Tampere University

Kalle Aaltonen

TOPIC EVOLUTION IN SCIENTIFIC PUBLICATIONS OVER TIME

A data pipeline

Faculty of Information Technology and Communication Sciences
Master's thesis
October 2020

ABSTRACT

Kalle Aaltonen: Topic evolution in scientific publications over time
Master's thesis
Tampere University
Computer Science
October 2020

This study aims to identify an optimal data pipeline for modelling topic evolution over time in scientific publications of the Tampere universities.

To define a pipeline we divided it into stages of data acquisition, preprocessing, persisting and topic modelling. We then compared alternative methods of executing the stages. The final pipeline was composed of the best performing methods. As the data set we used the English-language abstracts from the Master's theses. The data source was the Trepo repository for scientific papers of the Tampere universities.

Our results show the Dynamic Non-negative Matrix Factorization (DNMF) algorithm being significantly faster to train and more versatile an implementation than the Dynamic Topic Models (DTM) algorithm. The algorithms produce very similar latent topics, where technical fields of study are dominantly present. This seems to reflect the distribution of fields of study in our corpus. The evolution of individual terms inside topics follow the real world trends and technological advancements to some extent.

The results for the persisting layer comparison reveal PostgreSQL to be better performing than MongoDB on aggregate queries. Surprisingly this was also true for the queries targeted at the data that is stored as JSON data type inside Postgres. The fact that MongoDB is a dedicated document store and PostgreSQL is primarily a relational database management system makes this finding particularly interesting. Data acquisition results show that the most efficient way to ingest data from Trepo is through the provided OAI-PMH service. Our research does identify any reason to utilize web scraping over it.

The thesis proposes a pipeline mainly from the efficiency perspective. The time-inefficiency of training the topic models needs to be taken into account when implementing a system based on the proposed data pipeline. Additionally the study highlights the possibility of using PostgreSQL as a dedicated document store.

Keywords: topic model, topic evolution, NLP, machine learning, data, data pipeline, natural language processing, database, NMF, DTM, LDA

TIIVISTELMÄ

Kalle Aaltonen: Tieteellisten julkaisujen aiheiden kehitys ajan kuluessa
Pro gradu
Tampereen yliopisto
Tietojenkäsittelyoppi
Lokakuu 2020

Tutkielma määrittelee tietoputken Tampereen yliopistojen julkaisuista löytyvien piilevien aiheiden kehityksen mallintamiseksi.

Tietoputken määrittelyä varten se jaettiin osiin, joita ovat tiedonhankinta, tiedon esikäsittely, tiedon varastointi ja aihemallinnus. Tämän jälkeen osille valittiin vaihtoehtoisia tapoja suorittaa toiminto ja verrattiin niitä keskenään. Lopullinen tietoputki muodostui yhdistämällä parhaiten toimivat suoritustavat. Tietokokoelmana käytettiin Pro gradu -tutkimusten ja diplomitöiden englanninkielisiä tiivistelmiä. Tietolähteenä toimi Tampereen yliopiston avoin julkaisuarkisto, Trepo.

Tutkielman tulokset kertovat, että ajan yli tapahtuvassa mallinnuksessa epänegatiivisen matriisin faktorointiin perustuvan DNMF-algoritmin (engl. Dynamic Non-negative Matrix Factorization) opettaminen on huomattavasti nopeampaa kuin LDA-menetelmään perustuvan DTM-algoritmin (engl. Dynamic Topic Models). Tämän lisäksi ensin mainitun toteutus on monipuolisempi sisältäen esimerkiksi aihemääräsuosittelun. Algoritmit tuottavat hyvin samankaltaisia piileviä aiheita, joissa tekniset tutkimusalueet korostuvat. Tämä vaikuttaa myötäilevän jakaumaa eri tutkimusalueista peräisin olevien töiden suhteen tietokokoelmassa. Yksittäisten termien kehityksessä aiheen sisällä on havaittavissa jonkin verran korrelaatiota ulkomaailman tapahtumien ja teknologisen kehittymisen kanssa.

Tietokantahallintajärjestelmien vertailun tulokset paljastavat, että PostgreSQL suoriutuu yhdistämiskyselyistä paremmin kuin MongoDB. Yllättävästi näin on myös silloin, kun kyselyn kohteena oleva data on Postgresissa JSON-tietotyyppinä. MongoDB:n ollessa nimenomaan JSON-esitysmuodolle tarkoitettu dokumenttitietokanta ja PostgreSQL:n ollessa relaatiotietokannan hallintajärjestelmä, tämä löydös on erityisen kiinnostava. Tulostemme mukaan tiedon hankinta Treposta tapahtuu tehokkaimmin OAI-PMH -palvelun kautta. Tutkimuksessa ei paljastu syitä käyttää verkkosivun haravointia kyseisen palvelun sijaan.

Tutkielma esittää tietoputkea pääasiassa tehokkuuden näkökulmasta. Esitettyyn tietoputkeen pohjautuvaa järjestelmää rakennettaessa aihemallien opettamisen hitaus on otettava huomioon. Lisäksi tutkielma korostaa mahdollisuutta käyttää PostgreSQL-tietokantaa dokumenttitietokantana.

Avainsanat: aihemalli, aihekehitys, NLP, koneoppiminen, data, dataputki, tietoputki, luonnollisen kielen käsittely, tietokanta, NMF, DTM, LDA

CONTENTS

1 Introduction . . . 1
2 Theoretical background . . . 3
2.1 Collecting and storing data . . . 3
2.1.1 Data structure . . . 4
2.1.2 Database systems . . . 5
2.2 Knowledge discovery . . . 8
2.2.1 Natural Language Processing . . . 8
2.2.2 Topic modelling . . . 15
3 Research methodology and material . . . 21
3.1 Trepo repository - the data source . . . 21
3.2 Research environment . . . 24
3.3 Database comparison . . . 26
3.3.1 Setup . . . 27
3.3.2 Task definition . . . 28
3.4 Data acquisition and preprocessing . . . 30
3.4.1 Collecting metadata . . . 30
3.4.2 Acquiring full text content . . . 31
3.4.3 Applying NLP . . . 32
3.5 Topic modelling in practice . . . 35
3.5.1 Initial LDA and NMF modelling . . . 36
3.5.2 Modelling over time with LDA . . . 38
3.5.3 Modelling over time with NMF . . . 41
4 Results and analysis . . . 47
4.1 Comparison between PostgreSQL and MongoDB . . . 47
4.2 Topic modelling over time . . . 49
4.2.1 Topic modelling with LDA and NMF from sklearn . . . 49
4.2.2 Dynamic Topic Models . . . 52
4.2.3 Dynamic NMF . . . 56
4.2.4 Algorithm comparison . . . 64
4.3 Proposition for a pipeline . . . 69
5 Conclusions . . . 73
References . . . 77
Appendix A Database systems . . . 85
Appendix B Database comparison . . . 86
Appendix C Topic modelling . . . 91

LIST OF FIGURES

2.1 A hierarchical view of the same data as on Table 2.1 [7]. 7
2.2 The derivation of NLP 9
2.3 A parsed arithmetic statement. 10
2.4 A parsed natural language sentence presented as an arborescence [50]. . 11
2.5 A diagram of machine learning subsets [59]. 12
2.6 An illustration of an artificial neural network (below), and a detailed look into one of the neurons (above). 13
2.7 A Euclidean plane representing a Dirichlet distribution of $k = 3$ with the points θ_i defining the plane and satisfying Definition 2.8 [75]. 16

3.1 A bar chart showing the count of publications in Trepo by the year issued. The year is on x-axis and the count on y-axis. 23
3.2 A chart showing the total number of publications, the number we managed to download as PDF and the number we were able to extract text content from. 32
3.3 Tables used for storing the metadata (documents) and the refined data from abstracts (linguistic_features). 34
3.4 A bar chart visualizing the distribution of those Master's thesis level publications that have an English abstract present in the Trepo metadata [100] . 39
3.5 Jupyter notebook demonstrating the preprocessing for the gensim DTM wrapper. 41
3.6 Jupyter notebook showing the use of the gensim DTM wrapper. 42
3.7 Jupyter notebook showing some post-processing of gensim the DTM wrapper outputs. 43
3.8 Jupyter notebook showing the preprocessing for the DNMF algorithm. . . . 45
3.9 Jupyter notebook showing the post-processing step for the DNMF output. . 46

4.1 Results of the comparison between database systems. 48
4.2 DTM Topic 0 evolution. 54
4.3 DTM Topic 6 evolution. 55
4.4 DTM Topic 7 evolution. 55
4.5 DTM Topic 9 evolution. 56
4.6 DTM Topic 13 evolution. 56
4.7 DTM Topic 15 evolution. 57
4.8 DTM Topic 15 variance. 59
4.9 DNMF Topic 1 evolution. The words are plotted only for the years in which they appear in the topic. 60

4.10 DNMF Topic 2 evolution. The words are plotted only for the years in which they appear in the topic. 60
4.11 DNMF Topic 3 evolution. The words are plotted only for the years in which they appear in the topic. 61
4.12 DNMF Topic 8 evolution. The words are plotted only for the years in which they appear in the topic. 62
4.13 DNMF Topic 14 evolution. The words are plotted only for the years in which they appear in the topic. 63
4.14 Comparison 1 between evolution of similar words. Above DTM Topic 17 and below DNMF Topic 10. For DNMF the words are plotted only for the years in which they appear in the topic. 67
4.15 Comparison 2 between evolution of similar words. Above DTM Topic 7 and below DNMF Topic 2. For DNMF the words are plotted only for the years in which they appear in the topic. 68
4.16 The proposed data pipeline. 72

B.1 The model of application containerization adapted from Docker [26, 32]. . . 87
B.2 The schema used for mongoDB in testing with Suomi24 data. 88
B.3 The only table in the schema used for PostgreSQL in testing with Suomi24 data. 88

C.1 A jupyter notebook on topic modelling with sklearn LDA. 91
C.2 A jupyter notebook on topic modelling with sklearn NMF. 92

LIST OF TABLES

2.1 A relational view of data related to each other with redundant data. [7]. . . . 6
2.2 Categorized NLP applications [46] 9
2.3 A parsed natural language sentence. 11
2.4 A collection of text documents, vocabulary and a corresponding term frequency matrix. 14

3.1 The counts of publications by type, and the counts by the written language inside those types. 22
3.2 An example of the metadata available for a publication in the Trepo repository 25
3.3 The specs of the computing environment. 26
3.4 Some 3rd party Python libraries used in the research. 26
3.5 Some of the parameters for LDA and NMF models in scikit-learn. 38
3.6 A table demonstrating the bag-of-words representation of a corpus the gensim wrapper for DTM ingests. The representation is a list of lists having one row for each document. 40

4.1 20 topics inferred by the sklearn Latent Dirichlet Allocation model 49
4.2 20 topics inferred by the sklearn Non-negative Matrix Factorization model . 51
4.3 Terms describing the DTM model of 20 topics. Terms are chosen by using the cumulative probability from all time slices by topic. 52
4.4 Terms describing the DNMF model of 20 topics. Terms are chosen from the overall rank provided by the algorithm. 57
4.5 Combination of window topics that form a dynamic topic in a time window representing the year 2015. 62
4.6 Top 3 suggestions for topic number by year given by the automatic recommending feature. The recommendation for the number of dynamic topics is at the end of the table. 64
4.7 Time consumption of training the DTM and DNMF models. Times for three separate runs and their average. 65

LIST OF SYMBOLS AND ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
arborescence	A directed rooted tree. A special case of a directed acyclic graph, where there is exactly one node from which all other nodes are reachable, each via exactly one directed path
Bag-of-Words	Bag of Words model has the text document presented as a multiset of words (containing the multiplicity information). The words can e.g. be in a lemmatized form
big data	Big data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time
BLOBS	Binary Large Objects
corpus	This is a collection of text documents
CRF	Conditional Random Fields
data	Data (singular datum) are individual units of information. A datum describes a single quality or quantity of some object or phenomenon. In analytical processes, data are represented by variables
DBMS	A piece of software known as a database management system that translates between the user's request for data and the physical data storage. [47]
deep learning	Is a branch of Artificial Intelligence attempting to mimic neurons in the (human) neocortex. The word deep comes from having many layers of nonlinear feature transformation [13]
DNMF	Dynamic Non-negative Matrix Factorization
DTM	Dynamic Topic Models
fork	A fork is an independent project from a copied source code. This is usual in open source development

Frobenius norm	A matrix norm of an $m \times n$ matrix A defined as the square root of the sum of the absolute squares of its elements $$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} \|a_{ij}\|^2 \right)^{1/2}$$ [109]
GSL	The GNU Scientific Library
HDFS	Hadoop Distributed File System
HMM	Hidden Markov Model
horizontal scaling	Scaling horizontally (out/in) means adding more nodes to (or removing nodes from) a system, such as adding a new computer to a distributed software application
intellectual property	A category of property that includes intangible creations of the human intellect
JSON	JavaScript Object Notation is an open-standard file format or data interchange format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value)
knowledge discovery	Knowledge discovery from data (KDD) is a process of extracting meaningful knowledge from data [45]
Kullback–Leibler divergence	This is a measure of how one probability distribution is different from a second, reference probability distribution
LDA	Latent Dirichlet Allocation
lemma	The base form of a word
machine learning	This is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead
metadata	This means data about data. Data that describes other data.
METS	Metadata Encoding and Transmission Standard
natural language	Language that has developed in the usual way as a method of communicating between people, rather than language that has been created, for example for computers [22]
NER	Named Entity Recognition

NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NMF	Non-negative Matrix Factorization
NoSQL	A NoSQL (originally referring to ”non SQL” or ”non relational”) database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases
NTFS	New Technology File System
OAI-DC	Open Access Initiative Dublin Core
OAI-PMH	The Open Archives Initiative Protocol for Metadata Harvesting
OCR	Optical Character Recognition
open source	Products that are released under an open source licence. Everyone has permission to use the product and alter its source code
PDF	Portable Document Format
pdf2image	A python module wrapping pdftoppm and pdftocairo to convert PDF to a PIL Image object.
PIL	Python Imaging Library
POS	Part of Speech
Python-tesseract	Python-tesseract is a Python wrapper for Google’s OCR engine.
raster image	Aka bitmap image, a dot matrix data structure representing a grid of pixels (points of color).
RDBMS	Relational Database Management System
regularization	This is the process of adding information in order to solve an ill-posed problem or to prevent overfitting
relational database	the place where data are stored, which contains not only the data but also information about the relationships between those data [47]
REST	Representational State Transfer
semi-structured data	Semi-structured data has a structure but does not conform to the formal definition of structured data, that is, tables with rows and columns. Examples of semi-structured include tab and delimited text files, XML, other markup languages such as HTML and XSL and JavaScript Object Notation (JSON) [30]

SMiLE	Statistical Machine Learning and Exploratory Data Analysis
SOAP	Simple Object Access Protocol
storage device	In a computing environment, devices designed for storing data are termed storage devices or simply storage [93]
structured data	This is data that has a predefined data model and fits well into the tables of relational databases with rows and columns
TC-W2V	Topic Coherence via Word2Vec
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
time slice	Part of the whole time line with a beginning and an end. A synonym to time window.
topic modelling	”Text mining by topic modelling aims to discover topics that occur in a collection of documents in order to explore hidden semantic structures in the body of the texts” [102]
treebank	A treebank is a parsed text document collection, annotating sentence structure.
Trepo	This is the open institutional repository of Tampere University at trepo.tuni.fi. It includes open access publications of Tampere University.
TUNI	Tampere Universities
TUT	Tampere University of Technology
unstructured data	The term refers to data that does not have a predefined data model and/or does not fit well into traditional relational database tables. Typically has no identifiable structure and may include bitmap images, text, audio, video, and other data types [52]
URL	Uniform Resource Locator
UTA	University of Tampere
vector graphic	This means computer graphics images defined in terms of points, connected by curves and lines and forming polygons and other shapes.
vocabulary	The set of words of a corpus
web crawler	A program that traverses web pages

web scraping	The practice of gathering data from web pages through any means other than a program interacting with an API [73]
word2vec	Word2vec models are shallow neural networks used to produce word embeddings. The words that share common context in the document collection are situated close to each other in the word vector space
WWW	World Wide Web
XML	Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

1 INTRODUCTION

There are hundreds of universities in Europe, some of which are very old such as the universities of Cambridge and Paris, dating back to the 13th century [48]. Each university produces multiple kinds of scientific papers, and as the years go by the research interests evolve and publications pile up. Would it not be intriguing to know how the themes in research have evolved?

Topic modelling is a way to capture meaning from a collection of documents. A topic model captures the concepts a text corpus consists of. The documents can then be organized into topics or categories using these concepts [40]. Latent Dirichlet Allocation (LDA) is a popular topic modelling algorithm we will utilize in our research. Along with it we use another approach based on Non-negative Matrix Factorization (NMF). Both of these techniques produce static topics, but we also apply dynamic versions of them both, to achieve topics that evolve over time.

By looking at a title of a book, article, report or thesis you get an understanding of the theme it is about. By reading the abstract you get deeper understanding. But somewhere inside the scientific texts there are themes harder to get a grasp of, themes that may come as a surprise to you. By capturing those themes, and grouping documents according to them, you may find connections between publications you had never believed being connected. Those latent topics can give insight into the political or social environment of the society at some particular time period. Or they can let you have a peek at the hot technical advances during some geopolitical conflict that shaped history.

In 1962 Thomas Kuhn talks about paradigms [60], which he explained to be scientific achievements recognized universally having two things in common. The two characteristics defining a paradigm are, for one, it being unprecedented enough to attract people away from competing scientific modes permanently. The second characteristic defining a paradigm is it being open-ended enough to leave problems for its practitioners to resolve. Kuhn introduced a model of scientific change, where science shifts from a paradigm to another. When scientific imagination is transformed enough you eventually have to describe it as ”a transformation of the world within which scientific work was done”, as Kuhn stated. These scientific revolutions come with new framing and new vocabularies. Topic modelling over time is also about capturing the change with the help of the vocabulary utilized and the terms emphasized. Maybe the scope in which it captures the change of topics is smaller than Kuhn’s revolutions.

Some work in the field of text classifying and clustering include supervised algorithms for classification in the 1990's [53, 67], probabilistic latent semantic indexing by Thomas Hofmann in 1999 [49] and a text classifier algorithm using both labeled and unlabeled documents for learning by a Nigam et al. in 2000 [80]. In 2003 Blei et al. introduced their famous generative model, the Latent Dirichlet Allocation (LDA) [18], which uses unsupervised method to find the topics. In 2006 Blei et al. introduce an LDA model that tracks topics over time [16], calling it dynamic topic models, and show it performs better than two static models at predicting the topics of an unseen year in a time series of articles. In 2012 an algorithm generalizing to topic models with correlations among topics, based upon Non-negative Matrix Factorization (NMF), was introduced [5]. In 2017 Greene et al. introduced a dynamic topic model based on NMF.

The data source in our research is the *Trepo* repository. It consists of thousands of scientific publications from Tampere Universities (TUNI) ever since the 1970's. Our research focus is on *discovering an optimal pipeline leading from the original data in Trepo to modelling the latent topics of the publications over time.* The research goal is to find answers to the following questions: *What is the optimal way to extract data from Trepo? What is the best approach for storing the raw and the preprocessed data for the purpose of further consuming it in topic modelling? What is the most suitable algorithmic approach to topic modelling our data set over time?*

The data pipeline we seek to define could then be fully automated and scheduled to work as a backbone for a Business Intelligence solution or for a micro service offering a *REST API* for a web app.

A data pipeline involves dealing with many dimensions of computer science and thus require at least basic understanding of several concepts. The majority of those concepts are covered in Section 2, *Theoretical background*. The setup for research and the qualities and features of the data, the steps the pipeline consists of and key research questions are discussed in Section 3, *Research methodology and material*. Then in Section 4, *Results and analysis*, we talk about the outcomes of the practical experiments, and our findings are introduced in detail. In Section 5, *Conclusions*, we sum up the research.

2 THEORETICAL BACKGROUND

Creating a pipeline for an apparatus of topic modeling over time involves many stages. In this chapter we discuss some relevant concepts and background for making informed choices in those stages. First we talk about collecting data in Section 2.1 *Collecting and storing data* and about different data structures, which relate to enriching and refining data, in Subsection 2.1.1 *Data structure*. Next we introduce some data storage devices in Subsection 2.1.2 *Database systems*. Lastly we discuss the knowledge discovery pipeline in Section 2.2 *Knowledge discovery*. This all is done from the perspective of introducing a data pipeline for topic modelling purposes later on.

2.1 Collecting and storing data

Any *data* related work, study or research has some tasks in common. One most likely obligatory task is the data collection, which is often done by using an electrical data source such as a web service. Having the possibility to collect data through something like Representational State Transfer (REST) Application Programming Interface (API) or Simple Object Access Protocol (SOAP) API provided by the service consumed is nowadays very common. Both of the mentioned web communication protocols provide a standardized means to request and retrieve data from a service, making them reliable and relatively stable and thus enable building applications that consume the services an easy task. An example of such a service is the Twitter developer API which provides many endpoints for retrieval of tweets and related metadata [101]. The data formats that these APIs use to respond to the requests include *JSON* and *XML*. Twitter API for instance responses with JSON formatted data. One method for collecting data is by utilizing *the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)* framework. OAI-PMH is a model consisting of data providers and service providers for offering repository metadata in XML format over HTTP [62].

Sometimes, for some reason, the service consumed does not provide any above mentioned API. Reasons for a service not to provide an API can be e.g. lack of infrastructure or technical ability to create an API and the data being considered valuable and not intended to spread widely [73]. If the service does not provide an API, the data can be acquired by scraping. In this scenario one must be especially careful to respect the *intellectual properties* and copyright laws. One clear definition of web scraping is ”the practice of gathering data through any means other than a program interacting with an API (or, obviously, through a human using a web browser)" [73]. As the definition suggests web scraping is a big field for study on its own.

2.1.1 Data structure

The best options for effectively storing data depend, among other things, on the structure the data holds. To create business value from unstructured data most likely requires transforming it to semi-structured format first. Let's now discuss the concepts of structured, semi-structured and unstructured data.

Structured data

Structured data is considered well refined and not easy to come by in most of the real life data sources one stumbles upon. Structured data goes well with *relational databases* such as MariaDB or Microsoft Azure SQL Database because of the predefined model it follows. These SQL type data sources consisting of tables of columns and rows, and holding structured data, are for example the easiest ones for business intelligence tools (Tableau, Power BI) to further consume [30].

Semi-structured data

Semi-structured data on the other hand complicates or opens up the choice of the data storage device. Earlier we talked about REST APIs and concluded that they can respond with data in JSON or XML formats. Both of these can be considered semi-structured [30]. For this type of data having the schema contained within the data and being possible to produce a representation of the data as some kind of graph-like or tree-like structure [21] NoSQL usually work quite well as a data store. Many times one can dump the semi-structured data to a NoSQL database as it is, especially in the case the data follows the JSON specification. To use a relational database with semi-structured data one might need to apply at least some preprocessing before inserting it. This is because a Relational Database Management System (RDBMS) strictly follows the predefined data format but the semi-structured data does not necessarily do so.

Unstructured data

Unstructured data data can be defined as something with no metadata available at all [85] or as having no identifiable structure and possibly including bitmap images, text, audio, video, and other data types [52]. This type of data are best and cheapest stored as Binary Large Objects (BLOBS) or as a file in a file system like New Technology File System (NTFS) [37] and are the most difficult types of data for business intelligence tools to consume [30]. The World Wide Web (WWW) is considered one example of unstructured data [21]. A popular term *big data*, with diverse and contradicting definitions [106], can for one be considered describing the huge amounts of unstructured data produced by any sources, e.g. the WWW. Combining the need for cheap file storage and the need for *horizontal scaling* to store the piling ”big data” makes businesses eye beyond the conventional databases. Here comes to play the distributed file storage e.g. Hadoop Distributed File System (HDFS) and cloud storage such as Azure storage offerings. Also SQL and NoSQL databases can be distributed across a cluster of servers.

Two things are worth highlighting here:

1. The distinction between unstructured and semi-structured data is a line drawn in water. By adding one piece of metadata to a set of unstructured data one can argue it now is semi-structured. The distinction between semi-structured (or unstructured) and structural data on the other hand can be defined unambiguously
2. Data can indeed be transformed from unstructured to semi-structured and even from semi-structured to structural. This can be thought as refining the data. It is possible to refine complitely unstructured data without having any metadata attached by e.g. applying *machine learning* to it.

2.1.2 Database systems

”Data created by individuals or businesses must be stored so that it is easily accessible for further processing” [93].

There exists a vast amount of different data *storage devices*: from USB flash drives and hard drives in PCs with terabytes of storage space, to cloud environments on virtualized data centers, these are nowadays available to businesses as well as to individual people. In this subsection we introduce briefly a couple of selected means to effectively store data on a server or a cluster of servers.

In general, a database of some kind has a good probability of topping the list for the best option to store your business data, and to consume it from. Which database to use depends at least on the structure, size and use case of your data set. One definition of a database states that ”to be considered a database, the place where data are stored must contain not only the data but also information about the relationships between those data” [47]. Any data store or a software package which is not able to ”represent relationships between data, much less use such relationships to retrieve data” should be called a *data store* [47]. Next let’s go through a few database systems we are going to need later on. We will also have a small example of transaction data to give insight into how the different database systems handle equivalent data. The equivalent data means here data carrying exactly the same information, but having slightly different representation.

Relational database

A relational database must be the most conventional type of a database. First mention of the relational model was in the paper ”A Relational Model of Data for Large Shared Data Banks” from 1970 by Dr. E.F.Codd at IBM [28]. In a relational database data are presented as sets of tables consisting of columns and rows. Redundant data is used to link records between the tables [7]. In Table 2.1, fairly simple tables are linked to each other by columns of the same name. Each row on each table can be uniquely identified by the primary key of the table. In this example the primary key of each table is the first column on that table.

Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	$0

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500	2004-03-27

Table 2.1. *A relational view of data related to each other with redundant data. [7].*

Data are queried from the tables with a non-procedural programming language called SQL. Usually for clean structured data a relational database is a very good option for a data store / source to use. When the data grows to a big data size the limits of horizontal scaling capabilities of relational databases complicate things. Today there are solutions for this problem. For instance PostgreSQL can be distributed by using an extension called *Citus*, which provides scaling capabilities over a cluster of servers [27, 55]. Postgres is *open source* and there exists *forks* from it to enable its use in cloud environment. Even inserting dirty semi-structured data to a relational database is not a big problem with newer versions of the popular RDBMS such as MariaDB or Postgres, since they have column types for JSON and also database functions to make queries to data inside these columns [71, 86].

NoSQL database

NoSQL database is an umbrella term used in this study for a database other than a relational one. A NoSQL database is a DBMS for data stored otherwise than strictly as described previously when discussing the RDBMS (relational databases). The NoSQL includes a variety of approaches to modeling data. These approaches include key-value APIs, document model, column-family, columnar model, graph and polygot databases and also cloud based databases that are difficult to classify [84, 110]. Often the NoSQL databases are more flexible than relational ones, with unrestricting, dynamic schemas and good horizontal scaling capabilities. Relational databases provide Atomicity, Consistency, Isolation and Durability (ACID) properties but most of the NoSQL databases fall short doing this [110]. The account data mentioned earlier and shown in Table 2.1 can be presented in a 3 level hierarchy of Customers, Accounts and Transactions as show in Figure 2.1.

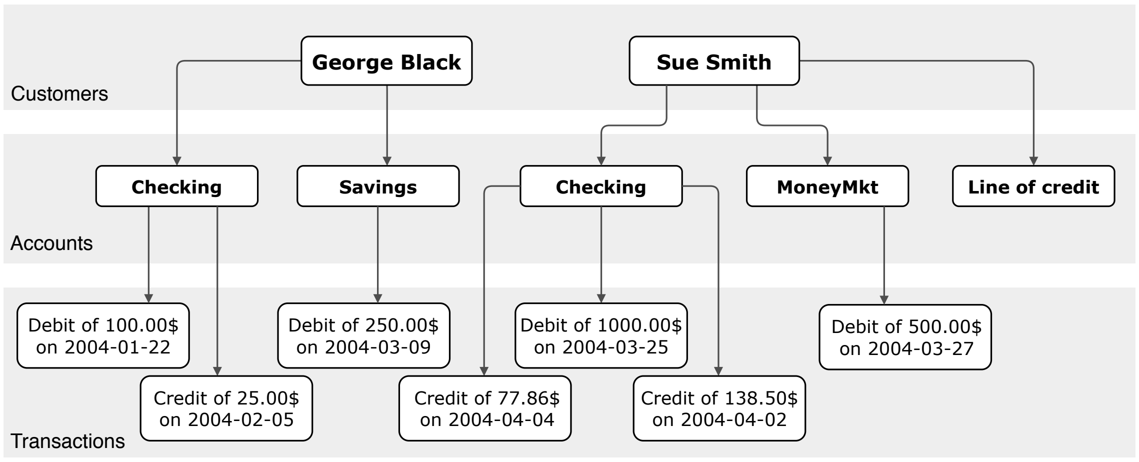

Customers
George Black
Sue Smith
Accounts
Checking
Savings
Checking
MoneyMkt
Line of credit
Transactions
Debit of 100.00$ on 2004-01-22
Credit of 25.00$ on 2004-02-05
Debit of 250.00$ on 2004-03-09
Credit of 77.86$ on 2004-04-04
Debit of 1000.00$ on 2004-03-25
Credit of 138.50$ on 2004-04-02
Debit of 500.00$ on 2004-03-27

Figure 2.1. *A hierarchical view of the same data as on Table 2.1 [7].*

This can be interpreted as follows:

- 1st level is a list of customers
- 2nd level is a list of accounts nested in each individual customer
- 3rd level is a list of transactions found withing each account

In a database using the document model this structure can be presented in practice in JSON format or in XML format. This sort document format in JSON notation is used in MongoDB [74]. An example with the same transaction data we have already seen in Table 2.1 and in Figure 2.1 presented now in JSON format is visible in Listing 4. The listing is found in Appendix A.

On database popularity

When looking at the popularity of all Database Management Systems in use, the top 3 of February 2020 is clearly distinguishable from others. According to [29] Oracle, MySQL and Microsoft SQL Server are the most popular based on a method of calculating scores that includes number of results in search engine queries, Google trends, number of job offerings etc. Following the top 3 are PostgreSQL and also MongoDB as the leader of the NoSQL databases. Looking at another review, the same 3 software products are topping the list of the DBMSs in use in medium to large businesses, tailed by IBM DB2, PostgreSQL and MongoDB [47]. In the category of document databases MongoDB seems to have established a solid position. As an example of this is the fact that Microsoft's Azure based multi model Cosmos DB implements a MongoDB API among a few others [6].

2.2 Knowledge discovery

After the data is obtained and stored we apply the knowledge discovery process to it to extract meaningful knowledge and value. In general the process consists of the following tasks [45] of preprocessing (steps 1-4), data mining (steps 5-6) and post-processing (step 7).

1. **Data cleaning** to remove noise and inconsistent data
2. **Data integration** where multiple data sources may be combined
3. **Data selection** where data relevant to the analysis task are retrieved from the database
4. **Data transformation** where data are transformed and consolidated into forms appropriate for mining by performing summary or aggregation operations
5. **Data mining** an essential process where intelligent methods are applied to extract data patterns
6. **Pattern evaluation** to identify the truly interesting patterns representing knowledge based on interestingness measures
7. **Knowledge presentation** where visualization and knowledge representation techniques are used to present mined knowledge to users

Next we introduce some pre-processing, data mining and post-processing tasks relevant to this thesis. Subsection 2.2.1 *Natural Language Processing* covers what we talk about steps 1 through 4 and Subsection 2.2.2 *Topic modelling* covers the steps 5-6.

