School of Engineering Science
Business Analytics

Juho Kerppola

Semi-automated document indexing

Thesis for the degree of Master of Science (Technology) in Business Analytics

Examiners: Prof. Pasi Luukka
Postdoctoral researcher Jyrki Savolainen

Tiivistelmä

Kirjoittaja:	Juho Kerppola
Työn nimi:	Semi-automated document indexing
Yliopisto:	LUT-yliopisto
Tiedekunta:	School of engineering science
Tutkinto:	Master of science (Tech.) in Business Analytics
Diplomityö:	89 pages, 36 figures and 16 tables
Vuosi:	2020
Ohjaaja:	Prof. Pasi Luukka
Tarkastajat:	Prof. Pasi Luukka, Postdoctoral researcher Jyrki Savolainen
Asiasanat:	Subject indexing, term assignment, Natural Language Processing (NLP), Machine Learning (ML), ontology, metadata, document indexing

Tämä opinnäytetyö tehtiin yhteistyössä TE-palveluiden kanssa. Tavoitteena oli kehittää menetelmä, jolla voidaan tuottaa automaattisia asiasanaehdotuksia työvoimapalveluista olemassa olevien kuvausten perusteella. Tähän pääsemiseksi tutkimuskysymykset aseteltiin kolmen konkreettisen tavoitteen ympärille. Ensimmäisenä tavoitteena oli tunnistaa sopivia lähestymistapoja automaattisen asiasanoituksen toteuttamiseksi sekä tarkastella näihin liittyviä algoritmeja. Toisena kokonaisuutena olivat algoritmien evaluointimenetelmiin perehtyminen, jotta algoritmien keskinäistä suorituskykyä voitiin arvioida. Kolmannella tutkimuskysymyksellä edelliset kokonaisuudet tuotiin yhteen ja tavoitteeksi asetettiin tunnistettujen algoritmien implementointi, suorituskyvyn evaluointi sekä näiden pohjalta parhaan asiasanoitusalgoritmin suosittelu.

Diplomityössä tarkasteltiin yhteensä viittä eri asiasanoitus algoritmia (kaksi yhdistelmä mallia (eng. ensemble models) sekä kolme yksittäistä algoritmia) käyttäen avoimen lähdekoodin työkalua nimeltä Annif. Annif hyödyntää luonnollisen kielenkäsittelyn ja koneoppimisen työkalujen yhdistelmää, ja sitä voidaan käyttää asiasanoitus algoritmien kouluttamiseen ja evaluointiin. Tulokset osoittavat, että yhdistelmä mallit (eli mallit, joissa yhdistetään useiden yksittäisten algoritmien indeksointiehdotuksia) tuottavat hieman parempia tuloksia kuin yksittäiset asiasanoitus algoritmit. Yksittäisistä algoritmeista Omikuji - koneoppimisalgoritmi asiasanoitukseen - ylitti selvästi VSM- ja Maui-algoritmien tulokset. VSM:n tulokset olivat odotusten mukaisia, kun taas Mauin suorituskyky ei yltänyt kirjallisuuskatsauksessa olleiden tulosten tasolle. Mauin osalta tulokset olivat kuitenkin osittain odotettavissa, koska Maui on leksikaalinen algoritmi, joka etsii sopivia asiasanoitustermejä suoraan annetusta kuvaustekstistä. Tapaustutkimuksessa Mauille annetut syötteet koostuivat lyhyistä palvelukuvauksista, jotka eivät usein sisältäneet kaikkia indeksointitermejä, mikä johti huonoon suorituskykyyn. Vaikka tulokset Mauin osalta eivät olleet toivotunlaiset, muut algoritmit toimivat hyvin ja asiasanoitusehdotusten automatisointi saatiin toteutettua onnistuneesti.

Abstract

Author:	Juho Kerppola
Title:	Semi-automated document indexing
University:	Lappeenranta-Lahti University of Technology (LUT)
Faculty:	School of engineering
Degree:	Master of science (Tech.) in Business Analytics
Thesis:	89 pages, 36 figures and 16 tables
Year:	2020
Supervisor:	Prof. Pasi Luukka
Examiner:	Prof. Pasi Luukka, Postdoctoral researcher Jyrki Savolainen
Keywords:	Subject indexing, term assignment, Natural Language Processing (NLP), Machine Learning (ML), ontology, metadata, document indexing

This thesis was done in co-operation with the Finnish public employment services. The aim was to develop an approach to provide automated indexing suggestions based on employment service descriptions. To realize this the research questions were structured around three concrete goals. First, identify suitable approaches and study the related algorithms for term assignment. Second, research evaluation methods for term assignment algorithms, that can be ultimately used in model selection. And third, implement the most promising algorithms, evaluate their performance, and finally provide recommendation on best alternatives.

Altogether, five different term assignment algorithms (two ensemble models and three individual algorithms) are implemented using an open source tool called Annif. It uses a combination of natural language processing and machine learning tools, that can be used for term assignment and provides a command line interface for both training and evaluating different algorithms. The results show that ensemble models (i.e. models that combine indexing suggestions from multiple individual algorithms) slightly outperformed individual term assignment models. Out of the individual algorithms Omikuji – a state-of-the-art machine learning algorithm for extreme multi-label classification – clearly outperformed VSM and Maui. VSM's results were as expected whereas Maui's performance was not on par with previous evaluation results found in literature. However, these results were partly to be expected since Maui is a lexical algorithm that uses dictionary mapping to prune candidate terms from the input text. Here the inputs consisted of short descriptions and often did not include all indexing terms in the description resulting to poor performance. Overall, the implemented term assignment models performed sufficiently well and provide added value in semi-automatic metadata generation.

Table of contents

Abbreviations

API	Application programming interface
ATC	Automatic text classification
CD	Cumulative gain
DCG	Discounted cumulative gain
JUHO	Finnish Ontology for Public Administration
JUPO	Finnish Ontology for Public Administration Services
KE	Knowledge engineering
ML	Machine learning
MSE	Mean squared error
NDCG	Normalized discounted cumulative gain
NLP	Natural language processing
NN	Neural network
POS-tagging	Part-of-speech tagging
PTV	Finnish service catalogue
RDF	Resource description framework
TF-IDF	Term frequency inverse document frequency
TPV	Employment service catalogue
VSM	Vector Space Model
YSO	General Finnish Ontology

1 Introduction

Searching information from the internet seems like a trivial task. You spin up the browser of your choosing, head to Google, Yahoo, DuckDuckGo or any other search engine you might prefer. There you find a simple text box, write what you are looking for and after pressing enter you are linked to tens of millions of pages with information on the topic you searched for. This, undeniably, simple process of using a search engine, effectively conceals many complexities - overcome by decades of research and development - that go towards choosing most relevant content for the user.

The difficulties of automated search often boil down to the intricacies of human language, though natural to us, foreign and unpredictable to computer programs. The effort to make computers understand human language better - coined Natural language processing (NLP)[1] - has been an ongoing research area from the post-war era of the 1950's when the first studies were conducted around machine translation with the hopes of automatically translating Russian text to English. (Jones, 2001)

NLP is a broad research area. It combines multiple domains from computer science, artificial intelligence, and computational linguistics, and studies the interactions between computers and human languages (Jurafsky, 2012). Some of the main application areas in NLP are in machine translation (Johnson et al., 2017; Chéragui, 2012), sentiment analysis (Jurafsky, Martin, 2019), speech recognition (Nadkarni et al., 2011; Chadha et al., 2015), information retrieval and extraction (Rijsbergen, 1979), automatic summarization (Allahyari et al., 2017) and question answering (Siblini et al., 2019). Today these technologies are ubiquitous and woven deep into the services we use every day - such as the simple search engine.

This thesis will focus on *term assignment* - also known as semantic annotation (Martinez et al., 2017) - and *keyphrase extraction* (Medelyan, 2009). Both techniques for automatically extracting information from unstructured text documents using either a controlled vocabulary (e.g. thesaurus) or the document's vocabulary. These techniques can be seen as a part of information retrieval and extraction - under the NLP umbrella - and are often employed to enhance information storage and search capabilities (Martinez et al., 2017). The aim of this

[1] NLP is the main term used today, also known as Computational Linguistics (CL), Human language Technology or Natural language engineering

thesis is to implement a technique that provides semi-automated subject indexing based on the Finnish KOKO-ontology[2] when prompted with a user inputted piece of text. Based on this goal, three research questions have been formed

1) What techniques/algorithms can be used for term assignment?
2) How to evaluate the results between different algorithmic approaches?
3) How to implement a working solution for term assignment?

These questions are used as a guide in structuring this thesis. First two questions are dependent on a thorough literature review of the subject. To have a good understanding of the landscape around term assignment it is necessary to study both past and contemporary algorithms and evaluation methods. After suitable candidate algorithms and evaluation methods for terms assignment have been identified, the implementation phase necessitates a detailed understanding of the theory behind the chosen methods to both build and evaluate a solution for term assignment. Although the ultimate goal for implementing a term assignment tool is to have a production ready version that can be deployed to an online platform, the scope for this thesis is limited to experimenting and evaluating the identified methods in order to provide recommendations on which approaches would be most suitable for production use. Decision to reduce the scope was taken to focus the efforts on exploring term assignment approaches rather than on web service creation and implementation.

1.1 Scope of thesis

According to Medelyan (2009) different tasks related to *topic indexing* - also known as automated text classification - can be organized using two criteria:

1. Source of terminology used to refer to a document's concepts
2. Number of topics assigned per document

This criteria is depicted in figure 1, where the "number of topics" -axis ranges from only a few fixed topics to all possible and the "Source of terminology" lists different sources for topics like a vocabulary, the document itself or any other source. The scope of this thesis is around *term assignment* and *keyphrase extraction*. (Medelyan, 2009) *Term assignment* is a task of expressing the document's main topics using a predefined set of terms, for example a thesaurus. Terms chosen to express the main topics do not necessarily have to exist in the

[2] Koko-ontologia: https://finto.fi/koko/fi/

text document. Term assignment is considered separate from text categorization, where the number of predefined classes is considerably smaller. *Keyphrase extraction* broadens the scope from term assignment by allowing the use of any phrase or word appearing in the document. Using a controlled vocabulary is optional as the relevant terms can be extracted by using only the in-document vocabulary. (Medelyan, 2009) Table 1 provides a summary of the tasks related to topic indexing in figure 1 - including out-of-scope terms.

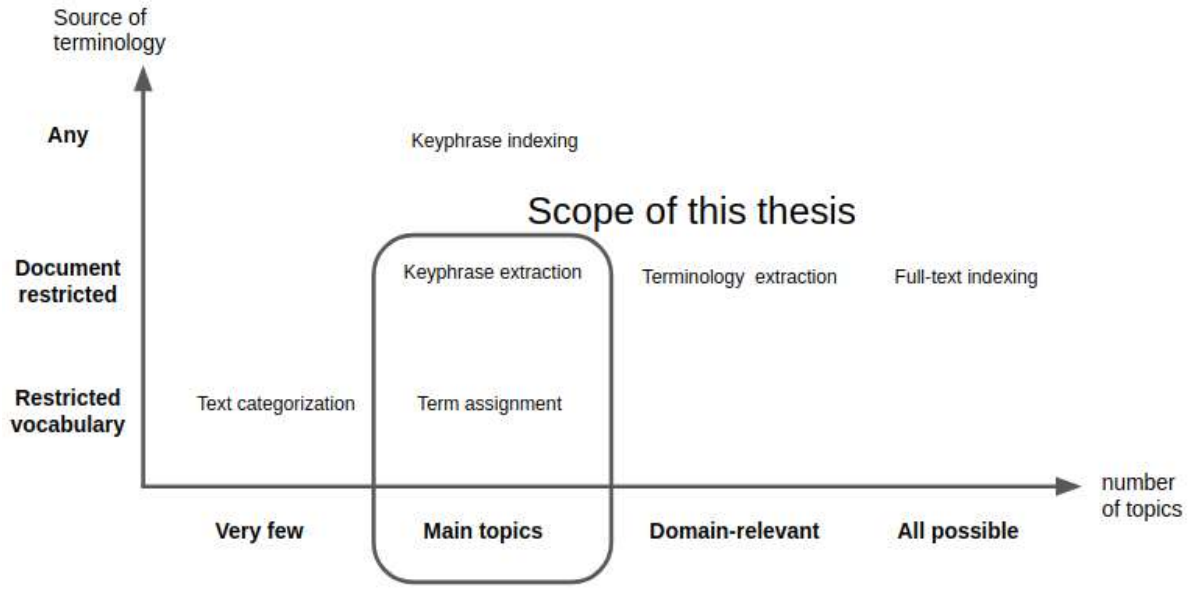

Figure 1 Scope of thesis and relevant terms. Adapted from Medelyan (2009).

Table 1 Different concepts, their alternate names and a brief description of tasks related to topic indexing. Adapted from Medelyan (2009).

Task name	Alias	Description
Text categorization	Text classification	Assigning documents to a general category. Number of categories is small.
Term assignment	Subject indexing, semantic annotation, concept indexing	Assigning documents by its main topics using a large controlled vocabulary for example a thesaurus.
Keyphrase extraction	Keyword extraction, key term extraction, Free-term indexing	Main topics are assigned using most suitable words and phrases from the document. Use of a vocabulary is optional.
Terminology extraction	Back-of-the-book indexing	All domain relevant words and phrases are extracted from a document.
Full-text indexing	full indexing, free-text indexing	All words and phrases - possibly excluding stopwords - are extracted from a document.
Keyphrase indexing	Keyphrase assignment	Main topics are assigned from a non-restricted terminology. Generalization of term assignment and keyphrase extraction.

Term assignment of a text document refers to a process where we attempt to find mappings between a document and the instances of a predefined ontology. These mappings are valuable since ontologies have a capability to express meaning and relationship between different entities. (Martinez et al., 2017) This enhances the annotated document as it can be linked to other associative documents which possess similar concepts that are semantically equivalent and linked through broader concepts. Semantically annotated documents would thereby have superior search capabilities compared to keyword-based search that has no underlying concept of word relationship, context or meaning and therefore can't link to other related content that is not expressed verbatim in document text. (Martinez et al., 2017)

There are two main approaches for automated subject indexing, *lexical* and *associative*. *Lexical approaches* are based on word frequency, where most occurring words are paired with the words in the vocabulary and used as the subject index. These techniques are simple and effective but lack all semantic capabilities of linking between associative subject categories. *Associative approaches* (including machine learning techniques) are used to find correlations between the analyzed document and subject vocabulary, based on large

amounts of training data, by using a classifier. To yield best possible results, often aspects of both techniques are used together. These are known as *ensemble* or *fusion* architectures. (Suominen, 2019)

1.2 Structure of the thesis

This thesis can be roughly divided to two individual parts. The first part (chapters 1 – 5) provide a detailed introduction to term assignment literature and covers a multitude of different approaches as well as evaluation methods for those approaches. In the first chapter the research questions are introduced, and the scope of the thesis defined. The second chapter will provide a literary review that covers previous research on the subject and gives a quick overview of past results with different method of attacks to term assignment. In the third chapter basic concepts of ontologies and controlled vocabularies are introduced as they are a vital part of term assignment. Fourth chapter will take a deeper look at different techniques and algorithms used for semantic annotation. In the fifth chapter a set of evaluation metrics and their use is discussed in length.

The second part of the thesis (chapters 6-7) is more directed towards implementation and experimentation using the algorithms and evaluation methods identified in the first section of the thesis. The sixth chapter will introduce the TE-digi business case and need for semantic annotation. The project phase of this thesis will be presented by following - somewhat loosely - CRISP-DM[3] framework that consists of *business and data understanding, data preparation, modelling, evaluation, and deployment*. However, the deployment to an online platform - including maintenance plan, version control, microservice architecture and creation, etc. - is considered out of scope, so that research efforts can be directed towards automated subject indexing instead of web service creation. Using this framework will provide a bird's eye view of planning and implementation of the annotation tool. Finally, in the seventh chapter conclusions and summary of findings and possible next steps are presented.

[3] https://www.sv-europe.com/crisp-dm-methodology/

2 Previous research on the subject

Automatic text classification (ATC) has been researched since the late 1950's (Luhn, 1958) and some of the first studies were centered around text categorization (Maron, 1961; Borko and Bernick, 1963) i.e. automatically assigning a document its corresponding category based on predetermined set of classes. Driving force behind this work was - as Maron (1961, p.405) puts it - the premise of an "obvious information explosion" that the scientific and intelligence communities were facing. The hope was, that with automated indexing this mass of information - which at the time presented in the form of punch cards[4] - could be efficiently categorized and sorted, aiding information retrieval and the "automatic dissemination" of these documents.

Already in the late 1950's Luhn (1958) aimed at automatically obtaining abstracts from science and technology -related articles. The method of attack was to investigate word frequencies so that a "significance" factor for article's sentences could be estimated and consequently the most relevant sentences extracted to create a "auto-abstract". Although these type of "auto-abstracts" were somewhat clumsy, his ideas on the use of word frequencies (also known as term weighting) was groundbreaking and some of the techniques - or its derivatives, most notably the *inverse document frequency* by Jones (1972) - are still applied in modern NLP procedures (see for example Kowsari et al. (2019) on word frequency). In the 60's multiple different theoretical approaches for automatic indexing were developed. Some notable experiments included Maron's (1961) probabilistic approach using clue-words - derived using words frequencies - with pre-established subject categories, Borko and Bernick's (1963) use of factor analysis and Williams' (1966) discriminant analysis for automated indexing. These early experiments proved somewhat successful - achieving results of around 50 - 60% (summary of results can be found from Stevens (1965, p.101-103)) accuracy within the tested document sets - , but as Stevens (1965, p.104) noted already in 1965 the experiments were carried out on very limited and specialized bodies of data, so extrapolating from these successes should be done with caution.

In the 1970's automated indexing blended in with information retrieval (IR) research as multiple different IR systems were built where automated indexing approaches were implemented to provide automatic categorization of stored collections and ease manual indexing efforts. Multiple research projects - such as Cranfield experiments (Cleverdon,

[4] https://en.wikipedia.org/wiki/Punched_card

1967), ISILT[5] (Keen, 1973), UKCIS (Baker et al., 1972), Medusa (Barber et al., 1973) and SMART (Salton, 1971; Salton and McGill, 1983) - were simultaneously ongoing and focused on enhancing information retrieval practices. Although some advances were made (e.g. invention of the Vector Space Model by Salton, Yang and Wong (1975) in the SMART project) overall the results were disappointing. Sparck-Jones and Rijsbergen (1975, p. 3) note that many of these projects were working with different collections which made it difficult to compare research results between different projects. Also, using separate collections meant that each project was dealing with unnecessary data preparation which scattered the efforts as similar work was done in parallel in each of the projects (Stevens and Rijsbergen, 1975, p. 3).

In the 1980's operational systems for automatic document categorization and indexing were handcrafted using *knowledge engineering (KE)* approaches (Sebastiani, 2002). These systems comprised of meticulously crafted logical statements (i.e. rule-bases) to categorize and label data. Characteristic examples of such rule-base systems from the 1980's are AIR/X (Fuhr and Knortz, 1984) and Construe (Hayes and Weinstein, 1990). Although the implementation details in both projects are quite different (AIR/X employed term-descriptor rules coupled with a probabilistic classification procedure to compute indexing weights (Fuhr and Knortz, 1984) and Construe used pattern matching techniques and rigorous if-then rules (Hayes et al., 1988; Hayes and Weinstein, 1990) for category identification) both had an extensive, domain specific rule-base which took considerable manual effort to build - for example the Construe team reported 9.5 person-years for system development (including the pilot) of which around 4 person-years was devoted to rule-development. (Hayes and Weinstein, 1990) Considering the development effort of building such systems the precision and recall metrics for AIR/X were fairly on par with previous works (Hayes and Weinstein, 1990) and provided little advancement on the accuracy of indexing, for Construe on the other hand, comparative evaluations showed significant improvements over earlier works - improving both recall and precision considerably (Hayes and Weinstein, 1990) - but these results have since been challenged as no other classifier has been tested on the same dataset as Construe (Sebastini, 2002).

In the 90’s and early 2000’s more powerful hardware was starting to be generally available. This made it possible to test the computationally heavy theories which in-turn generated lots of applicative interest in the research community. Research pivoted from hard-coding rules of expert knowledge to learning based technologies and started to develop machine learning

[5] Also known as *the Aberystwyth Index Language Tes*t

(ML) approaches to solve text classification tasks. (Sebastiani, 2002) Examples of experiments with learning based methods included machine learning techniques such as Bayesian classifiers (Lewis and Ringuette, 1994; Larkey, 1998) , k-nearest neighbors (Larkey, 1998; Larkey, 1999), decision trees (Lewis and Ringuette, 1994) and boosting methods such as AdaBoost (Wilbur and Kim, 2003). ML approaches made it possible to build automatic text classifiers using manually labeled training data, which again made automated indexing and text categorizing a viable choice for a larger group of users as the prohibitive cost of codifying human expert knowledge could be automated by suitable learning algorithms. (Sebastiani, 2002)

This trend of using ML based approaches has continued to grow in the recent years. In the past 10 years multiple tools for automated subject indexing have been developed. In the past 5 years most - if not all - notable solutions achieving state-of-the-art performance have employed ML techniques. Implementations that are freely available include: Magpie (Berger, 2015; Kim, 2014), FastXML (Prabhu and Varma, 2014), PD-Sparse (Yen et al., 2016), fastText (Joulin et al., 2017), Quadflor (Galke et al., 2017), AnnexML (Tagami, 2017), Parabel (Prabhu et al., 2018) and Slice (Jain et al., 2019). Some well-known solutions using non-ML based techniques include KEA, it's successor KEA++ and Maui (Medelyan, 2009). These solutions use mainly lexical approaches in matching document and vocabulary terms (Suominen, 2019). A recent work by a finnish research group from the National Library of Finland is implementing an open source tool for automatic subject indexing based on an ensemble of the state-of-the-art subject indexing techniques. (Suominen, 2019) The tool - called Annif - implements both a non-ML based algorithm Maui and multiple different ML algorithms for automated subject indexing, such as: TF-IDF, FastText and Omikuiji and it also provides possibilities for trying out fusion architectures by combining multiple different algorithms to create one classifier. In the project phase of this thesis, Annif will be used as a test bed for evaluating different approaches. (Suominen, 2019)

3 Semantic Web techniques

Machines have no underlying understanding of semantics. Data is only ingested, stored and presented "as is" when queried. Without proper semantic models, computers are not capable of reasoning anything from the stored data, there exists no concepts of entities, relationships or categories. To make data machine-readable - and ultimately machine-understandable - metadata (i.e. data about data) is needed.

