace is needed. Now we assume that pure clusters are namespace-defining clusters. But the namespace candidates should adhere to the namespace definition as much as possible, and therefore a good is needed.

6 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. Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
10. Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.
11. Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.
12. 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.
13. Alfio Gliozzo and Carlo Strapparava. *Semantic domains in computational linguistics*. Springer Science & Business Media, 2009.
14. Dan Jurafsky and James H Martin. *Speech & language processing*. Pearson Education India, 2000.
15. 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.
16. Nora Oikonomakou and Michalis Vazirgiannis. A review of web document clustering approaches. In *Data mining and knowledge discovery handbook*, pages 921–943. Springer, 2005.
17. Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining Text Data*, pages 77–128. Springer, 2012.
18. 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.
19. Ulf Schöneberg and Wolfram Sperber. POS tagging and its applications for mathematics. In *Intelligent Computer Mathematics*, pages 213–223. Springer, 2014.
20. Robert Pagael and Moritz Schubotz. Mathematical language processing project. *arXiv preprint arXiv:1407.0167*, 2014.
21. Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project (3rd revision). 1990.

22. 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.
23. Mihai Grigore, Magdalena Wolska, and Michael Kohlhase. Towards context-based disambiguation of mathematical expressions. In *The Joint Conference of ASCM*, pages 262–271, 2009.
24. 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.
25. 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.
26. Jerzy Trzeciak. *Writing mathematical papers in English: a practical guide*. European Mathematical Society, 1995.
27. Giovanni Yoko Kristianto, Akiko Aizawa, et al. Extracting textual descriptions of mathematical expressions in scientific papers. *D-Lib Magazine*, 20(11):9, 2014.
28. Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.
29. Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
30. 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.
31. Deborah Hughes-Hallett, William G. McCallum, Andrew M. Gleason, et al. *Calculus: Single and Multivariable, 6th Edition*. Wiley, 2013.
32. 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.
33. 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.
34. 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.
35. Rui Xu, Donald Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.
36. 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.
37. David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.
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. Mikhail Korobov. Morphological analyzer and generator for russian and ukrainian languages. *arXiv preprint arXiv:1503.07283*, 2015.
58. Martin F Porter. Snowball: A language for stemming algorithms, 2001.
59. 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.
60. 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.
61. Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. http://www.scipy.org/, accessed: 2015-02-01.
62. 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.
63. A Tropp, N Halko, and PG Martinsson. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical report, 2009.

64. Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.
65. SeatGeek. FuzzyWuzzy 0.6.0. `https://pypi.python.org/pypi/fuzzywuzzy/0.6.0`, accessed: 2015-07-01.
66. American Mathematical Society. AMS mathematics subject classification 2010, 2009. `http://msc2010.org/`, accessed: 2015-06-01.
67. American Physical Society. PACS 2010 regular edition, 2009. `http://www.aip.org/publishing/pacs/pacs-2010-regular-edition/`, accessed: 2015-06-01.
68. Bernard Rous. Major update to ACM's computing classification system. *Commun. ACM*, 55(11):12–12, November 2012.
69. 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.
70. Association for Computing Machinery. ACM computing classification system, 2012. `https://www.acm.org/about/class/2012`, accessed: 2015-06-21.
71. Daniel Krech. RDFLib 4.2.0. `https://rdflib.readthedocs.org/en/latest/`, accessed: 2015-06-01.
72. V. I. Feodosimov. Государственный рубрикатор научно-технической информации (state categorizator of scientific and technical information). 2000.
73. Sreenivasa Viswanadha, Danny van Bruggen, and Nicholas Smith. JavaParser 2.1.0, 2015. `http://javaparser.github.io/javaparser/`, accessed: 2015-06-15.
74. Apache Software Foundation. Apache Mahout 0.10.1. `http://mahout.apache.org/`, accessed: 2015-06-15.
75. Jure Leskovec, Anand Rajaraman, and Jeffrey Ullman. *Mining of massive datasets, 2nd edition*. Cambridge University Press Cambridge, 2014.
76. Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 622–629. Association for Computational Linguistics, 2005.
77. Andrea Baraldi and Palma Blonda. A survey of fuzzy clustering algorithms for pattern recognition. i. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(6):778–785, 1999.
78. 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.
79. Fei Wang and Ping Li. Efficient nonnegative matrix factorization with random projections. In *SDM*, pages 281–292. SIAM, 2010.
80. Anil Damle and Yuekai Sun. Random projections for non-negative matrix factorization. *arXiv preprint arXiv:1405.4275*, 2014.
81. Piotr Mirowski, M Ranzato, and Yann LeCun. Dynamic auto-encoders for semantic indexing. In *Proceedings of the NIPS 2010 Workshop on Deep Learning*, 2010.
82. David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.
83. 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.
84. Takeaki Uno, Hiroki Maegawa, Takanobu Nakahara, Yukinobu Hamuro, Ryo Yoshinaka, and Makoto Tatsuta. Micro-clustering: Finding small clusters in large diversity. *arXiv preprint arXiv:1507.03067*, 2015.
85. Rodrigo A. Botafogo and Ben Shneiderman. Identifying aggregates in hypertext structures. In *Proceedings of the Third Annual ACM Conference on Hypertext*, HYPERTEXT '91, pages 63–74, New York, NY, USA, 1991. ACM.

86. Andrew Johnson and Farshad Fotouhi. Adaptive clustering of hypermedia documents. *Information Systems*, 21(6):459–473, 1996.
87. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013.
88. 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.
89. Anatoly Anisimov, Oleksandr Marchenko, Volodymyr Taranukha, and Taras Vozniuk. Semantic and syntactic model of natural language based on tensor factorization. In *Natural Language Processing and Information Systems*, pages 51–54. Springer, 2014.