2.2.1 Natural Language Processing

”The meaning of a word is its use in the language.” [108]

Natural language processing (NLP) is a wide and multidisciplinary field of study on its own. It is a discipline consisting of linguistics, computer science (software engineering), and machine learning [97] as seen in Figure 2.2.

The process of NLP ”generally involves translating natural language into data (numbers) that a computer can use to learn about the world” [46]. While NLP could as well be a data mining task, in the context of this paper we consider it as preprocessing and preparation to our data mining. Practical applications of NLP can be found anywhere from news to law and finance as seen in Table 2.2. These applications include summarization and knowledge extraction in text mining, which is what this study leans towards. Without trying to cover the whole field of study let us focus on a couple of concepts important to us.

Search	Web	Documents	Autocomplete
Editing	Spelling	Grammar	Style
Dialog	Chatbot	Assistant	Scheduling
Writing	Index	Concordance	Table of contents
Email	Spam filter	Classification	Prioritization
Text mining	Summarization	Knowledge extraction	Medical diagnoses
Law	Legal inference	Precedent search	Subpoena classification
News	Event detection	Fact checking	Headline composition
Attribution	Plagiarism detection	Literary forensics	Style coaching
Sentiment analysis	Community morale monitoring	Product review triage	Customer care
Behavior prediction	Finance	Election forecasting	Marketing
Creative writing	Movie scripts	Poetry	Song lyrics

Table 2.2. *Categorized NLP applications [46]*

Lemmatization and Stemming

For the words ”goes”, ”going”, ”went”, and ”gone” there exists one common base form ”go”, which also represents all the former in a dictionary. This form representing a set of words is called word’s *lemma* [22] in morphology. Lemmatization (or lemmatisation) is the process of finding or identifying the lemmas from inflected words.

Lemmatization should not be confused with stemming. Stemming is removing suffixes, or basically chopping off the ends, of the words to combine them under a common *stem* [46]. This is done without knowing of the words context. Whereas ”lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words ... If confronted with the token saw, stemming might return just s, whereas lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun” [70].

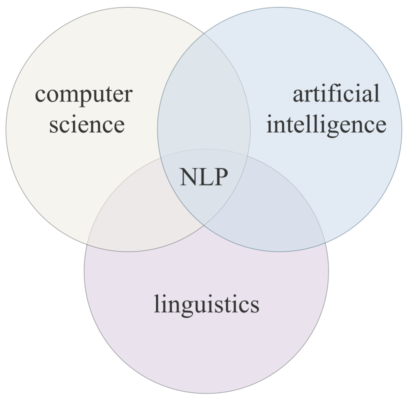


Figure 2.2. *The derivation of NLP*

Using lemmatization or stemming for a NLP will reduce the vocabulary size but increase ambiguity [46]. At least for an information retrieval task in english language it might be better not to use either normalization method [70]. On the other hand in the Finnish language there are results showing lemmatization is a better method than stemming, when clustering documents for information retrieval [58].

POS tagging, NER tagging and Dependency parsing

Part of Speech (POS) tagging in NLP signifies resolving to which part of speech each word in a text document belongs to. This can be useful when wanting to reduce dimensions of a large corpus by dropping some parts of speech, or wanting to get only nouns from an essay to get an idea of its topic.

There are also more complex needs. Let's say we want to translate the sentence *"I want to teach a fly to fly"* to Finnish. In Finnish *"to fly"* is *"lentää"* and *"a fly"* is *"kärpänen"*, two complitely different words there. The translated sentence would be *"Haluan opettaa kärpäsen lentämään"*. So for a machine to succesfully do this it is beneficial to find out the parts of speech for each word first. Some earlier probabilistic models to predict the POS tags used *Hidden Markov Model (HMM)* [61]. Other popular methods in use from the 1990's were rule-based algorithms [20]. State-of-the-art results have been reached using *neural networks* [95].

Other similar kind of a process is **Named Entity Recognition (NER)**. Named entities are the names of particular things or classes and numeric expressions [92] such as Finland, Teemu Pukki and Tampere University. But many a such proper noun or noun phrase can refer to the same named entity. For example John Fitzgerald Kennedy, Jack Kennedy, President Kennedy, and JFK can all be referring to the same entity. In an NLP pipeline POS tagging usually precedes NER tagging and the results of the former can be used to predict the latter [95].

As was the case with POS tagging, statistical modeling is a good approach for NER as well. Some popular approaches are *Conditional Random Fields (CRF)* and again neural networks [95]. Also rule based attempts exist but statistical methods tend to give better results [95].

Dependency parsing means breaking up a sentence or any other statement with a formal grammar to understand its structure [95]. The statement can be an arithmetic one such as $((8 + 3) * (6 - 2))$.

This could be parsed as a binary tree according to the arithmetic rules - having the numbers as leaves and operations as inner nodes. This is also visible in Figure 2.3.

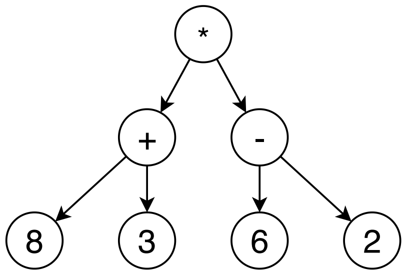


Figure 2.3. *A parsed arithmetic statement.*

Another example would be a natural language sentence such as *"Autonomous cars shift insurance liability toward manufacturers"*. The dependencies in this sentence are parsed by the *spaCy* library [50] and the dependencies shown in detail in Table 2.3. The parsed graph is a directed rooted tree having the verb *shift* as the root. All other words are dependent of it in some manner: *cars* is a nominal subject, *liability* is a nominal object etc. Looking at the table, the values of the DEP column are opened up using Cambridge English Grammar [23], and shown in the DEP EXPLANATION column. These same dependencies are also visualized in Figure 2.4 making it easier to understand the structure.

TEXT	DEP	DEP EXPLANATION	HEAD TEXT	HEAD POS	CHILDREN
Autonomous	amod	adjectively modifying	cars	NOUN	
cars	nsubj	nominal subject	shift	VERB	Autonomous
shift	ROOT	root verb of the sentence	shift	VERB	cars, liability, toward
insurance	compound	a noun acting as a modifier	liability	NOUN	
liability	dobj	direct nominal object	shift	VERB	insurance
toward	prep	preposition	shift	NOUN	manufacturers
manufacturers	pobj	object of the preposition	toward	ADP	

Table 2.3. *A parsed natural language sentence.*

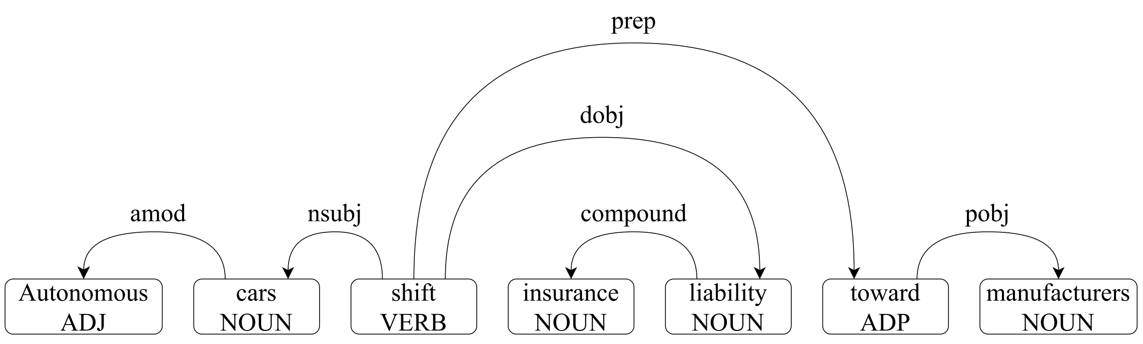


Figure 2.4. *A parsed natural language sentence presented as an arborescence [50].*

Deep learning for NLP

”One cannot guess how a word functions. One has to look at its use, and learn from that.” [108]

The aforementioned tasks of lemmatization, POS and NER tagging and dependency parsing can be performed using *deep learning*. For example the popular spaCy library uses deep learning for performing these tasks [50]. We will be get to know spaCy later on but for now let’s focus a bit on the concepts behind it. Deep learning is one subset of machine learning. It can be seen as a subset of *representation learning* as shown in Figure 2.5. This is not the only way to subset machine learning, it can also be done by learning method (e.g. unsupervised vs. supervised) or by use (e.g. classification vs. regression), which the attached diagram fails to capture.

Representation learning is about learning such representations of the data that can make it easier to extract meaningful information when building classifiers and other predictive models. In the context of probabilistic models, which neural network also is, a good representation captures the underlying distribution of the latent explanatory factors for the observed input. A good representation can be then fed as an input to a predictor [8]. It can even be an unsupervised model such as a topic model of a news article collection. In that case a good representation could be a useful reduction of dimensionality from the original text documents, comprising of the noun phrases only.

One definition for deep learning is it being layering of simple algorithms, artificial neurons, into networks several layers deep [59] as shown in Figure 2.6 of a simple sparse (not fully

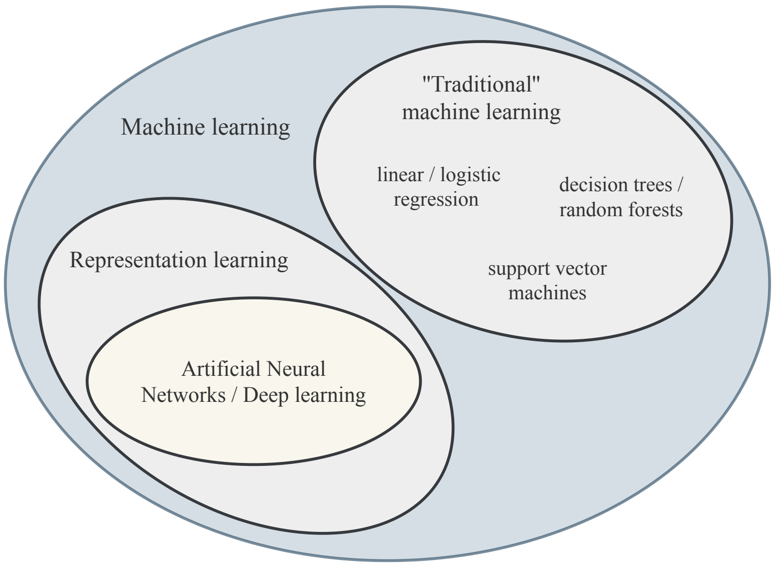

Machine learning
"Traditional" machine learning
linear / logistic regression
decision trees / random forests
support vector machines
Representation learning
Artificial Neural Networks / Deep learning

Figure 2.5. *A diagram of machine learning subsets [59].*

connected) neural network. In the figure there is an input layer of 3 neurons, two hidden layers of 4 neurons and an output layer having two neurons.

A neuron is basically a *logistic regression* model and the neural network is a stack of them [51]. The input of a single neuron is a numerical vector $\overline{x}$ and a weight vector $\overline{w}$. In the case of deep learning for NLP the input vector $\overline{x}$ in the input layer is a vector representation of a text document. The two vectors are then summed up. The summation can be written as

$$z = \sum_{i=1}^{n} x_i w_i + b \tag{2.1}$$

where $\overline{x} = (x_1, x_2, ..., x_n)$ is a vector of real values, $\overline{w} = (w_1, w_2, ..., w_n)$ is a *weight vector* and b is a constant named *bias*. Another way to write this operation is the dot product of the input vectors added to the bias

$$z = \overline{w}^T \overline{x} + b \tag{2.2}$$

The neuron also consists of an activation function f_a, which can be for example *hyperbolic tangent (tanh)*, *logarithmic sigmoid (logsig)* or a *rectified linear unit (ReLu)* function. For example a ReLu function can be written as

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & otherwise \end{cases} \tag{2.3}$$

The summation z is fed to the activation function to produce a scalar y.

$$f_a(z) = y \tag{2.4}$$

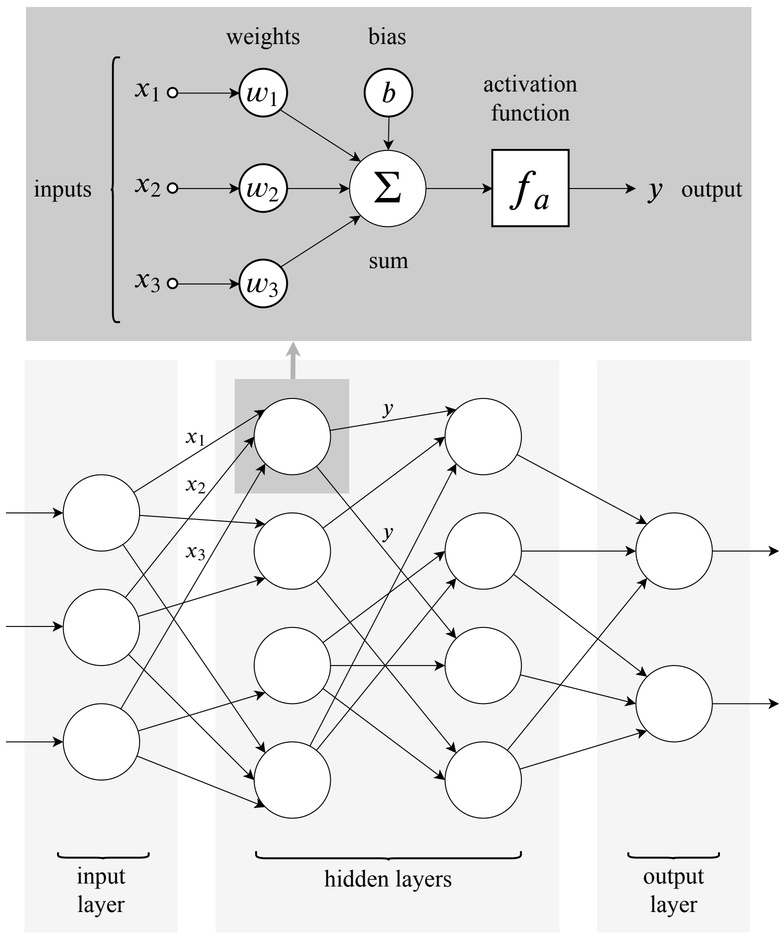


Figure 2.6. *An illustration of an artificial neural network (below), and a detailed look into one of the neurons (above).*

The y will then be sent to the next layer of neurons (as $x_i \in \overline{x}$) as a part of their input vector. If the said scalar y has a value of 0, it will not have any contribution in the next layer, meaning the neuron did not activate. The inner functions of a single neuron are also visible in Figure 2.6.

In the training phase of a neural network the purpose is to minimize the *loss function (cost function)*, the error between the actual known value and the prediction of the model. The only thing that is changed between the training iterations are the weights. A loss function is used to calculate the error between model predictions and actual values after the forward pass. Then during the backward pass the partial derivatives of the loss function are calculated with respect to all weights in the net. Then the gradients are used to adjust the weights for the next forward pass. An *epoch* is one training ”cycle” during which all training samples are shown to the net. One epoch may consist of many forward and backward passes. Training a neural network typically takes thousands of epochs [19, 46, 51, 107].

Deep learning methods ”have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics” [63]. What is the connection of our research to deep learn-

	TERMS								
DOCUMENTS	and	document	first	is	one	second	the	third	this
This is the first document.	0	1	1	1	0	0	1	0	1
This document is the second document.	0	2	0	1	0	1	1	0	1
And this is the third one.	1	0	0	1	1	0	1	1	1
Is this the first document?	0	1	1	1	0	0	1	0	1

Table 2.4. *A collection of text documents, vocabulary and a corresponding term frequency matrix.*

ing then? Well, also NLP has benefited from it as it's efficiently used by many notable projects such as spaCy [50], Stanford NLP [90] and Turku NLP [57]. These NLP libraries implement fully neural pipelines for lemmatization, POS tagging, dependency parsing, NER tagging, language detection and so forth.

Vectorizing with TF and TF-IDF

Vectorizing in NLP signifies transforming text to a numerical vector representation [83]. In our study we focus on Term Frequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF) presentations.

From a document collection we can form a *vocabulary*, a set of all the words in the corpus. Using this vocabulary we can then build our matrices. If we have already done lemmatization and dependency parsing, we can choose to use for example only lemmatized nouns to reduce dimensionality. Or we can choose to use the raw text as we do in the simple examples shown here.

A document-term matrix, also TF matrix, is the simplest word matrix. The rows of a document-term matrix represent the documents in the collection and the columns represent the terms (words) in the dictionary. Each cell value in the matrix represents the frequency of that word in that document. If some word doesn't exist in a document, the value for that word is a 0. A document-term matrix can be formed from a corpus of four documents [*This is the first document.*; *This document is the second document.*; *And this is the third one.*; *Is this the first document?*] so that first we produce a vocabulary: [*'and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this'*]. By applying one-hot encoding to our vocabulary we then get a vector representations for each document, and stacking the vectors gives us a term frequency matrix as shown on Table 2.4.

Term frequency tf_{ij} for a given term i in the document j having a frequency f_{ij} in a corpus of N documents, is calculated by dividing f_{ij} by the maximum number of occurrences of any word in the said document [66]

$$\mathrm{tf}_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{2.5}$$

Now **a term-by-document matrix**, also Term Frequency-Inverse Document Frequency (TF-IDF) matrix, can be calculated from the document-term matrix. TF-IDF means term frequency *times* inverse document frequency and measures how concentrated the occurrences of a given word are [66]. Inverse document frequency measures the importance of a term [82] and can be computed for the term i as

$$\mathrm{idf}_i = \log \frac{N}{n_i} \tag{2.6}$$

where N was the number of documents in the corpus and n_i the number of documents in which the given term i occurs in. Then from here it is defined that

$$\mathrm{tfidf}_{ij} = \mathrm{tf}_{ij} \times \mathrm{idf}_i \tag{2.7}$$

is the TF-IDF for term i in document j [66].

2.2.2 Topic modelling

"Text mining by topic modelling aims to discover topics that occur in a collection of documents in order to explore hidden semantic structures in the body of the texts" [102].

We have now reached the data mining part of the knowledge discovery pipeline. In this subsection we talk about some approaches to *topic modelling* which can be defined as stated in the quote above. By applying topic modelling to a collection of documents, we can find out the topics, or themes, the documents are about. As an example a collection, or a *corpus*, of articles from a newspaper may consist of topics of politics, sports, tech etc. We don't know beforehand what these topics are and they are unnamed and implicit [40]. To make a conclusion a discovered topic is, let's say, about sports, we will have to make a subjective judgement. Topic modelling is an unsupervised method and an good tool for exploratory data analysis and information retrieval. You can also utilize it for clustering, dimensionality reduction, historical analysis and tracking changes in topics (e.g. in a newspaper) over a period of time [40].

LDA

Arguably the most popular method in this field is a hierarchical Bayesian model called Latent Dirichlet Allocation (LDA). The intuition behind LDA is that documents display multiple topics [15]. LDA can be used on any collection with discrete data [18], meaning a collection that consists of things made up of parts [40]. Examples of these are athletes and their skills and text documents and their words. LDA uses statistical analysis based on the multivariate Dirichlet distribution to infer topics. In the context of natural language it assumes the topics to be probability distributions over a fixed vocabulary [15]. Documents in their part are then distributions over topics. Dirichlet distribution is a generalization of the beta distribution. It has support S_k over the $k-1$ probability simplex defined as

$$S_k = \{\theta : 0 \leq \theta_i \leq 1, \sum_{i=1}^{k} \theta_i = 1\} \tag{2.8}$$

where θ is a k-dimensional Dirichlet random variable [18, 75]. This is illustrated in Figure 2.7 with a 3-dimensional distribution represented as a triangular (2-simplex) surface. If the simplex is a topic simplex, then each vertex θ_i represents a topic and the distance of a vertex from the origin is the probability of the topic it represents.

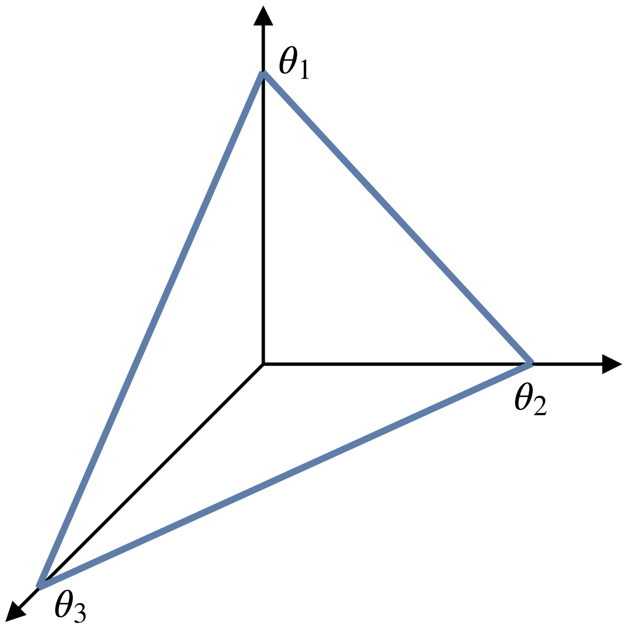


Figure 2.7. *A Euclidean plane representing a Dirichlet distribution of $k = 3$ with the points θ_i defining the plane and satisfying Definition 2.8 [75].*

LDA was first described by *Blei et al.* in 2003 [18]. In the paper they state that TF-IDF has shortcomings and that LDA is based on the ”bag-of-words” (TF) assumption where the order of words in a document does not matter. Also they state their methods can be applied to larger structural units than just separate words, such as n-grams and paragraphs. On the other hand in a later paper Blei uses TF-IDF for some tasks with LDA [17]. These tasks include visualization of topics and pruning the vocabulary.

NMF

Another approach to topic modelling is to use linear algebra and Non-negative Matrix Factorization (NMF) [64] to reduce dimensionality of non-negative matrices [43]. There are numerous fields of application for NMF including astronomy [10, 14], text mining [4, 11, 79], population genetics [38], bioinformatics [31] and more [103].

Given a non-negative matrix $A \in \mathbb{R}^{n \times m}$ the objective of NMF is to find an approximate of A by decomposing it to two non-negative matrices $W \in \mathbb{R}^{n \times k}$ and $H \in \mathbb{R}^{k \times m}$ and k being a (predefined) positive integer $k < \min(m, n)$). This is done by optimizing the distance d between A and the matrix product WH. The most widely used distance function is the *squared Frobenius norm*. In the case of the distance function being the squared Frobenius norm, the non-negative matrix factorization problem of finding W and H that minimize d_{Fro} can be calculated from

$$d_{\text{Fro}}(A, WH) = \frac{1}{2}||A - WH||^2_{\text{Fro}} = \frac{1}{2}\sum_{i,j}(A_{ij} - (WH)_{ij})^2 \tag{2.9}$$

[12, 64, 83]. Also other distance functions such as *Kullback–Leibler divergence* can be used and *regularization* added to the loss function [83].

In the context of topic modelling, the *document-term* and *term-by-document* matrices both consist of non-negative values only, and hence NMF can be applied to them. After finding the minimum for d in 2.9, the matrices W and H hold information for the k topics. Matrix $H \in \mathbb{R}^{k \times m}$ comprises of k rows representing the topics and n columns where each column represents a word (term) in the vocabulary. Each cell value then represents the importance of that word in that topic: the higher the cell value is the higher is the word's importance to the said topic. Matrix $W \in \mathbb{R}^{n \times k}$ has information on n rows representing documents and k columns representing topics. Each cell value represents the importance of a document to a topic [43].

Topic modelling over time

There are different ways of modelling topics over time. Topic models are designed to be used with categorical data [16]. This is the case with decomposition using LDA or NMF. These next two ways described to model topics over time use different perspective, but both consider data to be completely discrete. In Algorithm 1 the result is one topic model M . The topic distributions of M are then studied separately in each time slice t of length l (e.g. one year).

Algorithm 1: model first, divide last -approach to topic modelling over time

```
Result: a topic model $M$ and subsets of documents
  1. take a collection of documents and train a topic model over it;
  2. order the documents in time;
  3. choose a time span $l$;
  4. divide the ordered set of documents into $t$ discrete subsets of length $l$
     each;
```

This approach was taken for example by *Griffiths et al.* [44]. In this approach the topic space stays the same over time.

In Algorithm 2 the result is a set of topic models $\{M_1, ..., M_t\}$ ordered in time and each modelled over the documents coming from a separate t time slices of equal length l.

This sort of approach was taken for example by *Wang et al.* [105]. Here the topics are evolving, completely different topics may emerge and old topics vanish. This approach

```
Algorithm 2: divide first, model last -approach to topic modelling over time
Result: a set of topic models ${M_1, ..., M_t}$
  1. take a collection of documents and order the documents in time;
  2. choose a time span $l$;
  3. divide the ordered set of documents into discrete subsets by the chosen
     time span;
  4. train a separate topic model over each $t$ time slices;
```

has a difficulty to align the topics from different subsets [104].

Both of these approaches use static topic models. In such a (probabilistic) model each document can be considered to be drawn from the following process [16]

1. Choose θ from a distribution over the $K - 1$ -simplex.
2. For each word
   - Choose a $Z \sim \mathrm{Mult}(\theta)$
   - Choose a $W \sim (\beta_z)$

where θ is K dimensional topic proportions, z is latent topic, w is an observed word and β_z is a distribution of words generated from topic z. Especially for the process described above with Algorithm 1 this is problematic because all documents are drawn from the same set of topics. In reality, in many corpora, the order of the documents reflect evolving topics [16]. The process described with Algorithm 2 avoids the said problem better, because it generates separate topics for every time slice.

Dynamic Topic Models derived by LDA

Blei et al. propose [16] an approach to topic modelling over time called *Dynamic Topic Model*, where they have data divided in time slices just as in the two approaches presented above eventually have. They also model the subset of documents of each slice with a K -component topic model, similarly as happened with the process described with Algorithm 2. But their approach also has a feature where the topics associated with slice t evolve from the topics associated with slice $t - 1$. Blei et al. use LDA as the basis for their dynamic model.

This feature is achieved by Blei et al. [16] by chaining natural parameters of topics in a *state space model*. The model they present for this is

$$\beta_{t,k} | \beta_{t-1,k} \sim \mathcal{N}(\beta_{t-1,k}, \sigma^2 \mathbb{I}) \tag{2.10}$$

where $\beta_{t,k}$ stands for the word distribution of topic $k \in \{1, ..., K\}$ in time slice t.

In their work Blei et al. [16] also capture the sequential structure between consecutive time slices with the model

$$\alpha_t | \alpha_{t-1} \sim \mathcal{N}(\alpha_{t-1}, \delta^2 \mathbb{I}) \tag{2.11}$$

where α_t is the per-document topic distribution at time slice t.

Then the distributions over document topic proportions, and topic word proportions are replaced with a dynamic model [16]

1. Draw topics $\beta_{t,k}|\beta_{t-1,k} \sim \mathcal{N}(\beta_{t-1,k}, \sigma^2\mathbb{I})$
2. Draw $\alpha_t|\alpha_{t-1} \sim \mathcal{N}(\alpha_{t-1}, \delta^2\mathbb{I})$
3. For each document
    - Draw $\eta \sim \mathcal{N}(\alpha_t, a^2\mathbb{I})$
    - For each word
        (a) Draw $Z_{t,d,n} \sim \mathrm{Mult}(\pi(\eta))$
        (b) Draw $W_{t,d,n} \sim \mathrm{Mult}(\pi(\beta_{t,z}))$

Here the π is a softmax function, η is the the topic distribution, $w_{t,d,n}$ is the observed word, $z_{t,d,n}$ is the topic for the word n in the document d in time slice t.

Dynamic topic modelling with NMF

A solution to the problem of modelling over time is proposed by *Greene and Cross*. It uses a two-layer dynamic topic modelling method based on NMF [43]. We will call this approach Dynamic NMF (DNMF).

In the layer one Greene and Cross divide the data to time slices of a fixed and equal length. Before applying NMF they use *Topic Coherence via Word2Vec (TC-W2V)* by O'callaghan et al. [81] to select the number of topics to include in the model. TC-W2V is a method to evaluate how related a set of top terms describing a topic is. It uses *word2vec* by Mikolov et al. [72] to achieve this. The TC-W2V coherence score for a topic is determined by the mean pairwise cosine similarity of two word vectors

$$\mathrm{tcv2w}(z_k) = \frac{1}{\binom{N}{2}} \sum_{j=2}^{N} \sum_{i=1}^{j-1} \mathrm{similarity}(wv_j, wv_i) \tag{2.12}$$

where z_k is one of the K topics ($k \in \{1, ..., K\}$), N is the number of the top terms and $\mathrm{similarity}$ is the cosine similarity [81]. Then the coherence of the whole model is aggregated as the mean of the coherence scores of the individual topics

$$\mathrm{tcv2w}(Z) = \frac{1}{K} \sum_{k=1}^{K} \mathrm{similarity}(z_k) \tag{2.13}$$

where Z denotes the entire model of K topics [81] [43]. The NMF process is then applied individually to all time slices and it produces a set of time-slice-topic-models $\{M_1, ..., M_t\}$ where t is the number of time slices. In the second layer Greene and Cross [43] create a new representation of the document collection. They view the rows of each time-slice-matrix H and call them *topic documents*. They assume that topics coming from different time slices will have similar topic documents, if they share a common theme. They then construct a topic-term matrix A' as described in Algorithm 3.

Algorithm 3: Constructing a topic-term matrix for a DNMF model

```
Result: topic-term matrix $A'$
start with an empty topic-term matrix $A'$;
for time-slice-topic-model $M_i$ do
    for time-slice-topic $z_k$ within $M_i$ do
        1. select the $N$ top-ranked terms from the
           corresponding row of the NMF factor $H_i$;
        2. set all weights for all other terms in that vector to 0;
        3. add the vector as a new row in $A'$;
    end
end
remove any columns with only zero values from $A'$;
```

After constructing A' Greene and Cross [43] decompose it to two new matrices W' and H' by applying NMF for a second time. TC-W2V is also used again to find out the suitable number K' of dynamic topics. The factors of the approximate decomposition $A' \approx W'H'$ can be interpreted as:

1. the highest scoring terms in each row of H' describe the dynamic topics
2. the column values of W' show how well each time-slice-topic relates to each dynamic topic.

Then Greene and Cross [43] track the evolution of the topics as follows:

- Each time-slice-topic is assigned to the dynamic topic for which it has the most weight according to the row values of the factor W'
- *Temporal frequency* of a dynamic topic is defined as the count of distinct time slices in which the dynamic topic appears
- The set of all documents related to a given dynamic topic across the whole collection of documents is corresponding to the union of the documents assigned to the individual time-slice-topics.
- Time-slice-topics are for their part assigned to a dynamic topic.

The output of the two-layer NMF model of Greene and Cross [43] is:

1. A set of time-slice-topic-models, each containing K time-slice-topics. These are described using their N highest scoring terms (words) and the set of all associated documents
2. A set of K' dynamic topics. Each having a set of time-slice-topics associated to it. These dynamic topics are described using their N highest scoring terms (words) and the set of all associated documents.