Semantic Web architecture offers one possible avenue to model existing concepts and relationships in a machine-readable format. A bedrock of this architecture is the Resource Description Framework (RDF), which provides a simple and an efficient model for presenting semantic information using *resources* and *statements* (explained in length in the next section). This framework is then extended to create ontologies that ultimately allow for specifying different knowledge areas (domains) in machine-readable format.

In this chapter we'll present the basic technologies - RDF and ontologies - surrounding semantic web and the use of ontologies. Technologies discussed in this chapter represent only a subset of the semantic web technology sphere and the motive for their introduction is based on relevance in the context of automated subject indexing and semantic annotation - the use of controlled vocabularies for term assignment - in particular.

3.1 Resource Description Framework

The Resource Description Framework (RDF) is a framework that can be used to express information about *resources* (RDF 1.1 Primer, 2020). Resources can be divided to two specific types: *referent* and *literal* (RDF 1.1 Concepts and Abstract Syntax, 2020). *Referent* describes a resource that is denoted by an unique IRI (International Resource Identifier) and they are used to identify resources such as people, physical objects, documents or abstract concepts. *Literals* on the other hand are used to describe any other resource apart from referents. These include literal values such as dates, numbers, and strings.

RDF provides a standard data model to express relationships - i.e. semantic information - between different resources. This is achieved with *statements*. A *statement* is a three-part construct - also known as a *triple* - and consists of a *subject, predicate, and* an *object.* Both *subject* and *object* represent resources that are related, whereas the *predicate* expresses the nature or type of the relationship. A statement therefore says that a certain subject has some *property* (predicate) with a certain object. An example of a simple statement can be found in figure 2 where we have a statement: "Bob lives in Helsinki" consisting of a subject Bob, object Helsinki and a predicate (relationship) Lives in. Statements can be visualized as directional graphs - called semantic graphs - where the nodes represent subjects and objects and arcs between the nodes express the relationships. A single triple creates the smallest unit of information consisting of a graph with two nodes and an arc - subject, object, and a predicate respectfully. (RDF 1.1 Primer, 2020)

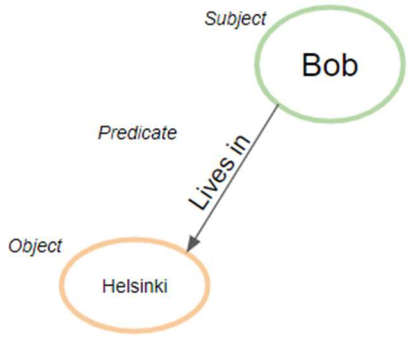

Figure 2 RDF triple representing a statement where subject: Bob, predicate: Lives in, object: Helsinki.

By adding more information to the RDF model, we can create a semantic model of the described resources. This semantic information can then be used for *inference* to derive new facts that are not explicitly expressed but arise from the predetermined relationships. Figure 3 extends the simple example in fig. 2 and adds information that Helsinki *is in* Finland. Using the RDF data model, we can now infer the fact that "Bob lives in Finland" although this is not explicitly expressed within the model.

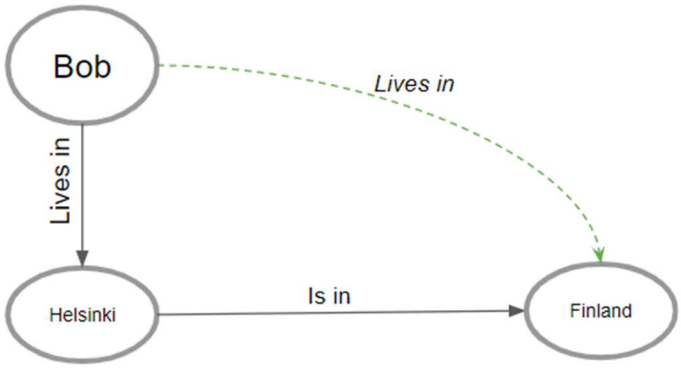

Figure 3 Simple model of inference through semantic information. Known facts are that "Bob lives in Helsinki" and that "Helsinki is in Finland". Using the model, we can infer that "Bob lives in Finland".

These properties make RDF a powerful model for expressing semantic information. The RDF data model itself does not enforce any schema and only describes the core building blocks for modelling semantically entact data. By enforcing a specific schema - such as RDFS or OWL - it is possible to create *RDF vocabularies* or *ontologies* described in length in the next section.

3.2 Ontologies

The word “Ontology” has a long history and carries different meanings in different communities. In philosophical discourse Ontology refers to the study of nature and relations of being, it focuses on what exists and what is real (Definition of ONTOLOGY, 2020). Here we will look at the definition of ontology from the perspective of computational linguistics, in which the scope is somewhat narrower. Gruber (1993) defines *ontology* - in the realm of computational linguistics and AI research - as an explicit specification of a conceptualization. Simply put, this definition holds two distinct characterizations for an ontology. First, an ontology defines (*specifies)* the concepts and relationships that are essential to modelling a domain. And second, these concepts and relationships are presented (*conceptualized)* through a vocabulary that holds formal constraints on its use (Gruber, 2009).

Based on Gruber’s (1993) definition of ontology Lassila and McGuinness (2001) point out three characteristics that an ontology must have, and these are:

1) finite controlled vocabulary
2) an unambiguous definition of classes and term relationships
3) strict hierarchical subclass relationships between classes.

This description - together with Gruber’s - leave a broad range of different specifications that satisfy the definition but differ in precision of the model. Taxonomies and thesauri are considered as less formal - and less specific - instances that fulfill the criteria and are found in the lower end of the ontology spectrum - shown in figure 4.

Taxonomies are a way to represent a hierarchical domain structure. It consists of hierarchical relationships relating concepts from general to specific. The relationships are *transitive:* properties that hold for a general concept also apply to a more specific concept if the concepts are related. Taxonomies are often presented as a tree-like entity (fig.4) consisting of a root - top most node - that reflects the domain, multiple nodes that denote the concepts and are connected by paths between the nodes which relate the structure and relationships between the different concepts. Taxonomies have no standardized way of modelling and therefore sit on the least formal end of the ontology spectrum (Schwarz, 2005, p.5-6)

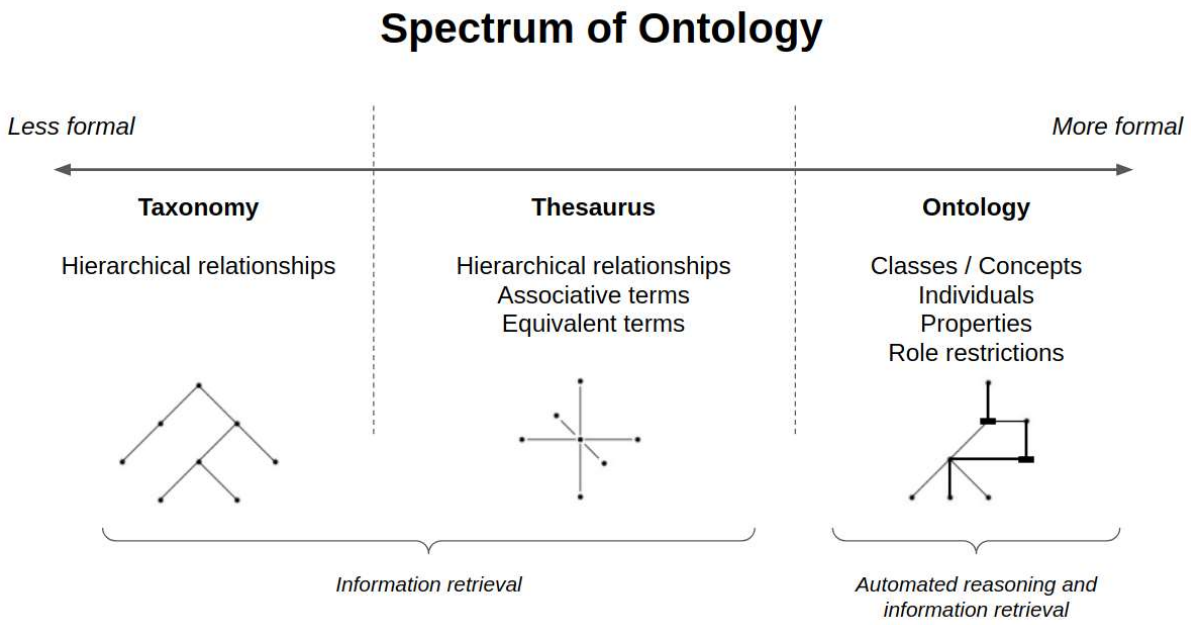

Figure 4 Spectrum of ontology. Adapted from Schwarz (2005, p. 1).

Thesaurus extends the definition of a taxonomy. Harping (2010, p.24) describes thesaurus as a "semantic network of unique concepts, including relationships between synonyms, broader and narrower (parent/child) contexts, and other related concepts". There are multiple generally accepted standards for constructing a thesaurus - such as ISO 25964[6] and NISO Z39.19-2005 (R2010)[7] - which state conventions for documenting both relationships and form (e.g. use of singular and plural) of the terms in the thesauri to achieve consistency in indexing. The types of relationships allowed are *hierarchical, associative* and *equivalent. Equivalence* relationship states the preferred term for a specific concept and alternate terms (e.g. synonyms, colloquial/informal terms, and culturally different terms) describing the same concept. (Schwarz, 2005, p.6-8) *Hierarchical* relationships define broader and narrower (parent/child) relationships between different concepts. There are different types of *hierarchical* relationships such as whole/part (e.g. Finland - Lapland), genus/species (e.g. mammal - human) and instance-of (e.g. Mountains - Mount Everest) relationships. *Associative* relationships are added when the terms are conceptually close but not equivalent or in a hierarchical relationship. (Harping, 2010) There are numerous *associative* relationships and they can be used for example to distinguish terms from each other or relate similar terms. A detailed list of different associative relationships can be found from Harping (2010 p.45).

[6] https://www.niso.org/schemas/iso25964
[7] https://www.niso.org/publications/ansiniso-z3919-2005-r2010

Ontology is a formalized vocabulary of terms that covers a specific domain or an area of interest (OWL 2 Web Ontology Language Document Overview (Second Edition), 2012). It is defined as a collection of concepts and their relationships and it comprises classes*, properties, role restrictions* (i.e. restrictions on properties, sometimes called *facets*) and optionally includes *individuals* or *instances* (Noy and McGuinness, 2001). Though ontologies have many common traits with thesauri - e.g. both describe domain specific information, compose of a terms (or concepts) that have various relationships, and use hierarchical structures - they are fundamentally different as ontologies provide far more expressiveness (i.e. can capture and describe various relationships beyond what is possible using a thesauri) and formality in logical constraints to facilitate automated machine reasoning. (Kim and Beck, 2006) The latter is an important difference as the choice of using either an ontology or thesauri often depends on the intended use. Thesauri and taxonomy are generally a valid choice for information retrieval and extraction (e.g. use in libraries to catalog information). However, if the emphasis is on automated reasoning (e.g. software development and human-machine interaction) ontologies are needed to build necessary logical constraints. (Kim and Beck, 2006)

Ontologies center around *classes*. Classes are used to define concepts that exist in the domain the ontology describes. Classes are arranged hierarchically in super- and subclasses, in similar fashion to Thesauri's narrow and broader relationships. (Noy and McGuinness, 2001) Superclasses depict the broad concepts and subclasses capture narrower and less abstract concepts that are related to the broader concept. In figure 4 an example from Movie-ontology[8] depicts the subclass relationship between CreativeWork and Movie where CreativeWork is the superclass and Movie subclass. Superclass can have multiple different subclasses, for example CreativeWork[9] also has other subclasses such as SheetMusic, Drawing, Book, etc., but these are not depicted in figure 5.

A class represents a collection of multiple instances of the same concept (e.g. class Movie is a collection of all individual movies like Matrix or Die Hard). *Properties* (also called attributes or slots) are used to characterize common traits between instances within one class as well as to describe the different relationships that a certain class has to other concepts defined in the same ontology. (Kim and Beck, 2006) These relationships are defined either between classes (more formally between instances of classes) - called object properties - or a class and a data type - called datatype properties. *Properties* like Author and Director are

[8] https://schema.org/Movie

[9] Check https://schema.org/CreativeWork for full list of properties, subclasses and restrictions

examples of relationships between classes (i.e. object properties) as they both belong to class Person. Award and IsFamilyFriendly on the other hand detail a relationship to different data types (i.e. datatype properties) as they are represented by text and boolean type of values respectively. Properties are also transitive between super- and subclasses, as all properties belonging to superclass are inherited by subclasses. (Noy and McGuinness, 2001; Kim and Beck, 2006)

Important aspect related to properties are *role restrictions* that enable a fine grain description of allowed values, such as data type (e.g. IsFamilyFriendly is asserted to be a boolean value), cardinality (e.g. specifying minimum cardinality of 1 for property Award would mean that all listed CreativeWorks are required to have at least one award), domain (i.e. restrict the classes a property describes) and range (i.e. restrict the values a property can have). (Noy and McGuinness, 2001) Example of domain and range restrictions in figure 4 shows the property Actor which has a domain of Person and range from Movie to VideoGame. This enables an automatic reasoner to *infer* that an actor is always a person. The goal in using different types of properties and role restrictions is to make implicit information explicit (e.g. actor is a person). The reason for this is that machines lack intuitive knowledge of the world (i.e. every human knows that an actor is a person) and it needs to be modeled explicitly so that correct information can be inferred using the ontology.

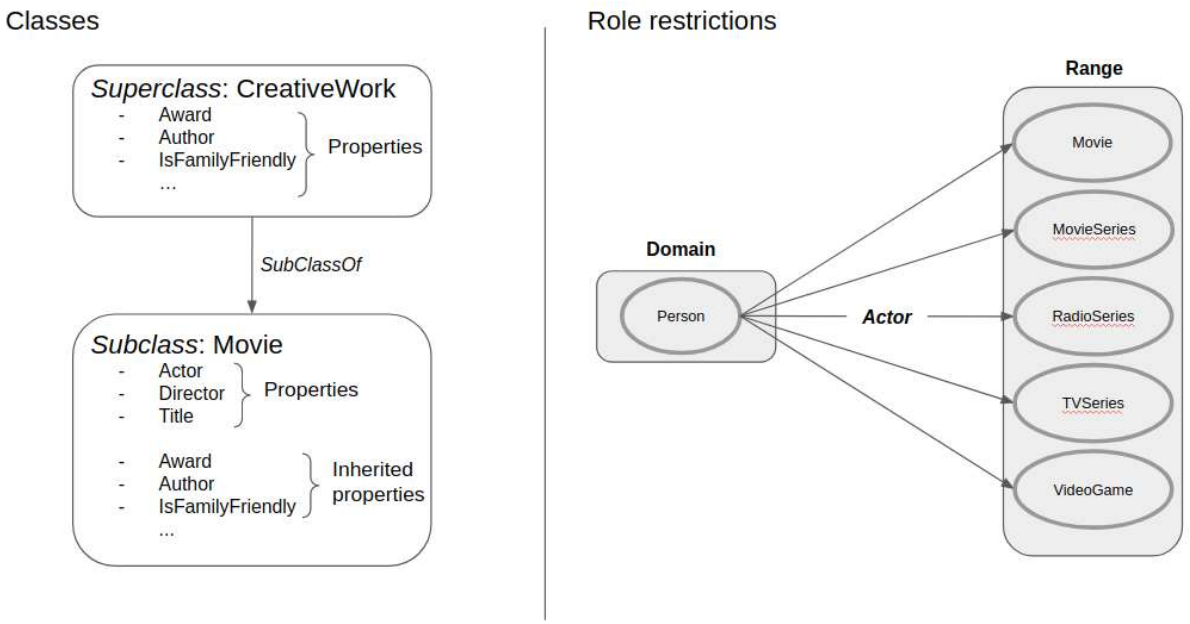

Figure 5 Example of classes, properties, and role restrictions in a Movie-ontology. Example depicted from https://schema.org/Movie

4 Automatic subject indexing

In this chapter the aim is to familiarize with the basic strategy of Natural Language Processing (NLP) and lay out the theory of different subject indexing approaches. First, we'll look at the standard NLP process, which is used as a preliminary step to turning text from human-readable documents to machine-readable tokens that can then be used for further analysis. After a brief explanation of the different phases of NPL-procedure three different subject indexing algorithms and the concept of ensemble models are introduced. Motive behind selecting the introduced models is based on a literature review of contemporary approaches as well as the freely available open source contributions of previous researchers, since the overall goal is implementation and evaluation of different algorithmic approaches.

4.1 Standard NLP-procedure

Written text is considered an *unstructured* form of data. Computers on the other hand are well suited for analysis using *structured* data - e.g. data in a table format or other strictly defined format. To process text into suitable form - from *unstructured* to *structured* -, often a standard pre-processing scheme is applied. Next, we'll overview the different phases of this procedure to build a general understanding of each processing step. Individual techniques for implementing these steps are not discussed in detail, but instead the focus is only on the overall process and what type of modifications are performed on the input text. A pictorial summary of the process is given in figure 6.

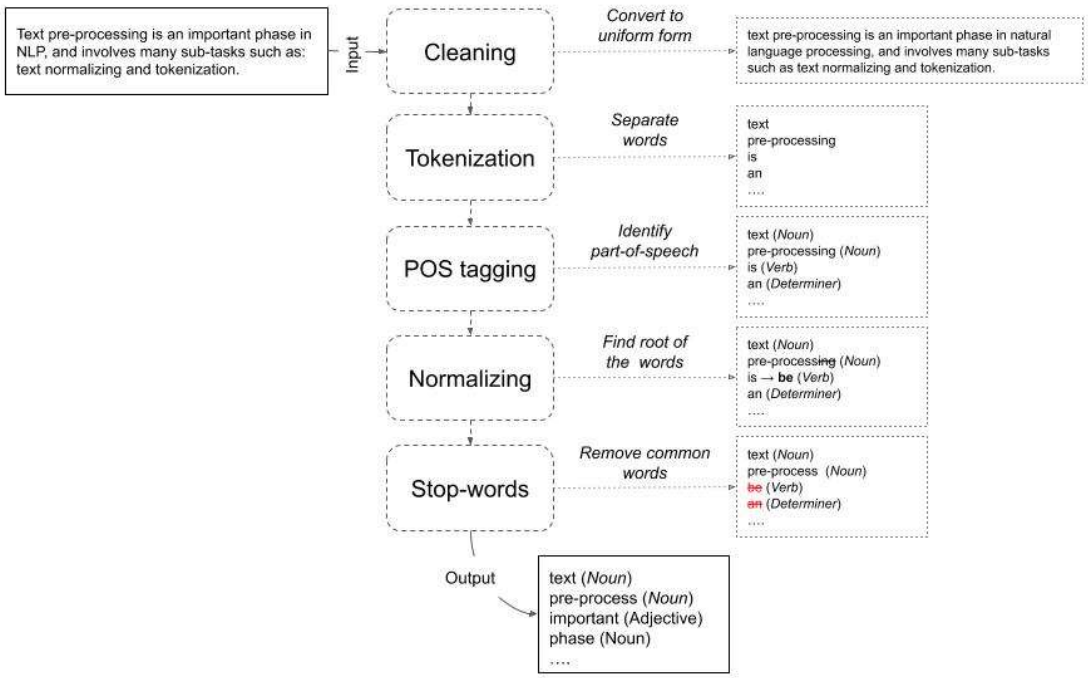

Figure 6 Pictorial summary of often used pre-processing steps for different NLP-related tasks. Input is a text document and output a curated list of tokens with part-of-speech tagging.

4.1.1 Data cleaning

Data cleaning is the first step in text processing and its objective is to transform the given text input to a uniform state. The applied methods are dependent on the input text, as it may vary from more formal forms, such as articles, books or news to less formal like tweets, text messages, email or raw HTML content downloaded from the internet. Often the more informal the source text the more cleaning steps it will require.

Common steps include modifying text encoding to a standard format (e.g. UTF-8), converting all text to lower or upper case, converting numbers to words, or removing them if not needed, expanding abbreviations and removing extra whitespace (Davydova, 2018). Other processing steps - depending on the input text - might include working with HTML-tags and characters, transforming slang words and abbreviations (e.g. LOL or 4ever) to standard text or translating emoticons or emojis to text. In figure 6 a simple example is presented, where text is transformed to lowercase, and the ‘NLP’ abbreviation is expanded.

4.1.2 Tokenization

Tokenization is the first linguistic preprocessing step as the goal is to identify the basic units of which the text is composed (Michelbacher, 2013). Tokens represent meaningful elements such as words, phrases or symbols. A popular tokenization method is to separate words to individual tokens using white space as a delimiter - called *single word tokenization (SWT)* (Michelbacher, 2013, p.26-28). However, using SWT can be problematic since tokens should represent *meaningful* elements, separating words using only white space loses some of this property as multi-word units (MWU) are overlooked and in the process semantic information is lost. MWUs comprise of collocations, habitual word combinations that co-occur together and carry special meaning (e.g. *crystal clear*, *hot dog* or *there's no use crying over spilled milk*). (Michelbacher, 2013, p.23-26) For example tokenizing the word "Hot dog" to two distinct tokens "hot" and "dog" will erase the initial semantic relationship to food. Typically, tokenization should be considered as a language specific task. Different languages present different issues and knowing the language offers additional information about word formation and grammatic. As an extreme example some East Asian languages (e.g. Chinese, Japanese, Korean, and Thai) employ no explicit word boundaries in written text, which means that tokenizing these languages requires additional lexical and morphological information. (Palmer, 2000)

4.1.3 Part-of-speech tagging

Part-of-speech tagging (POS-tagging) is used to attach each word in the input text to its correct word class. Input to a tagging algorithm consists of a sequence of tokens (see previous section about tokenization) and a set of tags (i.e. word classes) and the output is a sequence of tags, one for each input token. (Jurafsky, Martin, 2019 p.148) Word classes can be divided into two separate super categories: *closed class type* and *open class type*. (Jurafsky, Martin, 2019, p.144) *Closed class* refers to a set of word classes that are relatively fixed, in the sense that their lexicon doesn't usually change, these include for example *prepositions* (under, over, at, etc.), *pronouns* (she, we, others, etc.) and *numerals* (one, two, etc.). Open classes by comparison have a more fluid lexicon and change over time as new words (i.e. neologisms) enter common use. There are four major open classes: nouns, verbs, adjectives, and adverbs. (Jurafsky, Martin, 2019 p.144-145) Use of word classes vary between languages and not all languages share the same word classes. For example, in the Finnish language there are only few words that can be used as a preposition and mostly postpositions and case suffixes are used. (Kielitoimiston ohjepankki, 2020)

These major classes are then divided into various subclasses which can then be used for tagging the input tokens. For example, a widely used part-of-speech tag set for English is the Penn Treebank tag set (Marcus, Marcinkiewicz and Santorini, 1993) that consists of 45 different tags, when symbols and punctuations are included. Figure 7 includes a complete list of tags together with examples for each tag. POS-tagging is often considered a language specific task and separate tag sets exist for most major languages[10]. Size of the tag set is always dependent of the particular treebank used and a summary by Petrov et al. (2011) consisting of 25 different treebanks from 22 different languages had variation from 11 tags in a Russian Treebank to 294 tags used in a Chinese Treebank - median was 42 tags, which is close with the Penn Treebank. Recently research towards universal (i.e. multi-lingual) POS-tagging has gained growing interest (Snyder et al., 2009; Petrov et al, 2011; Nivre et al., 2016). A standard framework for multilingual tagging is the Universal POS tag set developed by the Universal Dependencies project[11].

POS-tagging is mainly used as a *disambiguation task* as separate tokens can have multiple different meanings and are therefore ambiguous. For example, the word 'book' can be used as a *verb* "Did you *book* those tickets?" or as a *noun* "I am reading this *book*." and using POS-tagging can resolve these ambiguities. (Jurafsky, Martin, 2019 p.148-149)

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	"	left quote	' or "
LS	list item marker	*1, 2, One*	TO	"to"	*to*	"	right quote	' or "
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Figure 7 Penn Treebank part-of-speech tags, their description, and examples (Reproduced from Jurafsky, Martin, 2019)

[10] A table of 25 language specific tagsets (+ a derived universal POS-tag scheme) can be found at https://www.researchgate.net/publication/51887367_A_Universal_Part-of-Speech_Tagset

[11] https://universaldependencies.org/

4.1.4 Word form normalization

Written text often contains multiple different forms of a single word. Morphological variants (e.g. measurement, measuring, measures, measure, etc.) are usually most prevalent, but also valid alternative spellings, misspellings and other variants from transliteration and abbreviations occur (Willett, 2006). This has a dampening effect on information retrieval as the amount of different search terms would need to be inflated to find all mentions of a single subject. Conflating different terms under a single - appropriate - variant, would therefore provide significant benefits in information retrieval and extraction. (Willett, 2006) To accomplish this, *word form normalization* is applied.

Word form normalization refers to a process where the input token is converted to its standard form (Jurafsky, Martin, 2019 p.3-4). There are two main techniques for word form normalization: *stemming* and *lemmatization (Toman et al., 2006)*. Both techniques are aimed at determining a basic form of a token, however there are important differences as the derived basic form is often different based on which approach is chosen.

Lemmatization is used to identify to which *lexeme* each individual token belongs to, and then replace this term with the corresponding *lemma*. Different word forms that are used to denote the same concept, belong to the same lexeme (Silfverberg, 2016, p.20). For example, word forms such as “cat”, “cats” and “cat’s” all refer to a common concept “cat” and therefore belong to the same *lexeme*. Each lexeme has a *lemma,* a specific word form used to denote the entire lexeme. In the described example lemma would be “cat” and more generally for English nouns, the singular form of a word is used. (Silfverberg, 2016, p.20-21) Lemmatization is a challenging task as morphological information (i.e. information of word stems/basic forms and different affixes and inflections a word may have) on individual words are needed. (Jurafsky, Martin, 2019 p.21) For example words “*am”, “are”* and “*is”* all share a common lemma ‘*be’* and the words “*singing”, “sang” and “sung”* have a lemma ‘*sing’.*

Stemming on the other hand is a coarser way of attaining token’s standard form. The aim of stemming is not to produce a morphologically correct basic form - as in lemmatization - but only an approximation of this form, called a *stem*. (Toman et al., 2006) A widely used method for stemming is suffix-stripping, where the end of word affixes is removed using a standard heuristic. One of the most popular rule sets for stemming was defined by Porter (1980) and is coined *Porter stemmer* and is still widely used today. Applying Porter stemmer to words “*measure”, “measurement”* and *“measuring”* would yield a common stem ‘*measur*’ although the correct basic form is ‘*measure’*. When Porter stemmer is applied to the previous

examples from lemmatization, the stemming procedure produces "*am"*, "*ar"* and "*is"* - when the correct lemma would be 'be' - and "*sing", "sang", "sung"*, as 'sing' should be used.

4.1.5 Stopword removal

Often most frequently occurring terms in any text document are insignificant, as they have low discrimination power between different subject areas (Lo et al., 2005). These words are called *stopwords* and they mainly consist of function words (e.g. conjunctions, prepositions, pronouns, determiners, etc.) that are used to build relationships between other words in the text (Function word, 2020). According to Chung and Pennebaker (2007) 50 most frequently occuring words in the English language make up over 40% of any given text. In the context of information retrieval and extraction this can be considered noise, as these are poor index terms and are unlikely to produce any meaningful search results (Lo et al., 2005).

Since stopwords constitute a large portion of any given document, and at the same time carry little information, it is often a good idea to remove these terms. Often used methods for stopword removal include using *stoplists* or different *statistical approaches. Stoplist* is a pre-compiled list that includes *stopwords* which are simply removed from the document. Examples of early English *stoplists* in literature include Van Rijsbergen's (1979) list of 250-stopwords and Fox's (1989) stoplist of 421-words based on the Brown corpus, sometimes referred to as *classic* and *standard* stoplists respectively. Although popularly used, stoplists have two major limitations. First, pre-compiled lists become outdated quickly as lexicon changes and new stopwords enter vocabulary - especially with web and social media (Saif et al., 2014). Second, stoplists suffer from generality, as standard lists only consist of frequent words and often need to be tailored for specific purposes, using stoplists can create considerable manual work to account for different types of texts. (Saif et al., 2014) Depending on the domain, using a off-the-shell *stoplist* can even be hampering, for example if one is searching information on rock-music (or more broadly from the music domain) removing the word 'The' means that many band names are cut from the text (e.g. The Rolling Stones, The The, The Who, etc.). Before using stoplists these domain constraints need to be understood and the use of stopwords checked for applicability.

Statistical approaches have been developed to solve some of the shortcomings that stoplists suffer from. These approaches are aimed at automatic stoplist generation, which in theory should remove the need for manual curation of topic/domain specific stoplist and also keep the stoplist up-to-date even as lexicon changes (Saif et al., 2014). Different methods for statistical stoplist generation have been proposed, but they can be categorized into two

distinct groups: methods based on *Zipf's law* and methods based on *information gain criteria* (Saif et al., 2014).

Zipf's law (Zipf, 1949, p. 19 - 55) states that *when given a corpus of natural language, the frequency of any word is inversely proportional to its rank in the frequency table*. This fundamental property - or orderliness, as Zipf puts it - has been used as an axiom for constructing different word-frequency based techniques for stopword removal. As we observed earlier, most frequent words tend to be used as function words and serve very little content or information. This fact combined with Zipf's law (i.e. that frequency and rank are inversely proportional) makes it possible to automatically craft a threshold for word significance and use it as a heuristic for stopword removal (fig. 8). Upper threshold is chosen by investigating the frequency differences between consecutive words (Lo et al., 2005). This is often done using the "elbow" method (used also as an heuristic for example in clustering and principal component analysis - see for example Kodinariya and Makwana (2013) or Syakur et al. (2018)), where the threshold is chosen at a point where the frequency difference between consecutive words starts to diminish. Lower threshold on the other hand is often used to cut words that occur only once, i.e. singletons (Saif et al., 2014). Third method is to use the *inverse document frequency* (IDF), which is calculated by weighting terms based on how often they appear in a given document versus the whole collection. The intuition is that a term that occurs often in multiple documents is not a good discriminator and should therefore have a lower weight compared to a term that is document specific. (Robertson, 2004)

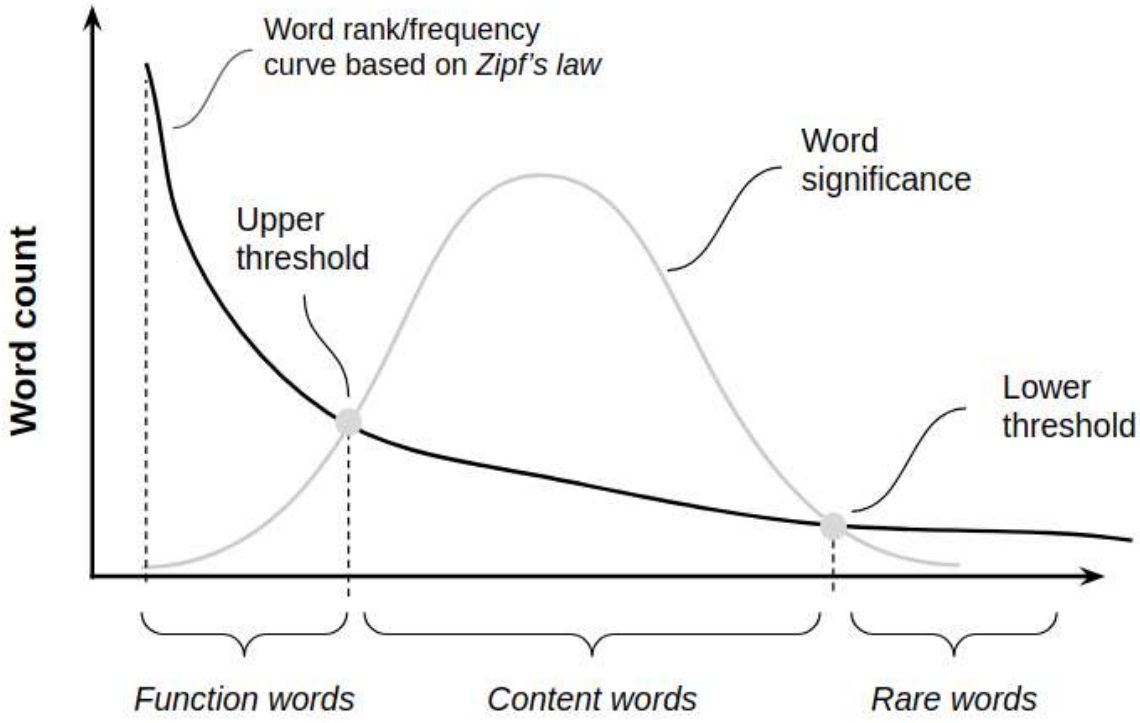

Figure 8 Stopword removal using a statistical approach based on Zipf's law. Word significance is dependent on word-rank, most- and least-frequent words are eliminated. Adapted from Luhn (1958).

As methods based on Zipf's law are centered around word frequency, methods based on information gain are used to assess how much information a term carries. The intent is to detect words with low informativeness and then remove them. Different methods such as *term entropy measure* - also known as *term based random sampling* - (Sinka, Corne 2003), that measures frequency variance of terms between multiple documents (similar to IDF) and *divergence measure* which uses Kullback-Leibler divergence measure to estimate the informativeness of a given word in a document collection by comparing term distribution from a randomly selected sample to the whole document collection (Lo et al., 2005) are also used.

4.2 Approaches to term assignment

The preprocessing pipeline (introduced in previous section, pictorial summary in fig. 6) is used to tokenize and structure the document collection. Next, we can focus on the actual task of term assignment and subsequently consider different approaches to assigning descriptive keywords to a document based on a controlled vocabulary. In this chapter the aim is to lay out the fundamental approaches (*lexical* and *associative*) to term assignment and discuss the advantages and drawbacks of individual algorithms belonging to either camp.

4.2.1 Overview of different approaches

Approaches for term assignment can be split to two main avenues: *lexical* and *associative* (Toepfer and Seifert, 2018). *Lexical approaches* rely on the knowledge specified in thesauri to identify and rank suitable keywords or concepts. Typically, these algorithms require little training data as suitable candidates are extracted through dictionary mapping and ranked using simple feature values (e.g. TF-IDF score, length, first-occurrence, etc.) calculated for each candidate word. (Toepfer and Seifert, 2018, p.9) However, since the candidate words are extracted from thesaurus using dictionary mapping, the main drawback from lexical approach is quite evident. For dictionary mapping to work, proposed candidate words need to exist verbatim in the document. Using a lexical approach, we are restricted to the matches found between the document and used vocabulary. (Toepfer and Seifert, 2018, p.2) From a practical viewpoint, this means that the used vocabulary needs to exhaustively cover the terminology of the domain since no association between the terms exist (i.e. document covering aviation and using for example terms: *airliner, jet* or *aircraft* need to all be explicitly found from the vocabulary as no internal association to - perhaps more common word - *airplane* is made). This fact hinders the effectiveness of lexical approaches as building and maintaining such an extensive thesaurus is costly and often not possible. (Toepfer and Seifert, 2018, p.9) Moreover, even if such extensive thesauri would exist, documents often don't include all relevant subjects in text so these subjects will not be suggested when using a lexical algorithm (Suominen, 2019).

In contrast to lexical algorithms, *associative approaches* employ machine learning (ML) methods to find mappings and correlations between document words and vocabulary subjects (Suominen, 2019). Mappings between terms and subjects are learned by using appropriate ML algorithms together with an extensive amount of professionally pre-labeled training data. (Toepfer and Seifert, 2018, p.2) From this point of view term assignment can be considered a multi-label learning task where the controlled vocabulary provides labels which are then used to categorize any given document. For this approach to work, a classifier needs to be trained for each label (i.e. subject in the controlled vocabulary) separately, meaning that for every subject in the vocabulary there needs to exist some pre-labeled training data so that mappings can be associated. (Toepfer and Seifert, 2018, p.2) The evident drawback is, that for many applications such an extensive training set is not available, and creating one is costly.

Lexical and associative approaches can be considered complementary as both approaches have different drawbacks and merits. (Suominen, 2019) To leverage this, *fusion architectures* have been proposed combining elements from both lexical and associative approaches. These techniques alleviate the different disadvantages and can fully leverage the strengths from individual approaches. (Toepfer and Seifert, 2018, p.4)

4.2.2 Lexical algorithms

Recently, research and implementations of different lexical approaches to term assignment have seen a downturn as research has pivoted towards machine learning approaches[12]. However, lexical approaches can be used to complement ML based solutions as they provide valuable predictions based on keywords expressed verbatim in the document text. In the project phase of this thesis, the idea is to implement a fusion architecture where both lexical and associative algorithms are used in combination. For this reason, Maui - a lexical algorithm for term assignment - is introduced.

Maui indexing

Maui is a lexical automated subject indexing tool developed by Olena Medelyan in her 2009 PhD work “Human-competitive automatic topic indexing”. Name *“Maui”* stems from **m**ulti-purpose **au**tomatic topic **i**ndexing, and it is named after the Polynesian mythological hero known in the Māori mythology as Māui (Medelyan, 2009, p.7). As the name suggests Maui is a multi-purpose tool as it is able to perform three different types of topic indexing - indexing with a controlled vocabulary (term assignment), topic indexing using Wikipedia terms (keyphrase extraction) and automatic tagging (Medelyan, 2009, p.8). In the context of this work, the main interest is in indexing with a controlled vocabulary (term assignment), but the algorithm Maui implements for the different tasks varies only slightly, meaning that the overview of the algorithmic process of term assignment also generalizes to other indexing schemes. However, even though the general process is similar there are some variations in the implementation details (e.g. in automated tagging and keyphrase extraction schemes no controlled vocabulary is available, so possible term candidates are extracted in slightly different manner). (Medelyan, 2009, p.79) These differences are not discussed here - as

[12] such as Magpie (Berger, 2015; Kim, 2014), FastXML (Prabhu and Varma, 2014), PD-Sparse (Yen et al., 2016), fastText (Joulin et al., 2017), Quadflor (Galke et al., 2017), AnnexML (Tagami, 2017), Parabel (Prabhu et al., 2018), Slice (Jain et al., 2019)

term assignment is the primary focus -, but an interested reader can refer to Medelyan's thesis[13] for a more thorough examination.

Maui implements a two-stage topic indexing algorithm consisting of *candidate generation* and *filtering*. The first phase, *candidate generation,* is a five-step process that is used to identify and extract keywords and -phrases from document text by mapping the input text to the used vocabulary. (Medelyan, 2009, p.79) Main goal of the *candidate generation* -phase is to detect and extract as many candidate topics as possible (i.e. achieving high recall). These candidates are then pruned in the *filtering*-phase using various different features, so that false positives that were attained in the first phase can be excluded from the actual results of the algorithm. (Medelyan, 2009, p.79-81) A high level summary of Maui's indexing process is given in figure 9.

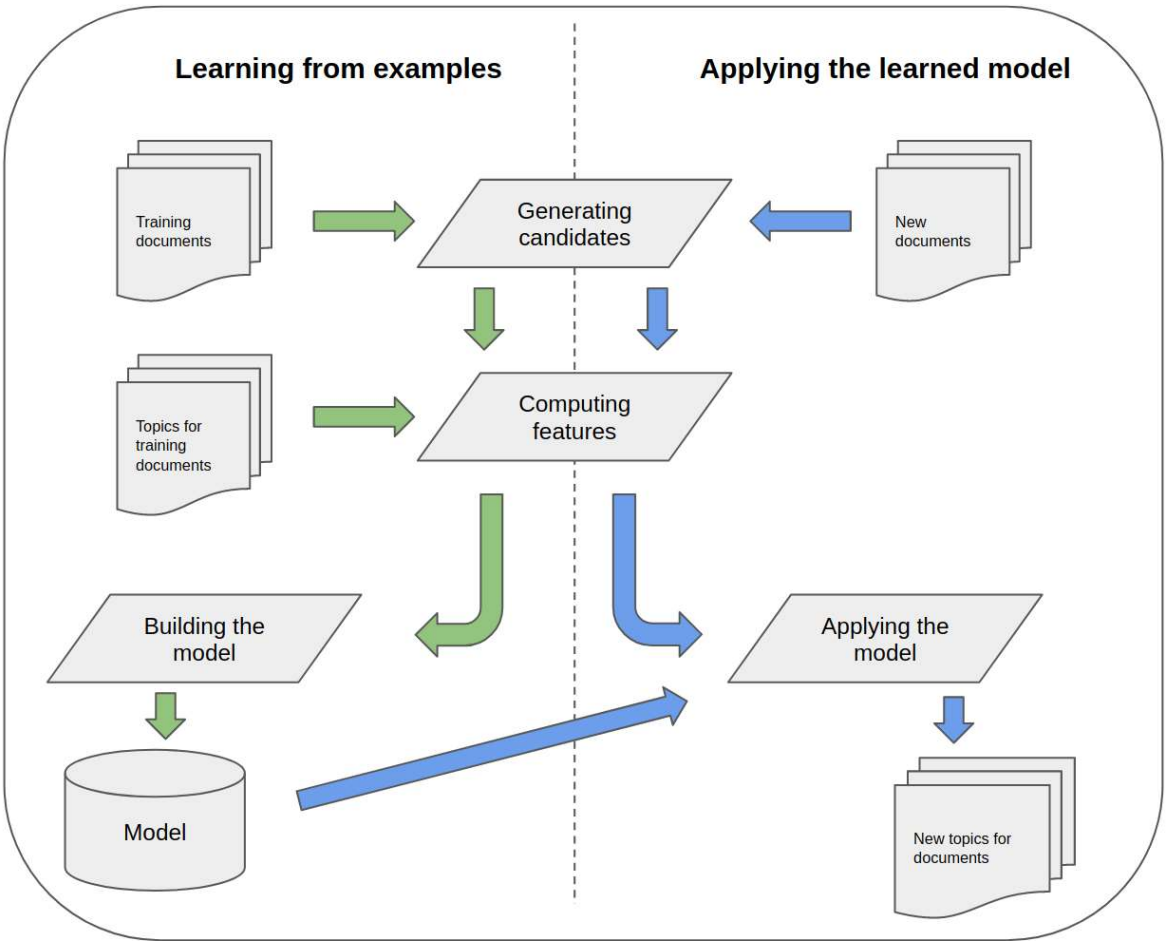

Figure 9 Summary of the different phases when using Maui indexing scheme. Adapted from Medelyan (2009, p.108)

[13] Olena Medelyan, Human competitive automatic topic indexing, accessible: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.178.2104&rep=rep1&type=pdf

A document text is given as an input to the *candidate generation* algorithm. The output of this process is a list of candidate keywords and -phrases found both in the document text and the controlled vocabulary. The terms are derived using a five-step process:

1. Extract all n-grams (short sequences of words[14]) where n should match the longest term in the controlled vocabulary.
2. Normalize both n-grams and vocabulary terms. Normalization includes conversion to lowercase, stopword removal, stemming and word re-ordering (see chapter standard NLP-procedure for more information).
3. Map extracted n-grams to vocabulary terms.
4. Apply semantic conflation: if extracted n-gram maps to a non-descriptor (i.e. alternative label of a defined concept in the vocabulary), replace it with the preferred descriptor (e.g. document contains word “Timber” which is alternative label to concept “Wood”: “Timber” is replaced by “Wood”).
5. For a given n-gram, compute all vocabulary descriptors that represent a possible meaning of the phrase (extracted n-grams can be ambiguous and match more than one descriptor e.g. *Jaguar* is mapped to both car and animal, stemming also amplifies ambiguousness, e.g. *Product* is mapped to *Products, Productivity* and *Production* when stemming is applied). (Medelyan, 2009, p.80–81)

In the *candidate generation* -phase all matching vocabulary terms are used as possible candidates for indexing. Here it is important to note that this process constraints the possible keywords and -phrases to the ones found verbatim from the document text, since the algorithm contains no associative features between words i.e. the *meaning* of words or sentences is not considered when indexing terms are extracted. (Medelyan, 2009, p.79-80) For example a sentence: “Urho Kaleva Kekkonen was the president of Finland between the years 1956 - 1982” would not produce key phrases such as “Political history” or “Politics” as these are not mentioned directly in text.

After a pool of candidate keywords is obtained from the document text, the collection needs to be filtered to detect the most suitable terms for indexing. The second phase, *filtering*, ranks candidate terms according to a selected set of features that are computed based on both training and input documents. The aim is to separate positive and negative candidate topics based on the computed feature values (Medelyan, 2009, p.102). The different

[14] E.g. n-grams from sentence “Mary likes John”: 1-grams are individual words “Mary”, “likes”, “John”; 2-grams “Mary likes” and “likes John”; and 3-grams “Mary likes John”

features include term frequency metrics, term occurrence statistics, domain keyphraseness, semantic relatedness and term specificity measures. These metrics are explained shortly in table 2 for a more thorough explanation refer to Medelyan (2009, p.90-101).

Table 2 Maui indexing - features for filtering and ranking the candidate topics. (Medelyan, 2009, p.90–101; p.119)

Feature	Explanation
Term frequency (TF)	Identify terms that appear most frequently in each document.
Inverse document frequency (IDF)	Identify how common terms are in the whole document collection. Words that appear in most documents will be assigned low scores and rare words appearing in only a few documents are assigned high scores.
TF-IDF	Combines TF and IDF information to identify words that are both rare in the whole collection and frequent in particular document(s).
Occurrence position (first & last)	The position of the first or last occurrence for each candidate relative to the number of words in the document. Candidates with extreme (high or low) values are deemed more likely to be good indexing terms.
Domain keyphraseness	Record the number of times a specific candidate appears in the training set as a keyphrase. Terms appearing more often in the training set are considered good *domain-specific* keywords and used when possible.
Node degree	Computed for candidates whose semantic relations are described in a vocabulary. The number represents their related candidates in the given document divided by the total number of all candidates. Candidates with higher node degrees are likely to be better keywords.
Term length	Captures the length of candidate terms. Longer terms are considered more specific and therefore better keywords.