3 RESEARCH METHODOLOGY AND MATERIAL

A data pipeline for an apparatus of topic modelling over time involves quite a few stages. It begins from data acquisition and ends in presenting the results in an suitable manner. In this chapter we walk through most of those stages and talk about the means to perform each task at hand in an adequate way. However we are not designing a complete automated tool or software, so a requirements analysis for that is not conducted. The research is quantitative by nature as we measure speed of execution. On the other hand the research is qualitative as we ponder upon the goodness of the results of different topic modelling algorithms in the context of our use case. The main research question is:

What is the optimal pipeline leading from the original data in the Trepo repository to modelling topics over time?

Our pipeline can be divided to natural steps of extracting data from Trepo, processing data, storing data and eventually topic modelling and visualizing results. We will compare selected methods for performing each step as we build the pipeline in practice. Thus our main question can be divided into the following questions

1. *What is the best data storing solution for the data set from the perspective of further consumption?*
2. *What is the optimal way to extract data from Trepo?*
3. *What is the most suitable algorithmic approach to topic modelling our data set over time?*

We will follow the knowledge discovery process described earlier in Section 2.2. In the following subsections we define the process in practice for our pipeline and take a closer look at the research questions.

3.1 Trepo repository - the data source

As a data source for this study we use the open institutional repository of Tampere Universities (TUNI) called Trepo [100]. This repository includes open access material: self-archived articles and publications of the TUNI staff, Master's and Bachelor's theses of the University and even books of Tampere University Press are published to Trepo's open access books collection. The total amount of publications in Trepo is 43401 at the moment of writing this. Most of them are written in Finnish (21513 pieces) or English (12610). Other languages having at least 10 publications written in the language are Swedish

(254), German (233), Russian (143) and French (76). These amounts are acquired by using the search service provided by Trepo itself.

The number of publications by the type of the publication is shown in Table 3.1. The column named *Type* tells the publication type, *Count* gives the total number of publications of the given type and *% of Total* is the percentage from total amount of 43401 publications in Trepo. Then the columns *Language* and *Lang Count* show the counts by writing language amongst the given type. The publications with limited access are discarded from the table. Also languages having less than 10 publications written in them in a publication type are discarded.

Type	**Count**	**% of Total**	**Language**	**Lang Count**
Master's theses	30974	71.4	Finnish	17027
			English	4829
			Swedish	219
			German	213
			Russian	136
			French	65
Articles	5378	12.4	English	2976
			Finnish	908
Doctoral dissertations	3930	9.1	English	2976
			Finnish	908
Monographs and series	1057	2.4	Finnish	644
			English	404
Bachelor's theses	843	1.9	Finnish	768
			English	81
Open Access books	235	0.5	Finnish	196
			English	34

Table 3.1. *The counts of publications by type, and the counts by the written language inside those types.*

Navigating the Trepo web site is easy for a human user. It includes a search service providing a keyword search functionality. Additionally you can browse the site by author's name, publication title, faculty name, programme name, subject, date issued and collection type. The user can refine her search filtering with a faculty name from a list of faculty names or with a programme name from another list shown to her. Also the user can filter with point-and-click style from a list of writing languages or from a list of publication years. Since we are planning on topic modelling over time we are interested in the publication years. There are 25817 (59.5% of total) publications in Trepo issued between the years 2010-2020 and 12793 (29.5%) pieces between the years 2000-2009. From the 1990's there are 4742 (10.9%) publications, from the 1980's 47 (0.1%) and from the 1970's the number is only 12 (0.0%) publications. The count of publications issued by year between years 1990-2020 is visible in Figure 3.1. As you can see the count of publications per year has been increasing almost constantly with only a couple of clearer exceptions most notably in 2005 and 2008.

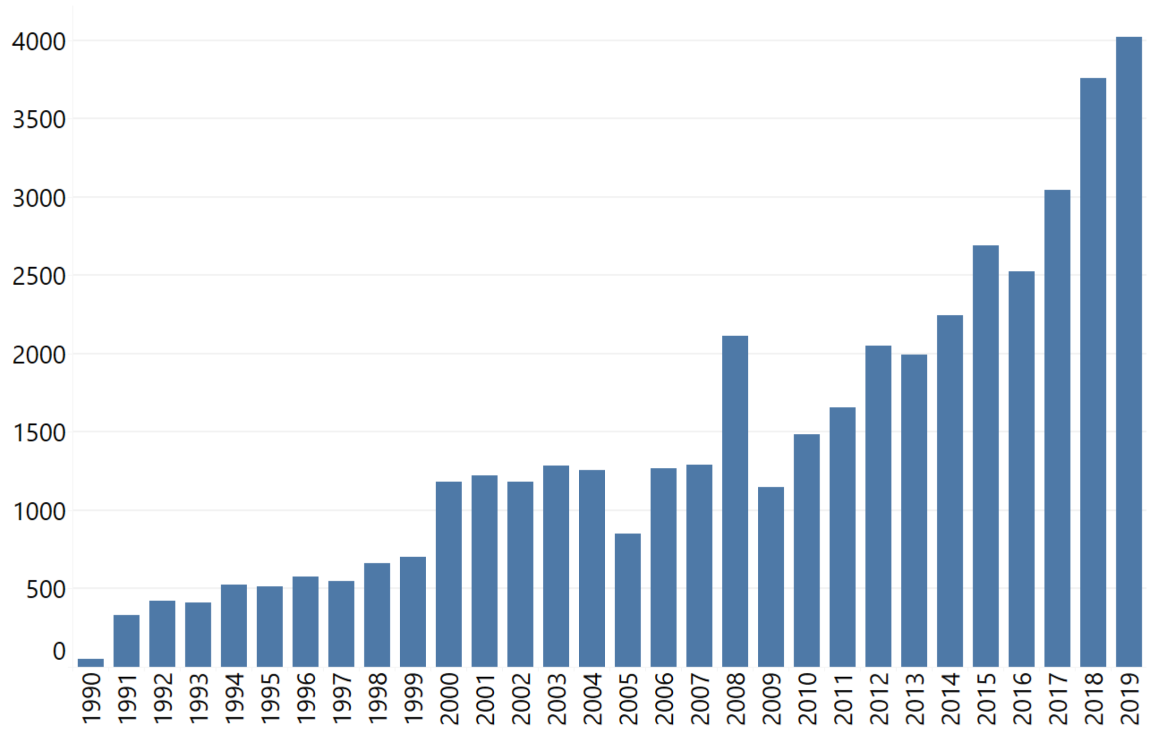


Figure 3.1. *A bar chart showing the count of publications in Trepo by the year issued. The year is on x-axis and the count on y-axis.*

Looking at the metadata present in Trepo repository, there are approximately 15-25 fields describing each scientific paper. The metadata fields present for a publication are not constant throughout the repository. Some interesting metadata include: date issued, title, written language, abstract text, abstract text language, keywords, information on copyrights, organization name, faculty name, programme name, publication type, information on the availability of the full paper in Trepo. Also if the full content is available as a PDF file there is a download link present. There are also some recently added metadata fields that are present only for the publications issued in 2019 or 2020. Most notable of them is the permission information regarding mining. This is an important factor with the newer publications.

It is possible for a metadata field to be present multiple times for the same publication. For example an abstract field to be present two times with the exact same name: one holding the English abstract and the other the Finnish abstract. Both of the fields are called ”dc.description.abstract”. In these cases there often is additional information on the language available nearby. If the abstract field is present only once, it often lacks the language information completely. Then at times the abstract field is not present at all. This kind of instability has to be considered when processing the data.

There exists a data provider utilizing the *The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)* metadata harvesting framework for Trepo repository - an API to collect metadata on the publications [98]. The metadata formats provided include 12 schemas of which two comply with OpenAIRE rules: *Metadata Encoding and Transmission Standard (METS)* and *Open Access Initiative Dublin Core (OAI-DC)*. The latter one follows the *Dublin Core (DC)* Schema. Dublin Core metadata is grouped in XML inside a *<dc>* element. All of the metadata in DC format is repeatable, optional and can appear in any order [35]. Unfortunately it seems by exploring the documentation and examples provided [98] that some schemas available for Trepo do not provide the metadata information needed to download the full PDF file of a publication even if it is available with a web browser. The METS schema provide that information as well as do many of the other schemas available. It seems that at least for some of the publications where METS provides the link for the full PDF the OAI-DC does not. In many cases METS also provides a link to a plain text version of the full publications in addition to the PDF file link. This is useful for the task at hand if we want to reduce the number of preprocessing steps needed.

The metadata can also be scraped directly from the Trepo web site by navigating the site programmatically and parsing the HTML content. The metadata fields are quite easily interpretable from the web page elements as they are presented as a table with the first column having the name (key) of the metadata field and the second column having the value. This is demonstrated with Table 3.2.

If there is additional information present, for instance specifying the language the abstract is written in, it is situated in a third column. The table does not include a key for downloading the full content PDF file when available. This is found as separate hyperlink tag. The main problem with using this approach to harvest the metadata or the full content files are the possible future changes to the web page structure: what works today may not work tomorrow. Using a dedicated API is more stable an option.

3.2 Research environment

All the computational tasks in the research are carried out on the same computer platform, which is a Hewlett-Packard HP Z420 desktop PC consisting of the parts shown in Table 3.3.

The data pipeline is built entirely using the Python 3 programming language. Python offers a rich ecosystem for data intensive programming and machine learning. Also Python offers good frameworks for building web applications and REST API services to offer the machine learning models for end user consumption. The Python code written in Jupyter Notebook running on IPython kernel is utilizing Docker containerization. Other Python scripts are executed from PyCharm IDE using an isolated virtual environment. The specific Python versions are subversions of Python 3.7 in all of the cases. Some important 3rd party Python libraries and modules used in the research are shown in Table 3.4.

Metadata key	Metadata value
dc.contributor.author	Lastname, Firstname
dc.date.accessioned	2019-01-31T08:11:12Z
dc.date.available	2019-01-31T08:11:12Z
dc.date.issued	2019
dc.identifier.uri	https://trepo.tuni.fi/handle/xxxxx/yyyyyy
dc.description.abstract	This is the abstract text of the publication.
dc.format.extent	78
dc.language.iso	fi
dc.rights	This publication is copyrighted. You may download, display and print it for Your own personal use. Commercial use is prohibited.
dc.subject	keyword 1
dc.subject	keyword 2
dc.subject	keyword 3
dc.title	My title
dc.type.ontasot	fi=Syventävä työ | en=Master's thesis |
dc.identifier.urn	URN:NBN:fi:tuni-zzzzzzzzzzz
dc.subject.degreeprogramme	Degree Programme in Journalism and Communication
dc.date.thesis	2019-01-29
dc.contributor.faculty	Faculty of Information Technology and Communication Sciences
dc.format.content	fulltext
dc.contributor.organization	Tampere University
dc.rights.accesslevel	openAccess
dc.type.publication	masterThesis

Table 3.2. *An example of the metadata available for a publication in the Trepo repository*

Comparison between PostgreSQL and MongoDB on chapter 3.3 is done using containerized versions of the databases. The platform visible in Table 3.3 is also hosting the database containers. We are not going dive deep into containers, but a short definition is given by Kane et al. [56] as "A container is a self-contained execution environment that shares the kernel of the host system and which is (optionally) isolated from other containers in the system". For understanding Docker better, good resources are for example the official Docker documentation [32] and a book by Nickoloff [78]. If you are running a Docker container on a Linux machine, there is no need to run a virtual machine anywhere on the system. Docker for Mac and Windows needs a Linux virtual machine to run on, which is handled seamlessly by the Docker Desktop application. A picture demonstrating the containerization model for Docker is adapted from Docker documentation and Chung [26, 32] and can be found as Figure B.1 in the appendices.

Hewlett-Packard HP Z420 Workstation (LJ449AV)	
CPU	Intel® Xeon® Processor E5-1620 v2 @ 3.70 GHz 4 Cores, 8 Threads
Memory	32 Gb of 1866 MHz DDR3
Hard disk	1000 Gb, ata st1000lm024 hn-m scsi
GPU	NVIDIA GeForce® GTX 1070, 8 Gb
OS	Linux Ubuntu 18.04 LTS

Table 3.3. *The specs of the computing environment.*

Library / module	**Purpose**
beautifulsoup4	HTML and XML parsing
jsonpickle	Serialization and deserialization of Python objects
pandas	Manipulating numerical tables and time series
NLTK	Statistical natural language processing
gensim	Statistical semantics and topic modeling
pdf2image	Convert PDF file to a PIL image
psycopg2	PostgreSQL database adapter
PyMongo	Tools for working with MongoDB
Python-tesseract	Optical character recognition (OCR) tool
Requests	HTTP requests simply and human-friendly
scikit-learn	Machine learning
spaCy	library for NLP using neural networks
tika-python	A Python port of the Apache Tika library

Table 3.4. *Some 3rd party Python libraries used in the research.*

3.3 Database comparison

A data pipeline can be built between two systems by using a stream processing software. If you have a data source outputting data with high velocity and volume, you can build a pipeline utilizing Apache Kafka for example. One use case could be first getting the data from Twitter to Kafka and then from Kafka to Elasticsearch [76]. We have a similar use case: to get the data from Trepo repository and to store it to a database of some kind. In our case the velocity we acquire new data is not high, there are significantly less new scientific publications coming out than there are tweets coming out.

As we have discussed in Subsection 3.1, there are a little over 40000 publications in Trepo. The amount of papers published each year has been constantly increasing, as can be seen in Figure 3.1. The year 2019 was the top year with approximately 4000 new publications added to Trepo. The amount has doubled in six years, but even if it continues doubling every six years, the amount will stay moderate for a long time. Also there will likely be limits to the constant growth, that will be hit at some point. Trepo is relatively new system and three Tampere universities have merged to one Tampere Universities (TUNI) recently. It is possible that we are experiencing a surge in the growth at the moment and it will be flattened in the next years. Because of these possible restrictions to growth and the nature of scientific publications being quite slow to produce, we will not be building a stream processing solution here. We will still need to be prepared for a larger number of publications than presently available, and thus we perform the testing with a larger data set acquired from Suomi24.fi discussion forums.

3.3.1 Setup

As the data ingestion speed from Trepo to our database is not high our research is not focused on that. Instead we focus on the output speed from the database as our pipeline needs to be able to fetch the data from the database for topic modelling efficiently. For this we are testing aggregation speed from two databases: PostgreSQL and MongoDB. The database systems are quite different as PostgreSQL is a relational RDBMS and MongoDB is a document database. We have chosen these two, because they are both suitable for our use case and are equally popular database systems as discussed in Subsection 2.1.2. Also both are free to use. If need be they can be scaled horizontally [27, 74]. Both of the DBMSs are running on containers spun up with a simple docker-compose.yml, visible in Appendix B as Listing 5. The reason for running both inside a container is making the environment as stable as possible. The versions of the databases we use for this research are MongoDB 4.0.4 and Postgres 11.1. We do not create any indices for the databases.

Python 3.7 is used to run the tests and measure the results. For communicating with MongoDB we use PyMongo library [88]. For working with PostgreSQL we use Psycopg2, which is a Python-PostgreSQL database adapter [87].

The Suomi24.fi data set is acquired earlier by the Statistical Machine Learning and Exploratory Data Analysis (SMiLE) research group from Tampere Universities (TUNI). The data set has already been preprocessed by applying lemmatization, part-of-speech tagging and dependency parsing. The data resides in a MongoDB in TUNI servers. We take a dump of the data set consisting only of posts from the subforum of *Matkailu*, which translates to *Travelling*. This subset consists of 1048629 hierarchically arranged (5 levels of hierarchy) posts written in Finnish language.

The schema used to store the data set in MongoDB is shown as Appendix B.2. MongoDB uses JSON schema, all the values shown in the left column of the figure are keys in the JSON document. The figure shows the hierarchy of the JSON document flattened, in reality some keys are nested. The *dot* (.) indicates a hierarchical relationship. An example of a Suomi24 document in JSON format can be seen in Appendix B as Listing 6, also demonstrating the hierarchy.

To perform comparison, we need to insert the data to a PostgreSQL instance. Postgres gives us options for an efficient schema, but we opt to use only one table. We also insert the data to columns maintaining the same data type as found in the MongoDB schema as closely as possible. The nested fields we insert in columns of the type *json* so that we simultaneously mimic the MongoDB and are able to test how well Postgres performs with JSON formatted data. Postgres comes with built-in JSON functions and operations [86]. The schema used in the sole Postgres table is shown as Appendix B.3.

3.3.2 Task definition

Now that we have the database instances up and data in place, we are ready to plan how to carry out the comparison. What are we comparing exactly?

We compare the speed of aggregating data from the database systems.

The aggregation speed is definitely not the only aspect to take into consideration when choosing a database system. Other aspects are for example the memory consumption and the disk space requirement, but we will assume that we have plenty of both in our disposal. We also want to choose a basic standalone setup, we assume this is all we need for the job at hand. Both of the compared databases can be expanded to a cluster of servers or to a cloud environment if need be. As we compare, we keep in mind that for topic modelling we need to create a Bag-of-Words model. For that we need the words present in each document and the number those words appear in each document. To simplify testing, we aggregate the word counts in the collection level, discarding the document level information. We choose to query for nouns only, because nouns are likely to be important in topic modelling later on. Both of the database systems offer multiple ways to query data. We choose six different strategies in total, three for MongoDB and three for Postgres.

MongoDB strategies in detail

For MongoDB the strategies we choose are:

1. Get the nouns from each document to a list with a basic find query and use Python to aggregate the counts of each word
2. Aggregate the word counts using MongoDB's map-reduce functionality
3. Aggregate the word counts using MongoDB's Aggregation pipeline functionality.

Each of the methods are applied to two fields, nouns and word_counts.noun_counts, in the schema presented as Appendix B.2. The former is the nouns of the document as a list and the latter is a list of dictionaries. These are also visible in Appendix B as Listing 6, for understanding the structure better. Both of these fields contain exactly the same information, only in a different form.

Strategy 1 in detail means querying data from MongoDB with the basic *find()* query

[74] seen in Appendix B as Listing 7. The result set from MongoDB is piped to Python *Counter* instance from the standard library module *collections*, to aggregate the number each word is present in the collection. The counter instance also sorts the word counts in descending order. This can be considered a baseline method for MongoDB.

Strategy 2 uses the MongoDB's map-reduce [74]. A map-reduce operation has two phases. In map phase each document is processed and a result is emitted forward. In our case the map phase simply emits the counts for each noun in the document. Then the reduce phase combines the result of the map phase. In our case it means combining the count of each word in document level to the count of that word in collection level. This method is again applied on two fields. MongoDB uses JavaScript functions of a predefined form as the mapper and the reducer in the map-reduce framework it implements, these functions for both fields we query are shown in Appendix B as Listing 8. The aggregation happens completely inside the database system.

Strategy 3 uses the Aggregation pipeline MongoDB implements. It is a multi-stage pipeline of different operations [74] to produce the end result. The aggregation pipeline can be used in a sharded MongoDB as can the map-reduce framework. We will use a three stage pipeline of *unwind*, *group* and *sort* to aggregate from the list of dictionaries and also from the list of words. These pipelines are almost identical and are visible in Appendix B as Listing 9. Using the pipeline, the aggregation takes place completely inside the database system.

PostgreSQL strategies in detail

The strategies of querying from PostgreSQL:

1. Get the nouns from each document to a list with a basic select query and use Python to aggregate the counts of each word
2. Aggregate the word counts from a JSON column using Postgres's JSON functionality
3. Aggregate the word counts from an array column using Postgres's array functionality.

For PostgreSQL the testing is a bit clearer and more straight forward. We only apply each of the three strategies to one column, instead of applying them on two columns as we did with MongoDB. There is no point in using other built-in functions or strategies on the JSON columns than the those specifically designed for doing this. As well as there is no point in trying to apply JSON functions on an array column, when Postgres comes with functions designed especially for this [86].

Strategy 1 means retrieving the text array column called *nouns* for each document with a very basic select query. Then extending the result to a one long list and using *collections.Counter* once again to aggregate the word counts and sort them. The basic query is visible in Appendix B as Listing 10. This is the baseline for PostgreSQL.

Strategy 2 uses Postgres' built-in functionality to query inside a column type JSON or JSONB [86]. The query is substantially more complex than was with the 1. approach. Still querying solely one table keeps this relatively simple, even though we need a nested query. We use the function *json_array_elements* to expand a JSON array to a set of values of JSON. Then we use the JSON operator -» to get an element as text. Then we group the result and sort it. The query can be seen in Appendix B as Listing 11. All the aggregation is done neatly inside the RDBMS.

Strategy 3 is utilizing the functionality Postgres implements for working with an *Array* of valid data types [86]. Again we are able to produce the aggregation completely inside the database system. We will take advantage of the *unnest* function, which expands an array so that each element becomes a row [86]. This relatively simple query is shown in Appendix B as Listing 12.

The results of the database system comparison are discussed in Section 4.

3.4 Data acquisition and preprocessing

The research begins with collecting the raw material and processing it to a suitable form to gain knowledge from it. In this case the raw material consists of the scientific publications in Trepo repository and the related metadata. To answer the second research question *"What is the optimal way to extract data from Trepo?"* we will try out two strategies of harvesting the metadata from the Trepo repository. The first option is by using the provided service of OAI-PMH. The second option is by getting the web site's HTML code with a script and parsing the metadata from there. After the metadata is collected, we download the full text publication if possible.

3.4.1 Collecting metadata

Metadata can be fetched from Trepo by using the offered OAI-PMH service. The service is located at *trepo.tuni.fi/oai*. The location of this resource is not easilly available at the Trepo web site but the service is mentioned in the privacy statement of Trepo repository [99]. We will use a Python library called *Sickle* for this [94]. It is dependent on the libraries *requests* and *lxml*. The latter is an XML toolkit and a Python binding for two libraries written in C: libxml2 and libxslt [69]. Sickle provides an easy to use API and a request to the resource returns a Python's iterator object, which can be iterated over lazily and thus memory effieciently. Retrieving the metadata is very simple as can be seen in Listing 1. You only need one request to the API to retrieve metadata for the whole collection.

Another way to fetch metadata is to scrape the Trepo web site *https://trepo.tuni.fi/*. It provides all the necessary meta information for a human user, who is utilizing a web browser, and thus we can collect that same information with a web scraper. In the privacy statement of Trepo [99] it is stated that the information regarding the publications is openly

```python
from sickle import Sickle

if __name__ == '__main__':
    sickle = Sickle('https://trepo.tuni.fi/oai/request')
    params = {
        'metadataPrefix': 'mets',  # xml schema to use
        'from': '2020-02-20'  # harvest from this date on
    }
    for record in sickle.ListRecords(**params):
        print(record.metadata)  # do something with the metadata
```

Listing 1. *Fetching documents is simple with Python and Sickle using OAI-PMH*

readable with a web browser. For metadata collection by web scraping the Trepo site we use *Requests*, which is a HTTP library for Python. After retrieving a single web page we pause for one second, because we do not want to put too much load on the Trepo servers. We then extract all the metadata information on individual publications, from the fetched HTML pages, using the *Beautiful Soup* library. All the other data in the HTML pages we discard. As was mentioned in Subsection 3.2 metadata fields in Trepo are repeatable, meaning they are can be present more than once in a publication's meta info. We solve this problem usually by turning a value to a list of values, whenever a repeating field is encountered, so that we do not lose information. Next we serialize the parsed data to JSON format and save it to hard disk, compressed in *gzip* file format. The size of the compressed file, consisting of the metadata for over 43400 publications, is only about 37 Mb.

3.4.2 Acquiring full text content

If we get the metadata using OAI-PMH we have a possibility to download the full text as a plain text file in some cases. We can do that with the Requests library and be done with it. It the plain text is not available a PDF version might be. If that is the case we need to download the PDF and then extract text from it. First we filter out the publications that do not have the full content available, which leaves us with 30487 publications to download. For downloading we use the Requests library once again and save the acquired PDF files to hard disk. From the 30487 publications tagged with the availability of a downloadable full text, we actually managed to download 30459. The space these file occupy on disk is 82 Gb. Then we need to extract the text content.

Portable Document Format (PDF) file format is more complex than a plain text file, consisting also of vector graphics, raster images, fonts and more. We could insert the complete PDF files to Postgres database as a binary string, using the *bytea* data type. For sure we want to keep hold of the files, because it may be in our interest to extract images from them later on for some other task. Right now we are interested in text only, and to be able to use it we need to separate it. For this we use *Apache Tika*. Tika is a framework for content detection and analysis. It is funded by the Apache Software Foundation and written in Java. *Tika-Python* is a Python library offering a binding to Tika REST server. To use Tika-Python you also need to have Java Runtime Environment installed, so that Tika-Python can launch the Tika server it uses. With Tika you can extract text from different file types. It can also extract metadata from a PDF with it, so we could try to add more metadata information to our publications also if we can find some from the PDF files. In an article published by Forbes in 2016 [36], Tika is mentioned being one of the technologies used to analyze millions leaked documents known as the Panama Papers.

We used Python modules *pdf2image*, *PIL* and *Python-tesseract* to back Tika up in case it doesn't find any text from a PDF file. Here we were mainly thinking of the case were a publication from the 1970's or 1980's was scanned to PDF from a paper version. The combination functions so that first the pdf2image module converts the PDF files to PIL images page by page. Then Python-tesseract uses Google's Tesseract-OCR Engine to recognize and extract text that is embedded in those images. Using this setting we managed to get text extracted from 29133 of the total number of 30459 PDF files as shown in Figure 3.2. This operation took 6 hours 25 minutes to perform on our computing environment.

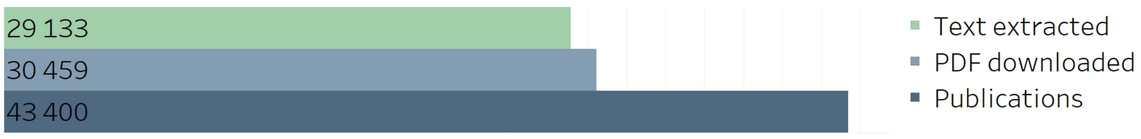


Figure 3.2. *A chart showing the total number of publications, the number we managed to download as PDF and the number we were able to extract text content from.*

At this point we have the metadata and the associated full texts extracted. Next we want to combine them and insert to a database for later use.

3.4.3 Applying NLP

The data set in whole consists of various lengths of publications from articles to books. The abstracts on the other hand are alike in length in all types of scientific papers. It would most definitely be interesting to topic model the whole data set, and for instance by using the abstracts this could be carried out. Also it would be interesting to topic model the abstracts of Master's theses side by side with the whole text content and make a comparison. Those research subjects will be saved for later, yet the ground work for the data pipeline to use in those interesting subjects is prepared here. As of now, we have the data collected and we want to sharpen out research focus to what we defined earlier in the introduction 1. As stated, the research interest here is on the Master's thesis level publications written in English, and their abstracts. All of these pieces of information are available in the harvested metadata 3.2. The publication type information is using a field name *dc.type.publication*, the abstract can be found as the field called *dc.description.abstract*, and the language information is identifiable by the key *dc.language.iso*. Sometimes there are many abstracts in different languages. In those cases we will use a key found as an extra information from the metadata and combine that with the abstract key to create a new field. In the case of an English abstract this field is called *dc.description.abstract_en*.

We want to be applying topic modelling on the abstracts and one way to do that is on the raw texts. The abstracts we use are not as long as the full texts, this should definitely reduce the dimensionality of our vocabulary and the bag-of-words models. But the least we should do is to remove punctuation and lowercase the words before we split the collection to a word list representations and start building the word vectors. If we don't remove the punctuation and lowercase letters our vocabulary will have for example terms like "*cat.*", "*cat,*", "*cat*" and "*Cat*" as four different terms. Still after these operations we will end up with an unnecessarily huge vocabulary as words like "*nebula*" and "*nebulae*" will be identified as two different terms. A solution to this problem is normalisation with lemmatization or with stemming. Stemming will only shorten the words and this might make us lose too much information. For example the Porter stemmer will stem all of the words operate, operative, operation and operator to *oper* [70]. Lemmatization on the other hand can make a difference between these terms, and save precious information by not reducing them to a one single term. Lemmatization will provide us dimensionality reduction and in addition information gain compared to stemming, so we will choose that option.

If we want to reduce dimension even more, we can perform Part of Speech (POS) tagging and then apply topic modelling on a selected subset of nouns, noun phrases, adjectives, verbs, adverbs etc. All of this Natural Language Processing (NLP) and more is provided to us by various powerful NLP frameworks and libraries such as *gensim Natural Language Toolkit (NLTK)*, *spaCy*, *StanfordNLP*, the neural parser pipeline by *TurkuNLP*. We will go with spaCy, mainly because of a personal preference and the intuitive and easy to use API it provides. It also has performed well in a dependency parser comparison by Choi et al. in 2015 [25]. SpaCy does not have support for finnish language at the moment. There exists a work around for this as the StanfordNLP has two Finnish treebanks to use with its state-of-the-art neural pipeline. Also there exists a Python package that takes the StanfordNLP library and wraps it so that you can use those models as a spaCy pipeline, with the spaCy interface. The said package is called *spacy-stanfordnlp*. In our case we don't have to worry about this since we are using English abstracts, but this is nice to know for future reference, if we want to use this same pipeline for documents written in Finnish.

SpaCy uses convolutional neural network models for performing POS tagging, dependency parsing and entity recognition. Neural networks are discussed in Subsection 2.2.1. The statistical components of spaCy pipeline are independent of each other, so you can swap the order of them or leave some components out of the pipeline [50]. It also is worth mentioning that spaCy usable with CPU only, you don't need a separate GPU to utilize it. It is not self-evident when using neural networks, because using a GPU will often speed up the computing quite a lot [9]. By adding the POS tags to the document collection metadata we are on our way of adding value and possible ways to utilize the data set. We can add for example Named Entity Recognition (NER) tags and dependencies with spaCy if we want to, but this we will not do for now. One feature spaCy has, and we are using, is language detection. This we want to utilize to make sure that the abstract is written in English, especially in the cases when the publication has two abstracts this is a necessary backup plan to make sure we labelled the correct abstract as the English one when we extracted metadata from the Trepo repository.

After extracting the linguistic features with spaCy it is wise to insert them to the database also. This can be done using only one table for now, as adding another table will not be that beneficial because all the relations are one-to-one relations. Yet, for the sake of clarity, we divide the table in two. We also change the names for some fields introduced earlier, because the raw names vary depending on which XML schema is used when harvesting through OAI-PMH and are quite cumbersome. As an example, the original field telling the top level organization of the publication was called *dc.contributor.organization*. This likely comes from a nested XML or JSON, so we have replaced it here with a simpler field name *organization*. The schema we propose to use is shown in Figure 3.3. In the table *documents* there is the content before extracting the linguistic features. In the table *linguistic_features* we have inserted the results from NLP processing with spaCy on the abstracts. Here the primary key column *id* of the documents table is a foreign key in the *linguistic_features* table. Now we are ready to consume the pre-processed data from the database to perform topic modelling.