After feature values for all candidate terms are calculated, the comparison and ranking are executed by using a machine learning algorithm. Training data is used to create a model for separating positive candidates from negative based on the above mentioned - numeric - feature values (Medelyan, 2009, p.102-104). The usefulness - in separating the negative terms from positive - of each individual feature is often dependent on the used vocabulary and domain. Machine learning algorithms are used to discover the proper weighting scheme for feature values (i.e. a mapping function from numeric feature values to a probability of being a keyword) based on the available training data. (Medelyan, 2009, p.104)

4.2.3 Associative algorithms

Next two associative subject indexing algorithms are introduced. Vector Space Model (VSM) is introduced as it develops basic understanding of the concepts surrounding vector representations and text similarity measures. After VSM, Omikuji - an efficient implementation of a tree type algorithm using machine learning classifiers - is presented. It builds label representations on similar concepts as VSM but uses more sophisticated methods for model training and label prediction. In the project phase of this thesis, both Omikuji and VSM models are evaluated in combination with Maui to implement an efficient algorithm for term assignment.

In addition to subject indexing algorithms, an unsupervised machine learning method for clustering is introduced. The method is leveraged in Omikuji to partition the label presentations into a tree structure based on the vectorized TF-IDF text documents. To properly introduce the indexing process of Omikuji, it is also necessary to separately acquaint with the basic concepts of the spherical k-means clustering method.

4.2.3.1 Vector Space Model

Vector Space Model (VSM) is a statistical model for representing information (e.g. terms, documents, images, audio, etc.) using vectors. (Kim, 2015) In the context of term assignment VSM provides a model for representing the terms in a controlled vocabulary through weighted vectors, making it possible to calculate similarity measures between document text and vocabulary terms. The obtained similarity measures can then be used for ranking most suitable candidate terms from the controlled vocabulary. Using VSM requires a controlled vocabulary and an existing set of manually indexed documents - i.e. training data (Suominen, 2019). Ideally every term in the vocabulary should be linked to a set of indexed documents as this data is used to build vectors that ultimately represent each vocabulary term. Terms that aren't linked to any indexed documents will result as a zero-vector and thus won't be assigned to any new queries (i.e. these terms are not used for indexing).

To create the term specific vectors (e.g. figure 9 vectors $t_1, t_2\ or\ t_3$) words contained in the manually indexed documents are concatenated and some scheme of *term weighting* is applied. Simplest approach is to use a boolean indicator (true or false; 1 or 0) that describes the presence or absence of a word - as in figure 9 top most vector t_1 where words $w_1\ and\ w_3$ are present - marked with one - and $w_2\ and\ w_n$ are absent - indicated by zero. (Munteanu, 2007, p.44-45) Since the boolean vector representation only captures information of word existence, an obvious refinement to this is to also include word frequency (Term Frequency,

TF) information (Munteanu, 2007, p.45). This can be done by counting the times a specific word appears in the indexed documents and then storing this information to the corresponding term vector - example of this in figure 9 where the middle t_2 vector shows that w_1 appears five and w_3 three times in the term specific documents.

However, as we have already seen with stopwords (see section Stopword removal) relevance of a word does not increase proportionally with word frequency - i.e. words that appear most frequently are often not most relevant. To account for this term frequency (TF) can be combined with *inverse document frequency (IDF)* to represent a heuristic for term importance (Munteanu, 2007, p.45). Where TF-part focuses on the local frequency of a particular word (i.e. how many times does word w_1 appear in the indexed documents belonging to a particular term, for example t_2), IDF-part on the other hand focuses at the global frequency of the word (i.e. how many documents in the whole collection contain the word w_1 - called document frequency), and as the name suggests calculates an (logarithmic) inverse of that result. (Munteanu, 2007, p.45; Kim, 2015, p.2) When combined to a single metric (*TF-IDF*) words that appear often in a specific document (locally frequent) but rarely in the collection (globally rare) are given high value, which seems logical as these words have high discrimination power over the document collection and are likely to be descriptive of a specific document's content. In figure 10 the bottom t_3 vector is created using a TF-IDF metric where w_1 has a weight of .3 and w_3 weight of .53 - this indicates that even though the word w_3 appears only three times (compared to $w_1$which appears five times) it is less common in the whole collection and is therefore given a higher weight than w_1. Here it should also be noted that the presented term weighting schemes are all separate and only one of the vector representations is used at any one time.

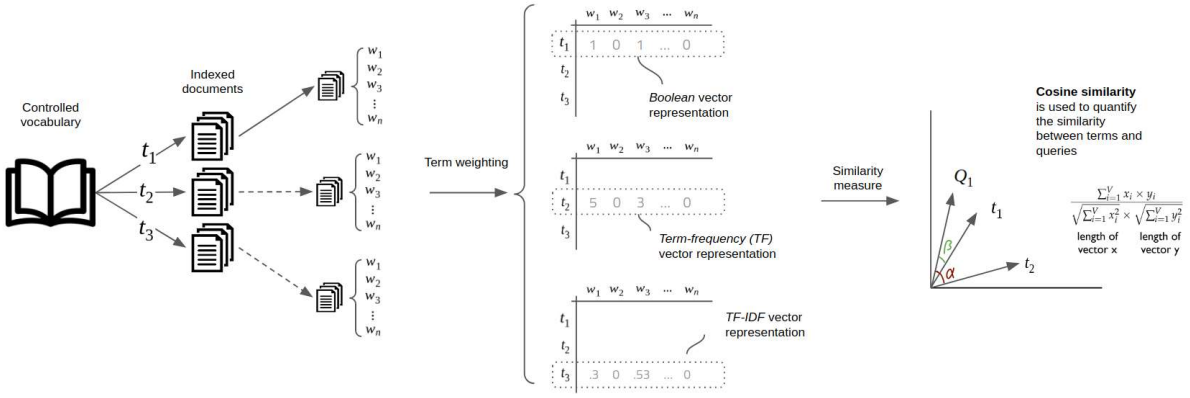

Figure 10 Vector Space Model (VSM) in the context of term assignment. Vocabulary terms are represented through weighted vectors and new queries are indexed by computing cosine similarity (i.e. angle between the vectors) between query and term vectors, making it p

A vectorized representation of the vocabulary terms allows to build a geometric definition of similarity between different information objects (Kim, 2015, p.2). Similarly, to vocabulary terms, also query terms can be vectorized, which in turn leads to the possibility of correlating queries to existing vocabulary terms, making term assignment possible. A standard approach for assessing similarity between two documents is to calculate the *cosine similarity* of the documents' vector representation. (Manning et al., 2009, p. 120-121) This works by first calculating the dot product (numerator in figure 10) between the documents - which essentially multiplies together term weights for words which exist in both documents (e.g. using boolean term weighting computing the *dot product* gives the amount of mutual words in the documents[15]) - and then length-normalizing the document vectors' using Euclidean length (denominator in fig. 10) - consequently eliminating effects stemming from the fact that longer documents tend to always have higher term frequencies than short documents (Manning et al., 2009, p. 121-122). The result of this computation is the cosine angle (marked with symbols α and β in figure 10) between the documents, which can be used as a proxy for similarity - the smaller the angle between query and term, the more they are alike - to rank different terms in order from *"most* to *least* like*"* the given query. For term assignment usually the top K terms are chosen or suggested as index terms.

4.2.3.2 K-means clustering

K-means is an *unsupervised method* for clustering data. It is used to detect structures in the dataset and find subgroups of similar data points in situations where no labeled examples are available - which is in contrast to *supervised* learning approaches, where the aim is to train a classifier to detect such groupings based on existing labeled examples (e.g. neural networks). (Alashwal et al., 2019) K-means is a relatively simple, yet effective algorithm for clustering: as an input it takes vectorized data (e.g. document vectors) and a freely specified parameter k for the amount of clusters, as an output k-means provides a partitioning of the dataset based on an optimization scheme (e.g. minimizing mean-squared error (MSE) or maximizing cosine similarity). Next a brief overview of the standard k-means - using MSE as the optimization goal - is given. Furthermore, a modification of the standard k-means called *spherical k-means* is also discussed, as this technique is used for label presentation in an indexing algorithm called Omikuji (see section Omikuji - Parabel). However, the intent of this introduction is to convey basic intuitions of k-means by summarizing the basic steps involved, a rigorous mathematical explanation is beyond the scope and an interested reader

[15] For an in-depth explanation see https://ils.unc.edu/courses/2013_fall/inls509_001/lectures/06-VectorSpaceModel.pdf slides 13 - 30

may refer to Dhillon et al. (2002), Zhong (2005) or Hornik et al. (2012) for a more in-depth presentation.

Standard k-means is an iterative process which consists of assigning data points to cluster centroids, re-calculating cluster centroid positions, moving them and then repeating the process until it converges so that cluster centroids are stationary (i.e. do not move - or move very little - when the process is repeated) and data points have a constant centroid. (Alashwal et al., 2019) In figure 11 this process is summarized: first the data is vectorized - using for example VSM (see Vector Space Model) - and cluster centroids are initialized randomly (steps 1 and 2). Next the data points are assigned to the closest centroid by calculating the Euclidean distance between each centroid and data point (step 3). After each data point is assigned to a centroid, the centroids' positions are re-calculated and moved to the mean of the points that are assigned to them (step 4). This procedure - of assignment and centroid shifting - is iterated until the process converges and both the centroids and data points are stationary or move sufficiently little (steps 5 and 6). The overall optimization goal in the standard k-means is to minimize the mean-squared error which is the square of the sum of the distances from each cluster centroid to data points assigned to them. Trivially, the smaller this error gets, the more clustered the data points are around the cluster centroid. (Alashwal et al., 2019)

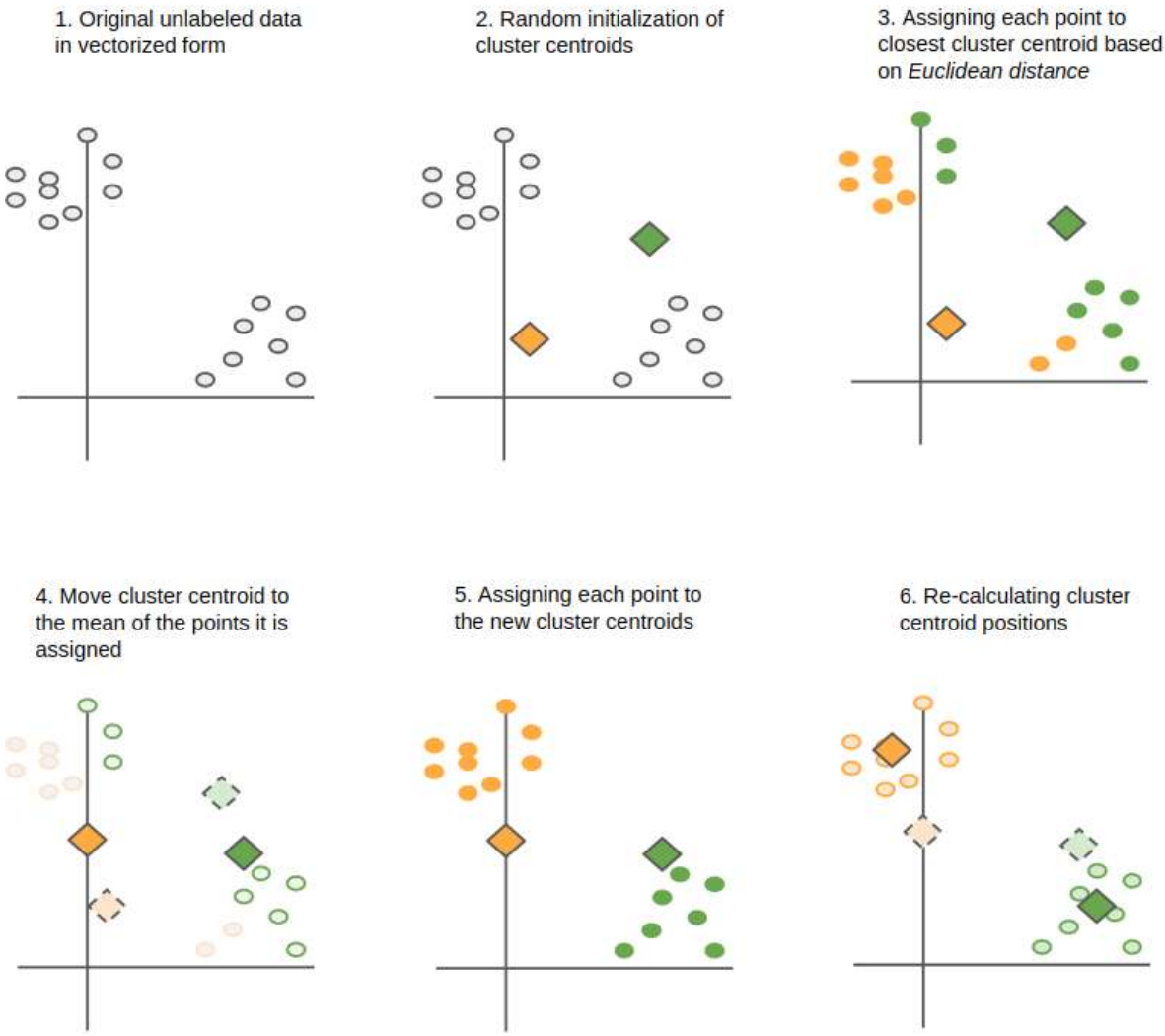

Figure 11 Basics steps of standard k-means.

In contrast to standard k-means, spherical k-means normalizes the data to unit vectors, randomly initializes concept vectors (i.e. same as cluster centroids in standard k-means) and uses cosine similarity (i.e. the angle between data point and concept vector) instead of Euclidean distance to assign data points to different clusters (Zhong, 2005). When clustering high-dimensional and sparse data (e.g. text documents in TF-IDF representation) it has been shown (Strehl et al, 2000) that cosine similarity should be preferred over metric distance measure - such as Euclidean - since it captures a scale invariant representation of similarity and is not dependent on the magnitude of the vector (e.g. length of the document). This means that different documents composed of similar content, but of different length, are treated identically and represented by a similar vector (Strehl et al, 2000; Zhong, 2005).

The overall process of spherical k-means is identical to the standard, apart from the optimization objective and representation of the data points and cluster centroids. This

process is summarized in figure 12. First the data points are normalized to unit vectors and represented on a unit sphere (in the figure 2-dimensional circle is plotted for visualization). Next concept vectors are randomly initialized (step 2). Unit vectors are assigned to concept vectors based on cosine similarity - concept vectors with the most similar direction (i.e. smallest angle between concept vector and unit vector) will be assigned to the unit vector (step 3). The directions of concept vectors are re-calculated based on the normalized expectation of the cluster (step 4). Again, this process of assigning unit vectors and shifting concept vectors is iterated until the process converges (steps 5 and 6). Here the optimization objective is to maximize the cosine similarities of different clusters so that clusters consist of unit vectors that have a similar direction with each other (as can be seen in step 6). (Zhong, 2005)

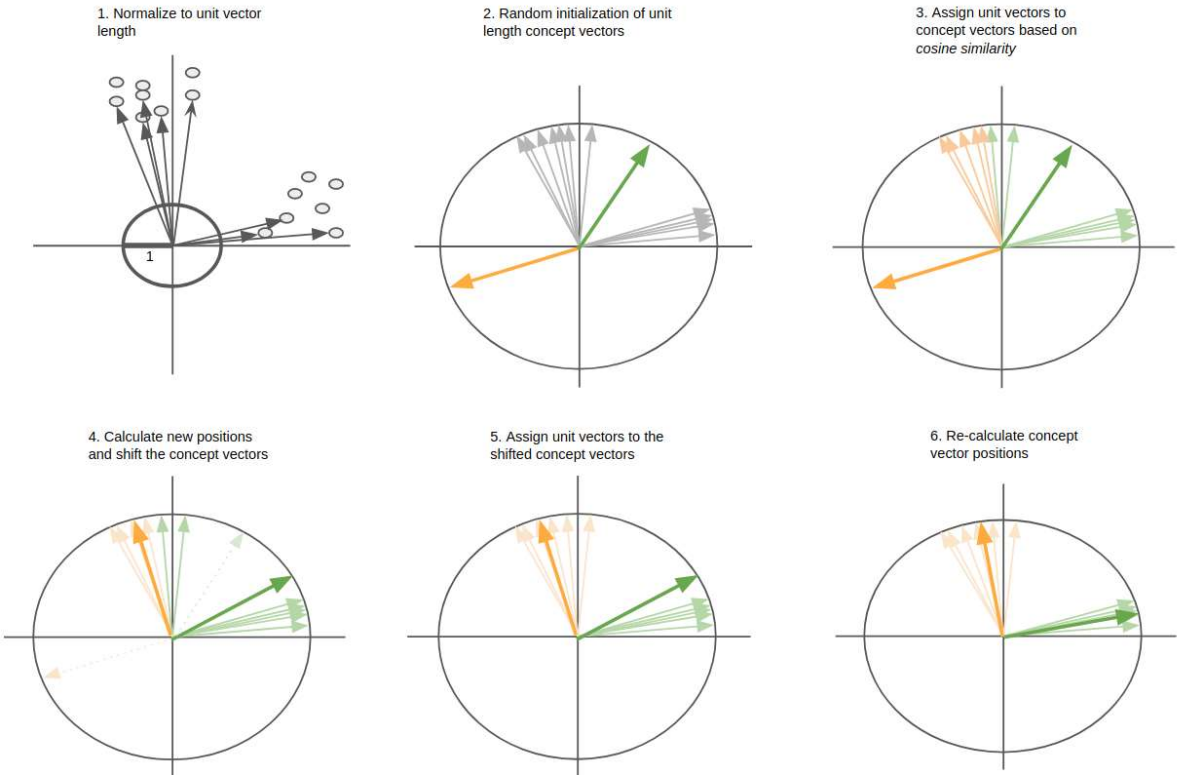

Figure 12 Basic steps of spherical k-means algorithm.

4.2.3.3 Omikuji - Parabel

Parabel is a tree-based machine learning algorithm for *extreme multi-label learning.* The algorithm is designed to annotate data with a relevant subset of labels from an extremely

large set of labels. (Prabhu et al., 2018) Omikuji - more precisely Omikuji Parabel - is the project name for an efficient open source implementation of the Parabel algorithm[16]. The Omikuji project also includes different variations of the Parabel algorithm - called Bonsai and AttentionXML - but these are not discussed in length. A ported implementation of Omikuji Parabel is estimated for term assignment in the project phase of this thesis.

The objective of extreme multi-label learning (also called extreme multi-label classification, XMC) is to predict the most relevant subset of labels when provided with a text input (Prabhu et al., 2018). The number of possible labels can range up to millions of indexing terms (see for example Amazon or Wikipedia tag sets[17]). An efficient approach to XMC can certainly be used for term assignment since the indexing vocabularies (i.e. thesaurus or ontologies) are considerably smaller than that of the benchmark tag sets used for XMC (for example the General Finnish Ontology consists of around 30 000 concepts[18]). The algorithm itself can be broken down to three high level phases: *Label presentation and hierarchy, model training* and *prediction*. Next a coarse overview of the main steps is given, for a more thorough explanation - with supplementary material on mathematical proof for used theorems - interested readers should refer to Prabhu et al. (2018) research paper on Parabel.

Label presentations are created in a two-step process after which spherical k-means is used to partition the labels (i.e. building the hierarchy of labels) to a balanced tree structure, an ensemble of up to 3 separate trees is usually learned. (Prabhu et al., 2018) First training points are transformed to vector representations by applying Vector Space Model (for more detailed explanation see section Vector Space Model). The exact method for term weighting is not specified in Prabhu et al. (2018) paper on Parabel, but most likely a scheme involving TF-IDF weighting is used. After the vector representations of training points (i.e. input text data) are created, label vectors are calculated as a mean of the training points containing the label and then normalized to a unit vector. (Prabhu et al., 2018) In Figure 13 L_1represents a label vector and x_3, x_4, x_5 are training points which contain the same label L_1. This type of representation captures the intuition that two labels are similar if they are active in similar training points (Prabhu et al., 2018). This means that the unit vectors for similar concepts will generally point in the same direction (e.g. if x_3, x_4, x_5 would share another label L_2labels' $L_1$and L_2unit vector representations would overlap) and contrasting labels unit vectors will point in different directions. Now with this premise, spherical k-means (with k=2) is used to partition the labels to a balanced tree representation. Spherical k-means uses cosine

[16] Implementation of Parabel https://github.com/tomtung/omikuji
[17] Datasets for XMC: http://manikvarma.org/downloads/XC/XMLRepository.html
[18] YSO - General Finnish Ontology: https://finto.fi/yso/en/

similarity (see section Vector Space Model for reference) as the similarity measure for clustering the concept vectors, which is a fitting heuristic for clustering similar concepts under shared nodes. Label partitioning is done recursively and terminated when a leaf contains less than M labels. (Prabhu et al., 2018)

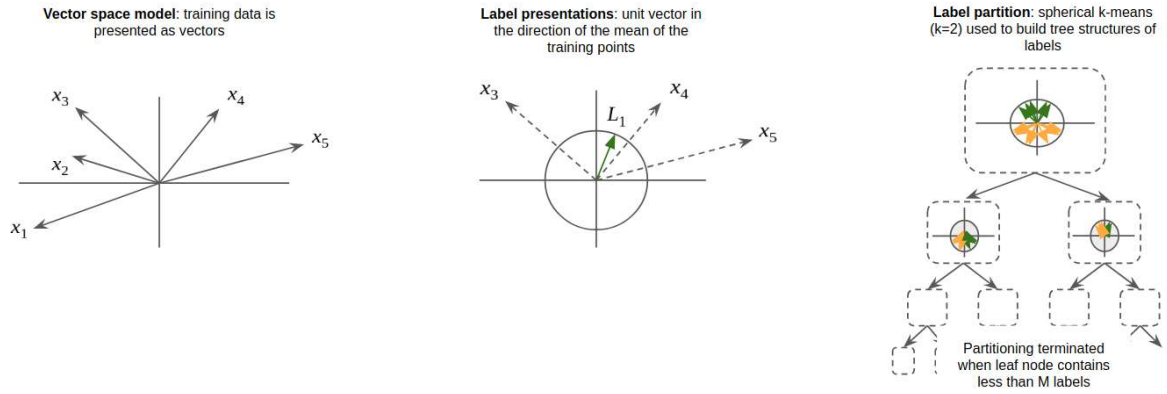

Figure 13 Label presentation is built using a vector space model (right), then grouping together training points that refer to the same label and representing the label with a unit vector in the direction of the mean of those training points (center). After all the label presentations are created spherical k-means (with k=2) is used to build a balanced tree structure (left).

After building an ensemble of trees (up to 3 label trees) training data is used to learn a hierarchical probabilistic model where the objective is to model the joint probability that a set of labels are relevant to a given data point (i.e. calculating the conditional probability of label L given some input text x) . (Prabhu et al., 2018) To achieve this, unique classifiers for both internal and leaf nodes are learned using the training data. For internal nodes the classifiers model the probability distribution of traversal from parent node to child node (or nodes) and the distribution can be sampled to determine which route to traverse (i.e. left, right or both child nodes). For leaf nodes 1-vs-All classifiers are trained for each label in the leaf node which can then be used for predicting the most relevant labels. (Prabhu et al., 2018)

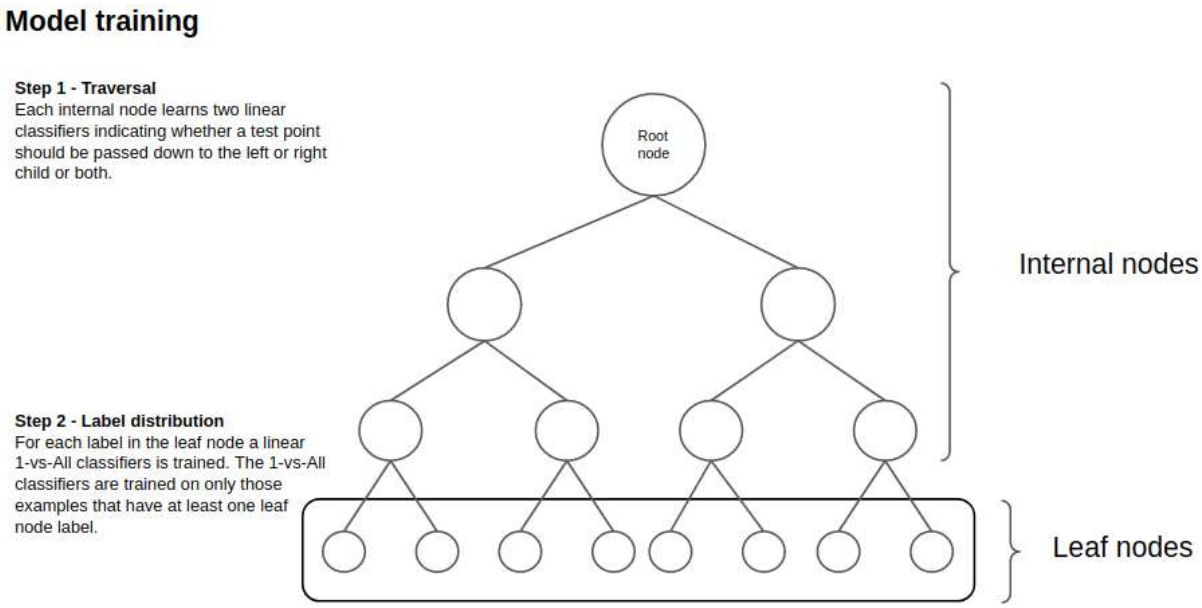

Figure 14 In the internal nodes training involves learning two linear classifiers for predicting whether to traverse left, right or both child nodes. In the leaf nodes 1-vs-All classifiers are trained for each label in the leaf separately.

After the internal and leaf node classifiers are trained, they can be used for traversing the label tree and sampling most relevant labels for new and unseen data points. The prediction method follows a greedy, breadth-first tree traversal strategy. (Prabhu et al., 2018) The assumption is that the most relevant labels for an unseen data point can be located by exploring the P most probable paths at each level of the label tree. First a test point goes through the internal nodes where classifiers are evaluated to predict which nodes should be traversed. Only the P most probable nodes are chosen at each level and other parts of the label tree discarded as non-relevant for the given test point. After leaf nodes are reached the 1-vs-All classifiers in these leaves are evaluated and as a result the corresponding labels can be ranked from most to least relevant based on the classifiers' probability measure. Finally, prediction of most relevant labels can be made by averaging the probabilities across all the trees in the ensemble. (Prabhu et al., 2018) Summary of the process is pictured in figure 15.

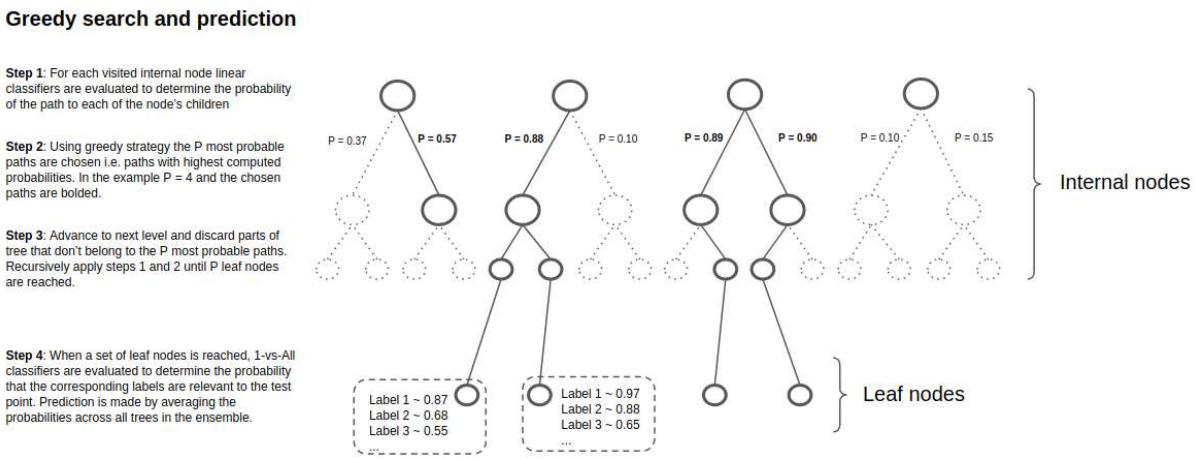

Figure 15 In the internal nodes two linear classifiers are evaluated to determine which nodes to traverse. Strategy of traversal is based on greedy search as only the P most probable paths are chosen at each level. After leaf nodes are reached, 1-vs-All classifiers are evaluated to determine the probability that a corresponding label is relevant to the input text

Annif's[19] implementation of Omikuji is based on an implementation[20] of Partioned Label Trees (Prabhu et al. 2018). In addition to Parabel-type label trees - introduced in the previous section - the Omikuji implementation supports hyperparameters that can be used to control the width and depth of the label trees. This allows to also build unbalanced label trees - known as Bonsai trees - which are shallow and wide (i.e. trees that have less depth but more width). These hyperparameters are introduced shortly in table 3.

[19] Subject indexing tool developed by the National Library of Finland http://annif.org/

[20] Further implementation details https://github.com/tomtung/omikuji

Table 3 Omikuji's adjustable hyperparameters and their short descriptions. Adapted from Suominen (2020).

Parameter	Description
limit	Maximum number of subjects to return
min_df	Number of documents a word must appear in to be considered. E.g. if value is 1 all words are evaluated, if value is 5 only words that appear in 5 different documents are used.
cluster_balanced	Perform balanced k-means clustering instead of regular k-means clustering.
cluster_k	Number of clusters.
max_depth	Maximum tree depth.
collapse_every_n_layers	Number of adjacent layers to collapse.

4.2.4 Ensemble algorithms

Neither lexical nor associative approaches work flawlessly on their own - at least at the time of writing - as both techniques and related algorithms all have their individual drawbacks. There is a multitude of factors that contribute to incorrect term assignments; Homonyms (similar words meaning different things e.g. "bat" can refer to a nightly animal or a baseball bat), misinterpreted names (e.g. "Smith" as a surname or a profession) biased training data or simply random noise all effect term assignment and can be sources for possible mistakes (Suominen, 2019).

As different algorithms tend to make different types of mistakes, a feasible strategy for increasing quality of term assignment is to combine different algorithms and average the individual predictions - analogues to for example tree ensembles where a group of weak learners are joint to form a strong learner. These techniques are generally referred to as *ensemble* or *fusion architectures*. (Toepfer and Seifert, 2018) Next the aim is to introduce two different ensemble methods, that can be used to combine previously examined lexical and associative algorithms. In addition to the ensemble models, a general overview of neural

networks is given, as they are leveraged in one of the ensemble models and having a basic understanding of the paradigm is necessary.

4.2.4.1 Simple ensemble

The simple ensemble merges together different predictions by calculating the mean value for each subject score predicted by the underlying algorithms (Suominen, 2019). It can be considered somewhat like a random forest approach where the final predictions are calculated as a mean score from the individual trees. Here only instead of decision trees the underlying algorithms are different indexing algorithms which output probability scores for a set of subjects. Using Annif it is also possible to set individual weights for each indexing algorithm, effectively calculating a weighted average. (Suominen, 2019) This feature can be used as an adjustable hyperparameter, to optimize for a specific metric. Simple ensemble requires no training since calculating the mean - or a weighted mean - of the score vectors is purely a mechanical task.

4.2.4.2 Neural Network

Neural networks provide a method of supervised learning meaning that they can be trained using manually labeled examples. In the context of subject indexing supervised learning methods are of interest since manually labeled data is readily available - e.g. metadata from library collections - and can be used as training data for supervised learning algorithms. In this section the aim is to briefly explain the main concepts around neural networks (NN) - such as basic structure of NN's, activation and loss functions, use of weights, etc. - and convey key intuitions of the training process of a neural network by introducing the process on a single neuron level. Although the training process is similar when the number of neurons and layers are scaled, there are a multitude of optimizations - such as efficient vectorization, regularization, etc. - which are not covered in this short introduction. For a more thorough explanation, readers should refer to Nielsen (2015), Hayak (2005) or Ng and Katanforoosh (2018).

Neural networks are presented as connected layers of nodes. A simple two layer[21] NN - such as the one depicted in figure 16 - consists of an *input layer, hidden layer,* and an *output layer*. Each layer consists of nodes (or neurons) that are connected to the next layer's nodes by a *weight*, which is a scalar value representing the strength of the relationship between the

[21] Note that the naming convention does not include input layer since no computations are performed there (Hayak, 2005, p.43)

two nodes. In the example the layers are *fully connected*[22] (i.e. each node in one layer is connected to every other node in the next layer) if some connections were missing the network would be called *partially connected*. (Hayak, 2005, p.43-44) Moreover, the output of a previous layer is always used as an input to the next layer, resulting to a *feedforward* architecture (i.e. the network does not include loops where information would be fed back to a previous layer) and the full name of the introduced network is "*Multilayer Feedforward Neural Network*". (Nielsen, 2015, p.11-12) Although architectures with such loops do exist - called recurrent neural networks - those are not introduced here since they are out of scope of this thesis work.

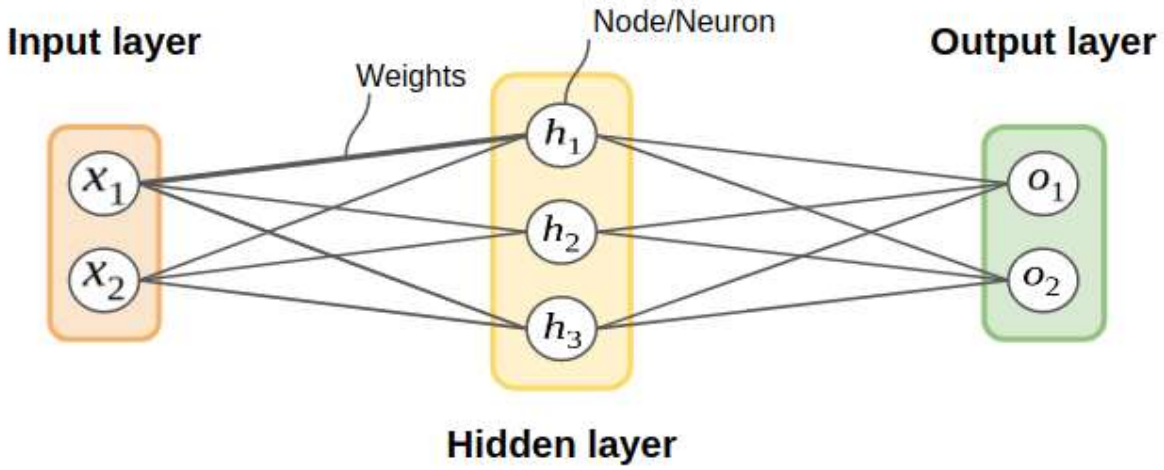

Figure 16 Structure of a fully connected multilayer feedforward neural network with one hidden layer.

The design of input and output layers is often straightforward. *Input layer* is responsible for supplying source elements or features to the network. For example in the case of a text document, this could be TF-IDF[23] vector, in which case the size of the network's input layer would reflect that of the size of the TF-IDF vector (1xn, where n is the amount of words in the document collection). On the other hand, the design of the output layer depends on the number of classes that are predicted. For example if the input is a text document and the aim is to index the document in one of 5 categories (e.g. a news article as an input and a prediction of a category such as politics, sports, finance, etc. as an output) the output layer should consist of 5 nodes - one for each category. (Nielsen, 2015, p.10-11) However, the design - size and amount - of the hidden layer(s) (i.e. the layers between input and output layers) is somewhat less straightforward and the amount of layers used is dependent on the classification task.

[22] Also referred to as "Dense layers"

[23] See section "Vector Space Model" for further clarification

A single-layer network (with no hidden layers) can only learn linearly separable decision boundaries. For this approach to work the classification problem should be relatively simple and the different classes quite distinctly different (e.g. in a 2-dimensional example the data should be separable by a line). On the other hand, a two-layer NN (one hidden layer with some activation function) can be used to approximate any function when a sufficiently large amount of nodes is used - this is called the *universal approximation theorem* and it was first proved by Cybenko (1989) with Sigmoidal activation functions and later expanded to include any activation function by Hornik (1991). However, even though two-layer networks have the capability to represent any function (i.e. create any input-output mapping) in many cases the layer size (i.e. the amount of hidden nodes) can grow infeasible or the network might fail to learn the correct mapping resulting to less than optimal performance. To mitigate these issues, networks with multiple hidden layers are often used as they can reduce the number of nodes needed to represent the desired mapping. (Goodfellow, Bengio and Courville, 2016, p.194 - 198)

Now that we have developed an understanding of the structure of a neural network, next we'll take a more granular look and discuss the training process on a single neuron level. Training consists of an iterative process where the network is constantly represented with labeled training data (e.g. the document vectors of which the indexing labels are known beforehand). The inputs are passed through the network in what is known as the *forward-propagation step* and the predicted output is then compared to the target label (i.e. the actual label of the document vector) using a loss function (Nielsen, 2015, p.16). The closer the predicted value is to the target value, the smaller the measured loss is. At first the loss will be high since the weights are randomly initialized. In the *back-propagation step* the weights are then adjusted so that the training error (i.e. measured loss) is eventually minimized, resulting to an input-output mapping that correctly predicts the target values based on the training data (i.e. produces correct indexing label or labels on a given text document). (Parker, 2007)

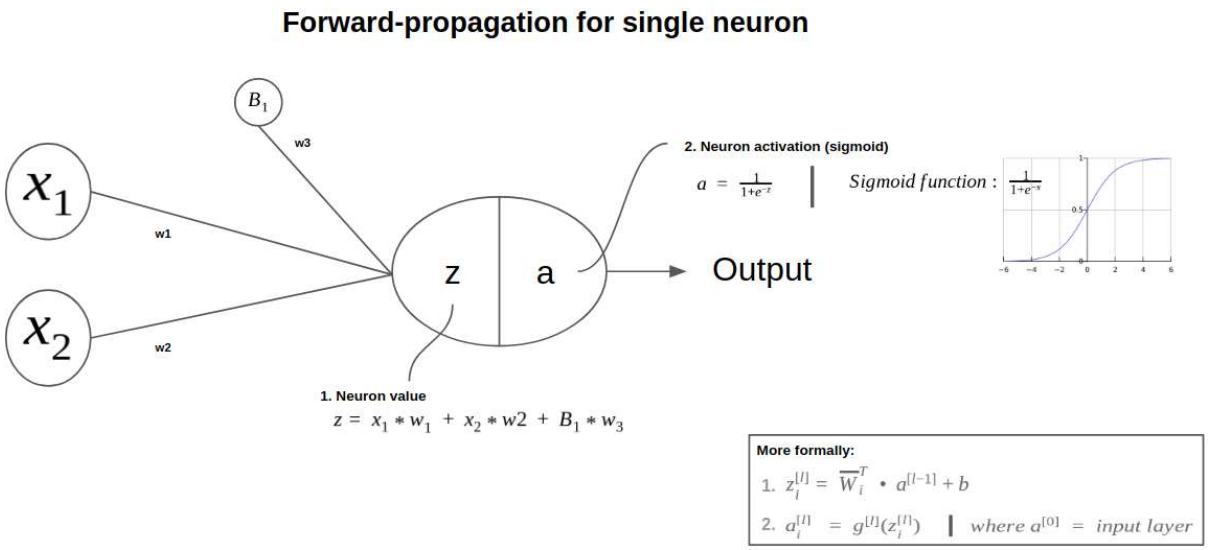

Figure 17 Example of feed forward phase with a single neuron.

To introduce the forward-propagation step a bit more formally, we'll look at an example (depicted in figure 17) consisting of two input neurons (x_1, x_2) and a bias-term (B_1) which are fed forward to a single neuron in the next layer. First the weights are combined with the input values used to calculate a weighted sum (called *neuron value* in figure 16) of the inputs and adding a bias term[24]. (Nielsen, 2015, p.6-8) Next a non-linear activation function (sigmoid function in figure 16) is applied to calculate the final output of the neuron. Activation function converts the *neuron value* - a linear input - to a non-linear output which is needed in order for the neural network to model high order polynomials - which is desired since modelling real world data generally requires learning complex patterns. (Nwankpa et al., 2018, p.3-5) If an activation function is not used (or the used activation function is linear), adding multiple hidden layers is redundant since the resulting NN will only perform linear regression (i.e. the model only explains linear relationships). (Ng and Katanforoosh, 2018, p.7) In the example Sigmoid function is used as the activation function since it is non-linear and differentiable - which is required for the backpropagation step to work. However, numerous other activation functions can be found in the literature - a comprehensive list of state-of-the-art activation functions and their use can be found from Nwankpa et al. (2018). After the activation of the neuron is calculated the result is passed on to the next layer where similar calculations are performed and passed forward.

[24] Bias term works similar to an intercept term in linear equation, allowing the activation function to move horizontally. This property enhances the flexibility of the node. For further details and an example, see Nielsen (2015, p.6-8).

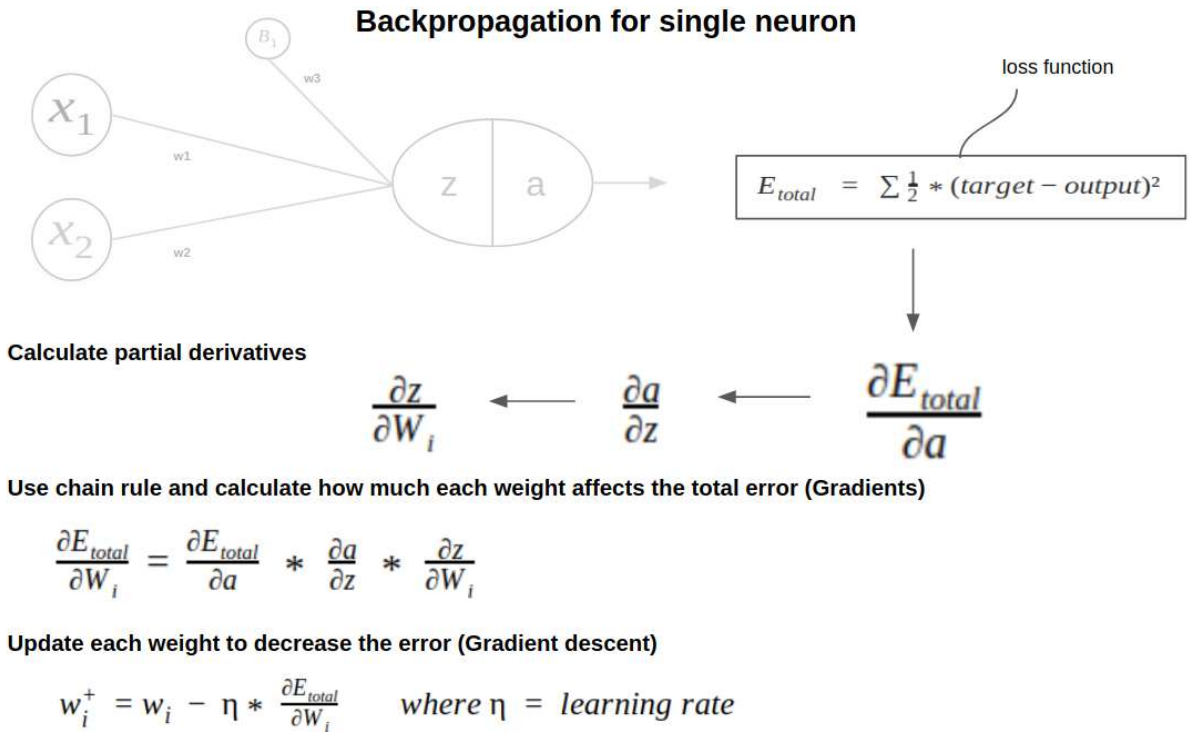

Figure 18 Example of a backpropagation step with a single neuron and a squared error loss function.