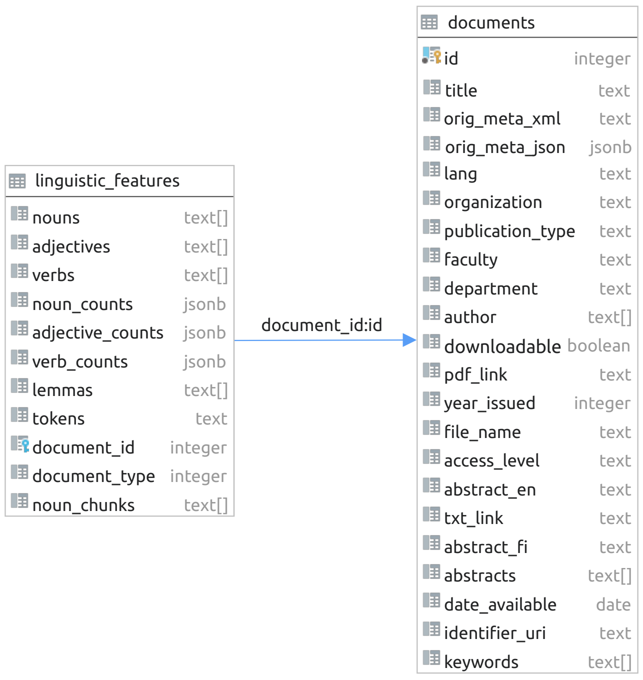


Figure 3.3. *Tables used for storing the metadata (documents) and the refined data from abstracts (linguistic_features).*

3.5 Topic modelling in practice

To see how different topic modelling algorithms behave with our data set we first omit the ”over time” demand and model the abstracts from Master’s thesis level publications in English using both Latent Dirichlet Allocation (LDA) and Non-negative Matrix Factorization (NMF) based topic modelling. We use nouns only to reduce dimensionality and expect to get good results, because nouns are important content words in revealing the topics [77, 111]. For the implementation we use the *scikit-learn* machine learning library for Python, as it comes with an easy to use implementation for both algorithms. We also use the same library for building the Bag-of-Word models and for turning the document collection into numerical feature vectors. The latter is known as vectorization. This approach is chosen because scikit-learn implements a Term Frequency (TF) vectorizer as well as a Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer, and they are quite simple to use. The vectorizers and the vocabulary created here can be reused later on when topic modelling over time.

Our strategy is to use TF vectorizer with the LDA model and then TF-IDF vectorizer with the NMF model, because from the literature we find suggestions for doing so [18, 24]. TF-IDF can be used with LDA to e.g. prune the vocabulary [17], but it likely doesn’t improve the results of the model. We already prune the vocabulary with lemmatization and use of nouns only. NMF also can be used with TF matrix, because it consists of non-negative numbers. Yet it lacks the additional measure TF-IDF weight normalization gives about the importance of a word to a document in the collection. Also in the literature we find that NMF tends to yield better results compared to LDA for short text data sets. Chen et al. state this is because NMF *”contains much more priors such as TF–IDF (term frequency and inverse document frequency) encodings for texts (instead of the only TF in LDA), Gaussian distribution for the noises with Frobenius norm, and implicit low-rank structures w.r.t. the original term–document datasets (while LDA does not have this); besides, the deterministic MU [65] algorithm in NMF also contributes to more stable and better results than the stochastic Gibbs sampling without enough word co-occurrences in LDA”* [24]. It is unclear how a *short text* is defined. An abstract might be a short text but if we move to modelling the full text publications we are not talking about short texts anymore.

After the initial familiarization with models covering the whole collection at once, we will move to modelling over time. This is done by applying the dynamic topic models using LDA as described by Blei et al. [16] and by applying similar concept using NMF as described by Greene et al [43]. Both of these algorithms are applied using a readily available implementation. Afterwards we compare the results. The comparison will then be reflected in the proposed pipeline.

3.5.1 Initial LDA and NMF modelling

For the exploratory analysis of the latent topics we use *scikit-learn* library, which is a machine learning library for Python. It is built on *NumPy*, *SciPy*, and *matplotlib* [91] and implements a simple and consistent interface.

Vectorization and preserving vectorizers

The vectorization process of the text corpus using the scikit-learn library is behind the following steps

1. Filter the publications that are of type Master's thesis and written in English
2. Get the nouns to a list of lists and join to a list of strings
3. Build the vocabulary and the CountVectorizer (TF vectorizer)
4. Build the TfidfVectorizer (TF-IDF vectorizer)
5. Save the vectorizers to disk or to database.

The first step is easily achieved with a Python script using the metadata we have stored in the PostgreSQL. We also have already extracted the nouns with spaCy as described in Subsection 3.4.3. In the second step we have to take care of joining the tokenized nouns of each document to one string representing that document, because the input for the vectorizer we use is a list of strings. The third step is building the vocabulary and the TF vectorizer. With the tools we have chosen, these two can be done together, as the scikit-learn implementation of TF vectorizer called *CountVectorizer* builds the vocabulary automatically from the corpus it is provided. It also is possible to give the vocabulary as a parameter to the vectorizer. This could be useful if we want to use a larger vocabulary than the one derived from the text corpus we feed to the vectorizer. We modify the default parameters for the constructor a little by setting the minimum document frequency for a word to 2 and maximum to 0.95. This means only words present in less than 2 documents are discarded and words present in more than 95% of documents are also discarded. The scikit-learn models usually come with a simple API of *fit* and *transform* and a combined method *fit_transform*, which does both at once. This is the case here also. We can initiate both vectorizers with the same parameters (Steps 3 and 4) and fit them by giving the corpus we constructed in Step 2. The TF-IDF vectorizer class is called *TfidfVectorizer* in scikit-learn. If we wanted to follow Steps 3 and 4 more strictly, we could produce the TfidfVectorizer from the CountVectorizer using a class called *TfidfTransformer* also from scikit-learn and especially made for this purpose [91]. In the case of Tfidfvectorizer the samples, matrix rows, are normalized individually to unit norm. The default normalization parameter is called ”l2” and means that the squares of (row) vector elements add up to 1. Also when using the l2 norm, the dot product of two vectors is the cosine similarity between the two.

Last step is saving the vectorizers. Python has modules in the standard library for serializing and deserializing objects in whole. Such modules are for example *marshal* and *pickle* [89]. The Python official documentation states that marshal is more primitive and pickle should always be preferred. Also the documentation warns that pickle is not safe, you only should unpickle trustworthy data. A third party library *jsonpickle* is available to serialize complex Python objects to JSON format. Since JSON is safer format than Pickle and more universal too, this is what we will be using as a baseline method. There exists another strategy for serialization in addition to serializing the whole Python object. For instance in the case of CountVectorizer you can only extract the vocabulary and the parameters from a trained vectorizer and serialize them. Then when loading the vectorizer, you deserialize the parameters and create a new instance of the CountVectorizer class using them. Then set the vocabulary attribute of the new instance from the serialized vocabulary. In the case of TfidfVectorizer you also need to serialize a *numpy* array containing the inverse document frequencies. Then as you deserialize, you make a sparse matrix from it and plug that matrix into the vectorizer. These processes are shown in Listing 13 found from Appendix C. Both of these strategies create a JSON object, which can be saved to a PostgreSQL database using JSONB datatype as well as stored to a file on disk.

We carried out a quick comparison between the two strategies using a collection of 3231 paper abstracts. The dictionary for this collection consists of 33736 distinct words. The number of words in the collection is 570942 in total. The TfidfVectorizer object serialized using *jsonpickle* was *28,8 Mb* on disk, while saving only the essential parts using the other strategy came out with a *2,0 Mb* JSON file in size. The latter serialization strategy can potentially save great amount of space on disk in comparison to serializing the whole Python object, and in many cases it should be preferred in our opinion. Especially in the case of topic modelling over time you create many vectorizer instances from different time slices and the difference in size on the disk will multiply.

LDA and NMF with scikit-learn

The scikit-learn implementations of LDA and NMF implement the same API we got to know with the vectorizers in Subsection 3.5.1. We decide to infer 10 topics with one model and 20 topics with another model for both LDA and NMF. Models are trained in a straight forward manner. You need to extract the TF matrix or the TF-IDF matrix from the vectorizer and then train the topic model with it using the *fit* function. Both models basically only need an instance of the model created with default parameters, and you are good to go. You can modify the parameters in quite a few ways, but the scikit-learn documentation states the default settings usually work well [91]. Other than changing the number of topics to infer we stick to the defaults. Some of the parameters are shown in the table with a description and the default value [91].

We use CountVectorizer and a term-document matrix to train the LDA model and TfidfVectorizer and a TF-IDF matrix to train the NMF model. Then we print the *top_n* terms for each topic to a CSV file and visualize using Tableau. The *top_n* terms for a topic

Model	Parameter	Default	Description
Common	n_components	10	Number of topics
	max_iter	200	Maximum number of iterations
LDA	doc_topic_prior	1/n_components	Prior of document topic distribution θ (discussed in Subsection 2.2.2)
	topic_word_prior	1/n_components	Prior of topic term distribution β (discussed in Subsection 2.2.2)
	learning_decay	0.7	Learning rate in the online learning; κ
	learning_offset	10	Downweights early iterations; τ_0
NMF	solver	'cd' coordinate descent	Numerical solver to use
	beta_loss	frobenius	Beta divergence to be minimized d_{Fro} (discussed in Subsection 2.2.2)
	alpha	0	Multiplier of the regularization terms
	l1_ratio	0	Penalty ratio between L1 and L2

Table 3.5. *Some of the parameters for LDA and NMF models in scikit-learn.*

means the $n \in \{1, 2, 3, ...\}$ words having the highest probability that the word is assigned, or belongs, to the topic. These terms describe the topic but inferring a title for it needs subjective judgement. To derive more objective a title this inferring process can be done in a quantitative manner: e.g. by using a majority vote among a sample from some target population. The results of the initial topic modelling are shown in Section 4.

3.5.2 Modelling over time with LDA

After the initial LDA and NMF modelling in Subsection 3.5.1 we have gotten some grasp of the topics these documents consist of. In addition we have created a vocabulary with the CountVectorizer from the scikit-learn library and we can reuse that when modelling over time. In addition we have the tokenized and lemmatized documents, which in addition we have POS tagged. We plan to use the nouns only once again for this task, so presumably we don't have need for stop word removal.

For the implementation of the modelling over time with Latent Dirichlet Allocation (LDA) based algorithm we use the Dynamic Topic Models (DTM) by Blei and Lafferty [16]. As already described in Subsection 2.2.2 DTM is an extension to the idea of LDA, which allows the representations of the topics to evolve over fixed time intervals. The length of a time interval can be e.g. an hour, a day, a month, a year or a decade. We feel that a natural starting point in the case of the scientific publications is one year, and that is what we will go forth with. One important factor using DTM is that the time interval is fixed. The amount of documents in each time slice on the other hand doesn't have to be fixed, DTM is not opining on or restricting this quality in the data set. Yet this might have impacts on the modelling quality, and we feel that after seeing the distribution of all the documents in Trepo in Figure 3.1, we want to take a look at the distribution of the Master's theses data set we are about to use.

Total amount of documents having an English abstract is 4018. This is our data set. As is visible in the figure the distribution between years is not close to even.

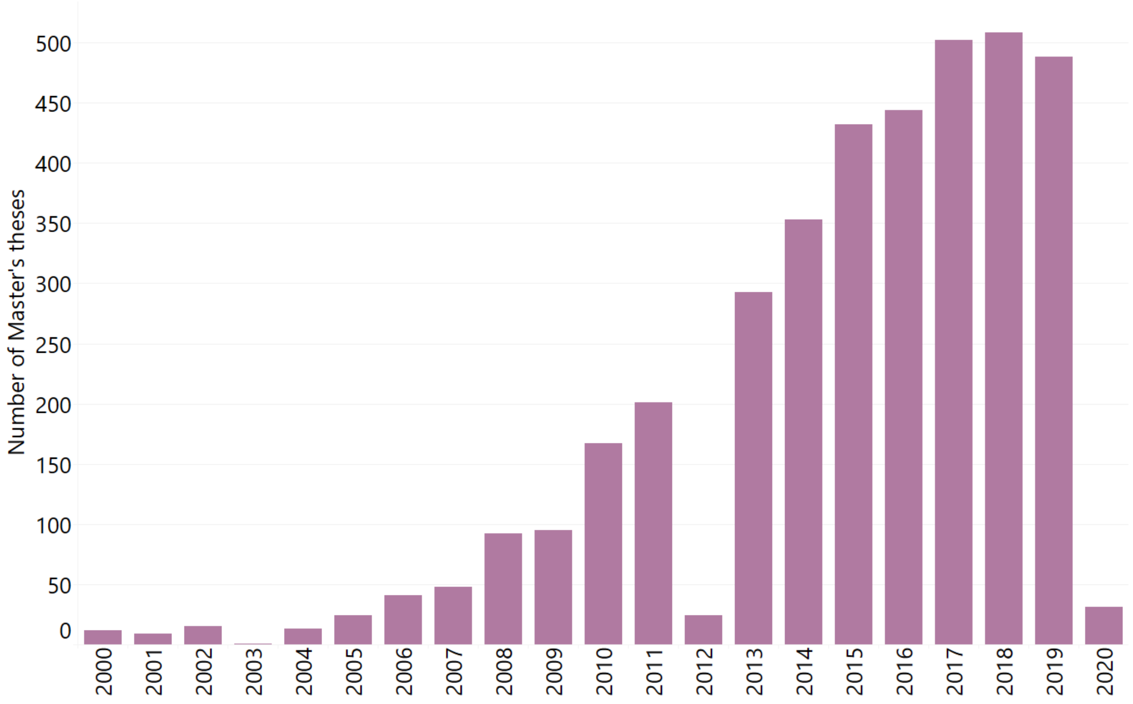


Figure 3.4. *A bar chart visualizing the distribution of those Master's thesis level publications that have an English abstract present in the Trepo metadata [100]*

The amount of publications is constantly ascending in big picture with a couple of years deviating from this trend for some reason. It seems only the publications from the 21st century have abstracts included in the metadata. This is unfortunate, especially because it would be interesting to see the evolution all the way from the 1970's and 1980's to this day. There are not that many publications from those decades in Trepo, but from the 1990's there are significantly more. Maybe we could get more English abstracts to our data set by extracting them directly from the full text contents instead of using metadata. This is something to consider in the future. Right now we decide to model the topics of this entire data set without excluding any years. We decide we use a python wrapper provided by the gensim NLP library [39] for the original C++ implementation of the Dynamic Topic Models [33] to implement topic modelling over time with LDA. To use the wrapper we need to compile the binaries of the C++ implementation first, but these steps are well documented in the gensim documentation. Workable bash commands for the deed for debian based Linux distibutions are visible in Listing 2. They include a mandatory installation of the The GNU Scientific Library [41].

The gensim DTM wrapper expects some additional preprocessing for the documents. The corpus needs to be an "iterable of iterable of (int, int)" [39]. This means a bag-of-words representation where the first element of the tuple is the term's index in the vocabulary. The second element is the number of times (count) the word appears in the

```
git clone https://github.com/blei-lab/dtm.git
sudo apt-get install libgsl0-dev
cd dtm/dtm
make
```

Listing 2. *Compiling the binaries for DTM C++ implementation [39]*

given document. This format only includes the terms with count greater than zero, so it factually is a sparse matrix in LIL [54] format. This format is demonstrated in Table 3.6. For example the text *"This document is the second document"* seen in the table gets the representation $[(1,2),(3,1),(5,1),(6,1),(8,1)]$ in this case.

	VOCABULARY [term | index]								
DOCUMENTS	and | 0	document | 1	first | 2	is | 3	one | 4	second | 5	the | 6	third | 7	this | 8
This is the first document.		(1, 1)	(2, 1)	(3, 1)			(6, 1)		(8, 1)
This document is the second document.		(1, 2)		(3, 1)		(5, 1)	(6, 1)		(8, 1)
And this is the third one.	(0, 1)			(3, 1)	(4, 1)		(6, 1)	(7, 1)	(8, 1)
Is this the first document?		(1, 1)	(2, 1)	(3, 1)			(6, 1)		(8, 1)

Table 3.6. *A table demonstrating the bag-of-words representation of a corpus the gensim wrapper for DTM ingests. The representation is a list of lists having one row for each document.*

In addition we need to arrange the documents in ascending order by the issuing year to be able to track topics in time. Also we must manually provide the gensim wrapper the information on how many documents belong to each time slice, it is not capable of inferring that otherwise. Some of the preprocessing is visible in Jupyter notebook 3.5.

The original C++ implementation expects an input of two files. First file *foo-mult.dat* is a one-document-per-line file consisting of bag-of-words representations of the documents. The documents need to be sorted in ascending order in time. The second file *foo-seq.dat* consists of information on the number of time slices and of lines indicating how many documents in *foo-mult.dat* belong to each time slice [33]. The gensim wrapper also creates these files temporarily but abstracts this away with the API it offers to the user.

We will build a DTM model using the same amount of topics we used earlier with sklearn models, which is 20. DTM will produce a constant number of topics for each time slice [33]. The process of inferring topics with the gensim wrapper is visible in Jupyter notebook 3.6.

After building the model we use the gensim wrapper's built-in functions to dig out word probability distributions for each topic at each time slice. From this we build a data source for a visualization tool to consume. We will use Tableau for visualizing the model outputs and helping in quality evaluation. To achieve this we need to create row-oriented data which Tableu and other business intelligence tools such as Power BI are optimized to consume. This process is shown in Jupyter notebook 3.7.

```
[ ]: import json
     from collections import Counter
```

Load the abstract data from a .json file.

```
[ ]: data_file_name_with_path = '/masters_abstract_en_lemma_nouns_only.json'
     with open(file=data_file_name_with_path, mode='r') as file:
         data = json.load(fp=file)
```

Change all 'dc.date.issued' values to year format yyyy only (some are in yyyy-mm-dd).

```
[ ]: for d in data:
         d['dc.date.issued'] = str(d['dc.date.issued']).split('-')[0]
```

Sort by the 'dc.date.issued' field. Then extract abstracts.

```
[ ]: data = sorted(data, key = lambda di: di['dc.date.issued'])
     abstracts = [di['abstract_lemma_nouns_only'] for di in data]
```

Use a Counter to extract counts for documents by year. Then destructure to get a sorted tuple where each element is a doc count for one year.

```
[ ]: c = Counter([d['dc.date.issued'] for d in data])
     years, counts = zip(*sorted(c.items()))
```

Create a dict consisting of the abstracts and time slice info.

```
[ ]: json_data = {
         'abstracts': abstracts,
         'time_slices': list(counts)
     }
```

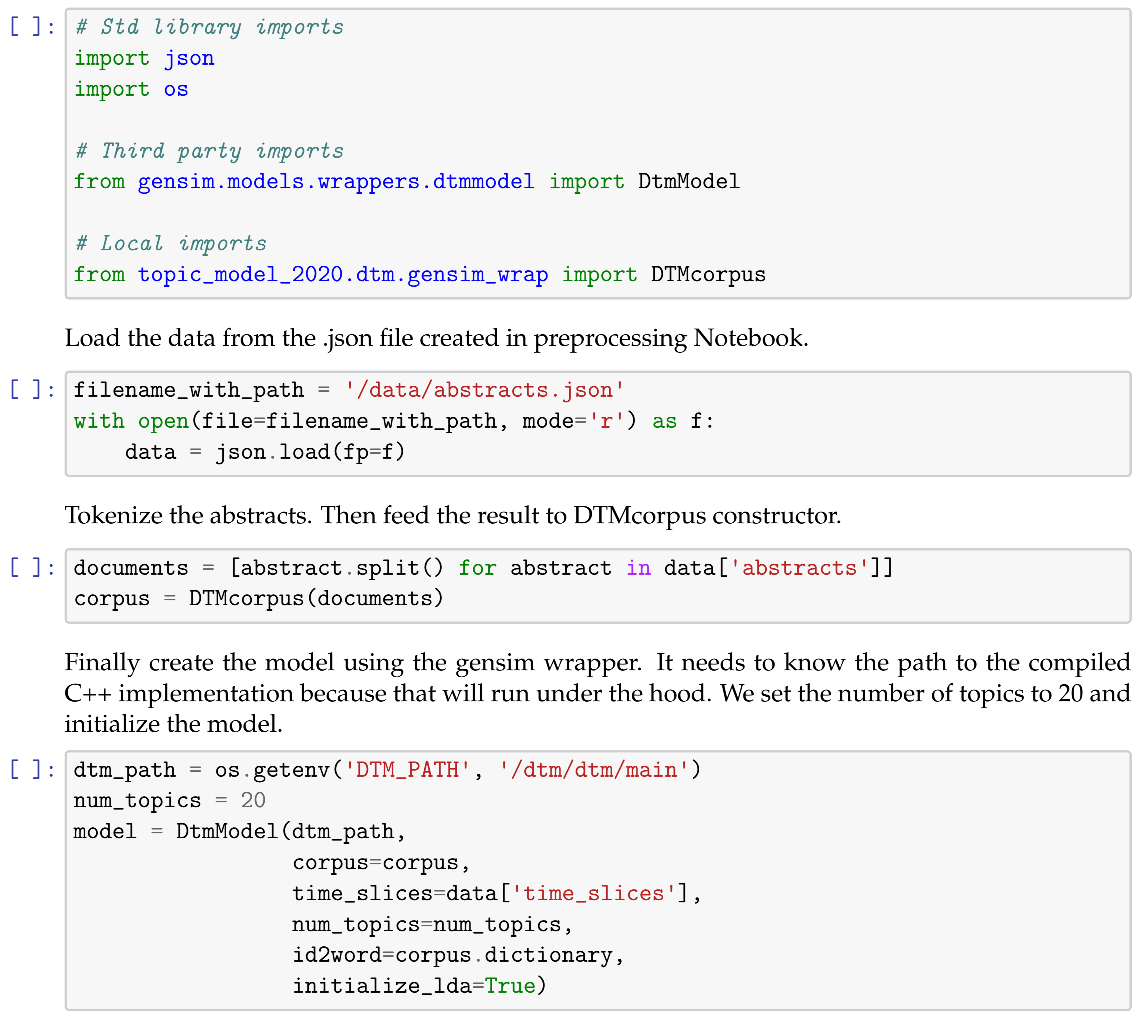
Finally dump the dict to a .json file.

```
[ ]: out_filename_with_path = '/data/abstracts.json'
     with open(file=out_filename_with_path, mode='w') as f:
         json.dump(json_data, fp=f, ensure_ascii=False, indent=2)
```

Figure 3.5. *Jupyter notebook demonstrating the preprocessing for the gensim DTM wrapper.*

3.5.3 Modelling over time with NMF

For modelling topics over time with Non-negative Matrix Factorization (NMF) we follow the setup described for DTM in Subsection 3.5.2 as closely as we can. Only this time we use a different algorithm of course. The data set is the same one described in Figure 3.4. For the algorithm there exists an implementation of the Dynamic NMF method proposed by Greene and Cross [43] provided also by the same authors. We will use this implementation written in Python to model over time with DNMF. What is nice here is that this implementation uses a similar approach for setting up the data as the Blei and Lafferty implementation of Dynamic Topic Models [33]. Only instead of arranging the documents in files that represent time windows the data needs to be divided in subfolders which correspond to the time windows. The order of the time windows is presented as the alphabetical order of the subfolder names. Each document is then placed in those



[ ]:
```
# Std library imports
import json
import os

# Third party imports
from gensim.models.wrappers.dtmmodel import DtmModel

# Local imports
from topic_model_2020.dtm.gensim_wrap import DTMcorpus
```

Load the data from the .json file created in preprocessing Notebook.

[ ]:
```
filename_with_path = '/data/abstracts.json'
with open(file=filename_with_path, mode='r') as f:
    data = json.load(fp=f)
```

Tokenize the abstracts. Then feed the result to DTMcorpus constructor.

[ ]:
```
documents = [abstract.split() for abstract in data['abstracts']]
corpus = DTMcorpus(documents)
```

Finally create the model using the gensim wrapper. It needs to know the path to the compiled C++ implementation because that will run under the hood. We set the number of topics to 20 and initialize the model.

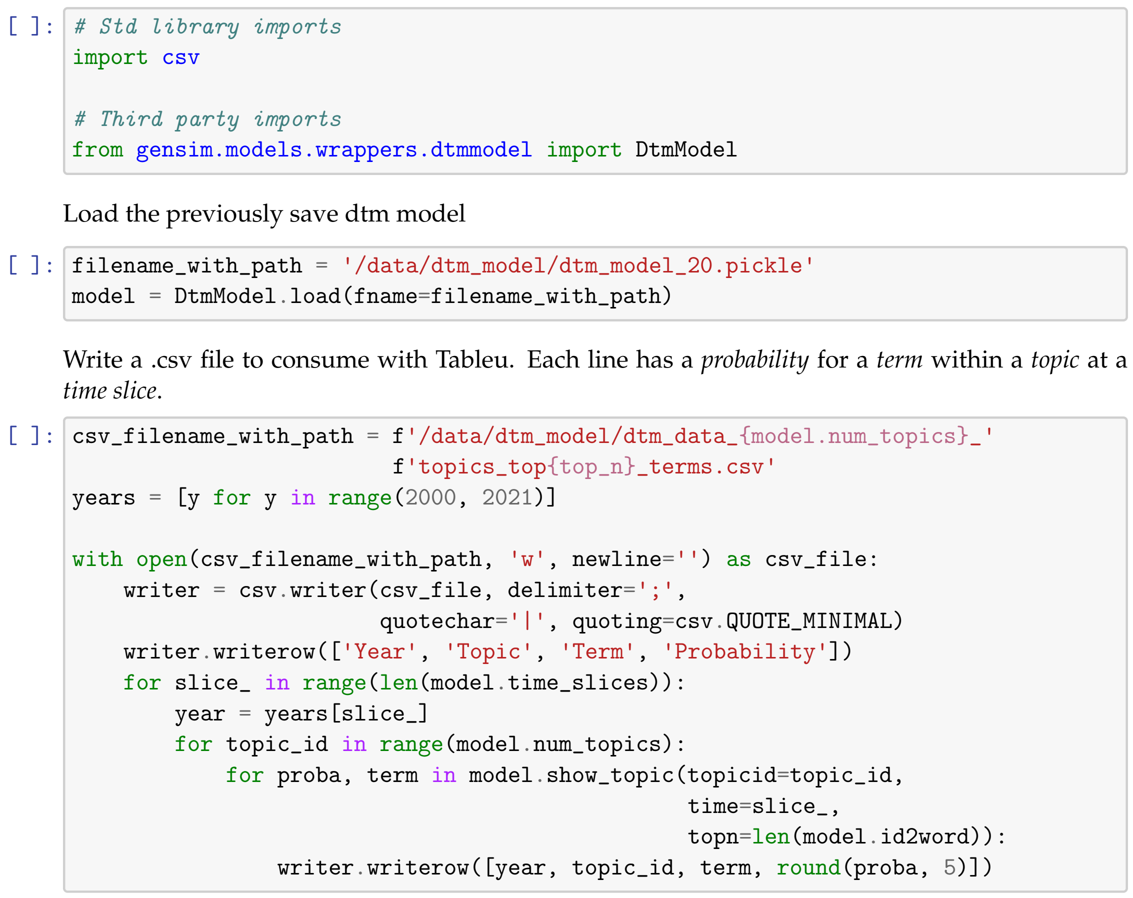
[ ]:
```
dtm_path = os.getenv('DTM_PATH', '/dtm/dtm/main')
num_topics = 20
model = DtmModel(dtm_path,
                 corpus=corpus,
                 time_slices=data['time_slices'],
                 num_topics=num_topics,
                 id2word=corpus.dictionary,
                 initialize_lda=True)
```

Figure 3.6. *Jupyter notebook showing the use of the gensim DTM wrapper.*

folders in separate plain text files - one document per file [34]. This preparation process is shown in Jupyter notebook 3.8. Note that are using term *time window* here because it is used by Greene and Cross in their work [34, 43]. This term is interchangeable with the term *time slice* we have used before. We will also use *window topic* synonymously with *time window topic*.

As DNMF is not a probabilistic approach the algorithm will not produce probability distributions for words over topics or topics over documents as the LDA models do [16, 18]. Instead it will produce ranks for the words in the dynamic topics at each time window [34]. Also it is notable that a dynamic topic at some time window may consist of many *window topics* and the number of existing topics on each time window can vary [34]. When training a model for a time window DNMF will by default only take into account the terms that are present in over 10 documents in that time window. We will change that to 2 documents as it has been with other topic modelling algorithms thus far.

Inferring topics with the implementation by Greene and Cross is fairly straight forward. They have created step by step instructions for their command line interface. You only need to clone the repository and prepare a Python interpreter with the required depen-



```
[ ]: # Std library imports
     import csv

     # Third party imports
     from gensim.models.wrappers.dtmmodel import DtmModel
```

Load the previously save dtm model

```
[ ]: filename_with_path = '/data/dtm_model/dtm_model_20.pickle'
     model = DtmModel.load(fname=filename_with_path)
```

Write a .csv file to consume with Tableu. Each line has a *probability* for a *term* within a *topic* at a *time slice*.

```
[ ]: csv_filename_with_path = f'/data/dtm_model/dtm_data_{model.num_topics}_'
                              f'topics_top{top_n}_terms.csv'
     years = [y for y in range(2000, 2021)]

     with open(csv_filename_with_path, 'w', newline='') as csv_file:
         writer = csv.writer(csv_file, delimiter=';',
                             quotechar='|', quoting=csv.QUOTE_MINIMAL)
         writer.writerow(['Year', 'Topic', 'Term', 'Probability'])
         for slice_ in range(len(model.time_slices)):
             year = years[slice_]
             for topic_id in range(model.num_topics):
                 for proba, term in model.show_topic(topicid=topic_id,
                                                     time=slice_,
                                                     topn=len(model.id2word)):
                     writer.writerow([year, topic_id, term, round(proba, 5)])
```

Figure 3.7. *Jupyter notebook showing some post-processing of gensim the DTM wrapper outputs.*

dencies [34].

The steps we will go through with the algorithm are listed below. The bash commands for the steps are shown in Listing 3.

1. **Preprocessing** consists of tokenizing, removing stop words and creating a document term matrix utilizing the *prep-text.py* script. We will use flags - *-tfidf* to create a TF-IDF matrix and - *-norm* flag to normalize document length. Inputs are the files we created earlier using the Jupyter notebook.
2. **Window Topic Modeling** consists of generating the time window topics by applying NMF on each preprocessed data file from Step 1. The script to use here is *find-window-topics.py*. We are using the flag *-k 20* to infer 20 topics.
3. **Dynamic Topic Modeling** is where the algorithm combines the time windows and generates dynamic topics spanning across multiple time windows. The script used for this step is *find-dynamic-topics.py*. We will use the flag *-k 20* to infer constant number of 20 topics.
4. **Display topics** is for showing the top ranked words for the inferred topics across all time windows. This is implemented in the script *display-topics.py*.
5. **Track dynamic topics** will output the top terms for each time window from the

dynamic model. We will use flag *- -top 1000* to output top 1000 terms at each moment in time. The flag *- -long* gives us a better output format for post-processing. We will also redirect the output to a text file using *tee*

6. **Choose topic number automatically** is an extra step. The implementation provided by Greene and Cross uses topic coherence based on gensim's Skipgram word2vec model [39] to try to infer the number of topics for the data set [34, 43]. Basically this means that the algorithm will build models using all numbers of topics from the given range of numbers. Then based on the topic coherence it will give three recommendations for each time window and for the dynamic model. There are three stages to achieve this

    - Build the word2vec model using the script *prep-word2vec.py* and save it
    - Use the word2vec model to automatically find best number of topics for each time window. The model to use is indicated with the flag *-m* and the interval of topic numbers we want to examine is given by *-k 5,25*. The script used in this step is *find-window-topics.py*
    - Lastly we automatically infer the best number of topics for the dynamic topics with *find-dynamic-topics.py*. We will also use the flag *-k 5,25* again to limit the automatic inference between 5 and 25 topics. The word2vec model is again given with the flag *-m* and is the same model we used for time window topics.