After the forward-propagation step the network outputs a prediction based on the input it received. At first the network's predictions are random, since they are dependent on the node weights which are randomly initialized. In the *backpropagation step* these weights are updated, so that the measured loss is minimized (i.e. the error between the prediction and the actual value is gradually reduced). (Nielsen, 2015, p.16-18) The loss - or error - of the output is measured using a loss function. In the figure 18 Mean Squared Error[25] (MSE) is used as an example, however, many other loss functions exist[26]. After a single forward pass an output value is predicted by the network, using the loss function we can compute a scalar value that represents the total error that the network produces. To minimize the error we first calculate the partial derivatives (for this to be possible all transformations in the forward-propagation step need to be differentiable, as mentioned already with activation functions) for each weight and use the chain rule to calculate the portion each weight has on the error. Finally, the weights are updated based on the calculated gradients, here a learning rate can be used to modify the speed that the weights are changed. If the chosen learning rate is too high the network might not converge (i.e. the error does not decrease) and if the chosen learning rate is too small, the learning process is very slow, and progress hindered. In practice the learning rate should be considered as a hyperparameter, that can be adjusted as needed. (Nielsen, 2015, p.20) This procedure of forward and back-propagation is then

[25] Also known as L2 loss or Quadratic Loss

[26] See Janocha and Czarnecki (2017) for a more thorough list of Loss Functions and their use

iterated until the error rate converges or a desired level of total error is achieved, and the training stopped. (Ng and Katanforoosh, 2018, p.10-15)

4.2.4.3 Neural Network ensemble

Neural Network (NN) ensemble implements - as the name suggests - a neural network-based solution to combining term suggestions from different backend algorithms. The process of combining and re-weighting suggestions from the backend algorithms is simple. It consists of a shallow neural network (with one hidden layer) and a procedure for calculating the mean vector based on the backend suggestions (process summarized in fig. 18). Initially, (before any training) NN ensemble behaves similar to a simple ensemble, where mean values are calculated for each subject based on the predictions from the different backend algorithms (Suominen, 2020). This happens since the neural network's output weights are initialized as zero, which means that at first the NN has no effect on the output when the computed mean values and output from the NN are summed to get the final result and ranking for subject terms. Training data is then used to fine-tune the output from the neural network so that the NN also contributes to the re-weighting process.

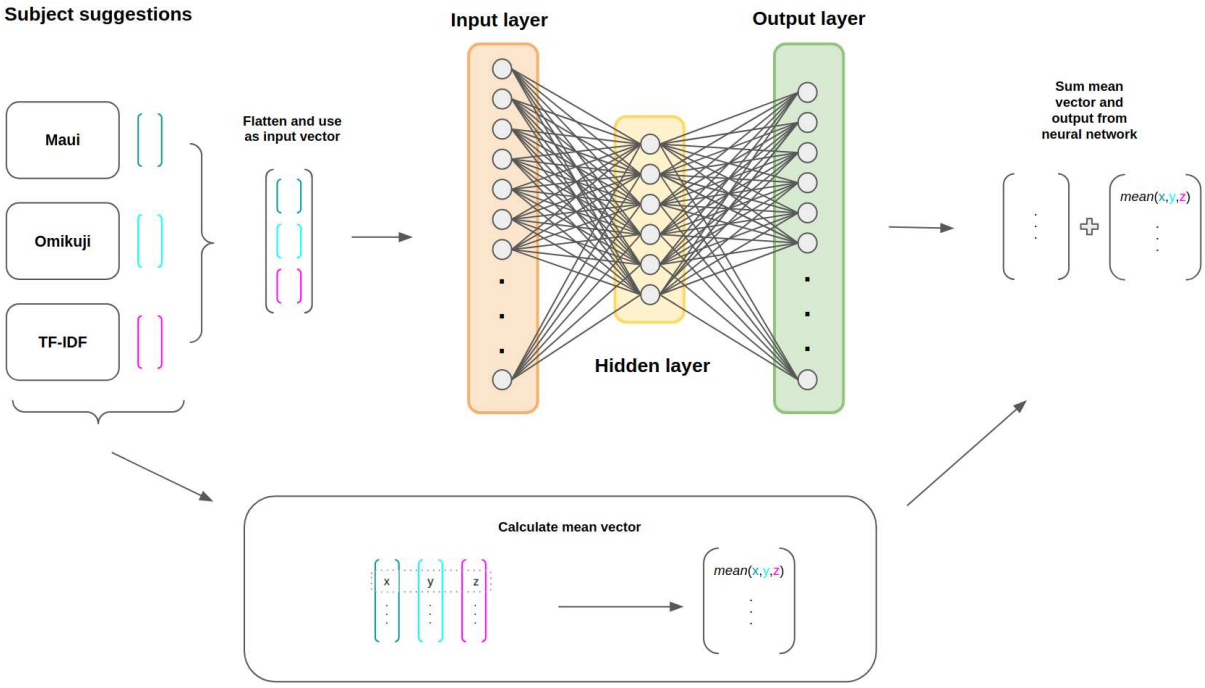

Figure 19 Neural network (NN) ensemble implements a trainable model for combining results from different indexing algorithms. First subject suggestions are queried from the backend projects (Maui, Omikuji and TF-IDF in the figure). Suggestion vectors are used as an input to both NN and for calculating the mean vector - as in simple ensemble. To obtain a ranking vector both the mean and output vector from NN are summed. Initially (i.e. with no training) NN ensemble works like a simple ensemble as the output weights from NN are initialized to zero, training is then used to fine-tune NN's output. (Suominen, 2020)

Training the neural network is a separate process from training the underlying algorithms. First the subject suggestion backends (Maui, Omikuji and TF-IDF in figure 19) are trained separately - using the same or a different set of training data for each algorithm - so that they can be used for indexing unseen pieces of text. This needs to be done first since training the neural network involves querying the backend projects for subject suggestions and then - through the use of training data - learning an efficient mapping from suggestion vector to the actual indexing terms. (Suominen, 2020) Here it is good to note that the neural network in itself does not participate in indexing (i.e. it does not come up with new indexing terms as it has no inner representation of the term space or of the given input text) and it's merely used for re-weighting the suggestions already given by the associated backends so that they better fit the given training data. The NN is implemented using two Python programming libraries: Keras[27] and Tensorflow 2[28] (Suominen, 2020). When training the neural network there are different hyperparameters that can be tuned to optimize the end results. These parameters are briefly explained in table 4.

Table 4 Optimization parameters for NN ensemble's neural network implementation (Suominen, 2020).

Parameter	Description
Nodes	The number of neurons in the hidden layer of the neural network. Default amount is 100.
Dropout rate	Randomly drop neurons during training. Dropout rate -parameter controls the amount of dropout to apply between layers (between 0 and 1). Used for regularization and avoiding overfitting. Dropout defaults to 0.2
Optimizer	Defines the Tensorflow optimizer[29] to use. Defaults to "Adam" (for details see Kingma and Ba (2014)).
Epochs	The number of passes over the initial training data to perform.

[27] https://keras.io/
[28] https://www.tensorflow.org/
[29] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

5 Evaluation of term assignment

Second research question - specified in the introduction of this thesis - is about evaluation of different algorithmic approaches to term assignment. In the next chapter this question is approached from two different angles. First a process for evaluation is defined. This process includes the steps that are needed to make relevant judgements about different approaches to term assignment. Following the same process for each algorithm ensures that the evaluation results are consistent and can be used to estimate which approach is most appropriate. Second, the evaluation metrics used to estimate the fitness of each algorithm are presented. The metrics are used to both optimize and test each model and ultimately ranking the different approaches and leading to the choice of a particular model.

5.1 Evaluation process

Classification models are used for predicting and categorizing unseen examples. They ingest a characterization of some object - usually in the form of a vector or a matrix - and output a prediction based on the internal model learned from the training data. As the emphasis of use is on categorizing unseen data, the evaluation process also needs to reflect this so that we can gain an estimate on how well the model *generalizes* to new and unseen points of data. In practice this means that the goal in creating a classification model is not to fully learn the training data, but instead to extract a general model that applies to both training data as well as new and unseen data. (Xu and Goodacre, 2018) Learning an overly simple or complex model - exhibiting high bias or high variance respectively (fig 20) - will hinder the model’s generalization capabilities and ultimately lead to less than optimal results.

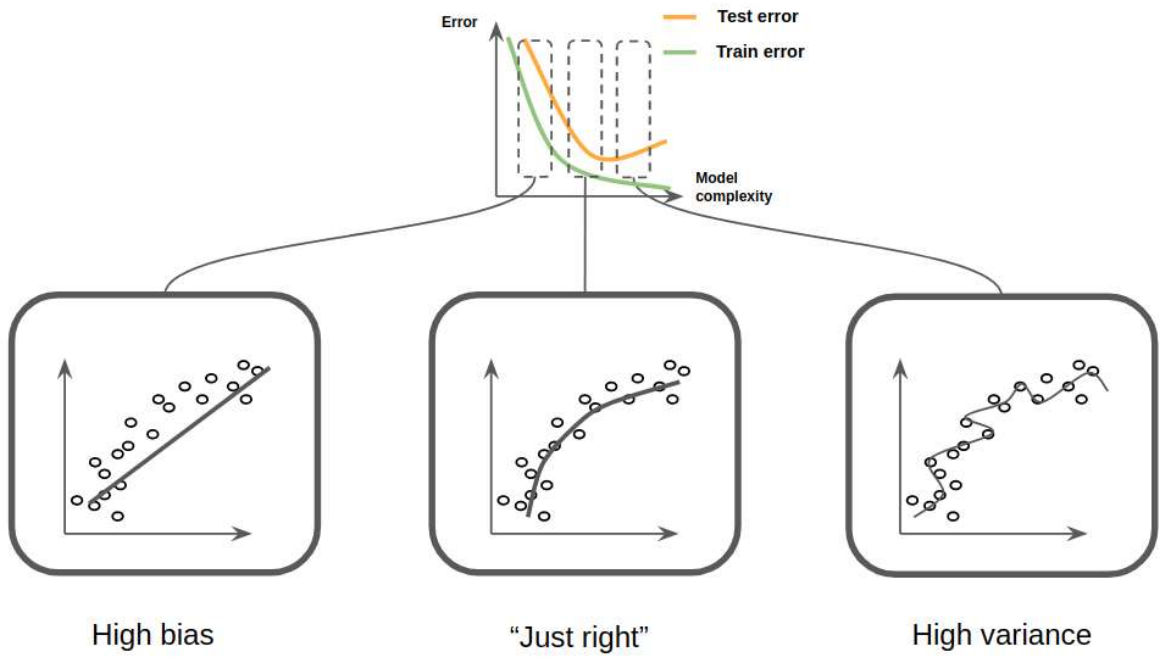

Figure 20 Shows the bias-variance tradeoff when creating a classification model. If the model doesn't have sufficient number of parameters (i.e. model's complexity is too low) it lacks representational power and bias is high - resulting to high error rate on both train and test data. If the model's complexity grows too high, the training data is overfitted and generalization properties lost - resulting to low training error and high-test error rate. Optimal model is therefore a tradeoff between bias and variance.

To evaluate an algorithm's generalization capabilities, available data needs to be partitioned into distinct sets so that both training and evaluation can be completed using separate sets of data. These two phases - training and evaluation - are often referred to as *model selection* and *error-rate estimation*. (Xu and Goodacre, 2018) First step in *model selection* is to partition the data to train, validation and test set. Majority of the data is designated to the training set and a marginal set is left out for the validation and test set (the exact train-validation-test ratio depends on the amount of available data). After the data is partitioned, training and validation data is used for model selection. Training data is first used to learn an initial model - with some default setting for hyperparameters such as amount of hidden neurons, drop-out rate, amount of trees, etc. - and after training model is evaluated using the validation set. This process of training and evaluation using the validation set is then repeated to fine-tune hyperparameter settings and achieve optimal performance. A simple method for hyperparameter optimization is to perform an exhaustive parameter sweep - called grid search - through a manually specified subset of different hyperparameter combinations. Another often used is *random search* where the combinations of hyperparameters are selected randomly and the model with the lowest validation error is chosen (Grosse, 2018).

After identifying the optimal model and tuning the hyperparameters to achieve best possible results, *error-rate estimation* needs to be performed. Since validation data is used repeatedly in optimizing the final model (i.e. tuning the hyperparameters), the error-rate estimated using validation data will be biased and report higher prediction accuracy than we should expect with new and unseen data (Gutierrez-Osuna, 2001). To achieve an unbiased estimate of the model's generalization capabilities, test data is used for error-rate estimation, as it is data that the model hasn't seen before and therefore predictions can be considered representative and unbiased. (Grosse, 2018) If there are multiple candidate models after model selection, the best performing (i.e. has the lowest error-rate) model should be chosen after error-rate estimation. Main steps of the training and evaluation process are summarized in figure 21.

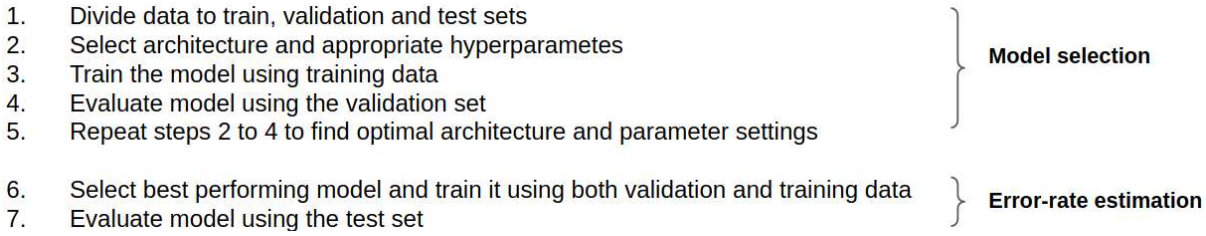

Figure 21 Training and evaluation process of a classification algorithm. Adapted from Gutierrez-Osuna (2001).

5.2 Evaluation metrics

In the previous section the evaluation process was broken down to incremental steps of model selection and error-rate estimation. In both these phases evaluation metrics are used to gain better understanding of how the implemented model is working and whether it can be further optimized. In this section the aim is to introduce suitable evaluation metrics for term assignment.

5.2.1 Accuracy, recall and precision

If asked to estimate the quality of a prediction algorithm, probably one of the first things that come to mind is to evaluate *accuracy* of the predictions. Accuracy of the model can be defined by calculating the fraction of predictions that the model predicted correctly i.e. by dividing the number of correct predictions by the total number of predictions. However, in cases where there is considerable *class-imbalance* between predicted classes, accuracy is not a sufficient metric for quality of the predictions (Classification: Accuracy | Machine Learning Crash Course, 2020). *Class-imbalance* refers to situations where proportions of negative and positive labels are highly divergent. For example, if we take a group of 100 patients of which we know that 5 have a malignant tumor - as it is a relatively rare event -, a

model predicting that all tumors are benign would result in an accuracy of 95%. The same dynamic is in play when evaluating term assignment, since a vocabulary holds tens of thousands of terms of which only a handful is used to index the input text at any given time, there exists considerable imbalance in the predicted classes. To better evaluate a model's prediction capabilities in these types of problems, *recall* and *precision* should be used.

To understand recall and precision, the concept of *confusion matrix* should first be introduced. *Confusion matrix* is a convenient way to summarize the results of a prediction algorithm by representing the actual and predicted classes in a 2x2 matrix and recording both correct and incorrect results. (Brownlee, 2016) As there are two classes (i.e. positive and negative) there are four different combinations of which two are correct and two erroneous. Trivially, predicting positive when the actual class is positive, will result in true *positive* and similarly for the *true negative* case. On the other hand, predicting positive when the actual class is negative will result in false *positive* prediction and predicting negative when the actual class is positive results in a *false negative* prediction. This breakdown of the prediction results gives a more detailed picture of the types of errors the classification algorithm is making and allows for calculating more specific evaluation metrics. An example of a confusion matrix is represented in figure 22.

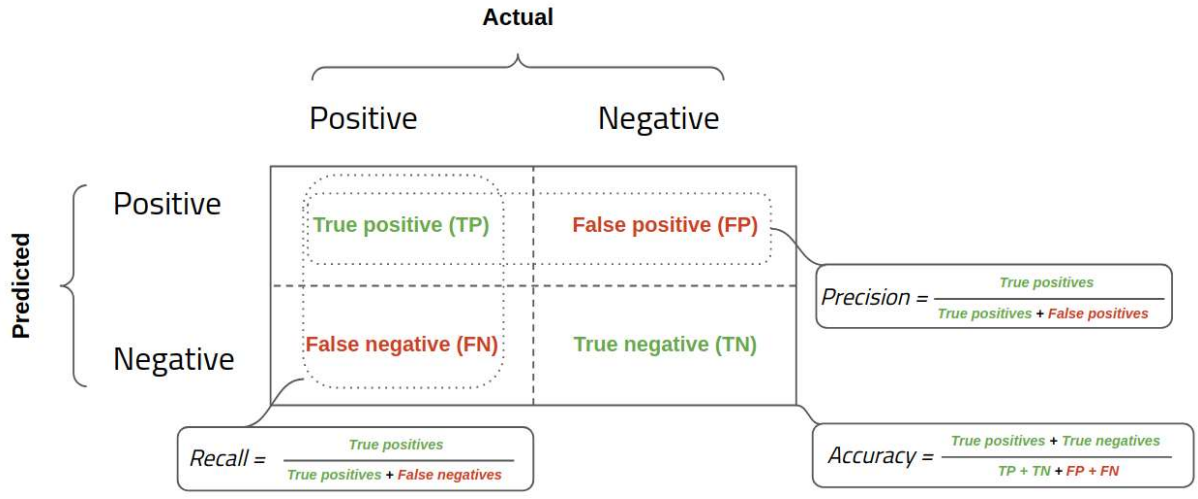

Figure 22 Confusion matrix for binary classification. Actual labels represent the true class (either negative or positive) and predicted class is given by the classification algorithm. Values can be used to calculate metrics such as accuracy, recall and precision.

Recall and precision are both evaluation metrics that focus in on the *true positive* classes (i.e. amount of correct predictions). However, both metrics zoom in from a different

perspective: *precision* measures the proportion of positive predictions that were actually correct (i.e. how precise the predictions were) and recall measures the proportion of actual positives that were correctly identified (i.e. how many actually positive data points were mislabeled as negative) (Classification: Precision and Recall | Machine Learning Crash Course, 2020). Going back to the class-imbalance problem discussed along with model accuracy, we can now easily see that for a model predicting only benign tumors both precision and recall would result in 0% as the model predicts zero true positive cases. When evaluating the performance of a classification algorithm both precision and recall should be examined. Usually improving both precision and recall at the same time is not feasible over a certain limit since both are dependent on the algorithm's classification threshold and optimizing one will hamper the other. If the threshold is increased (e.g. instead of 50% certainty, decision is made only when algorithm's certainty of a positive identification is over 70%) it is likely that the amount of false positives decreases and amount of false negatives increases, since increasing the threshold makes it more unlikely that the algorithm predicts a positive class. In turn, this results in higher precision and lower recall. Again, if the threshold is decreased the effect will be reversed (Classification: Precision and Recall | Machine Learning Crash Course, 2020).

When using precision and recall as metrics for estimating the effectiveness of a classification model, it is important to consider the classification task at hand to decide which of the two metrics is more suitable. For example in the first stages of screening whether a tumor is benign or malign achieving high recall is probably better than high precision, since it is important to detect all patients who actually have - or might have - a malign tumor. This can be achieved by decreasing the threshold for classification, which in turn will result in more false positives (i.e. predicting malign tumor when in fact the tumor is benign) predictions. This is acceptable, since the cost of a follow-up examination is considerably smaller than the cost of an unnoticed - and untreated - malign tumor (i.e. false negative). (Koehrsen, 2018) On the other hand, when classifying spam email precision should be optimized over recall, since having a high number of false positives (i.e. actually important emails classified as spam) wouldn't be user-friendly. Drawback from high precision would be that some spam email will get through, but again, the cost for removing spam messages occasionally is far lower than important mails being automatically removed. Precision and recall offer a more nuanced look into the errors the classification model is making and makes it possible to optimize the characteristic that is more suitable for the task.

5.2.2 F1-score

When there is no obvious reason for optimizing either recall or precision - as discussed in the previous section with tumor and spam detection - the main goal is often to find an optimal trade-off between the two metrics (i.e. to have both as high recall and precision as possible) . F1-score can be used to combine both metrics by calculating a harmonic mean of precision and recall. F1-score is a special case of the F-measure, where β is the relative importance given to recall over precision (Chinchor, 1992, p.25). In F1-score $\beta = 1$

$$F_\beta = \frac{(\beta^2 + 1) * Recall * Precision}{\beta^2 * Recall + Precision} \rightarrow F_{\beta=1} = 2 * \frac{Recall * Precision}{Recall + Precision}$$

Harmonic mean is used as it penalizes extreme values better than arithmetic mean. This behaviour is desired when measuring both recall and precision using a merged score. For example, an image recognition system having a 100% precision and 20% recall, would effectively have a very low 20% prediction rate. However, calculating arithmetic mean of recall and precision would result in a 0.6 score, whereas harmonic mean is only ⅓.

5.2.3 Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) is a measure of ranking quality which can be used to evaluate different information retrieval (IR) methods' (e.g. indexing or classification algorithms) ability to retrieve highly relevant documents. (Järvelin and Kekäläinen, 2002) It is calculated by accumulating the graded relevance judgements of the results, discounting the relevance score depending on the position of retrieval (i.e. devaluing the late-retrieved documents) and normalizing the result by dividing it with the ideal-retrieval gain (Järvelin and Kekäläinen, 2002). To give an overview of how the NDCG metric is calculated, we'll first look at *cumulative gain (CG)* and *discounted cumulative gain (DCG)*, which are separate metrics that are combined in calculating the NDCG.

Cumulative gain is a simple summing of document relevance scores. Documents' relevance values are acquired by examining the predicted documents and assigning them a score based on how relevant the content is. (Järvelin and Kekäläinen, 2002, p.424) Figure 22 shows a simple example where five documents are predicted, and their relevance scores are assigned so that 3 denotes a high relevance and documents with score 0 have no relevance. To calculate *cumulative gain* the relevance scores are simply summed as shown in figure 23. This metric gives a rudimentary ranking of different predictions, as it can be used to sort

out predictions with an overall higher relevance score. (Järvelin and Kekäläinen, 2002, p.424)

Document	Relevance	Discount factor	DCG
i	rel_i	$\log_2(i+1)$	$\frac{rel_i}{\log_2(i+1)}$
1	3	1	3
2	0	1.585	0
3	2	2	1
4	1	2.322	0.431
5	3	2.585	1.161

Cumulative gain

$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$ → 3+0+2+1+3 = **9**

Discounted cumulative gain

$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$ → 3+0+1+0.431 +1.161 = **5.592**

Normalized discounted cumulative gain

$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$ → Ideal ordering for results
i: 1,5,3,4,2 → rel: 3,3,2,1,0

IDCG = 3/1 + 3/1.585 + .. + 0/2.585 = 6,323

nDCG = 5.592 / 6.323 = **0.884**

Figure 23 Three different metrics for measuring the quality of ranking. Cumulative gain (CG) is a simple sum of human judged relevance scores. Discounted CG (DCG) uses a logarithmic discount factor to devalue later-retrieved documents. Normalized DCG uses the DCG of an ideal ordering as a normalizing factor (results in the range from 0 to 1) which makes it more suitable for comparing against other algorithms where the scale of relevance scores might differ.