    We will not use the result for the automatic inference in the actual topic modelling. It is something to consider in the future research but for now the only purpose is to see how much the algorithm's proposition differs from our choice of 20 topics.

The DNMF implementation includes a feature of automatic proposition for the best number of topics [34], which we will explore to see if our choice for 20 topics is close to the automated proposition. This is described as the step 6 in the list of steps for DNMF modelling.

After building the models and outputting the tracking of topics to a file we will post-process the output to produce a row-oriented table optimized for a Tableau visualization. This step of post-processing is shown in Jupyter notebook 3.9.

It is notable that for each time window it is possible for the dynamic topic to be a combination of many time window topics. This option is taken into account by using the *opt* variable in the notebook.

```
# Step 1: Preprocessing
python prep-text.py data/sample/* -o data --df 2 --tfidf --norm

# Step 2: Window topic modelling
python find-window-topics.py data/*.pkl -k 20 -o data/out

# Step 3: Dynamic topic modelling
python find-dynamic-topics.py data/out/*.pkl -k 20 -o data/out/dynamic

# Step 4: Display topics
python display-topics.py data/out/dynamic/dynamictopics_k20.pkl

# Step 5: Track dynamic topics, pipe to file
python track-dynamic-topics.py --top 1000 --long \
    data/out/dynamic/dynamictopics_k20.pkl data/out/*.pkl | tee track.log

# Step 6: Automatically choosing the best number of topics
python prep-word2vec.py data/sample -o data/adv-out -m sg

python find-window-topics.py data/*.pkl -k 5,25 -o data/adv-out \
    -m data/adv-out/w2v-model.bin -w selected.csv

python find-dynamic-topics.py data/adv-out/*.pkl -k 5,25 \
    -o data/adv-out/dynamic -m data/adv-out/w2v-model.bin
```

Listing 3. *Steps used to run the DNMF implementation by Greene and Cross [34].*

```
[ ]: import json
     from pathlib import Path
```

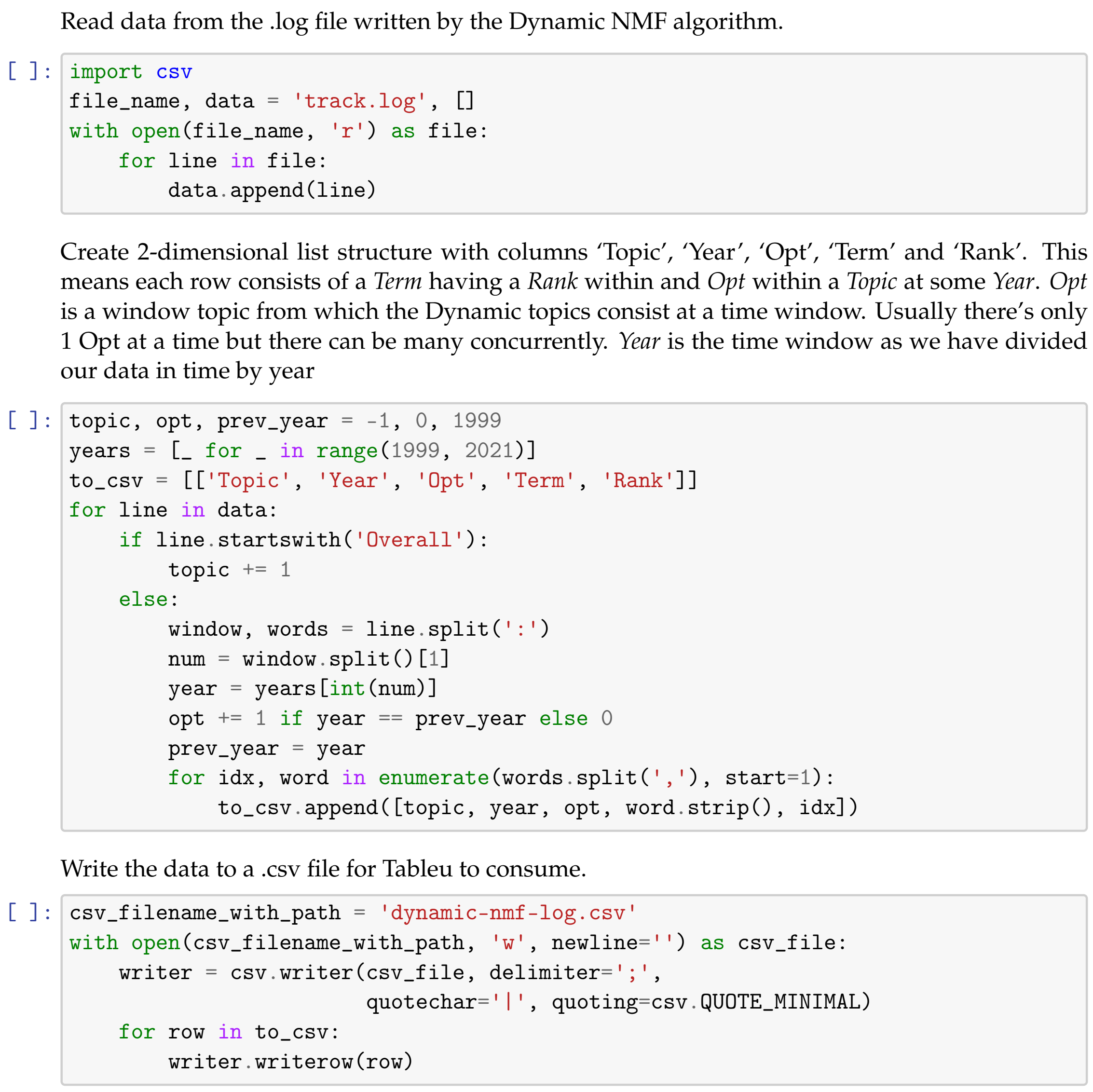
Load the data. Change all 'dc.date.issued' to year format only yyyy (some are in yyyy-mm-dd). Sort the data by the 'dc.date.issued'.

```
[ ]: data_file_name_with_path = 'masters_abstract_en_lemma_nouns_only.json'
     with open(file=data_file_name_with_path, mode='r') as file:
         data = json.load(fp=file)

     for d in data:
         d['dc.date.issued'] = str(d['dc.date.issued']).split('-')[0]

     data = sorted(data, key = lambda di: di['dc.date.issued'])
```

Write the abstract data to a folder structure the Dynamic NMF algorithm expects. For example one document to a file data/sample/2000/1.txt.

```
[ ]: p = Path('/data/sample')

     for idx, d_ in enumerate(data):
         folder = str(d_['dc.date.issued'])
         year_folder = p / folder
         year_folder.mkdir(exist_ok=True)
         with open(file=(year_folder / f'{idx}.txt'), mode='w') as file:
             file.write(d_['abstract_lemma_nouns_only'])
```

Figure 3.8. *Jupyter notebook showing the preprocessing for the DNMF algorithm.*



Read data from the .log file written by the Dynamic NMF algorithm.

```
[ ]: import csv
     file_name, data = 'track.log', []
     with open(file_name, 'r') as file:
         for line in file:
             data.append(line)
```

Create 2-dimensional list structure with columns 'Topic', 'Year', 'Opt', 'Term' and 'Rank'. This means each row consists of a *Term* having a *Rank* within and *Opt* within a *Topic* at some *Year*. *Opt* is a window topic from which the Dynamic topics consist at a time window. Usually there's only 1 Opt at a time but there can be many concurrently. *Year* is the time window as we have divided our data in time by year

```
[ ]: topic, opt, prev_year = -1, 0, 1999
     years = [_ for _ in range(1999, 2021)]
     to_csv = [['Topic', 'Year', 'Opt', 'Term', 'Rank']]
     for line in data:
         if line.startswith('Overall'):
             topic += 1
         else:
             window, words = line.split(':')
             num = window.split()[1]
             year = years[int(num)]
             opt += 1 if year == prev_year else 0
             prev_year = year
             for idx, word in enumerate(words.split(','), start=1):
                 to_csv.append([topic, year, opt, word.strip(), idx])
```

Write the data to a .csv file for Tableu to consume.

```
[ ]: csv_filename_with_path = 'dynamic-nmf-log.csv'
     with open(csv_filename_with_path, 'w', newline='') as csv_file:
         writer = csv.writer(csv_file, delimiter=';',
                             quotechar='|', quoting=csv.QUOTE_MINIMAL)
         for row in to_csv:
             writer.writerow(row)
```

Figure 3.9. *Jupyter notebook showing the post-processing step for the DNMF output.*

4 RESULTS AND ANALYSIS

This chapter is divided into three sections each dedicated to one research question. In Section 4.1 we discuss the results of the database comparison. Next is Section 4.2 where we review and discuss the results of topic modelling over time using DTM and DNMF. Finally is Section 4.3 in which we propose a data pipeline to build the automation of the topic modelling process of Trepo repository upon.

4.1 Comparison between PostgreSQL and MongoDB

Our first research question was *What is the best data storing solution for the data set from the perspective of further consumption?* We chose to compare two database management systems: one RDBMS and one document store that falls to the category of NoSQL databases.

The results of the comparison are visible in Figure 4.1. In the figure on x-axis is the database and the method of execution mapped to colors. The x-axis is divided by the target field: *Array* is a list e.g. *["house", "house", "child"]*. *JSON* is a list of dictionaries e.g. *[{"word": "house", "count": 2}, {"word": "child", "count": 1}]*. On y-axis is the average time of 5 runs.

When the query was targeted to the Array field the fastest method was PostgreSQL array functions with an average of 5,5 seconds. By far the worst performer was the MongoDB map-reduce functionality with the average of 203,9 seconds. This method is visible on the left in Figure 4.1. On the right in the same figure we can see the results when the queries are targeted to the JSON field. PostgreSQL has dedicated functions for handling arrays and json so we only used each dedicated method for the field (column) type it is meant to be used with. Again the Postgres dedicated functions gave the best result with an average of 10,3 seconds and the MongoDB map-reduce came last with 233,1 seconds and a huge gap to others.

In our research are only using the standalone versions of the DBMS' and we did not explicitly index any fields. A cluster of database instances and indexed fields might give different results. MongoDB natively supports sharded clusters to boost performance with large data sets and high throughput applications [74]. MongoDB map-reduce functionality is especially designed for sharded clusters and performs badly with a standalone instance. PostgreSQL community also has added a support for partitioning tables since the version 10 [86]. In addition there are many separate forks of Postgres implementing

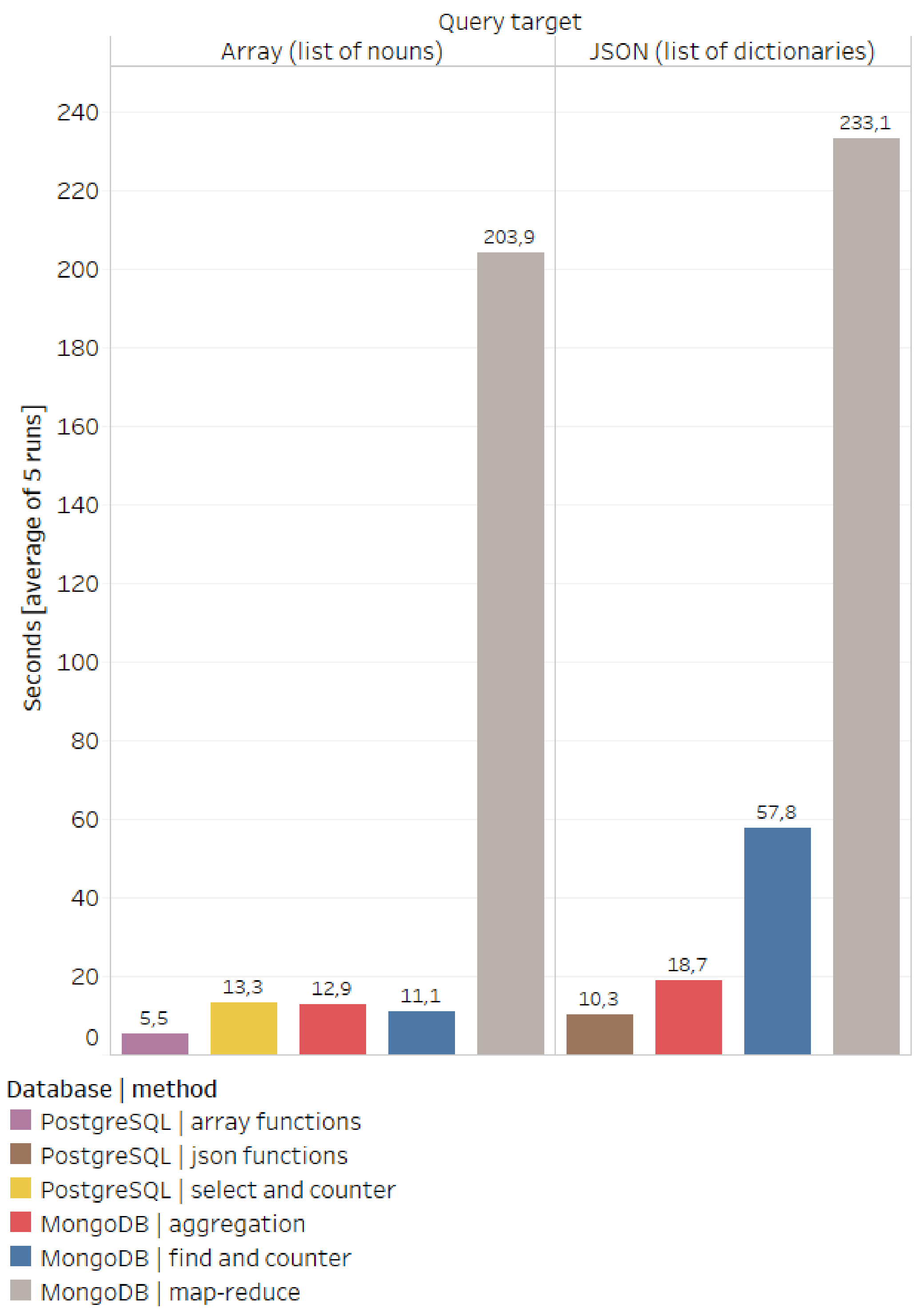


Figure 4.1. *Results of the comparison between database systems.*

sharding to help with scaling out. PostgreSQL sharding is more recent and it might be so that MongoDB has more mature an implementation for this.

Also it is notable that while the best results were achieved with PostgreSQL the aggregation pipeline of the MongoDB did well on both query targets. If the application the database is intended to be used with is not very time critical MongoDB with the aggregation pipeline seems like a valid choice also. That being said if we want to use a single instance database system with out-of-the-box settings in a time critical application for this type of data, our comparison indicates PostgreSQL to be a better performing option.

4.2 Topic modelling over time

This section focuses on the the third research question we stated as *What is the most suitable algorithmic approach to topic modelling our data set over time?* First we modelled topics over the entire data set using LDA and NMF algorithms from scikit-learn [91]. This was done without paying attention to the publication years of the documents, and is covered in Subsection 4.2.1. The topic modelling over time using DTM [16, 33] is discussed in Subsection 4.2.2. Then we cover the results of modelling with Dynamic NMF [34, 43] in Subsection 4.2.3. Finally we compare the results of the two approaches in Subsection 4.2.4.

4.2.1 Topic modelling with LDA and NMF from sklearn

For the initial testing of Latent Dirichlet Allocation and Non-negative Matrix Factorization we used the algorithms offered by the sklearn library. We derived 20 topics with each and the purpose was to see if they give comparable results. Both of the algorithms were trained by using the default parameters changing only the amount of topics to 20. The number of topics was chosen by using subjective judgement, it is probably not the best possible number to describe the data. The main idea behind this number was to use enough topics to find some clearly separating ones and also to keep the number low enough for the relatively small data set.

The Jupyter notebook used to derive topics with LDA is found as Appendix C.1 and the notebook for the NMF model is added as Appendix C.2.

Table 4.1. *20 topics inferred by the sklearn Latent Dirichlet Allocation model*

Rank	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4
1	cell	market	network	company	system
2	coating	company	distribution	process	software
3	surface	business	fault	study	model
4	tissue	strategy	text	research	design
5	sample	model	operator	customer	process

Rank	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9
1	analysis	camera	child	system	game
2	policy	concentration	family	performance	video

Table 4.1 continued from previous page

3	community	algorithm	parent	network	player
4	society	ph	support	sensor	medium
5	state	oxidation	life	signal	content

Rank	**Topic 10**	**Topic 11**	**Topic 12**	**Topic 13**	**Topic 14**
1	image	power	system	innovation	education
2	method	energy	cloud	organization	student
3	gene	measurement	energy	management	research
4	cancer	system	data	job	teacher
5	protein	simulation	service	music	university

Rank	**Topic 15**	**Topic 16**	**Topic 17**	**Topic 18**	**Topic 19**
1	user	health	material	policy	building
2	application	patient	water	language	work
3	system	care	temperature	immigrant	construction
4	design	group	structure	vietnam	plan
5	web	woman	sample	migration	site

The top 5 terms for each derived topic are represented in Table 4.1 for LDA and in Table 4.2 for the NMF. We have taken the freedom to remove some general words present in many topics, such as *study* and *research*, by hand to show more character in five terms. It is clearly visible that both of the algoritmns have inferred information technology and natural sciences dominant topics. The amount of topics related to fields of technology or natural sciences is 12 (Topics 0, 2, 4, 6, 8, 9, 10, 11, 12, 15, 17) for LDA and 12 for NMF (Topics 0, 3, 4, 5, 6, 8, 12, 13, 14, 15, 17, 18). Both of the algorithms have derived some economics related topics (LDA 3, 13; NMF 1, 19), built environment (LDA 19; NMF 10) and some topics related to social sciences and humanistic fields of study. Still given the size of e.g. the Faculty of Social Sciences with 3000 students and 430 employees [96] we feel they are likely to be underrepresented here. Why do these fields have underrepresentation then? Maybe the Master's theses they produce are mainly written in some other language than English. Maybe the number of topics derived should be higher to catch those fields better. Or maybe the parameters of the algorithms need adjustment.

Table 4.2. *20 topics inferred by the sklearn Non-negative Matrix Factorization model*

Rank	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4
1	system	customer	policy	user	cell
2	robot	business	eu	experience	gene
3	control	company	discourse	interface	cancer
4	implementation	value	state	interaction	protein
5	automation	market	country	usability	prostate

Rank	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9
1	service	game	model	material	child
2	cloud	video	simulation	property	care
3	customer	player	business	surface	parent
4	provider	industry	machine	temperature	health
5	delivery	gamification	result	coating	family

Rank	Topic 10	Topic 11	Topic 12	Topic 13	Topic 14
1	building	student	software	power	web
2	design	education	testing	voltage	iot
3	city	teacher	project	converter	technology
4	housing	school	automation	grid	platform
5	space	university	integration	frequency	sensor

Rank	Topic 15	Topic 16	Topic 17	Topic 18	Topic 19
1	image	product	energy	network	management
2	algorithm	process	consumption	distribution	project
3	learning	company	electricity	lte	organization
4	feature	manufacturing	harvesting	fault	process
5	classification	cost	battery	traffic	risk

When comparing the quality of the derived topics by LDA and NMF we cannot see a clear difference. None of the topics have strikingly exciting words in top ranks, but we feel the top words give us some overall understanding on the subject of practically every topic. In fact only from Topic 7 of the NMF model we are unable to get hold of the subject domain. Then again we feel that in Topic 18 of the NMF model 4.2 we have the only interestinly specialized word in the whole set of top words of both models: *lte*. Given the context of the topic in general this seems to refer to a wireless broadband communication standard called LTE (Long Term Evolution). Based on this comparison we believe it is worthwhile to proceed to modelling topics over time with both algorithms.

4.2.2 Dynamic Topic Models

Having the option to utilize the *gensim* wrapper [39] for the Dynamic Topic Models (DTM) implementation [33] was a nice feature to have. Gensim made it possible for us to implement all of our code in Python. The functionality the wrapper api provides was sufficient for our task. We had to preprocess the data to bend it to the format the wrapper ingests, which resulted in a 3.8 Mb json file. Deriving 20 topics from our data took 28 minutes and 43 seconds on our research environment which we described earliear in Table 3.3.

DTM produces constant topics meaning that in each time slice the number of topics remain the same. As it is a probabilistic model and the topic-term proportions are drawn from a Dirichlet distribution all the terms are present in all of the topics, only their probabilities change [18]. Also the probabilities for each term change within each topic in different time slices. We have tried to present the most influential terms for each topic across the time slices in Table 4.3. This is achieved by summing up the probabilities for each word within a topic in all of the time slices. The result is very similar to what we saw earlier when modelling the data set using sklearn's LDA and NMF algorithms. The topics are technology and natural science heavy. We believe it is possible to recognize many topics that are basically identical to those inferred by the sklearn algorithms. There are some which seem to be showing a different point of view to a subject. For example Topic 8 seems to be labour or workplace related but the perspective it gives with top terms *safety, worker, knowledge, job, report* points to occupational safety and health. Also Topic 14 looks like telecommunications related, but the top terms *signal, communication, channel, frequency, receiver* possibly point to more abstract concepts than LDA Topic 8 with top terms *system, preformance, network, sensor, signal* shown in Table 4.1. These differences in capturing a tone are small and they may not even exist, but there is a possibility that DTM yields at least comparable if not better results than sklearn models when used this way.

Table 4.3. *Terms describing the DTM model of 20 topics. Terms are chosen by using the cumulative probability from all time slices by topic.*

Rank	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4
1	society	gene	health	cell	network
2	discourse	protein	woman	tissue	system

Table 4.3 continued from previous page

3	identity	cell	child	sample	energy
4	conflict	mutation	age	particle	production
5	state	disease	intervention	treatment	plant

Rank	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9
1	patient	area	software	safety	policy
2	cancer	sensor	application	worker	country
3	eye	radio	image	knowledge	EU
4	voice	tag	design	job	Finland
5	gaze	antenna	test	report	government

Rank	Topic 10	Topic 11	Topic 12	Topic 13	Topic 14
1	family	education	measurement	power	signal
2	care	study	surface	control	communication
3	child	school	material	voltage	channel
4	support	research	temperature	output	frequency
5	health	university	layer	wind	receiver

Rank	Topic 15	Topic 16	Topic 17	Topic 18	Topic 19
1	security	company	user	organization	community
2	market	management	application	innovation	child
3	risk	service	game	accounting	group
4	price	business	device	capital	participation
5	migration	product	web	stakeholder	role

Next let's take a look at the topic evolution more closely. We surveyed the evolution of terms inside topics and use line plot visualization implemented with Tableau. The selected terms shown here are a result of subjectively finding something of interest when examining the topic evolution. In the figures we have the years on the x-axis. On the y-axis we have the probability for that term in that topic in the given time slice. It's important to notice the scale of the probabilities: it differs from one topic to another and therefore the figures are not comparable with each other. The terms are chosen so that they have a relatively strong presence in the topic they are picked to represent here.

In Figure 4.2 we have some extracted terms from Topic 0. This topic seems to have something to do with conflicts, international relationships and maybe also on how societies are interpreting them. Maybe it is a historical topic or a journalistic one. There are many possibilities here, but as the terms related to Russia and politics are loosing meaning it doesn't seem to be mainly about those subjects; at least not anymore. Because terms *Syria* and *geopolitic* have flat zero probabilities this topic surely doesn't have anything to do with geopolitics of the Middle East. All in all this topic seems difficult to interpret because even the strong terms like conflict reach only 0,01 probability at most.

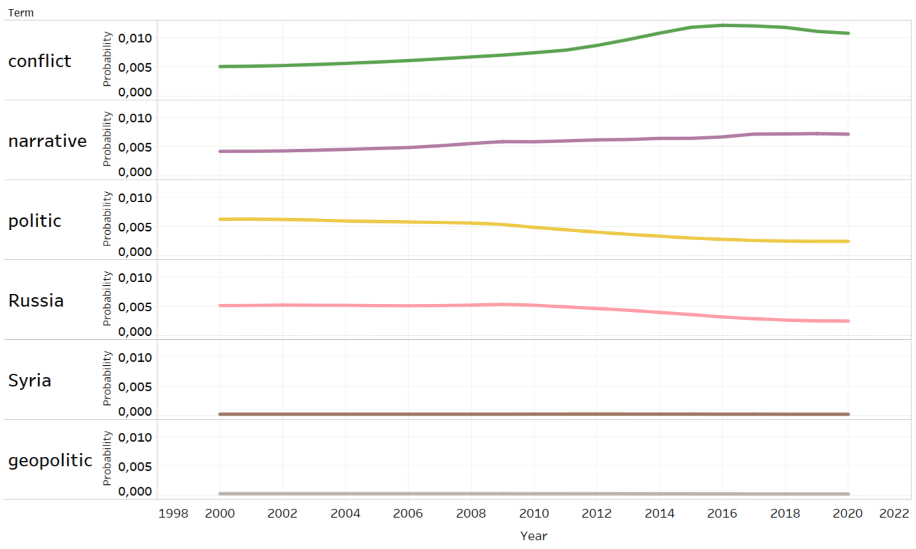


Figure 4.2. *DTM Topic 0 evolution.*

From Topic 6 we have picked four terms of interest. The topic seems to have something to do with hardware for radio waves and frequency ranges. Words representing this topic include *technology, tag, sensor, radio and antenna*. Interestingly the significance of the term *radio* has constantly decreased as can be seen in Figure 4.3. Radio waves are widely used in modern technology, so this might not be the reason here. At the same time the increase of terms *RDIF* and *tag* might point to Radio-frequency identification which are used in biometric passports for example. Also *sensor* has gained influence rapidly.

Topic 7 seems to have some interesting terms in it. This topic in general is likely about developing software. The increase in probability for the terms *cloud*, *robot* and *machine* seem to point to machine learning, artificial intelligence and cloud computing. The shift in this topic seems to be towards AI related publications as seen in Figure 4.4.

Even though it is difficult to interpret Topic 9, it seems to be about European Union and integration policies to some extend. The increase in use of term *Vietnam* is interesting and surprising in Figure 4.5. Maybe it has to do with the grown number of Vietnamese immigrants in Finland in the 2010's.

Our inferred Topic 13 is about electricity, batteries, power grids. Maybe the increase in use of battery as visible in Figure 4.6 is related to electric cars? It is curious that the term

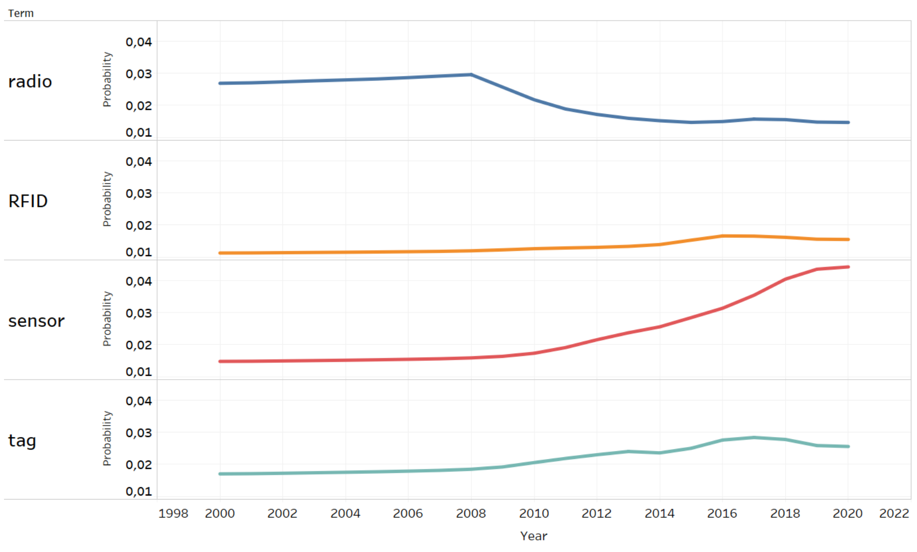


Figure 4.3. *DTM Topic 6 evolution.*

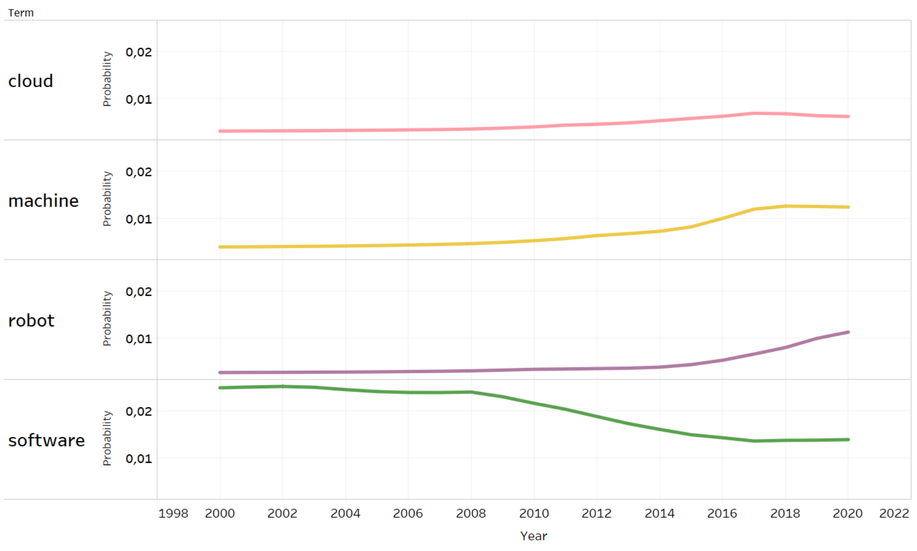


Figure 4.4. *DTM Topic 7 evolution.*

wind has come down even though renewable energy is a hot topic. This can also point to topic being about something else but energy sources. It should be certain that wind is referring to a noun, because we only use nouns for topic modelling here, and therefore it can't point to *winding* something as a verb.

What is interesting about Topic 15 is the increase in importance of the term *ecosystem*, which can point to presently increased concerns about climate change. This is visualized in Figure 4.7 This topic in general seems to have something to do with welfare society.

It seems that the evolution of terms inside the topics swifts beautifully but we have not captured many truly interesting changes in our examples. We did try to find terms with high variance to reveal interesting points. An example of such a *term-variance histogram* is shown in Figure 4.8. The relatively small and imbalanced data set (see Figure 3.1) is possibly causing problems to find interesting term evolution inside the topics.

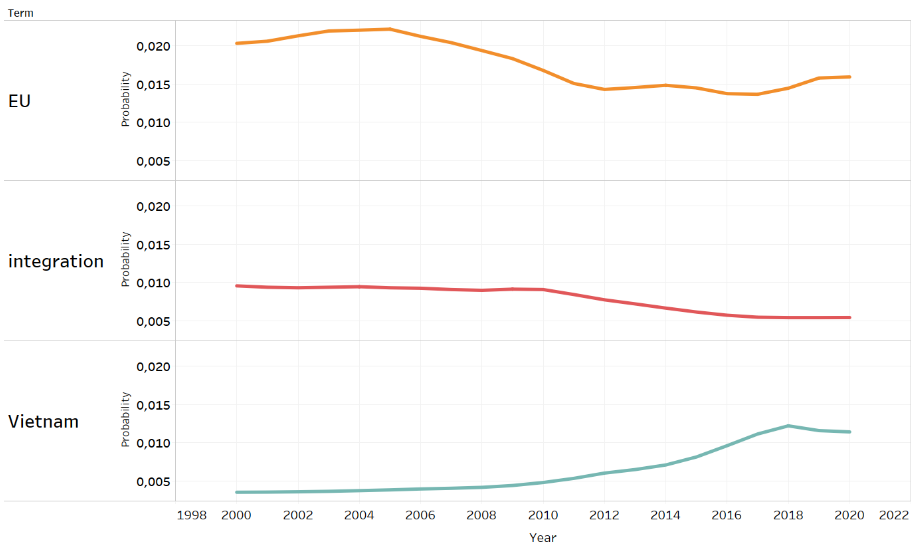


Figure 4.5. *DTM Topic 9 evolution.*

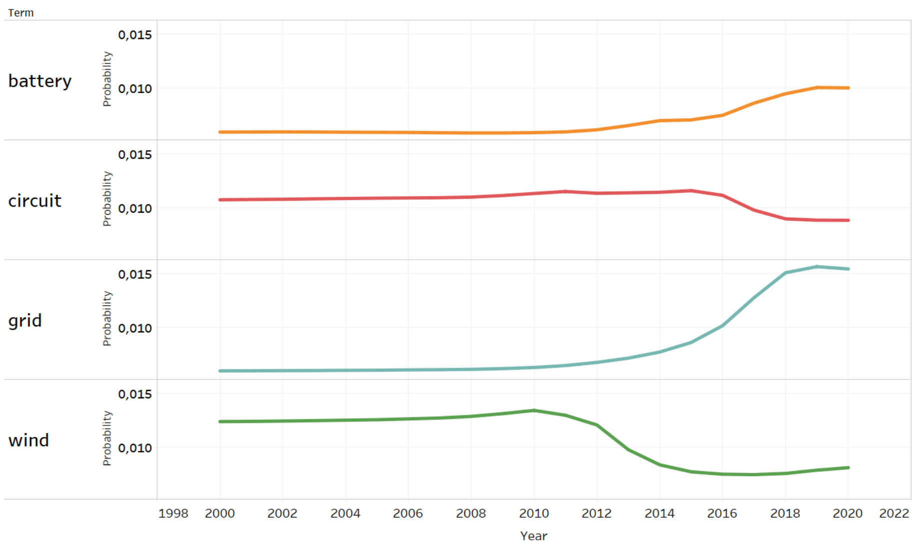


Figure 4.6. *DTM Topic 13 evolution.*

4.2.3 Dynamic NMF

The implementation we used for Dynamic NMF produced a dynamic topic model over all of the time windows in the data. The top words for the topics are shown in Table 4.4. Basically the results are very much in line with what we have seen before using other algorithms and implementations. The topics are again very technology weighted. Not many things come up as surprises here but we can find a few terms that seem more specific and interesting. In Topic 2 which seems to be about human biology or medicine there are terms *hydrogel* and *proliferation* present. These terms are perhaps not so generic as the terms seen in top positions for our topics tend to be. Maybe these terms shift the topic more towards cellular biology or research related to cancer? In Topic 19 the term *mrna* looks to be interesting. As the topic seems to be about genetics research it is likely *mrna* is short of *messenger RNA* and fits the topic description well.

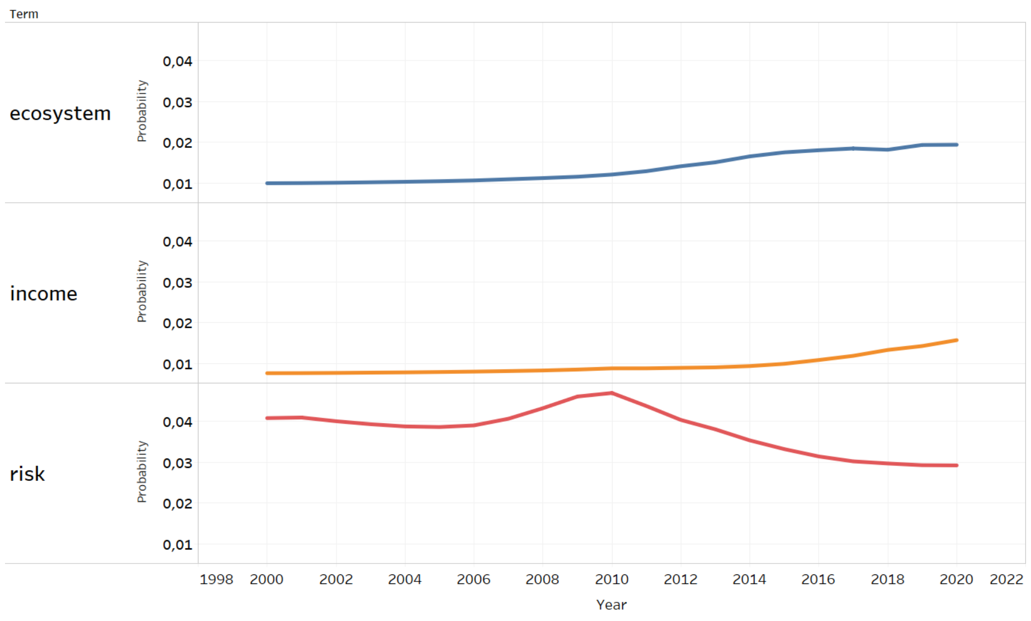


Figure 4.7. *DTM Topic 15 evolution.*

The DNMF algorithm also produced ranks for terms at each time window. We have again chosen terms from different topics and visualized their evolution.

Table 4.4. *Terms describing the DNMF model of 20 topics. Terms are chosen from the overall rank provided by the algorithm.*

Rank	**Topic 0**	**Topic 1**	**Topic 2**	**Topic 3**	**Topic 4**
1	society	software	cell	user	company
2	health	application	gene	interface	supplier
3	patient	test	cancer	experience	customer
4	woman	framework	hydrogel	interaction	product
5	questionnaire	engineering	proliferation	usability	market

Rank	**Topic 5**	**Topic 6**	**Topic 7**	**Topic 8**	**Topic 9**
1	power	system	child	sensor	building
2	voltage	implementation	care	protocol	architecture
3	converter	manufacturing	health	lte	city
4	grid	automation	parent	distribution	space
5	signal	requirement	family	communication	environment

Rank	**Topic 10**	**Topic 11**	**Topic 12**	**Topic 13**	**Topic 14**

Table 4.4 continued from previous page

1	game	service	material	model	image
2	player	customer	property	simulation	algorithm
3	video	quality	surface	process	video
4	experience	cloud	temperature	value	classification
5	control	web	coating	characteristic	feature

Rank	**Topic 15**	**Topic 16**	**Topic 17**	**Topic 18**	**Topic 19**
1	education	policy	energy	project	protein
2	student	state	electricity	management	gene
3	university	eu	consumption	process	tissue
4	teacher	discourse	water	organization	mrna
5	institution	russia	efficiency	risk	mutation

Looking at Topic 1 on Table 4.4 it seems quite clearly to be about software engineering. From Figure 4.9 we can spot some interesting evolution of terms inside this topic. If we look at terms *Java*, *JavaScript* and *Python* which all point to programming languages we can see that only Java has been around in this topic the whole span of years. One reason for the late emerge of JavaScript could be that there is another topic for web development, Topic 11 perhaps? JavaScript came to this topic only as recently as in 2015 but has had a high rank since then. For Python the reason for late appearance might be its use mainly in scripting even though it is well utilized in software engineering also. Python has increased its importance all the time since showing up in 2011. Looking at the terms referring to application frameworks we can see that *Spring* first emerged in 2008 an has climbed in rank since then. Curiously it has come down rapidly since year 2017 which is problematic for our interpretation of this topic's domain: Spring is nowadays at the peak of its popularity as a framework to build enterprise applications with. Python web framework *Flask* is present in this topic and chosen as another example of software development frameworks. It emerged in year 2013 and vanished again in 2018. The timing fits with the timing of Python's emerge but clearly Flask is not an important word in this topic. Lastly we take a look at some terms referring to modern technologies. *cloud* is present making appearance at the same time as one of the biggest public cloud providers *Amazon*. Cloud has actually been very highly ranked a term in recent years. This is fitting well with current trends of software development. The term *Azure* refers to another big public cloud provider but it has solely been in this topic in the year 2019. The term *container* points to a package of software that has lately become a standard for software development and deployment. A container image is a portable software package containing everything needed to run the application such as code, runtime, system tools,

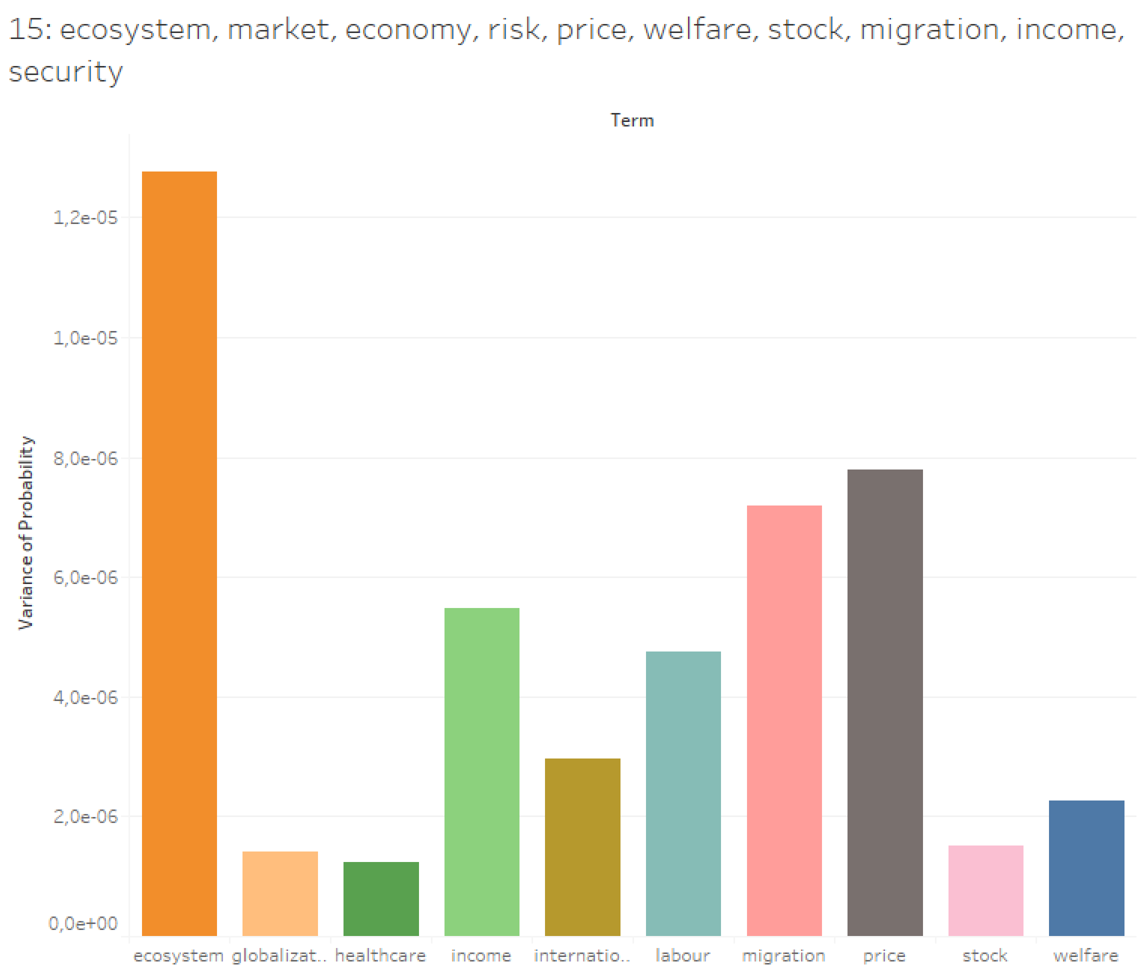


Figure 4.8. *DTM Topic 15 variance.*

system libraries etc. A model of containerized application is visible as Appendix B.1. In this topic *container* appeared in year 2016 but for some reason it has gone up and down in ranks since. This is also curious behaviour because nowadays containerization is the de facto industry standard.

Earlier we speculated that Topic 2 was about cellular biology or research related to cancer. Now having seen the evolution of some key terms inside this topic in Figure 4.10 it is more evident that the topic revolves around cancer quite a lot: both *prostate* and *breast* are well ranked. These terms seem likely to refer to two strongly gender dependent types of cancer that are also common among the human population. The term *hydrogel* had a very high rank in the overall term-importance, as seen in Table 4.4. Hydrogels are polymer networks extensively swollen with water [1]. The possible use of them in cancer or cellular biology related research is completely beyond the scope of this study. What we find curious is that the term *hydrogel* seems to be present in this topic only between the years 2015 and 2017. It is a top ranked term in 2016 and 2017 but its high importance in the overall ranking is surprising given this brief time span it appears at all.

In the evolution of our Topic 3 we have tried to find some terms that could have been trending at slightly different years. This topic is visualized in Figure 4.11 and the subject

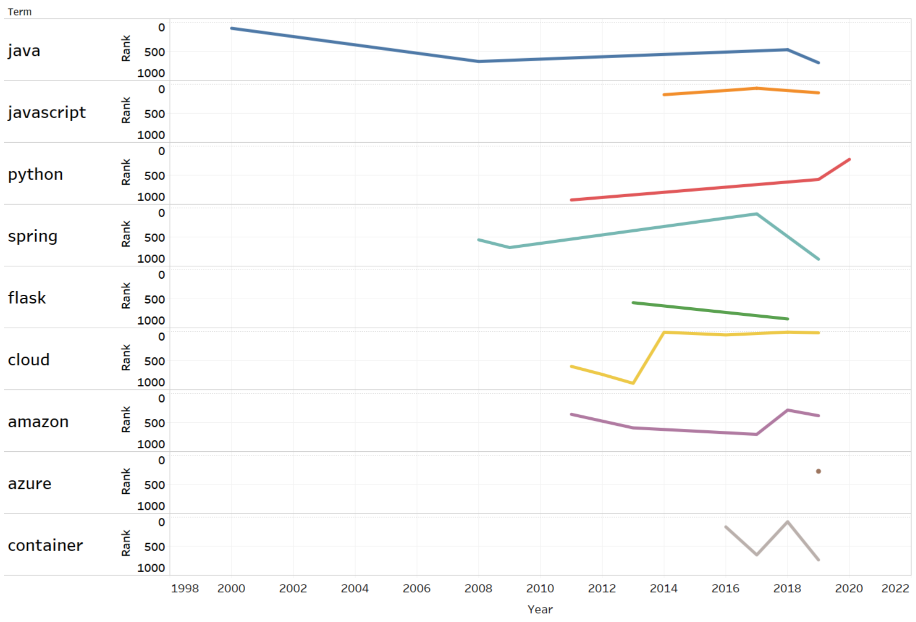


Figure 4.9. *DNMF Topic 1 evolution. The words are plotted only for the years in which they appear in the topic.*

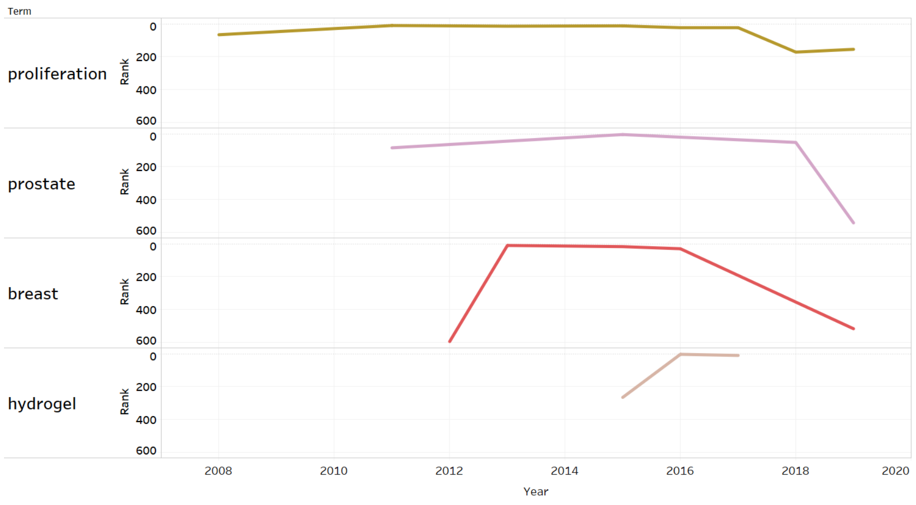


Figure 4.10. *DNMF Topic 2 evolution. The words are plotted only for the years in which they appear in the topic.*

of this topic seems to be user experience, user interfaces and usability. The terms *interface* and *usability* are high in the ranks from the beginning of the 2000s but the term *accessibility* emerges only in the year 2011. This seems to be indicating that accessibility in the user interfaces has gained importance only lately and has then been climbing in rank until today. Screen reader programs have been around for quite a long time but perhaps the late emergence of accessibility refers here to some more recent technologies like *Optical Character Recognition (OCR)*. Or maybe it refers to late realizations in web development of e.g. the importance of actually including a descriptive *alt* attribute to an *<img>* tag in a HTML page.

We also wanted to see if the terms *smartphone* and *touchscreen* appear later than the term *phone* and this was true. Regarding the different operating systems we chose to see how terms *Windows* and *Linux* compare. There was nothing really interesting there but some oddity in the late appearance of both of the terms. More interestingly we wanted to see the evolution of operating systems designed for mobile devices and chose to see *Android* and *Symbian*. As expected Android emerged later and is quite highly ranked today. At first glance Symbian seems to be peculiarly appearing between the years 2008 and 2012. This feels out of the place because Nokia used it extensively with their *S60* user interface already in the early 2000s. Perhaps this topic can't reach the use of Symbian in those days for some reason. In the end the timeline seen here with the evolution of Symbian emerging in 2008 and then disappearing as Android occupies the space since 2012 fits quite well with what has happened. In year 2008 Nokia purchases the Symbian Ltd and then published the Symbian 3 OS in year 2010. Then in year 2011 Nokia announced a partnership with Microsoft and plans to use Windows Phone as their main mobile operating system hence. Already in the end of the year 2010 Android had overtaken Symbian as the most popular mobile OS. So by the year 2012 Symbian had become obsolete in a sense and the plotted lines in Figure 4.11 are able to capture this. As an extra we have chosen the term *Nokia* to see how it plays with the evolution of Symbian inside this topic. It might be fitting the time line being hyped in the year 2010. Still it is unclear and unsettling even why Nokia is not appearing in the 2000s at all. For some reason this particular topic is not able to capture Nokia's importance in the end of their golden years.

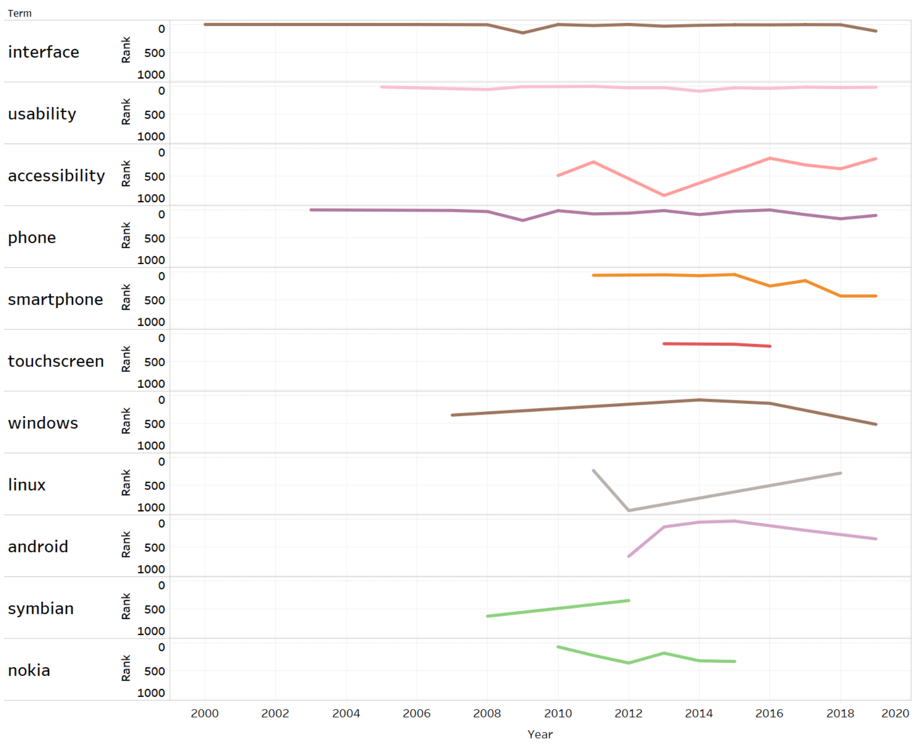


Figure 4.11. *DNMF Topic 3 evolution. The words are plotted only for the years in which they appear in the topic.*

The reason why we have chosen to visualize some terms from Topic 8 in Figure 4.12 is the year the timeline for this topic begins. Topic 8 emerges as a topic in the year 2005 with top term being *architecture*. The top term of this topic overall is *sensor* and that term has its first appearance in the year 2010 in the scope of this topic as is seen in the figure. Even though we did choose to infer 20 topics overall the DNMF algorithm behaves so that not all of those 20 topics exist in all of the time windows the data covers. The other terms plotted here are *ber* and *lte* which seems to point to *Bit Error Rate (BER)* and *Long Term Evolution (LTE)* in this order. Both of which along with the term *sensor* are revealing the domain of this topic to be somewhere near communication technologies.

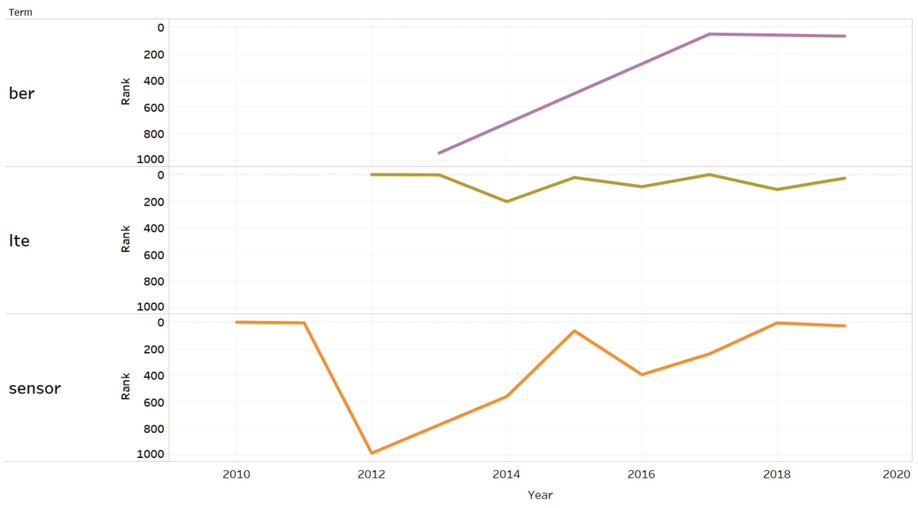


Figure 4.12. *DNMF Topic 8 evolution. The words are plotted only for the years in which they appear in the topic.*

Here is some interesting evolution in Topic 14 visualized in Figure 4.13. This topic seems to consist of publications in the fields of signal processing and pattern matching algorithms. The term *image* is very important to this topic ever since the year 2002. The term *classification* is also important but appeared only in the year 2011. Another term related to classification is *machine*. It could have non-machine-learning related meanings inside this topic also but because we have not included bigrams to our model we are not able to tell the difference. Given what the topic seems to be about it seems fairly safe to say it mostly refers to machine learning at least in the recent years. It also is a top ranked term since the year 2016. Lastly we have highlighted the term *cnn* which is likely to be an abbreviation of *convolutional neural network*. We discussed neural networks in Subsection *2.2.1 Deep learning for NLP* and a CNN is branch of deep neural networks. It fits here well since a common application of CNN is analyzing visual imagery. It seems that deep learning has strongly affected this topic since the year 2017.

	Terms representing the Window Topic					
WT 1	tool	process	information	BI	visualization	metric
WT 2	investment	cost	manufacturing	location	production	transportation
WT 3	customer	startup	market	sale	delivery	branding

Table 4.5. *Combination of window topics that form a dynamic topic in a time window representing the year 2015.*

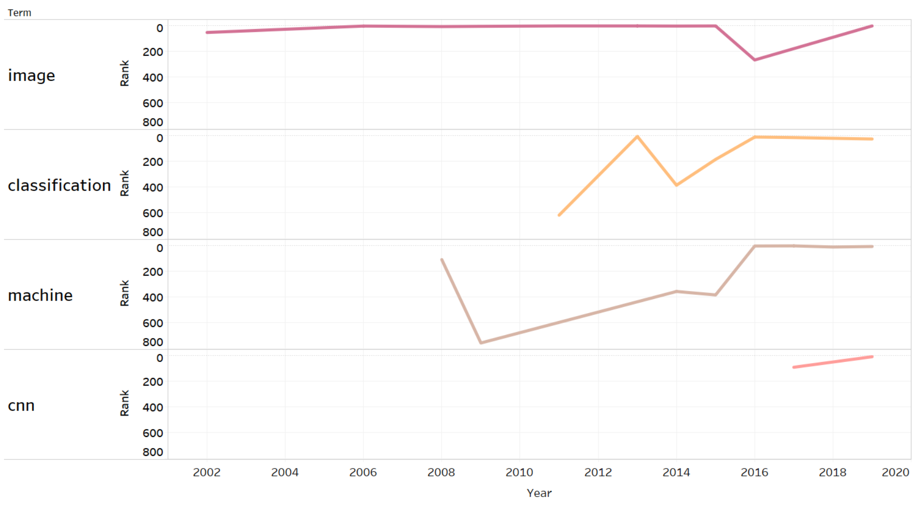


Figure 4.13. *DNMF Topic 14 evolution. The words are plotted only for the years in which they appear in the topic.*

Having plotted selected terms from a few topics we have discovered interesting and also curious behaviour. Finding an explanation to e.g. unexpected complete disappearances of terms inside a topic evolution is a subject for further study. The DNMF algorithm has a property of being able to capture a dynamic topic, at some given time window, consisting of several merged window topics. Using a *Sankey diagram* it might be possible to capture the flow of window topics merging and demerging inside a dynamic topic. In Table 4.5 we have illustrated our model's take on Dynamic Topic 4 at the time window pointing to the year 2015. The dynamic topic is consisting of many merged window topics. According to Table 4.4 the most influential terms for Topic 4 are *company*, *supplier*, *customer*, *product*, *market* and the domain of the topic derived from there seems to be business studies. Looking at Table 4.5 Window Topic 1 (WT1) consists of terms that relate to collecting, measuring and analysing the business data. The term *BI* in the WT1 is an abbreviation of business intelligence. The WT2 seems to have the viewpoint of manufacturing products at as low a cost as possible. The WT3 comes in from the angle of marketing and selling the products. This combination makes sense and it is also extremely interesting to see that the DNMF model has been able to capture these 3 different perspectives coming from 3 separate window topics. It is notable that most of the time the dynamic topics consist of only one time window topic at the time. Having a combination of 3 window topics is quite rare in our data. In this case it perhaps indicates that the year 2015 was relatively business heavy in the issued scientific publications.

The DNMF algorithm's implementation has an added feature of inferring a suggestion for the best number of topics for the data [34]. We explored that feature and illustrate the results on Table 4.6. As parameters we gave the algorithm a lower limit of 5 topics and an upper limit of 25 topics. Basically all of the recommended numbers for window topics are well inside those limits. As we can see from the table the years 2000-2010 have generally lower recommended numbers of topics than the years 2011-2020. This is the result of the number of publications-by-year having increased significantly in the data set as we

Year	1st	2nd	3rd
2000	8	10	9
2001	9	8	7
2002	7	6	8
2003	7	10	9
2004	13	12	8
2005	23	5	22
2006	5	6	7
2007	5	25	6
2008	9	8	7
2009	10	12	16
2010	6	9	17
2011	23	20	18
2012	25	23	24
2013	19	22	21
2014	23	21	22
2015	24	23	22
2016	20	22	18
2017	23	22	21
2018	18	19	25
2019	22	21	18
2020	15	17	21
Dynamic	24	23	25

Table 4.6. *Top 3 suggestions for topic number by year given by the automatic recommending feature. The recommendation for the number of dynamic topics is at the end of the table.*

showed in Table 3.1. When we look at the suggestions for the number of dynamic topics in the end of Table 4.6 we interestingly see that the recommendations are 24, 23 and 25 in that order. This tells us that the automatic topic recommendation feature suggests clearly higher number of topics we have used in our research. Another take away here is that the the term evolution inside the topics might suffer greatly from the topic number being chosen very poorly for some time slices. Our corpus is highly imbalanced with respect to documents belonging to each time slice and it is difficult to find one topic number that suits all the time slices. The same problem can affect the DTM algorithm as well.

4.2.4 Algorithm comparison

Let's start by comparing the training speed of the algorithms. As we trained the models we also measured the time consumption of the training. You can see from Table 4.7 that Dynamic NMF is significantly faster than DTM. For DTM the average of three training runs was 28 minutes and 43 seconds (1722.93 s) and for DNMF 5 minutes and 27 seconds (327.13 s). It is evident this big a difference is an important factor in favor of DNMF. As the corpus consists of a little over 3000 documents and we are using only nouns from the abstracts, the size of the data set is very modest. When having a corpus of thousandfold in size, choosing to use all the words and bigrams or full text content will have an impact on the model creation speeds. It seems that at least in the case of DTM an interactive web app implementing on-demand training with tolerable waiting time is difficult to achieve. Parallelizing the training might reduce the time consumption and be possible to implement. After all the time slices are separate and thus could be processed at least partly concurrently.

Run	**DTM**	**Dynamic NMF**
1st	1711.45 s	326.63 s
2nd	1766.05 s	327.53 s
3rd	1691.28 s	327.25 s
Average	1722.93 s	327.13 s

Table 4.7. *Time consumption of training the DTM and DNMF models. Times for three separate runs and their average.*

When comparing the implementations for the two algorithms we feel that the DNMF implementation [34] is more versatile than the DTM implementation [33]. The reasons behind this conclusion include the fact that DNMF is capable of inferring merging and separating window topics inside the dynamic topics. With further study this feature could prove beneficial as our demonstration in Table 4.5 hints. Another indication of better versatility comes with the automatic recommendation feature for the number of topics. Both of these features can be engineered into DTM but with DNMF they come out of the box. With the DTM implementation we had to calculate the overall ranks for the terms by ourselves while this feature is built-in to the DNMF implementation. We are speaking of the ranks over all time slices (time windows) here.

On the other hand DTM has the gensim wrapper implemented and that gives us a standardised gensim interface to work with. The algorithm itself is still implemented with C++ making it significantly faster than a pure Python implementation would be. The DNMF implementation is purely Python which is bound to be slower because it is interpreted rather than compiled. In our case the data set was so small that the speed of the implementation was not a factor. Also there is a good possibility that because DNMF uses only linear algebra for computations it can compete in speed with DTM even when implemented in a slower language. Being purely implemented in Python the DNMF is easy to include in a larger Python software body. You don't have a need to install C++ dependencies and everything can be handled with a Python package installer such as *pip*.

The programming interface of the DNMF implementation is less standardised compared to the gensim wrapper for DTM but the source code is licensed under the *Apache 2.0*

[34] and thus any modifications are allowed [3]. Speaking of the licensing the original implementation of DTM in C++ is released under the *GNU GPL v2.0* [33] which allows for modification [42]. The gensim wrapper is licensed under the *GNU LGPL v2.1* [39] so that could be modified too as long as you disclose the source code of those modifications [68]. Licensing is not a decisive factor between the algorithms.