An obvious way of augmenting the *cumulative gain* -metric is to add a discounting factor. The idea is to account for the fact that the greater the position of a relevant document, the less valuable it is for the user, since it takes more time and effort to find and examine the document (Järvelin and Kekäläinen, 2002, p.425). For example, web searches typically find millions of individual results, yet only a small fraction - usually only the first few pages - of this information is inspected, even though relevant information could be found further down. This type of search fatigue - resulting in inflation of relevance with later-retrieved documents - is accounted for by dividing the document's relevance score by the log of its rank which effectively decreases document's relevance proportionally to its rank as shown in figure 23 in calculating the *discounted cumulative gain*.

Finally, normalizing the above result makes it possible to compare different rankings and predictions. Since different algorithms might produce different output (i.e. not only predict the documents in different order but also new documents not predicted by some other classification algorithm) comparing the results using DCG doesn't provide a full picture since the scale of absolute DCG values could vary significantly. To counter this the DCG values are normalized using *ideal ordering* of the predicted results as a normalization factor. (Järvelin and Kekäläinen, 2002, p.426) The idea is to sort the predicted documents

descendingly according to their relevance, most relevant documents first and least relevant documents are added last. Using this type of ordering an ideal DCG value can be calculated, which in turn can be used as a normalizing factor for the predicted DCG value. In figure 23 an example is shown where the ideal ordering and associated ideal DCG value is calculated, after which the normalized DCG is calculated.

6 Case study: Public employment and business services - digitization project

Finland’s public employment and business services are - at time of writing - going through a massive digital transformation where the aim is to renew old ways of working by creating more efficient and automated - digital - services that support the employment efforts of both the applicant and the employer. Overall, this requires better use of available data in recognizing the applicant’s skills and then matching those to available positions in order to find and suggest suitable employment opportunities. In addition, based on individual interests and identified skill gaps on the market, valuable training opportunities could be introduced or recommended to the applicant’s in order to improve the prospects of finding interesting and meaningful job opportunities.

The goal of this thesis is to contribute to these digitalization efforts, by developing a viable solution for semi-automatic metadata gathering - more specifically term assignment - from employment services. Collecting metadata systematically will enable both better search and matching capabilities as different employment services will become easier for the users to find - due to more extensive indexing and tagging -, and a higher rate of personalization can be achieved by suggesting relevant content directly to the user based on their profile and selected preferences.

The scope of this study is limited to identifying and evaluating possible approaches for term assignment. Based on the evaluation results recommendations going forward are provided, but actual deployment to an online platform - including maintenance plan, version control, microservice architecture and creation, etc. - is deliberately left out of scope, so that efforts could be directed towards studying multiple different term assignment approaches and

finding best possible solutions. The project and its phases are described using the “cross-industry process for data mining” (CRISP-DM[30]) framework that consists of *business understanding, data understanding and preparation, modelling, evaluation,* and *deployment*. However, the *deployment* phase is not covered as it is out of scope.

6.1 Business Understanding

In this section we’ll briefly overview the TE-digi project by dissecting the relevant aspects in the context of this thesis. First a general outline of the project’s main elements and goals is represented, the introduction is concise since TE-digi covers a broad range of reforms of which most are out of scope.

TE-digi is an ambitious four-year (2016-2020) project that was set in order to renew the employment bureau’s digital operational capabilities. The core tenet of the project is simple: enhance matching between applicants and employers. Though easily stated, this vision is far from trivial to achieve. Ultimately, to affect matching on a national level in any meaningful way, a holistic view of both the job market - what skills are needed - and labor force - what skills are available - is required. Furthermore, gaining a view of the situation is only the first part of the puzzle, the second part - arguably even more important - is in providing a shared platform where this information can be leveraged and acted upon. These obstacles are of course acknowledged in the TE-digi project and different goalposts have been set to tackle each issue separately. The two main goals - derived from the vision statement - are as follows:

1. *Digitalize*: build shared platforms that enable networking and partnerships
2. *Analyze*: extract and use information proactively not reactively

The first goal, *digitalize*, sets a guiding boundary to the reform. The tools and procedures the employment bureau is currently using are insufficient in providing a detailed view of the job market. The digitization efforts are aimed at transforming the current closed system architecture to an open platform. This marks a paradigm shift in information exchange between the private and public sector as it enables a new type of ecosystem model, where the government provides an open infrastructure and common tools for employment services, that the private sector - both applicants and employers - can then leverage. Together this approach leads to a methodological way of gathering detailed employment data (private and

[30] https://www.sv-europe.com/crisp-dm-methodology/

public) from both the demand and supply side of the equation (i.e. what type of skills are needed in the job market and what is available). Since only *gathering* the data is not enough, the second goal, *analyze,* is needed so that the gathered data can be transformed to actionable insight. To achieve this, TE-digi is reforming both the internal and external tools (see figure 24). The external tools are comprised in a platform called the public job market[31], and the internal tools represent a set of different systems that employment officers use to find suitable work or training opportunities for customers.

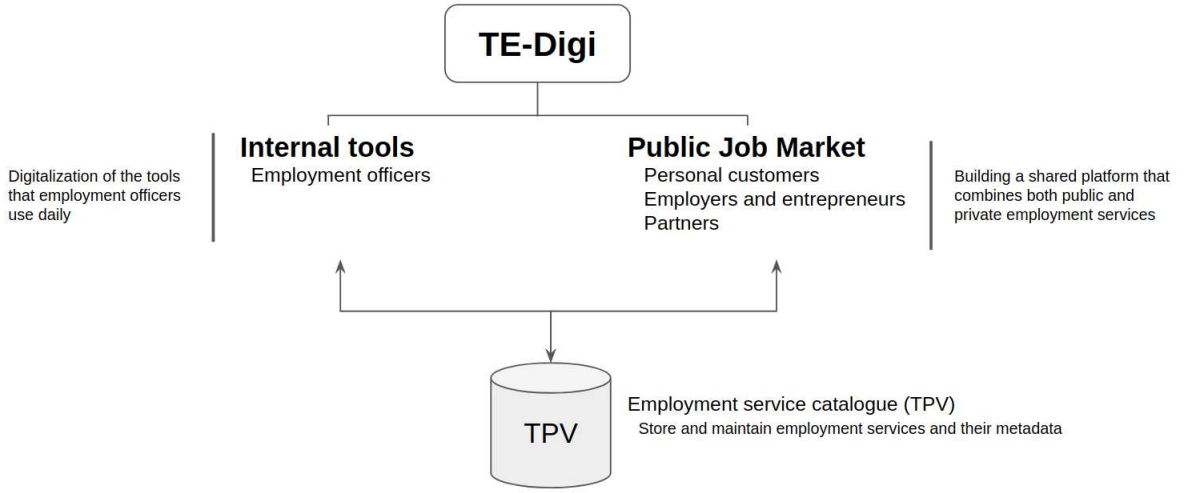

Figure 24 A simplified breakdown of the tools and systems that are built in the TE-digi project.

In the context of this thesis, the main interest is around the Employment Service Catalogue (TPV[32]). TPV is a database that comprises different employment services (e.g. training, recruitment, counselling, and mentoring services) and their metadata. It is used to store service information - such as name, description, location etc. - and the resulting metadata which can be used to make the services easier to find or even recommend to the user. Most of this metadata is created manually (see figure 25), since the amount of ontology[33] terms used for live events and service class and type metadata is sufficiently small, so that the user can easily select the correct labels from a list. However, this is not the case for the indexing terms, as the base ontology (a combination of YSO, JUHO and JUPO ontologies) covers over 40 000 terms. For this reason, it is necessary to aid the user in choosing suitable terms. The indexing term predictions are based on the service description, which the user gives. The indexing terms are generated semi-automatically. This means that after the user has provided the service description, he/she has the possibility to fetch term suggestions and then choose the terms that describe the service well enough. If the user

[31] https://kokeile.tyomarkkinatori.fi/en/Etusivu
[32] Työllisyyspalveluvaranto in finnish
[33] For each metadata type using a sub-section of the “Classification of public services” -ontology https://finto.fi/ptvl/en/

decides that none of the terms describe the service and chooses to not select any suggested terms, then none of the automatically created indexing terms are used. This leaves the user in control of the metadata creation and arguably generates more reliable indexing choices since the person adding the service is most often an expert in the service's domain.

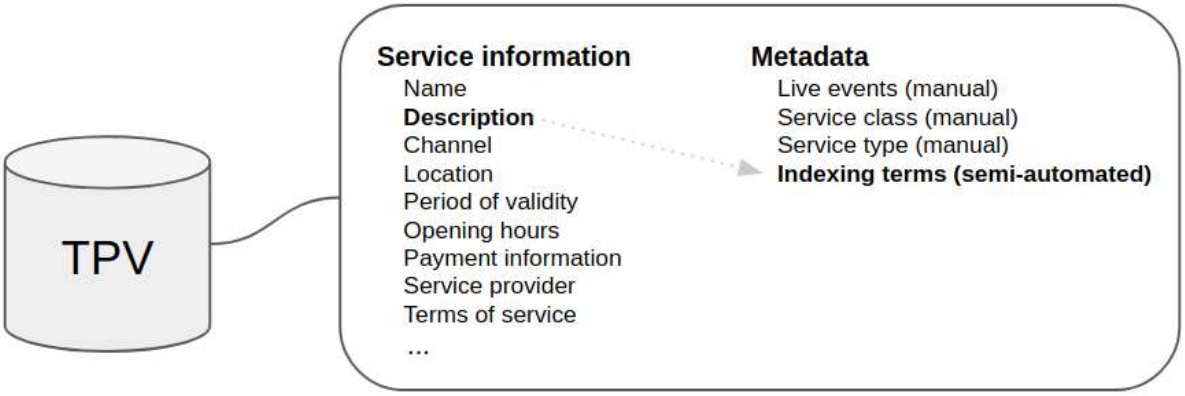

Figure 25 Service information and metadata stored in the Employment service catalogue (TPV). The gray arrow represents the term assignment task from service description to indexing terms, which defines the business problem in this thesis.

6.2 Data understanding and preparation

In this chapter the intent is to introduce the different datasets that are needed for building each term assignment model. This includes vocabulary data (i.e. the controlled set of indexing terms) where ontologies are used in order to create a robust vocabulary for different term assignment tasks. In addition to vocabularies training, validation and testing data is needed for each term assignment algorithm. This data is extracted from three open access services, which are *Finna, Jyväskylä University Digital Repository (JYX) and the Finnish Service Catalogue (PTV).*

6.2.1 Vocabulary building

Vocabulary defines the controlled set of terms the model can use to characterize a data point (e.g. a document, a sentence, etc.). If a specific term does not occur in the vocabulary, it will not be used for indexing even though the term would frequent in the predicted document and serve as an obvious indexing term for any human indexer. For this reason, the vocabulary used for term assignment should be exhaustive and specific enough to cover all - or at least the most important - terms in the predicted documents' domains. To build such vocabulary ontologies can be used, as they are designed to include an exhaustive set

of domain specific terminology and are also actively maintained and updated over time as terminology changes and adapts (i.e. new words are introduced and old ones omitted).

Finto[34] is a Finnish thesaurus and ontology service that offers a REST-style API to access vocabulary data and provides a service to download available vocabularies and ontologies in various serialization formats (such as RDF and Turtle). It maintains several different ontologies and vocabularies, including domains such as

- society,
- geography and geoinformation,
- science and medicine,
- art and culture,
- languages and literature and
- general. (Finto, 2020)

For this project three different ontologies - including general and society domains - were used. To build the necessary vocabulary that captures relevant terms for indexing the Finnish Public Employment Services a decision was made to include General Finnish ontology (YSO), Finnish Ontology for Public Administration (JUHO) and Finnish Ontology for Public Administration Services (JUPO). These include two special domain ontologies - JUHO and JUPO - and one general ontology - YSO. The general ontology YSO contains mainly general concepts and is by far the largest vocabulary with over 30 000 terms. YSO - as well as JUPO and JUHO - is trilingual (i.e. it includes also Swedish and English terms) which is beneficial if the term assignments is to be done in three different languages. However, the models for each language need to be trained separately, and training data for each language is needed. In this thesis all models are trained in Finnish and other languages are considered out of scope. JUHO and JUPO ontologies consist of a more specialized domain, as they describe different public services and terms for administration. JUPO and JUHO are both smaller ontologies, comprising about 2300 and 6500 terms respectively (see table 5).

[34] Finto-service website: https://finto.fi/en/

Table 5 Breakdown of the individual ontologies used for term assignment.

Ontology	**Description**	**Term count** (Finnish)	**Term count** (English)	**Term count** (Swedish)
YSO - General Finnish ontology	YSO consists mainly of general concepts in Finnish cultural sphere.	30 946	30 468	30 513
JUHO - Finnish Ontology for Public Administration	JUHO is a public administration ontology based on Finnish Government Subject Headings	6561	6552	6552
JUPO - Finnish Ontology for Public Administration Services	JUPO describes Finnish public services	2313	2270	2292

To build a unified vocabulary composed of YSO, JUHO and JUPO ontologies, a separate ontology called KOKO was used. KOKO is a collection of 16 core Finnish ontologies - described in figure 26 - that have been merged. (Finto, 2020) From these core ontologies only three were used as the goal was to drive down the vocabulary size and to omit terms unrelated to Finnish public employment services. To filter the KOKO-ontology a separate Python[35] script using RDFlib-library[36] was constructed. Using RDFlib KOKO-ontology could be filtered so that only terms from the desired ontologies were kept and then serialized to ttl-form which is used for vocabulary building.

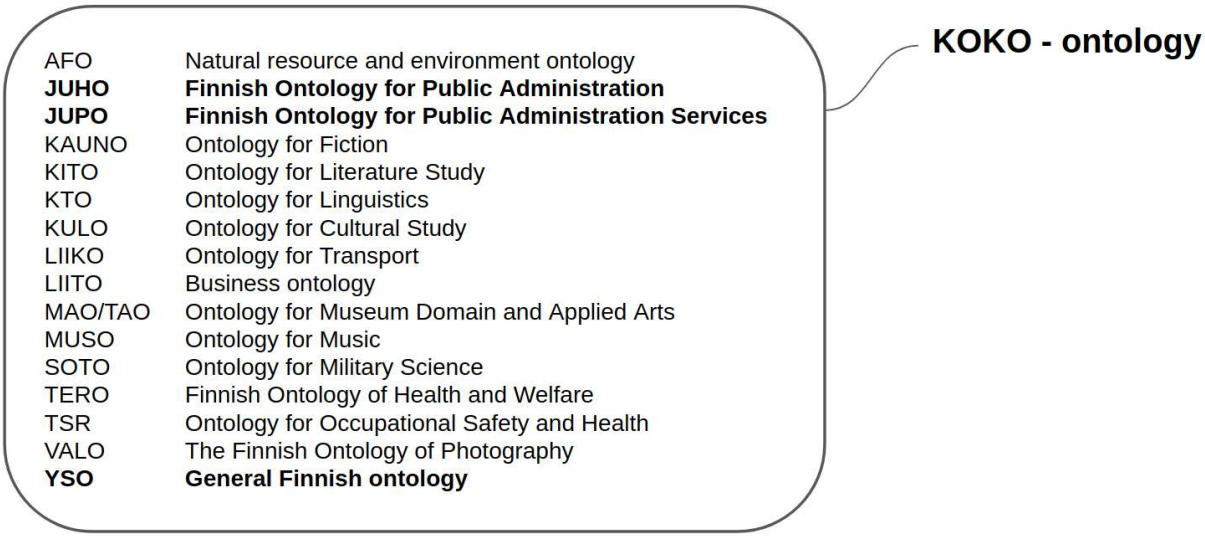

Figure 26 Core ontologies merged under KOKO-ontology. JUHO, JUPO and YSO are bolded as these ontologies are used for vocabulary building.

6.2.2 Data for modelling

Models used for term assignment in this project - Maui, VSM and Omikuji - are dependent on training, validation, and testing data. Training and testing data is needed for all models,

[35] https://www.python.org/
[36] https://rdflib.readthedocs.io/en/stable/index.html

whereas validation data is only used with models that have hyperparameters (e.g. Omikuji) which can be tuned in order to examine different architectures and then choose the one that performs best. Since YSO, JUHO and JUPO ontologies are used to construct the controlled vocabulary, training, validation, and test data also need to consist of manually labeled examples that are indexed using the same three ontologies. Otherwise, training the model would not be possible, as the algorithm couldn't assign correct labels (since only terms that are part of the vocabulary can be assigned) and therefore wouldn't learn anything.

Modelling is done using two different datasets that are extracted from two distinct services: *Finna API and Finnish Service Catalogue* (see table 6 for short descriptions). *Finna API* is an open access service to metadata from Finnish libraries, archives and museums. It is administered by the National Library of Finland and over 300 organizations publish their services and materials using Finna. It contains openly accessible metadata of nearly 6 million documents (composed of metadata on books, journals, video, audio, art works, etc.) which are indexed using YSO-ontology (Finna.fi, 2020). The Finnish Service Catalogue (PTV, abbreviation from the Finnish word Palvelutietovaranto) is used to extract Finnish public services' descriptions and indexing terms in YSO, JUHO and JUPO ontologies. PTV is a centralized data repository where organizations performing statutory tasks (e.g. municipalities, state administrative authorities and agencies, etc.) are obligated to describe and publish their services. (Palvelutietovaranto, 2020) Currently PTV holds over 20 000 indexed public service descriptions.

Table 6 Data extraction services and short descriptions.

Data extraction service	**Description**	**Form of data**	**Subject ontology** (extracted bolded)
Finna API	Finna API provides an open access to metadata from Finnish libraries, archives and museums. Collection contains over 5 million indexed documents which are partially used as training data.	Book title and subject URI	**YSO**
Finnish Service Catalogue (PTV)	Finnish Service Catalogue is a centralised data repository in which public organisations provide information on their services and service channels. It contains excess of 20 000 indexed public services.	Service description and subject URI	**YSO**, **JUPO**, LIITO, **JUHO**, TERO and TSR

The size and compositions of the two datasets vary quite considerably. Of the two, Finna-dataset is by far the larger one. It contains nearly 6 million manually labeled documents, indexed using over 25 000 unique YSO-ontology terms (YSO consists of 30 946 terms, meaning that Finna's indexing terms cover over 82% of the ontology). These documents are

indexed by specialists (e.g. library, museum, and archive staff) and of relatively good quality. Each document is indexed with a median of 4 terms, there is however considerable variance in the amount of indexing terms per document. In figure 27 the distribution and most common terms are plotted.

PTV-dataset consists of 20 426 public service descriptions which are indexed using 5749 unique indexing terms. The data quality of the descriptions and indexing terms is somewhat varying since there are over 2200 editors with varying skills and aptitude for indexing. From the statistics (table 6 and figure 27) we can see that most of the services are indexed with only two indexing terms and the variance is quite low, meaning that overall few indexing terms are assigned to the created services. Despite these shortcomings there are two main reasons that support the use of PTV-data. First PTV-data is indexed using YSO, JUHO and JUPO-ontologies, whereas Finna data comprises only YSO-terms. Indexing TPV-services will require a vocabulary that includes all three ontologies, which means that PTV-data is the only source of training data (to our knowledge) for JUHO and JUPO-ontologies. Second, it is safe to assume that service descriptions in PTV will have much resemblance to the future service descriptions created in TPV, due to the similarity in the organization of the indexing efforts (in both cases a big group of people with varying skill in indexing are creating both the service descriptions and indexing terms, which leads to high variance in quality). Therefore, training and testing on PTV-data should be a good indicator of the algorithms indexing quality.

One interesting aspect of the two datasets is that the theme between most common words is quite different in each dataset - which surely one might expect. In Finna-data the most common theme is history - with over three times more appearances than any other indexing term in the same dataset. Other common themes are for example children, memoirs, legislation, businesses, and cooking. PTV-data on the other hand is closely related to Finnish public services and most common themes are around the public-school system (both preschool and comprehensive school), construction permits and control and public grants. These findings are depicted using a bar plot in figure 27.

Table 7 Basic statistics of the Finna, PTV and JYX -datasets.

Dataset	Ontology	Indexed documents	Unique indexing terms	Mean (terms per document)	Median (terms per document)	Variance (terms per document)
Finna-data	YSO	5 960 724	25 603	5.3	4	24.41
PTV-data	YSO/JUHO/JUPO	20 426	5 749	2.9	2	4.13

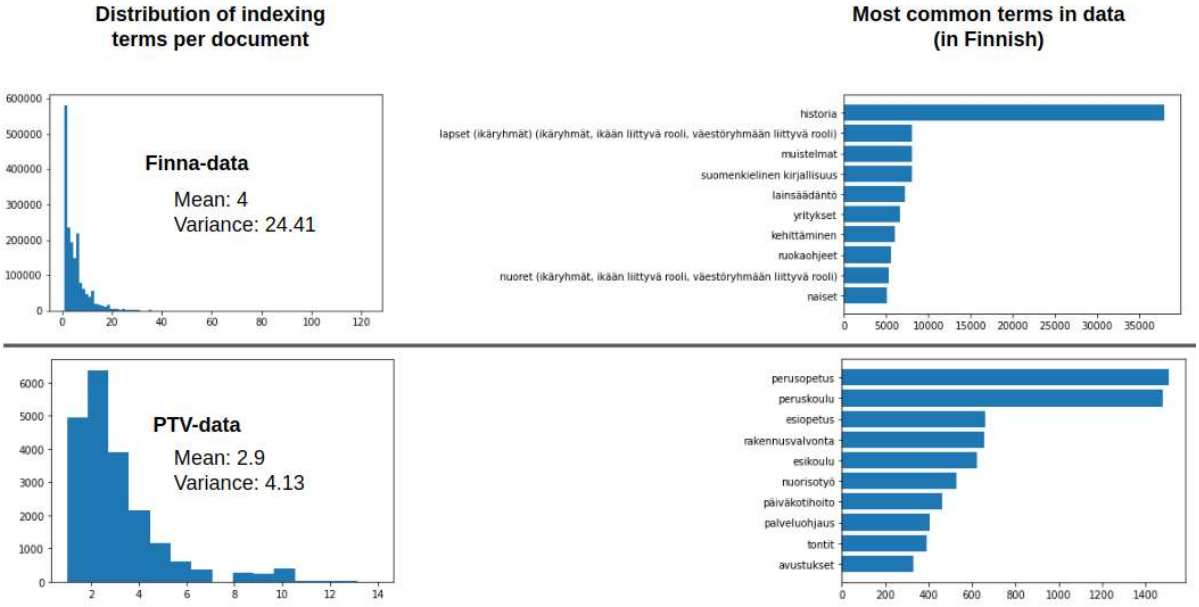

Figure 27 Distributions of indexing terms in different datasets and most common terms.