The fact that the DTM algorithm outputs probabilities while DNMF ouputs ranks plays in favor of DTM in our opinion. This is mainly because the probabilities incorporate more possibilities for interpreting the importance of words in topics and topics in corpus. From DTM we can often tell if the topic is full of *noise* without any strong terms that really define it. The ranks given by DNMF are discrete and bound to linearity and because of that the output of the algorithm doesn't have the same expression power as the output of DTM has. As an example let's say we have an arbitrary topic with four top words *summer, winter, spring, autumn*. DNMF then gives us overall ranks: *1. summer*, *2. winter*, *3. spring* and *4. autumn*. From this we could interpret that the topic is about *seasons* because we don't know how big a difference there is between the terms. But if the DTM gives us probabilities: *summer 0.87*, *winter 0.05*, *spring 0.04*, *autumn 0.04*. From this we can infer that the topic is not really about seasons but about summer.

Let's compare the the evolution for some terms in DTM and DNMF. Above in Figure 4.14 we have the evolution of terms *Android, JavaFX, JavaScript, Unity* within Topic 17 from DTM. This topic seems to be about web development and game development. Below in the same figure we have the terms *Android, JavaScript, Unity* from the DNMF Topic 10. That topic looks to be about video games. These topics are not completely comparable but the closest we were able to find. First if you look at the x-axis on both figures you can see that with the DTM it begins from the first year present in the data set as it does in all cases. The topics exist constantly throughout the time span present in the data and if a word is not present in a topic in some time slice the probability for that term is 0 in the time slice. In the case of NMF the x-axis begins from the year 2012. This is the first year the word *Android* appears in the topic. It could indicate that the topic appears in the year 2012 also but that's not the case here as Topic 10 first emerged in year 2010. In any case the possible emergence and disappearance of topics makes it more complicated to follow their evolution compared to the DTM topic evolution. In our opinion this is a clearly an advantage for DTM when wanting to keep things relatively simple. You can also see that the lines with DTM evolve smoother and the minimums and maximums are quite flat. In the case of NMF the local minimums and maximums are sharp very often. Both of the trends are repeating in our data. One reason for this is that the probabilities might evolve in slower rate than the ranks. Another reason is that between the first appearance and the last appearance of a term in a DNMF topic the term can disappear completely only to emerge again later. If a term is non-existent in some time window we don't have a rank for it at all at that moment. The upside for this is that there is a possibility to detect an important point in time for a term by looking at the moment it emerges the first time. For example in the case of *Unity* the probability within Topic 17 of DTM stays quite constant. If the term truly refers to the game engine, which we can't verify, this doesn't really tell

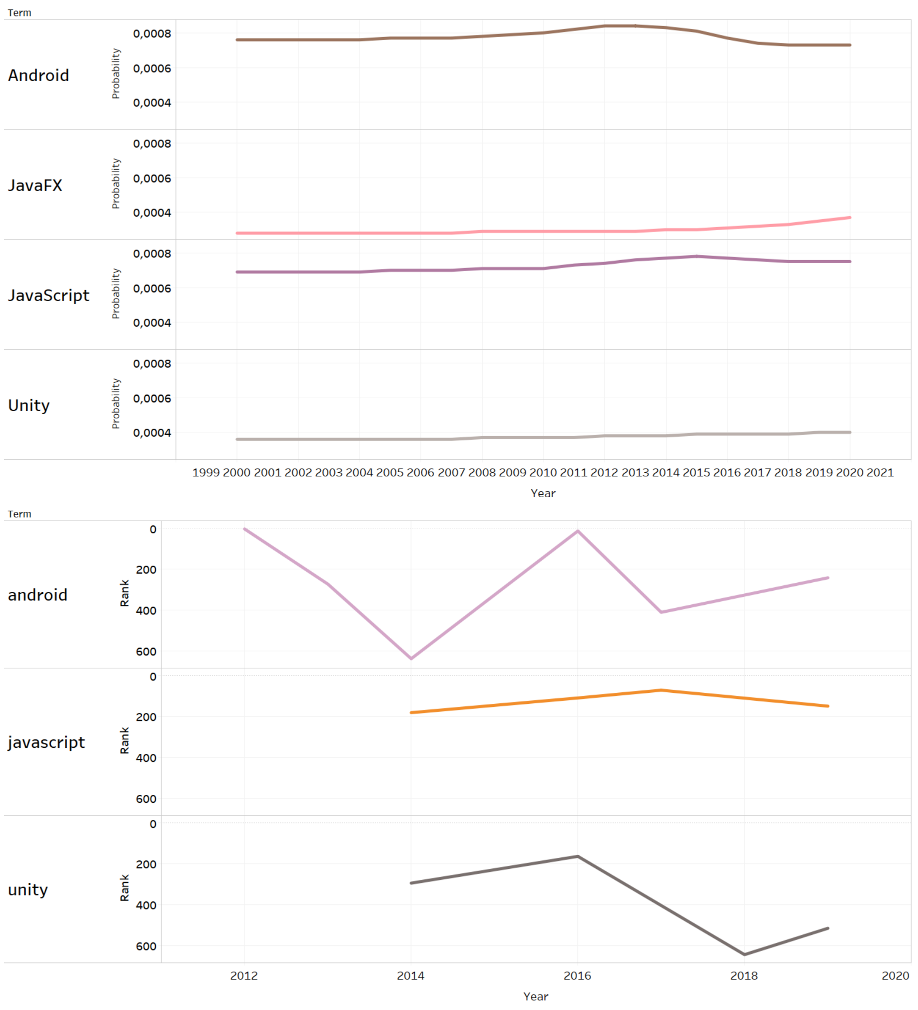


Figure 4.14. *Comparison 1 between evolution of similar words. Above DTM Topic 17 and below DNMF Topic 10. For DNMF the words are plotted only for the years in which they appear in the topic.*

us anything useful. On the other hand the appearance within Topic 10 of DNMF happens in the year 2014. Unity was first launched in 2005 but 2013 was important as Facebook then integrated an SDK for games using the Unity. Also 2015 was important as Unity 5 was released and that is the global maximum for the term in our plot. This is something DNMF possibly has captured but DTM defenitely has not.

Another comparison is between software development and programming related terms *Java, Python, Amazon, container*. In Figure 4.15 we have DTM Topic 7 plotted above and DNMF Topic 2 plotted below. As you can see the term Java is present in both models from the year 2000 and has decreased in importance. The other terms here behave quite differently. As the lines are basically flat within the DTM topic for Python, Amazon and container they have a large variance within the DNMF topic. We feel that the late appearance of all the three terms in DNMF actually reflect the real world events quite well since Python has become substantially more significant in the 2010s through the extensive use in data science. Also the public cloud technologies and container technologies are nowadays at the top of the heap in software industry. This behaviour

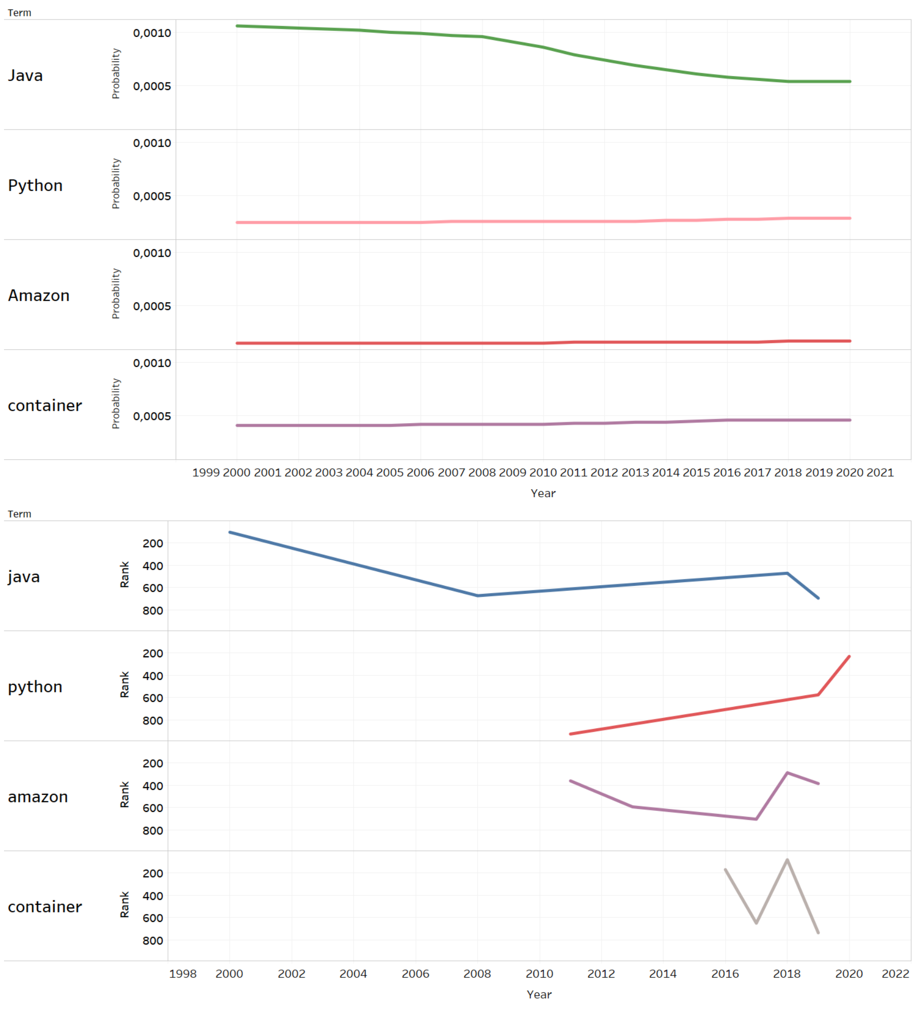


Figure 4.15. *Comparison 2 between evolution of similar words. Above DTM Topic 7 and below DNMF Topic 2. For DNMF the words are plotted only for the years in which they appear in the topic.*

is quite repeating in our data: the DTM curves tend to be flatter, smoother and more stable. This is not the case always as we have seen in Figures 4.3 and 4.6 where there is significant variance present with some terms. One possible reason for the repeating behaviour of flat curves is that the data set of 4018 abstracts we used is not big enough to capture interesting change with the DTM algorithm. Another possible reason are the clear imbalances of the data set. There are not enough publications from the 2000s to get a good result from those years. This distribution of publications by year was shown in Figure 3.1. Also there exists another imbalance which relates to the fields of study the data comes from. As we have seen in Figures 4.1, 4.2, 4.3 and 4.4 most of the topics are technology or hard-science related. This seems to indicate there are not enough publications from other fields of study to manifest as separate topics with the chosen topic number.

It's important to point out that we have not tried to find the best parameters for the algorithms but have mostly remained with the defaults. There definitely might be room for improvement by fine-tuning the parameters. Our research focused on a complete data pipeline and parameter tuning was deliberately left out of the scope. The preprocessing steps we took possibly fit one algorithm better than the other.

4.3 Proposition for a pipeline

Thus far we have covered quite a few stages that represent pipeline components.

1. Collecting metadata from Trepo repository.
2. Storing the acquired metadata to a database
3. Acquiring full text content
4. Natural Language Processing
5. Topic modelling over time
6. Visualization

By using well informed decision making to select the correct component at each stage to assembly the pipeline, we are revealing the answer to our main goal: *What is the optimal pipeline leading from the original data in Trepo repository to modelling topics over time?*

We have illustrated the pipeline in Figure 4.16. All of the components from 1 to 6 accompanied by a number reference to the enumeration above. The black arrows represent data flows. Some of the arrows are two-headed meaning in practice that after the data has being pushed to a pipeline component the output is fetched and stored back to the database. Our pipeline consists of separate steps that can be combined to produce the input for a visualization tool of choice. The pipeline can be automated and scheduled to run periodically with relatively small effort.

To achieve an automated run a *controller* can be built to orchestrate the sequential data processing stages. It could make sense to build the system so that the controller software is the only piece of software acting with the database in steps 1 to 5. The controller pulls and pushes the data between the database and the pipeline components and takes care of the correct timing for the pipeline steps. Thus the pipeline components are not aware of being a part of the Pipeline. Their sole purpose is to get an input and produce an output. In the pipeline illustration the controller is also pictured bolted to the side of the database. We have not implemented a controller in the scope of the study. The controller can be scheduled to run e.g. as a *cron job*. Earlier in the study we already used Docker and it could be a nice fit here also. A cron job that spins up Docker containers for the duration of running the pipeline.

The only other part of the pipeline communicating directly with the database besides the controller is the step 6 (visualization). This is a natural choice because the modern BI tools have a wide support for connecting with database systems. If the decision is to build a web application instead it is a separate software development project altogether consisting of proper back end and front end solutions. The back end can then very well access the database on it's own having read only rights. This brings us to a debate of whether or not the step 6 should be included in the pipeline at all. We decided to include it to be able to demonstrate an end-to-end solution.

1. Collecting metadata from Trepo repository The second research question was *What is the optimal way to extract data from Trepo?* The research indicates it is by using the provided The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) service. To begin with we had identified two possible ways for consuming the data source: by harvesting the OAI-PMH service and by scraping the Trepo web site. By taking advantage of e.g. the *sickle* Python package with OAI-PMH you can collect all the metadata between dates given as parameters to the harvester as was seen in Listing 1. When wanting to harvest only the fresh publications from Trepo it's easy to fetch the last date already obtained from the database and continue from there. You need one call to the resource to get an iterator of all the metadata records to work with. Meanwhile by scraping the web site it is much more difficult to fetch records between certain dates. Utilizing web scraping you will have to fetch another *html* page for each publication to get to its metadata and then find the correct tags within the page. On the other hand while using the OAI-PMH service you only need one call to the resource to fetch the metadata for the entire corpus. This plays greatly to the advantage of OAI-PMH since the number of calls to external services should generally be minimized. Also the OAI-PMH service is less likely to change than the web page structure so your code will work with higher probability in the future. We recommend using the OAI-PMH but one thing to consider then is the metadata XML schema, as some of the schema formats provided don't include links to full text content and might lack some other fields too. The *Metadata Encoding and Transmission Standard (METS)* format seemed to work well for our research having all the relevant fields present. You could consider implementing a web scraper to back up the OAI-PMH harvester. A combination could work so that if some of the important metadata fields are not available in XML you can try to get them through the Trepo web site then.

2. Storing the acquired metadata to a database The first research question was *What is the best data storing solution for the data set from the perspective of further consumption?*. Our answer to this question is PostreSQL. Because the amount of publications in Trepo was 43401 at the moment we collected the data it is manageable without parallel computing clusters and distributed data storage. As there was 4000 new publications in year 2019 as shown in Figure 3.1 this will be the case in the near future also. Thus we decided to compare standalone versions of PostreSQL and MongoDB as our database candidates. Both systems are scalable to a sharded cluster if need be and both work well for this type of a data set. The test results are visible in Figure 4.1 and we found that Postgres performed better in our test setup. In addition to storing the raw metadata from Trepo to it right after it's been collected, we also separately preserve the output of each of the following steps to the database which is implied in Figure 4.16 too.

3. Acquiring full text content If you want to use full text content instead of only abstracts for topic modelling you will need to download them separately. Even when using only the abstracts this could be beneficial. Especially older publications don't often have an abstract included in the metadata but you could extract it from the publication itself. Some publication metadata include a link to the full text context as a plain text file. Some only include a link to a PDF file. If the link is not available but the access level of the publication is *openAccess* (see metadata fields in Table 3.2) this is a case where you could use a web scraper as a backup to try getting a download link from the web site. If the provided link points to a text file it is simple to download it, read the contents and then store as text to Postgres. In the case of a PDF file we need to extract the text content using *Apache Tika* setup we described in Section 3.4. This setup worked well but took some time: we successfully extracted text from 30459 PDF files and it took 6 hours 25 minutes. In most of the cases where Tika failed to extract text our backup setup consisting of *pdf2image*, *PIL* and *Python-tesseract* also failed - so it mostly was not worth the while to utilize the backup. We also recommend storing the PDF files to the database along with the extracted text content because they contain other useful data such as images.

4. Natural Language Processing Whether you plan to use the full texts or the abstracts it helps with topic modelling to enrich and normalize the data by applying some Natural Language Processing (NLP) techniques on it beforehand. We utilize deep learning from *spaCy* neural pipeline to lemmatize and POS tag the corpus. This will reduce dimensionality and help the machine learning models to yield better results. You could also perform morphological tagging or dependency parsing to extract noun phrases. After the preprocessing we store the enriched data to a new relation in the PostgreSQL. In Figure 3.3 such a table is shown along with another table holding the metadata information.

5. Topic modelling over time The third research question we had was *What is the most suitable algorithmic approach to topic modelling our data set over time?* We compared two different algorithms here: DTM and Dynamic NMF. When the preprocessed texts are available we first create vector representations of the documents and train the models to track topic evolution. We can then save the models to the database by serializing them to JSON form. If you additionally want to serialize the TF or TF-IDF vectorizers you can save some space by using the techniques shown Appendix 13. Which algorithm to prefer then? Quality wise we were not able to find a decisive answer in the scope of this Master's thesis. The DNMF implementation was significantly faster to train and has more built-in features. Based on those we recommend it over the DTM implementation. A good option might be to build a system featuring both algorithms, and letting the end user then choose which one to use with the current data set. Both of the algorithms have quite nice printing functions bolted for outputting the topic evolution related data. Some post-processing is needed to get the data to a nice format for further use. After the post-processing step we once again insert the information obtained into the database.

6. Visualization The last part of our pipeline is the visualization of the results. We have used Tableau to achieve this and tracked the evolution of individual terms inside topics.

You could also visualize by tracking the change in top terms for each topic. Another way to visualize could be by tracking the evolution of topic importance inside the corpus: how many documents belong to each topic at each time slice etc. There are many possibilities here that we haven't explored. A good option for visualization is an interactive dashboard. We have our topic modelling results in the database and it can be consumed from there by a BI tool or a web app to build the dashboard with.

Finally here below, as an answer to the main research question, is the illustration of the proposition for an ideal pipeline (Figure 4.16).

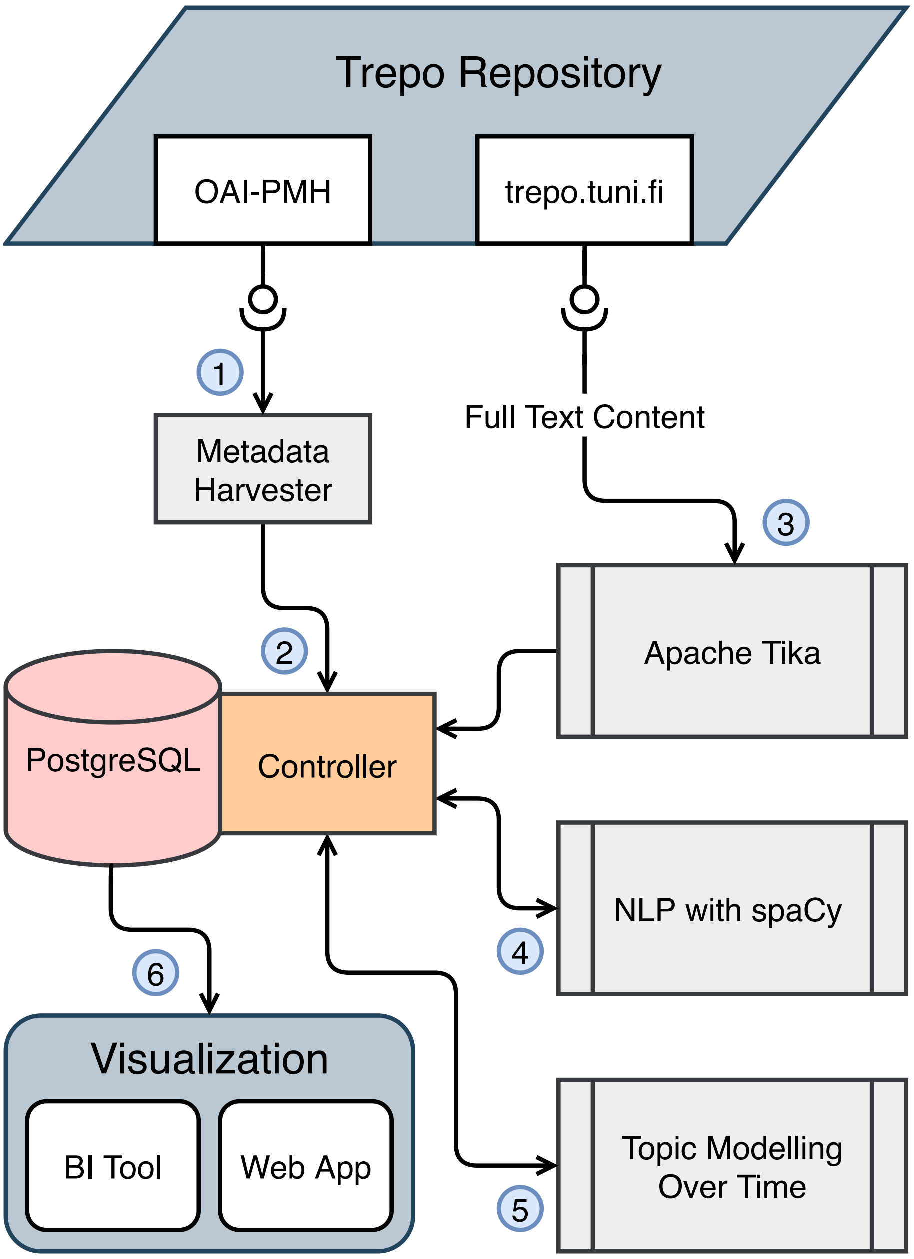


Figure 4.16. *The proposed data pipeline.*

5 CONCLUSIONS

Everyone loves a good story. There are numerous inspiring quotes about stories by intellectual giants throughout history. One such comes from the American poet Maya Angelou: *”There is no greater agony than bearing an untold story inside you”* [2]. A collection of scientific publications from a university is not capable of feeling agony but there are untold stories there certainly. This study went after those stories: the evolution of hidden topics over time within the scientific papers.

We propose a complete data pipeline from metadata repository to visualizing the topic modelling results. The data source for the research is *Trepo*, the scientific publication repository of Tampere Universities. The data set consists of the English-language abstracts of the Master’s thesis level publications between years 2000-2020. The research concentrates on finding an optimal pipeline leading from Trepo to modelling latent topics in the publications over time. We studied separate stages including data ingestion, preprocessing steps, data persisting and topic modelling. These components eventually compose the final pipeline. The resulting composition is shown in Figure 4.16.

Even though discovering a pipeline was the main goal, we had three research questions the optimal pipeline is then built upon. Some of the findings regarding these questions are the most intriguing output of this study. The thesis compared two methods of modelling topics over time: *Dynamic Topic Models (DTM)* and *Dynamic Non-negative Matrix Factorization (DNMF)*. Our research findings indicate DNMF to be the recommended algorithm. The recommendation is based on the model training speed and versatility of the algorithm implementation. The research could not find a clear recommendation based on the quality of the topics produced by the algorithms.

The topics discovered by DNMF and DTM were very similar overall. We were also able to identify the topics as well for both algorithms. An interesting finding was that among the topics technical themes were strongly emphasized. This can be a symptom of the data set originating mostly from Tampere University of Technology. It also indicates that the algorithms have trouble handling a corpus which is imbalanced in regard of the number of documents coming from different fields of study. The latent topics seem to strongly arise from the dominant fields and the minority fields among the collection are shadowed by the majority fields. Thus to efficiently take advantage of the algorithms it is beneficial build a balanced corpus in regard of the fields of study it consists of. One way to achieve this is to construct a corpus separately for each field. As we compared the topic qualities for the algorithms we tracked the evolution of individual terms inside a topic. We tried to identify evolution reflecting the events that happened in the real world at the given time. We were able to see terms reflecting the current happenings, trends and technological advancements to some extend. Still we were not able to make a clear difference in the quality in which the two algorithms reflect the real world events. Our corpus was imbalanced in terms of the number of documents belonging to each observed time slice. This could disturb the term evolution but it should be disturbing both algorithms then. For these reasons our research concludes that quality wise either algorithm is applicable but both have issues that need further attention and follow-up research.

While DNMF was considerably faster to train it still took minutes to build a model on our data set. This means interactive applications where a new model is trained on the selected data on-demand and the training result is expected to be in use while-you-wait in comfortable time is difficult to achieve. Because the training process is time consuming we suggest training the models as scheduled batch jobs in our the pipeline. The data set we used was relatively small and training a significantly larger data set might still increase the training time greatly. The possible impact of the increased data set size is mitigated by running the training as a scheduled job in any case. As we recommend batch training the significance of training time diminishes and as the quality of the topics the algorithms produce is comparable, you could also train both models and let the end user decide which one to use.

To find out the best alternative for storing the data topic models will consume and the models themselves, we compared *MongoDB* and *PostgreSQL*. We concentrated on aggregate queries and in our research environment PostgreSQL performed better. Our recommendation is Postgres but MongoDB performance was good also except for its map-reduce functionality. The extremely poor performance of MongoDB map-reduce is an interesting finding. Another interesting finding is that the JSON functions of PostgreSQL did perform faster than MongoDB even though Postgres is a relational database to begin with and MongoDB is a document store especially designed to work with data in JSON format. We conclude that, based on our results, when building a real world application you can safely use either one of the database systems as the data back-end. If you already have data in MongoDB there is no reason to move it to Postgres. That being said, based on our findings it's difficult to see a reason to choose MongoDB as a persistence layer for a new system.

As for ingesting the metadata from Trepo to the topic modelling pipeline we learnt that using the provided OAI-PMH service is the cleanest and most effective method. The compared method was web scraping the Trepo site. We couldn't find any reason to make use of web scraping in stead of the OAI-PMH service. Yet it is important to pick well the XML schema from the ones the OAI-PMH service offers. Some schemata seemed to lack fields such as the download link to the publication's full text content, but this finding is true in the Trepo context only.

We have identified many interesting paths to continue the research with. Instead of developing the pipeline as a whole these paths begin from the individual components studied in this thesis. Looking at the results of the DBMS comparison it is evident that PostgreSQL can be used as a document store in addition to using it as a relational database. Its JSON functions are efficient and Postgres offers a possibility of organizing part of the data in traditional relations and part of the data in JSON fields without a need to preprocess or normalize it. The results we had evoke a question: is there a reason to ever choose MongoDB over PostgreSQL? Is Postgres as user friendly as MongoDB when utilized as a document store? How does its memory consumption compare to that of a dedicated document database? This study compared standalone versions of the databases. Maybe a sharded cluster would yield more favorable results for MongoDB and especially for its map-reduce functionality. Our research focused on the output speed of aggregate queries and there might be other factors to consider more important in some particular use cases.

In the topic modelling algorithm comparison the model training was time consuming for both of the algorithms. It might be possible to parallelize parts of the training to reduce the time consumption. In the case of DTM it's hard to see enough improvement happening to make it possible to e.g. train new models interactively in a web app and get the resulting model in a reasonable time to immediately play with. For DNMF this could be achievable, depending on the time an average end user is willing to wait. Besides parallelism an interesting follow-up subject would be to study how the data set size correlates with the training time with each algorithm.

When we studied the quality of the topic modelling results we had trouble finding clear evidence in favor of either one of the algorithms. We trained the models using only nouns and mostly default parameters. Parameter tuning and using other parts of speech along with nouns might have an influence on topic qualities. One parameter is the number of topics. Choosing different amounts of topics to infer could reveal new insights. In addition it might be able to counter the bias the corpus had in the study field distribution it consists of. Our data set was skewed in regard of the amount of documents per year also. Would having a more evenly distributed data set generate different results on quality? Our perspective for the topic evolution quality was surveying the evolution of terms inside topics. There are other possible perspectives such as the evolution of topic proportions inside the documents over time. These are possible subjects for further research. In addition there is an interesting follow-up possibility in the preprocessing and NLP part of the pipeline. We performed the lemmatizing and POS tagging with spaCy. The data source for this research is a Finnish university and many of the publications it produces are written in Finnish. As spaCy doesn't have support for Finnish language yet, the inclusion of those publications would require studying the available options.

The pipeline we present can be used to build an automated backbone for a real world topic modelling application having Trepo or some other repository as the data source. The stages make the pipeline modular, individual components can be replaced without touching others. The pipeline can be used as a design for building a similar application by using the public cloud providers' offerings. For instance Azure Data Factory is suitable for implementing the pipeline components.

Results we had on individual stages can be used in other contexts as well. Almost any application has a persistence layer and our results indicate that PostgreSQL could be used as a pure document store even more efficiently than a dedicated document database. The findings from the comparison of the over-time topic modelling algorithms can be utilized in any application of the subject. If you dig deeper into this thesis you might find interesting insights from the NLP tasks and other preprocessing steps we applied on the data along the pipeline. If we are very lucky this work gives someone an inspiration to start exploring the ample domains of Natural Language Processing and Topic Modelling.

REFERENCES

[1] Ahmed, E. M. Hydrogel: Preparation, characterization, and applications: A review. *Journal of advanced research* 6.2 (2015), 105–121.

[2] Angelou, M. *I Know why the Caged Bird Sings*. A Bantam Trade Paperback. Bantam Books, 1997. ISBN: 9780553380019. URL: `https://books.google.fi/books?id=8Gz7dDRzqPkC`.

[3] *Apache-2.0*. May 27, 2020. URL: `https://www.apache.org/licenses/LICENSE-2.0` (visited on 05/27/2020).

[4] Arora, S., Ge, R., Halpern, Y., Mimno, D., Moitra, A., Sontag, D., Wu, Y. and Zhu, M. A practical algorithm for topic modeling with provable guarantees. *International Conference on Machine Learning*. 2013, 280–288.

[5] Arora, S., Ge, R. and Moitra, A. Learning topic models–going beyond SVD. *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*. IEEE. 2012, 1–10.

[6] *Azure docs*. Aug. 25, 2020. URL: `https://docs.microsoft.com/en-us/azure` (visited on 08/25/2020).

[7] Beaulieu, A. *Learning SQL, 3rd Edition*. O'Reilly Media, 2020. ISBN: 9781492057611. URL: `https://learning.oreilly.com/library/view/learning-sql-3rd/9781492057604/`.

[8] Bengio, Y., Courville, A. and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), 1798–1828.

[9] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D. and Bengio, Y. Theano: a CPU and GPU math expression compiler. *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. 3. Austin, TX. 2010.

[10] Berne, O., Joblin, C., Deville, Y., Smith, J., Rapacioli, M., Bernard, J., Thomas, J., Reach, W. and Abergel, A. Analysis of the emission of very small dust particles from Spitzer spectro-imagery data using blind signal separation methods. *Astronomy & Astrophysics* 469.2 (2007), 575–586.

[11] Berry, M. W. and Browne, M. Email surveillance using non-negative matrix factorization. *Computational & Mathematical Organization Theory* 11.3 (2005), 249–264.

[12] Berry, M. W., Browne, M., Langville, A. N., Pauca, V. P. and Plemmons, R. J. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis* 52.1 (2007), 155–173.

[13] Bhardwaj, A., Di, W. and Wei, J. *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing Ltd, 2018.

[14] Blanton, M. R. and Roweis, S. K-corrections and filter transformations in the ultraviolet, optical, and near-infrared. *The Astronomical Journal* 133.2 (2007), 734.

[15] Blei, D. M. Probabilistic topic models. *Communications of the ACM* 55.4 (2012), 77–84.

[16] Blei, D. M. and Lafferty, J. D. Dynamic topic models. *Proceedings of the 23rd international conference on Machine learning*. 2006, 113–120.

[17] Blei, D. M. and Lafferty, J. D. Topic models. *Text mining*. Chapman and Hall/CRC, 2009, 101–124.

[18] Blei, D. M., Ng, A. Y. and Jordan, M. I. Latent dirichlet allocation. *Journal of machine Learning research* 3.Jan (2003), 993–1022.

[19] Bonaccorso, G. *Mastering Machine Learning Algorithms*. Packt Publishing Ltd, 2018.

[20] Brill, E. A simple rule-based part of speech tagger. *Proceedings of the third conference on Applied natural language processing*. Association for Computational Linguistics. 1992, 152–155.

[21] Buneman, P. Semistructured Data. *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. PODS '97. Tucson, Arizona, USA: Association for Computing Machinery, 1997, 117–121. ISBN: 0897919106. DOI: 10.1145/263661.263675. URL: https://doi.org/10.1145/263661.263675.

[22] *Cambridge Dictionary*. Jan. 2020. URL: https://dictionary.cambridge.org (visited on 01/13/2020).

[23] *Cambridge Dictionary Grammar*. Feb. 2020. URL: https://dictionary.cambridge.org/grammar/british-grammar/ (visited on 02/09/2020).

[24] Chen, Y., Zhang, H., Liu, R., Ye, Z. and Lin, J. Experimental explorations on short text topic mining between LDA and NMF based Schemes. eng. *Knowledge-Based Systems* 163 (2019), 1–13. ISSN: 0950-7051.

[25] Choi, J. D., Tetreault, J. and Stent, A. It depends: Dependency parser comparison using a web-based evaluation tool. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, 387–396.

[26] Chung, M. T., Quang-Hung, N., Nguyen, M.-T. and Thoai, N. Using docker in high performance computing applications. *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. IEEE. 2016, 52–57.

[27] *Citus Community official*. Jan. 2020. URL: https://www.citusdata.com/product/community (visited on 01/24/2020).

[28] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Software Pioneers: Contributions to Software Engineering*. Ed. by M. Broy and E. Denert.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, 263–294. ISBN: 978-3-642-59412-0. DOI: 10.1007/978-3-642-59412-0_16. URL: https://doi.org/10.1007/978-3-642-59412-0_16.

[29] *DB-Engines*. Feb. 2020. URL: https://db-engines.com/en/ranking_trend (visited on 02/04/2020).

[30] Deckler, G. *Learn Power BI: A beginner's guide to developing interactive business intelligence solutions using Microsoft Power BI*. Packt Publishing, 2019. ISBN: 9781838646653. URL: https://books.google.fi/books?id=Z-mvDwAAQBAJ.

[31] Devarajan, K. Nonnegative matrix factorization: an analytical and interpretive tool in computational biology. *PLoS computational biology* 4.7 (2008).

[32] *Docker documentation*. Mar. 6, 2020. URL: https://docs.docker.com/ (visited on 03/06/2020).

[33] *DTM original c++ implementation*. Mar. 12, 2020. URL: https://github.com/blei-lab/dtm (visited on 03/12/2020).

[34] *Dynamic NMF implementation*. Mar. 12, 2020. URL: https://github.com/derekgreene/dynamic-nmf (visited on 03/12/2020).

[35] Eito-Brun, R. *XML-based Content Management: Integration, Methodologies and Tools*. Chandos Publishing, 2017.

[36] *Forbes article*. Mar. 6, 2020. URL: https://www.forbes.com/sites/thomasbrewster/2016/04/05/panama-papers-amazon-encryption-epic-leak/ (visited on 03/06/2020).

[37] Freato, R. and Parenzan, M. *Mastering Cloud Development using Microsoft Azure*. Packt Publishing, 2016. ISBN: 9781782173342. URL: https://books.google.fi/books?id=2vtvDQAAQBAJ.

[38] Frichot, E., Mathieu, F., Trouillon, T., Bouchard, G. and François, O. Fast and efficient estimation of individual ancestry coefficients. *Genetics* 196.4 (2014), 973–983.

[39] *gensim documentation*. Mar. 12, 2020. URL: https://radimrehurek.com/gensim/index.html (visited on 03/12/2020).

[40] Ghosh, S. and Gunning, D. *Natural Language Processing Fundamentals*. Packt Publishing, 2019. ISBN: 9781789954043. URL: https://learning.oreilly.com/library/view/natural-language-processing/9781789954043/.

[41] *GNU Scientific Library*. May 18, 2020. URL: https://www.gnu.org/software/gsl/ (visited on 05/18/2020).

[42] *GPLv2*. May 27, 2020. URL: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html (visited on 05/27/2020).

[43] Greene, D. and Cross, J. P. Exploring the political agenda of the european parliament using a dynamic topic modeling approach. *Political Analysis* 25.1 (2017), 77–94.

[44] Griffiths, T. L. and Steyvers, M. Finding scientific topics. *Proceedings of the National academy of Sciences* 101.suppl 1 (2004), 5228–5235.

[45] Han, J., Pei, J. and Kamber, M. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. ISBN: 9780123814807. URL: https://books.google.fi/books?id=pQws07tdpjoC.

[46] Hapke, H. M., Lane, H. and Howard, C. *Natural language processing in action*. Manning, 2019. URL: https://learning.oreilly.com/library/view/natural-language-processing/9781617294631/.

[47] Harrington, J. *Relational Database Design and Implementation*. Elsevier Science, 2016. ISBN: 9780128499023. URL: https://books.google.fi/books?id=yQgfCgAAQBAJ.

[48] Haskins, C. H. *The Rise of Universities*. Routledge, 2017.

[49] Hofmann, T. Probabilistic latent semantic indexing. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. 1999, 50–57.

[50] Honnibal, M. and Montani, I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear. 2017.

[51] Huttunen, H. Pattern Recognition and Machine Learning. University Lecture. 2019. URL: http://www.cs.tut.fi/courses/SGN-41007/ (visited on 02/14/2020).

[52] Isson, J. *Unstructured Data Analytics*. Wiley, 2018. ISBN: 9781119129752. URL: https://learning.oreilly.com/library/view/unstructured-data-analytics/9781119129752/.

[53] Joachims, T. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.* Tech. rep. Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[54] Johansson, R. *Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib*. Apress, 2018.

[55] Juba, S. and Volkov, A. *Learning PostgreSQL 11: A beginner's guide to building high-performance PostgreSQL database solutions, 3rd Edition*. Packt Publishing, 2019. ISBN: 9781789535211. URL: https://books.google.fi/books?id=ZtOGDwAAQBAJ.

[56] Kane, S. P. and Matthias, K. *Docker: Up & Running, 2nd Edition*. O'Reilly Media, 2018.

[57] Kanerva, J., Ginter, F., Miekka, N., Leino, A. and Salakoski, T. Turku Neural Parser Pipeline: An End-to-End System for the CoNLL 2018 Shared Task. *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, 2018.

[58] Korenius, T., Laurikkala, J., Järvelin, K. and Juhola, M. Stemming and lemmatization in the clustering of finnish text documents. *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. 2004, 625–633.

[59] Krohn, J., Beyleveld, G. and Bassens, A. *Deep Learning Illustrated: A Visual, Interactive Guide to Artificial Intelligence*. Addison-Wesley Professional, 2019.

[60] Kuhn, T. S. *The structure of scientific revolutions*. University of Chicago press, 2012.

[61] Kupiec, J. Robust part-of-speech tagging using a hidden Markov model. *Computer speech & language* 6.3 (1992), 225–242.

[62] Lagoze, C., Sompel, H., Nelson, M. and Warner, S. The Open Archives Initiative Protocol for Metadata Harvesting. (June 2002).

[63] LeCun, Y., Bengio, Y. and Hinton, G. Deep learning. *nature* 521.7553 (2015), 436–444.

[64] Lee, D. D. and Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* 401.6755 (1999), 788–791.

[65] Lee, D. D. and Seung, H. S. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*. 2001, 556–562.

[66] Leskovec, J., Rajaraman, A. and Ullman, J. D. *Mining of massive data sets*. Cambridge university press, 2019.

[67] Lewis, D. D. and Gale, W. A. A sequential algorithm for training text classifiers. *SIGIR'94*. Springer. 1994, 3–12.

[68] *LGPLv2*. May 27, 2020. URL: `https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html` (visited on 05/27/2020).

[69] *lxml library docs*. Mar. 6, 2020. URL: `https://lxml.de/` (visited on 03/06/2020).

[70] Manning, C. D., Raghavan, P. and Schütze, H. *Introduction to information retrieval*. Cambridge university press, 2008.

[71] *MariaDB Server documentation*. Jan. 2020. URL: `https://mariadb.com/kb/en/documentation/` (visited on 01/28/2020).

[72] Mikolov, T., Chen, K., Corrado, G. and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[73] Mitchell, R. *Web Scraping with Python: Collecting More Data from the Modern Web*. O'Reilly Media, 2018. ISBN: 9781491985526. URL: `https://books.google.fi/books?id=TYtSDwAAQBAJ`.

[74] *Mongodb official page*. Feb. 2020. URL: `https://www.mongodb.com/` (visited on 02/04/2020).

[75] Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[76] Narkhede, N., Shapira, G. and Palino, T. *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.", 2017.

[77] Nguyen, T. H. and Shirai, K. Topic modeling based sentiment analysis on social media for stock market prediction. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, 1354–1364.

[78] Nickoloff, J. *Docker in action*. Manning Publications Co., 2019.

[79] Nielsen, F. Å., Balslev, D. and Hansen, L. K. Mining the posterior cingulate: segregation between memory and pain components. *Neuroimage* 27.3 (2005), 520–532.

[80] Nigam, K., McCallum, A. K., Thrun, S. and Mitchell, T. Text classification from labeled and unlabeled documents using EM. *Machine learning* 39.2-3 (2000), 103–134.

[81] O'callaghan, D., Greene, D., Carthy, J. and Cunningham, P. An analysis of the coherence of descriptors in topic modeling. *Expert Systems with Applications* 42.13 (2015), 5645–5657.

[82] Ozdemir, S. and Susarla, D. *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing Ltd, 2018.

[83] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[84] Perkins, L., Redmond, E. and Wilson, J. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2018. ISBN: 9781680505979. URL: https://books.google.fi/books?id=rC5aDwAAQBAJ.

[85] Plejic, B., Vujnovic, B. and Penco, R. Transforming unstructured data from scattered sources into knowledge. *2008 IEEE International Symposium on Knowledge Acquisition and Modeling Workshop*. Dec. 2008, 924–927. DOI: 10.1109/KAMW.2008.4810643.

[86] *PostgreSQL documentation*. Jan. 2020. URL: https://www.postgresql.org/docs/current/index.html (visited on 01/19/2020).

[87] *Psycopg2 documentation*. Mar. 12, 2020. URL: https://pypi.org/project/psycopg2/ (visited on 03/12/2020).

[88] *PyMongo documentation*. Mar. 12, 2020. URL: https://api.mongodb.com/python/current/ (visited on 03/12/2020).

[89] *Python documentation*. Mar. 12, 2020. URL: https://docs.python.org/3/ (visited on 03/12/2020).

[90] Qi, P., Dozat, T., Zhang, Y. and Manning, C. D. Universal Dependency Parsing from Scratch. *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, 160–170. URL: https://nlp.stanford.edu/pubs/qi2018universal.pdf.

[91] *Scikit-learn documentation*. Mar. 12, 2020. URL: https://scikit-learn.org/ (visited on 03/12/2020).

[92] Sekine, S., Sudo, K. and Nobata, C. Extended Named Entity Hierarchy. *LREC*. 2002.

[93] Services, E. *Information Storage and Management: Storing, Managing, and Protecting Digital Information in Classic, Virtualized, and Cloud Environments*. EMC Education Services. Wiley, 2012. ISBN: 9781118094839. URL: `https://books.google.fi/books?id=tPLBUi8JSogC`.

[94] *Sickle library docs*. Mar. 6, 2020. URL: `https://sickle.readthedocs.io/en/latest/index.html` (visited on 03/06/2020).

[95] Srinivasa-Desikan, B. *Natural Language Processing and Computational Linguistics*. Packt Publishing Ltd, 2018.

[96] *Tampere Universities web site*. May 18, 2020. URL: `https://www.tuni.fi/fi` (visited on 05/18/2020).

[97] Thomas, A. *Natural Language Processing with Spark NLP*. O'Reilly Media, 2020. ISBN: 9781492047766. URL: `https://learning.oreilly.com/library/view/natural-language-processing/9781492047759/`.

[98] *Trepo OAI*. Mar. 6, 2020. URL: `https://trepo.tuni.fi/oai/` (visited on 03/06/2020).

[99] *Trepo privacy statement*. Mar. 6, 2020. URL: `https://www.tuni.fi/fi/yksityisyys/tietosuojailmoitus-trepo-sahkoinen-julkaisuarkisto` (visited on 03/06/2020).

[100] *Trepo repository*. Feb. 16, 2020. URL: `https://trepo.tuni.fi/` (visited on 02/16/2020).

[101] *Twitter developer API*. Jan. 2020. URL: `https://developer.twitter.com/` (visited on 01/08/2020).

[102] Vanhala, M., Lu, C., Peltonen, J., Sundqvist, S., Nummenmaa, J. and Järvelin, K. The usage of large data sets in online consumer behaviour: A bibliometric and computational text-mining–driven analysis of previous research. *Journal of Business Research* 106 (2020), 46–59.

[103] Wang, Q., Cao, Z., Xu, J. and Li, H. Group matrix factorization for scalable topic modeling. *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. 2012, 375–384.

[104] Wang, X. and McCallum, A. Topics over time: a non-Markov continuous-time model of topical trends. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, 424–433.

[105] Wang, X., Mohanty, N. and McCallum, A. Group and topic discovery from relations and text. *Proceedings of the 3rd international workshop on Link discovery*. 2005, 28–35.

[106] Ward, J. S. and Barker, A. *Undefined By Data: A Survey of Big Data Definitions*. 2013. arXiv: `1309.5821 [cs.DB]`.

[107] *Wikipedia*. Jan. 2020. URL: `https://www.wikipedia.org/` (visited on 01/13/2020).

[108] Wittgenstein, L. *Philosophical investigations*. John Wiley & Sons, 2009.

[109] *Wolfram MathWorld*. Feb. 16, 2020. URL: `http://mathworld.wolfram.com/` (visited on 02/16/2020).

[110] Wu, X., Kadambi, S., Kandhare, D. and Ploetz, A. *Seven NoSQL Databases in a Week: Get up and running with the fundamentals and functionalities of seven of*

the most popular NoSQL databases. Packt Publishing, 2018. ISBN: 9781787127142. URL: https://books.google.fi/books?id=irZTDwAAQBAJ.

[111] Xianghua, F., Guo, L., Yanyan, G. and Zhiqiang, W. Multi-aspect sentiment analysis for Chinese online social reviews based on topic modeling and HowNet lexicon. *Knowledge-Based Systems* 37 (2013), 186–195.

A DATABASE SYSTEMS

In Appendix A we have examples related to database systems introduced in Section 2.1.2 *Database systems*.

```
{
  'customers': [
    {
      'id': 1,
      'fname': 'George',
      'lname': 'Blake',
      'accounts': [
        {
          'account_id': 103,
          'balance': 75.00,
          'products': {
            'product_id': 'CHK',
            'name': 'Checking'
          },
          'transactions': [
            {
              'txn_id': 978,
              'txn_type_id': 'DBT',
              'amount': 100.00,
              'date': 2004-01-22
            }, {}
          ]
        }, {}
      ]
    },
    {
      'id': 2,
      'fname': 'Sue',
      'lname': 'Smith',
      'accounts': [{}, {}, {}]
    }
  ]
}
```

Listing 4. *A truncated example of a possible document database entry in JSON format of the same data as in Figure 2.1*

B DATABASE COMPARISON

In Appendix B we have JavaScript and SQL code examples, YML files and JSON files related to database comparison in Section 3.3. Also we have a figure demonstrating the Docker containerization model we talk about in Section 3.2.

```
version: '3'
services:
  postgres:
    image: postgres:11
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-postgres}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-postgres}
      PGDATA: /data/postgres
    volumes:
    - ./:/data/postgres
    ports:
    - "54321:5432"
    restart: unless-stopped
  mongo:
    image: mongo:4
    volumes:
    - ./:/data/db
    ports:
    - "270170:27017"
    restart: unless-stopped
```

Listing 5. *A docker-compose.yml file for spinning up a PostgreSQL and a Mongodb*

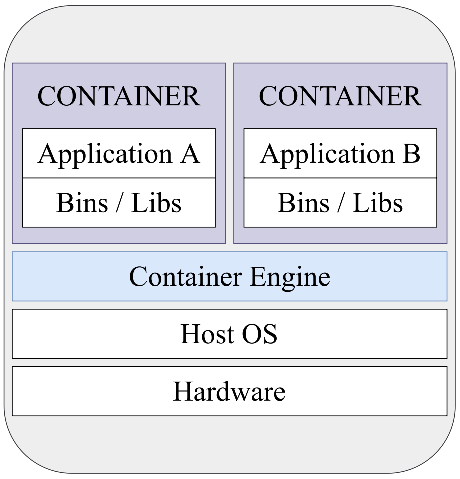
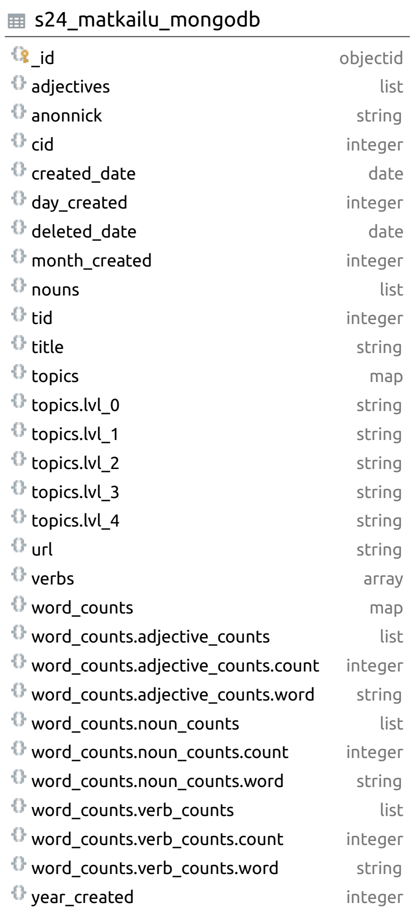


Figure B.1. *The model of application containerization adapted from Docker [26, 32].*

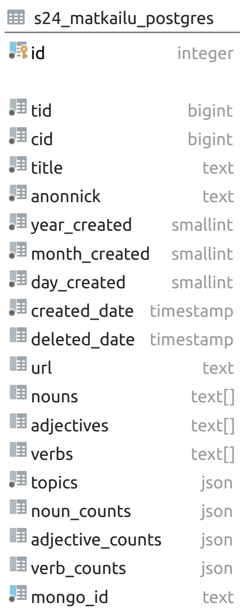
```
{
  'title': 'Sunny in Sevilla during winter?'
  'word_counts': {
    'noun_counts': [
      {
        'word': 'sun',
        'count': 2
      },
      {
        'word': 'river',
        'count': 1
      }
    ]
  'nouns': ['sun', 'river', 'sun']
  'topics': {
    'lvl_0': 'Travelling',
    'lvl_1': 'Europe',
    'lvl_2': 'Spain',
    'lvl_3': 'Andalucia',
    'lvl_4': 'Sevilla'
  }
}
```

Listing 6. *An example of the hierarchy in a Suomi24.fi document in JSON format.*



s24_matkailu_mongodb	
_id	objectid
adjectives	list
anonnick	string
cid	integer
created_date	date
day_created	integer
deleted_date	date
month_created	integer
nouns	list
tid	integer
title	string
topics	map
topics.lvl_0	string
topics.lvl_1	string
topics.lvl_2	string
topics.lvl_3	string
topics.lvl_4	string
url	string
verbs	array
word_counts	map
word_counts.adjective_counts	list
word_counts.adjective_counts.count	integer
word_counts.adjective_counts.word	string
word_counts.noun_counts	list
word_counts.noun_counts.count	integer
word_counts.noun_counts.word	string
word_counts.verb_counts	list
word_counts.verb_counts.count	integer
word_counts.verb_counts.word	string
year_created	integer

Figure B.2. *The schema used for mongoDB in testing with Suomi24 data.*



s24_matkailu_postgres	
id	integer
tid	bigint
cid	bigint
title	text
anonnick	text
year_created	smallint
month_created	smallint
day_created	smallint
created_date	timestamp
deleted_date	timestamp
url	text
nouns	text[]
adjectives	text[]
verbs	text[]
topics	json
noun_counts	json
adjective_counts	json
verb_counts	json
mongo_id	text

Figure B.3. *The only table in the schema used for PostgreSQL in testing with Suomi24 data.*

```
// Find from 'word_counts.noun_counts'
find(
  {},
  {
    'word_counts.noun_counts.word': 1,
    'word_counts.noun_counts.count': 1,
    '_id': 0
  }
)

// Find from 'nouns'
find(
  {},
  {
    'nouns': 1,
    '_id': 0
  }
)
```

Listing 7. *Simple find queries with MongoDB. The first is applied on a list of dictionaries and the second on a list of words.*

```
// Mapper for querying 'word_counts.noun_counts'
function () {
  for (var idx in this.word_counts.noun_counts) {
    var key = this.word_counts.noun_counts[idx].word
    var value = this.word_counts.noun_counts[idx].count
    emit(key, value)
  }
}
// Reducer for querying 'word_counts.noun_counts'
function (key, values) {
  return Array.sum(values)
}

// Mapper for querying 'nouns'
function () {
  for (var idx in this.nouns) {
    var key = this.nouns[idx]
    emit(key, 1)
  }
}
// Reducer for querying 'nouns'
function (key, values) {
  return Array.sum(values)
}
```

Listing 8. *JavaScript functions used for map-reduce in MongoDB. The first two are applied on a list of dictionaries and the last two on a list of words.*

```
// Aggregation pipeline of 3 stages for querying 'word_counts.noun_counts'
aggregate(
  [
    {'$unwind': '$word_counts.noun_counts'},
    {'$group': {
      '_id': '$word_counts.noun_counts.word',
      'count': {
        '$sum': '$word_counts.noun_counts.count'
        }
      }
    },
    {'$sort': SON([('count', -1), ('_id', -1)])}
  ]
)

// Aggregation pipeline of 3 stages for querying 'nouns'
aggregate(
  [
    {'$unwind': '$nouns'},
    {'$group': {'_id': '$nouns', 'count': {'$sum': 1}}},
    {'$sort': SON([('count', -1), ('_id', -1)])}
  ]
)
```

Listing 9. *Aggregation pipelines for MongoDB. Above is the pipeline applied on a list of dictionaries, and below is the one applied on a list of nouns.*

```
SELECT t.nouns
  FROM s24_matkailu_postgres t;
```

Listing 10. *A very simple sql query to retrieve the nouns for all the documents.*

```
SELECT t.word_count->>'word' as word,
       SUM(CAST(t.word_count->>'count' AS INTEGER)) as wcount
  FROM(
    SELECT json_array_elements(nouns) AS word_count
      FROM s24_matkailu_postgres
  ) t
GROUP BY word
ORDER BY wcount DESC;
```

Listing 11. *A query using Postgres' JSON functionality to aggregate word counts.*

```
SELECT word, count(word) as wcount
  FROM s24_matkailu_postgres, unnest(nouns) AS word
GROUP BY word
ORDER BY wcount DESC;
```

Listing 12. *A query that uses Postgres' array functionality to aggregate word counts.*

C TOPIC MODELLING

Appendix C includes code regarding the LDA and NMF modelling of the whole data set and serialization of vectorizers.

Load the data and extracts the abstracts from it.

```
[ ]: import json

     data_file_name_with_path = 'masters_abstract_en_lemma_nouns_only.json'
     with open(file=data_file_name_with_path, mode='r') as file:
         data = json.load(fp=file)

     abstracts = [di['abstract_lemma_nouns_only'] for di in data]
```

Create a *term-frequency vectorizer* and fit it with the data.

```
[ ]: from sklearn.feature_extraction.text import CountVectorizer

     tf = CountVectorizer(min_df=2, max_df=0.95, lowercase=True)
     X = tf.fit_transform(raw_documents=abstracts)
```

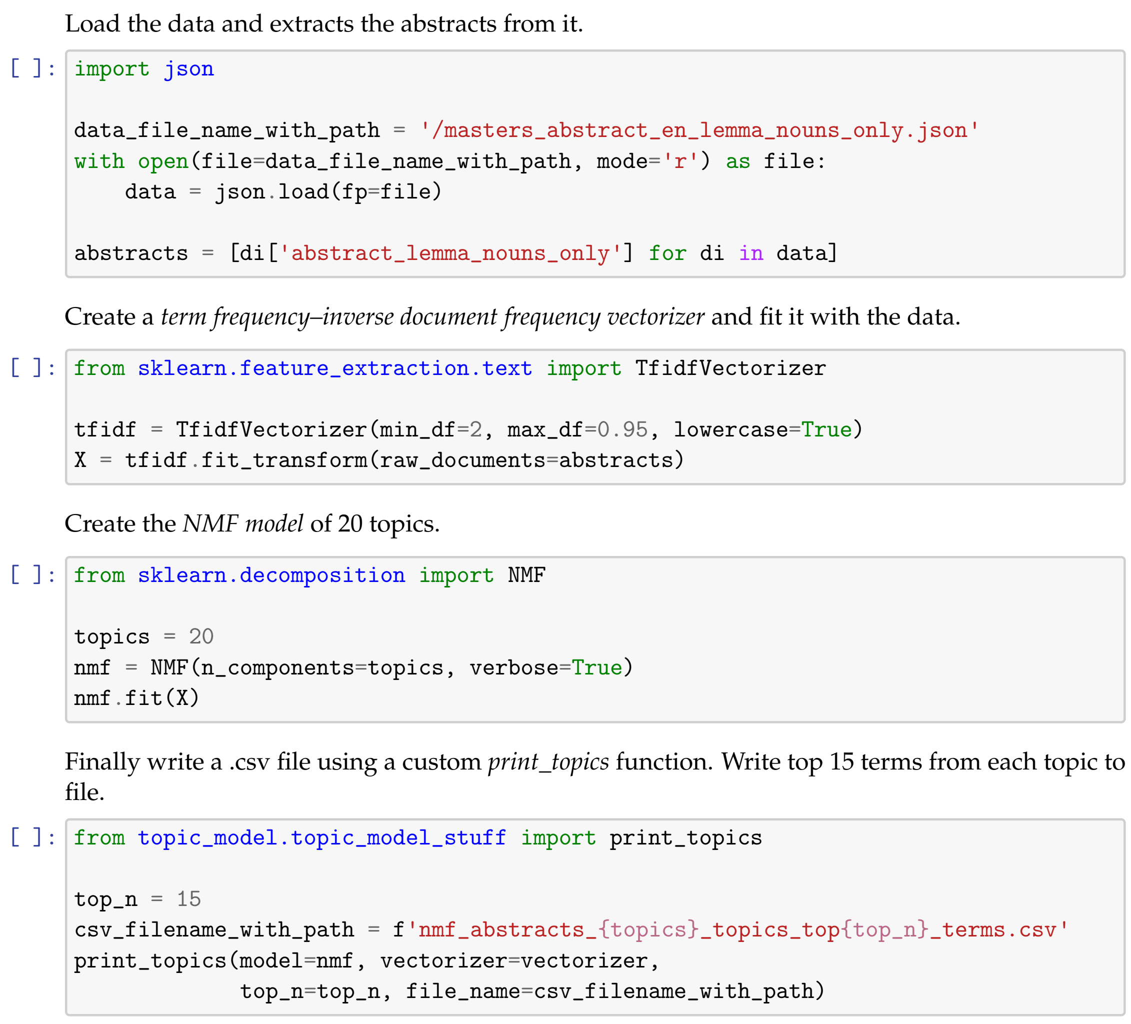
Create the *LDA model* of 20 topics.

```
[ ]: from sklearn.decomposition import LatentDirichletAllocation

     topics = 20
     lda = LatentDirichletAllocation(n_components=topics, verbose=True)
     lda.fit(X)
```

Finally write a .csv file using a custom *print_topics* function. Write top 15 terms from each topic to file.

```
[ ]: from topic_model.topic_model_stuff import print_topics

     top_n = 15
     csv_filename_with_path = 'lda_abstracts_{topics}_topics_top{top_n}_terms.csv'
     print_topics(model=lda, vectorizer=tf,
                  top_n=top_n, file_name=csv_filename_with_path)
```

Figure C.1. *A jupyter notebook on topic modelling with sklearn LDA.*



Load the data and extracts the abstracts from it.

```
[ ]: import json

     data_file_name_with_path = '/masters_abstract_en_lemma_nouns_only.json'
     with open(file=data_file_name_with_path, mode='r') as file:
         data = json.load(fp=file)

     abstracts = [di['abstract_lemma_nouns_only'] for di in data]
```

Create a *term frequency–inverse document frequency vectorizer* and fit it with the data.

```
[ ]: from sklearn.feature_extraction.text import TfidfVectorizer

     tfidf = TfidfVectorizer(min_df=2, max_df=0.95, lowercase=True)
     X = tfidf.fit_transform(raw_documents=abstracts)
```

Create the *NMF model* of 20 topics.

```
[ ]: from sklearn.decomposition import NMF

     topics = 20
     nmf = NMF(n_components=topics, verbose=True)
     nmf.fit(X)
```

Finally write a .csv file using a custom *print_topics* function. Write top 15 terms from each topic to file.

```
[ ]: from topic_model.topic_model_stuff import print_topics

     top_n = 15
     csv_filename_with_path = f'nmf_abstracts_{topics}_topics_top{top_n}_terms.csv'
     print_topics(model=nmf, vectorizer=vectorizer,
                  top_n=top_n, file_name=csv_filename_with_path)
```

Figure C.2. *A jupyter notebook on topic modelling with sklearn NMF.*

```python
import json
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import scipy.sparse as sp

def default(obj):
    """Helper for numpy serialization.
    """
    if type(obj).__module__ == np.__name__:
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        else:
            return obj.item()
    raise TypeError('Unknown type:', type(obj))

def serialize_vectorizer(vectorizer, file):
    """Serialize a vectorizer.
    """
    serialize = dict()
    if isinstance(vectorizer, TfidfVectorizer):
        serialize['idf_'] = vectorizer.idf_
    serialize['vocabulary'] = vectorizer.vocabulary_
    serialize['params'] = vectorizer.get_params().pop('dtype', None)
    try:
        with open(file=file, mode='w') as f:
            json.dump(obj=serialize, fp=f, indent=2, default=default)
    except TypeError:
        raise

def deserialize(file):
    """Deserialize a vectorizer.
    """
    with open(file=file, mode='r') as f:
        deserialize_dict = json.load(fp=f)
    if deserialize_dict.get('idf_'):
        vectorizer = TfidfVectorizer(**deserialize_dict.get('params'))
        idfs = np.asarray(deserialize_dict.get('idf_'))
        vectorizer._tfidf._idf_diag = sp.spdiags(idfs, diags=0,
                                                  m=len(idfs), n=len(idfs))
        vectorizer.vocabulary_ = deserialize_dict.get('vocabulary')
        return vectorizer
    vectorizer = CountVectorizer(**deserialize_dict.get('params'))
    vectorizer.vocabulary_ = deserialize_dict.get('vocabulary')
    return vectorizer
```

Listing 13. *Serialize and deserialize vectorizer properties instead of the whole vectorizer object to save space on disk.*