Of the two datasets only PTV-data is partitioned randomly to train, validation, and test datasets, whereas Finna-data is used merely for model training (see table 8). When comparing Finna and PTV-data it is likely that PTV-data resembles more the Finnish employment services data and therefore using PTV-data for both validation and testing should yield the best performing model as well as a good approximation of the actual performance on new and unseen data. On top of train, validation and test sets, a separate set is left out for training both the NN-ensemble and Maui model. Considering that both models use training data to optimize their output - Maui creates a weighting scheme for the feature values and NN-ensemble tunes its internal neural network - it is reasonable to train these architectures using a small amount of PTV-data.

Finna-data is used only partially, because of memory constraints. Training the model on the full Finna-dataset - comprising 5.9 million documents - requires more memory than currently

available on the machine used for model building. The used dataset is sampled randomly from the full dataset and has similar distribution of indexing terms when compared to the full dataset and over 90% of the indexing terms are used (compare Finna-train in table 8 to Finna-data in table 7). A separate train dataset is sampled randomly from the PTV-train data and used to train Maui-algorithm. Since Maui is a lexical algorithm (i.e. uses dictionary mapping to find candidate words and simple features to rank those) it requires little training data, as it is only used to find suitable weights for individual features - analogous to nn-ensemble creating a mapping for index term suggestions.

Table 8 Dataset partition, statistics, and usage.

Dataset	**Ontology**	**Indexed documents**	**Unique indexing terms**	**Mean** (terms per document)	**Median** (terms per document)	**Variance** (terms per document)	
Finna-train	YSO	1 766 497	23 115	5.3	4	24.41	Finna-data used only partially (memory constraint)
PTV-train	YSO/JUHO/JUPO	16 355	5 267	2.87	2	4.09	PTV-data partitioned to *train, validation* and *test set*
PTV-train (nn-ensemble)	YSO/JUHO/JUPO	1006	1153	2.88	2	3.87	
PTV-validation	YSO/JUHO/JUPO	1034	1235	2.93	2	4.30	
PTV-test	YSO/JUHO/JUPO	2031	1850	2.90	2	4.48	
PTV-maui-train	YSO/JUHO/JUPO	1041	1188	2.88	2	3.69	Maui is trained using a subset of PTV-train data

6.3 Modelling and evaluation

In the previous chapter we gained an understanding of the different datasets - both training and vocabulary - used for creating a term assignment model. Next the aim is to build and evaluate different models and ultimately choose the optimal model based on test results and accuracy metrics. This chapter will cover *test design, model validation* and *model selection*. *Test design* -section describes the process of model evaluation and summarizes the models (and their hyperparameters), datasets and metrics that are used for model selection. In the *model validation and selection* phase the evaluation process is employed to fine-tune, compare, and finally choose the optimal model.

6.3.1 Test design

Before the actual validation and evaluation phase, it is necessary to summarize the key aspects of the testing process and list the algorithms that will be evaluated. Overall, the testing process is simple, and its phases are depicted in figure 28. First each model is trained (excluding simple ensemble - as it requires no training and only uses suggestions from other term assignment algorithms) using a suitable set of training data and default hyperparameter settings. Next, if the model has hyperparameters, validation data is used to test different hyperparameter settings and evaluate performance between different architectures. This step is done iteratively by first using validation data to calculate the same metrics that are ultimately used for model selection. Evaluation of each architecture is done using *k-fold cross validation* where the validation data is split to k-chunks (of equal size) and for each chunk the metrics are calculated. This is done to calculate the mean, standard deviation and confidence interval of the results to see whether there is statistical difference between the different architectures. After cross validation the hyperparameters are adjusted and the new architecture is trained using training data and then evaluated using a similar k-fold cross validation step. This procedure is iterated over all the examined architectures. After different architectures have been evaluated, the best performing model is chosen based on the calculated statistics and then trained using both training and validation data.

The final evaluation of model performance is done using test data, which is a separate dataset used only for testing. Similarly, to the validation phase, k-fold cross validation is used to gain statistics of model performance. In case the model has no hyperparameters, the validation step is skipped and instead only model performance is evaluated. This process is applied separately for each algorithm with the goal of attaining comparable performance metrics which can be used for final model selection. Model selection is done using the calculated metrics (and their statistics) and comparing different algorithms performance.

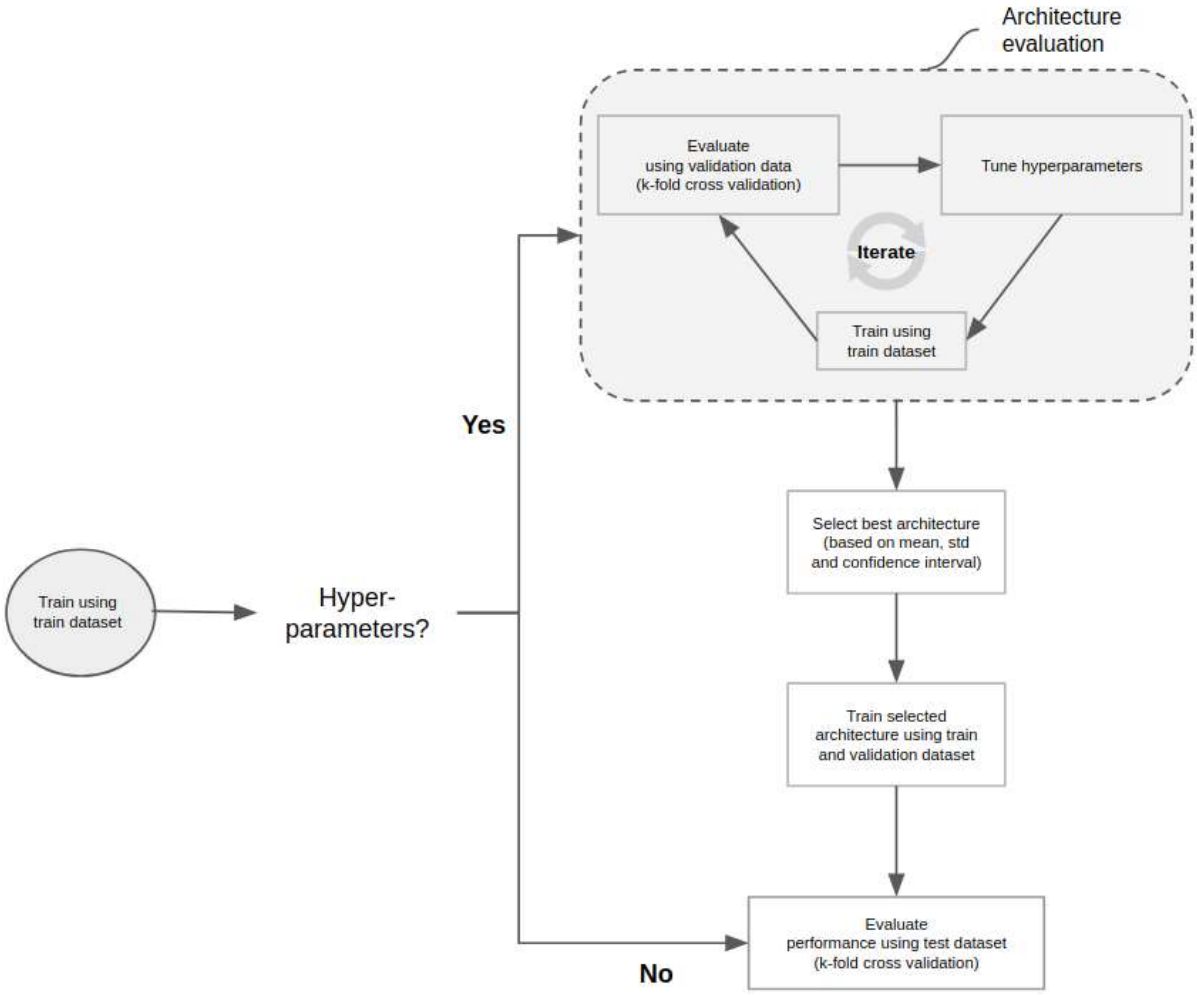

Figure 28 Model train, validation, and evaluation process.

Overall, five different algorithms are evaluated. These consist of two ensemble models - simple ensemble and neural network ensemble - and three term assignment algorithms - Maui, VSM and Omikuji. Of the models, three have adjustable hyperparameters. For evaluation - both in validation and testing steps - five different metrics are used to rank models based on performance and indexing quality (see table 9). The used metrics are introduced in detail in chapter “Evaluation of term assignment”. The notion “@[number]” (e.g. F1@5) signifies that the amount of evaluated indexing terms is limited to the given number (e.g. for F1@5 only the first five predicted terms are evaluated when calculating the F1-score). Both F1 and precision metrics are sensitive to the amount of predicted indexing terms, and since the validation/test data only includes a mean of 2-5 terms per document (table 8) calculating the metrics with 20 predicted terms (which is used as the limit for the other metrics) would result to several false positives, which in-turn would result to a lower score. For this reason, the amount of evaluated subject terms is limited for F1-score and precision.

In terms of evaluation, three distinct (and five all together) metrics are used to assess each algorithm's quality of indexing. Recall is used in order to detect the proportion of relevant terms the algorithm finds, NDCG gives an estimate of the ranking quality and precision provides an indication of how well the terms with the highest scores (i.e. terms the algorithm suggests first) correspond to the actual terms. Since the goal is to create an indexing algorithm for semi-automatic term assignment (i.e. give suggestions of indexing terms and have the user choose most relevant) it is important to assess whether the given suggestions are useful and that the most relevant terms are ranked high for ease of use. With this context both the NDCG and precision-metrics are useful in assessing whether the first suggested terms are relevant and give good indication of the usefulness of the algorithm in semi-automatic term assignment setting. Recall on the other hand is used to examine the overall amount of relevant terms the algorithm finds when limited to 20 suggestions, which can be considered as an upper limit of how many terms a user might browse when looking for a good indexing term candidate. Finally, the F1-score (limited to 5 suggestions) is a combination metric of precision and recall and gives an overall indication of the amount of relevant and non-relevant terms that were found. F1-score augments the precision and recall-metrics as it includes both the false positive (terms that are not actually relevant but were predicted) and false negative (terms that are actually relevant but weren't predicted) scores and gives an overall estimate of the indexing term quality of the first five predicted terms (which is a reasonable amount of terms to suggest to a user at any one time). A summary breakdown of the individual algorithms and their train/validation/test datasets, adjustable hyperparameters and used metrics can be found in table 9.

Table 9 Summary of algorithms, train/validation/test datasets, algorithm specific hyperparameters and evaluation metrics.

Algorithm	Data	Hyperparameters	Metrics
NN-ensemble	**Train**: PTV-train (nn-ensemble) **Validation**: PTV-validation **Test**: PTV-test	**Nodes**: Size of hidden layer **dropout_rate**: Proportion of neurons to drop **epochs**: Number of passes over training data	**Recall**: Proportion of relevant items selected **NDCG**: Ranking quality **F1@5**: F1-score - suggestions limited to top 5 **Precision@1**: Precision - suggestion limit 1 **Precision@3**: Precision - suggestion limit 3
Simple ensemble	**Train**: - **Validation**: PTV-validation **Test**: PTV-test	**Weight**: Determine weight for each source project	
Omikuji	**Train**: Finna-data and PTV-train **Validation**: PTV-validation **Test**: PTV-test	**Limit**: Amount of indexing terms to suggest **min_df**: Minimum document frequency **cluster_balanced**: Perform balanced k-means **Cluster_k**: Amount of clusters **max_depth**: Maximum tree depth **collapse_every_n_layers**: Num of layer to collapse	
VSM	**Train**: Finna-data and PTV-train **Validation**: - **Test**: PTV-test	No hyperparameters	
Maui	**Train**: PTV-data (PTV-maui-train) **Validation**: - **Test**: PTV-data	No hyperparameters	

6.3.2 Model validation

The aim in model validation is to examine different architectures and evaluate which hyperparameter settings are optimal for the given prediction task. This is done by iteratively modifying the existing hyperparameters, then re-training and re-evaluating the model. Finally, the different architectures of the same model are ranked based on predefined metrics and the overall best model is selected. In this section the validation results of Omikuji, Simple ensemble and Neural Network ensemble are introduced.

6.3.2.1 Omikuji

Out of all the models tested in the scope of this thesis, Omikuji has the highest amount of adjustable hyperparameters. These include (for a more thorough explanation see table 3 and chapter on Omikuji):

- Limit: Amount of indexing terms to suggest
- min_df: Minimum document frequency
- cluster_balanced: Perform balanced k-means
- cluster_k: Amount of clusters
- max_depth: Maximum tree depth

- collapse_every_n_layers: Number of layers to collapse.

Using the available hyperparameters there are three different types of architectures that are found in the literature, namely *Parabel, Bonsai* and *AttentionXML.* Here the aim is to evaluate each architecture with different hyperparameter settings, assess the performance differences between each architecture and choose optimal. The differences between the architecture types are as follows. *Parabel* builds a deep tree configuration using a balanced 2-means clustering algorithm, where tree depth (i.e. max_depth parameter) is adjusted to see what affects the number of layers have on algorithms performance. *Bonsai* builds a considerably shallower and wider tree representation using regular k-means, where both tree depth and number of clusters are adjusted to find optimal settings. (Omikuji, 2020) Last, we evaluate *AttenxionXML* style tree, where balanced 2-means clustering is used together with layer collapsing to build a shallow and wide tree. Finally, the best configurations from each architecture are compared and the overall best chosen.

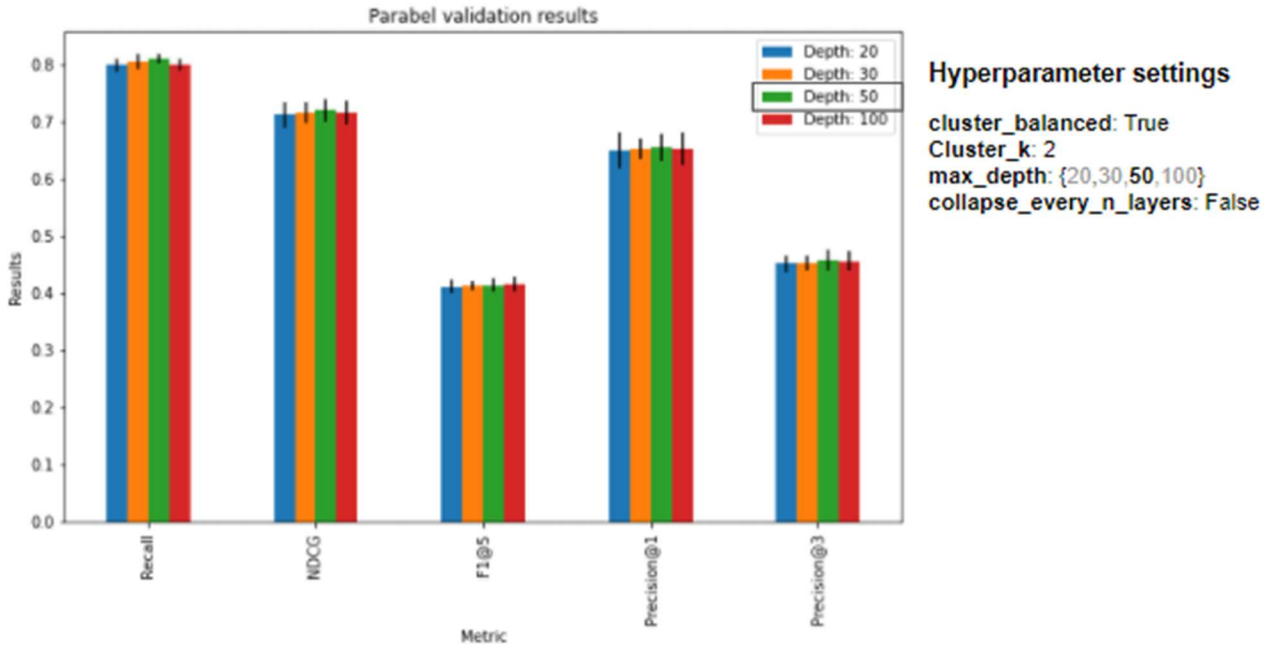

Figure 29 Validation results of Parabel architecture. Tree depth seems to have little effect on the overall performance.

Table 10 Parabel architecture mean results with 95% confidence interval calculated using a sample size of n=5

95% Confidence interval (with n=5)

	Depth: 20	Depth: 30	Depth: 50	Depth: 100
Recall	0.79749±0.0106	0.80335±0.01194	0.80874±0.00731	0.79793±0.00889
NDCG	0.71096±0.02005	0.71465±0.01503	0.71877±0.01791	0.71416±0.01803
F1@5	0.4101±0.01026	0.41177±0.00746	0.41317±0.01128	0.41448±0.01191
Precision@1	0.64892±0.02782	0.65084±0.01621	0.6547±0.02067	0.6518±0.02565
Precision@3	0.45131±0.01276	0.45165±0.01177	0.45679±0.0162	0.45551±0.01612

Validation results from Parabel are plotted in figure 29. For Parabel-architecture the only modified parameter was depth of the decision tree. From the metrics we can see, that modifying the depth has little effect on the overall performance. All the models are within 1% of each other - on all of the metrics - and given the standard deviation and confidence interval results (table 10) the models seem to have little statistical performance difference. Out of the evaluated Parabel architectures tree with depth 50 seems to have the best overall performance and is therefore selected as the best alternative.

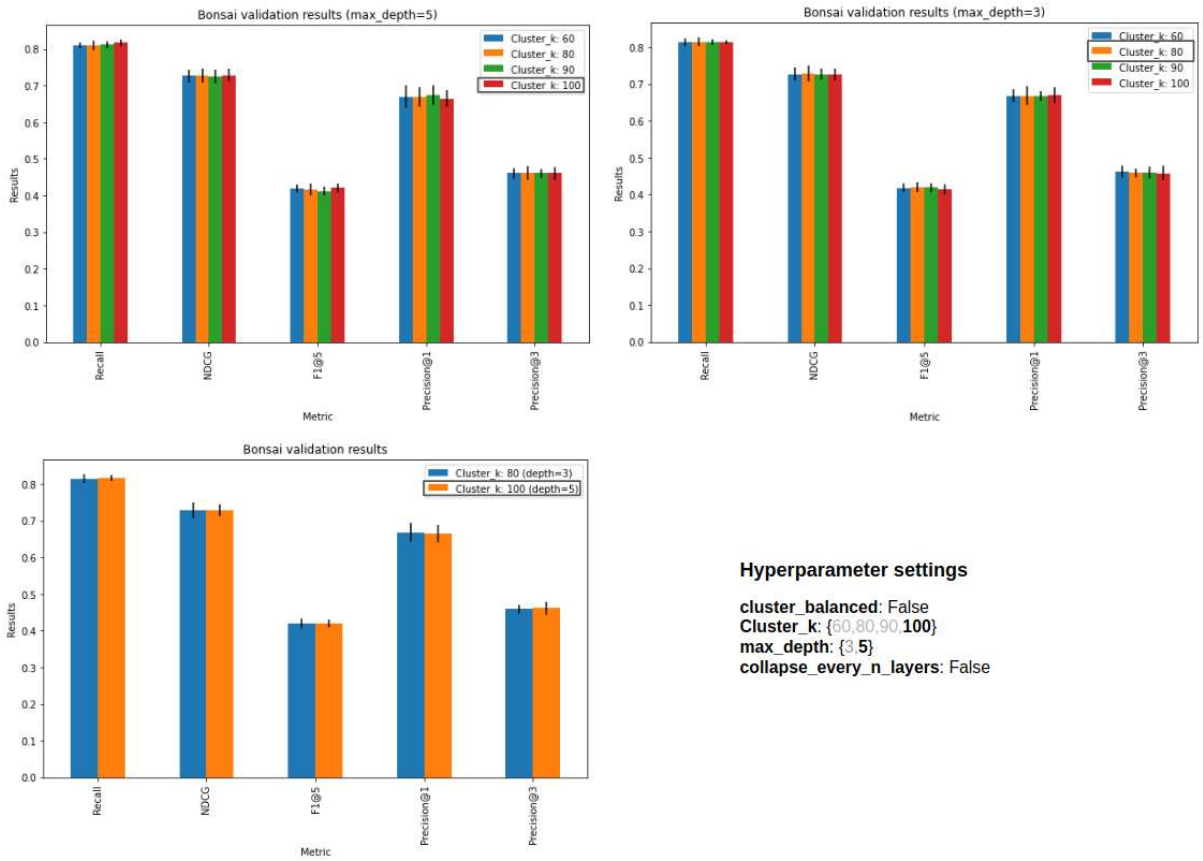

Figure 30 Bonsai validation results. Both the depth and cluster amount parameters adjusted. Again, adjusting the parameters have little effect on the overall performance.

Table 11 Bonsai architecture mean results with 95% confidence interval calculated using a sample size of n=5

95% Confidence interval (with n=5)

Max depth = 3

	Cluster_k: 60	Cluster_k: 80	Cluster_k: 90	Cluster_k: 100
Recall	0.8145±0.00912	0.81488±0.01048	0.81321±0.0067	0.81421±0.00512
NDCG	0.727±0.01536	0.72822±0.01827	0.72697±0.01184	0.72639±0.01398
F1@5	0.4177±0.01049	0.419±0.01205	0.41844±0.00955	0.41373±0.01144
Precision@1	0.66826±0.01589	0.66826±0.02276	0.66729±0.01237	0.67018±0.01787
Precision@3	0.46164±0.01428	0.45841±0.0101	0.46034±0.01384	0.45809±0.0167

Max depth = 5

	Cluster_k: 60	Cluster_k: 80	Cluster_k: 90	Cluster_k: 100
Recall	0.80948±0.00673	0.8094±0.01132	0.81107±0.00885	0.81648±0.00771
NDCG	0.72579±0.0149	0.72634±0.01597	0.72473±0.01689	0.72827±0.01489
F1@5	0.41932±0.00952	0.41594±0.01357	0.41146±0.01019	0.42045±0.00973
Precision@1	0.66922±0.02727	0.66826±0.02368	0.67309±0.02322	0.66438±0.02006
Precision@3	0.46035±0.01197	0.46196±0.01596	0.46034±0.0099	0.461±0.01504

	Cluster_k: 80 (depth=3)	Cluster_k: 100 (depth=5)
Recall	0.81488±0.01048	0.81648±0.00771
NDCG	0.72822±0.01827	0.72827±0.01489
F1@5	0.419±0.01205	0.42045±0.00973
Precision@1	0.66826±0.02276	0.66438±0.02006
Precision@3	0.45841±0.0101	0.461±0.01504

For Bonsai-architecture the results are similar to Parabel. Adjusting the hyperparameters (here number of clusters and depth) seems to make little difference on the end result which we can clearly see from figure 30. Again, all of the metrics are within 1% of each other, and choosing a model based purely on the metrics is not possible. Therefore, we choose the default configuration - with depth 3 and cluster amount 100 - as it is recommended in the literature and also seems to provide slightly less varying results (table 11) based on the standard deviation and confidence interval results. (Khandagale et al., 2019)

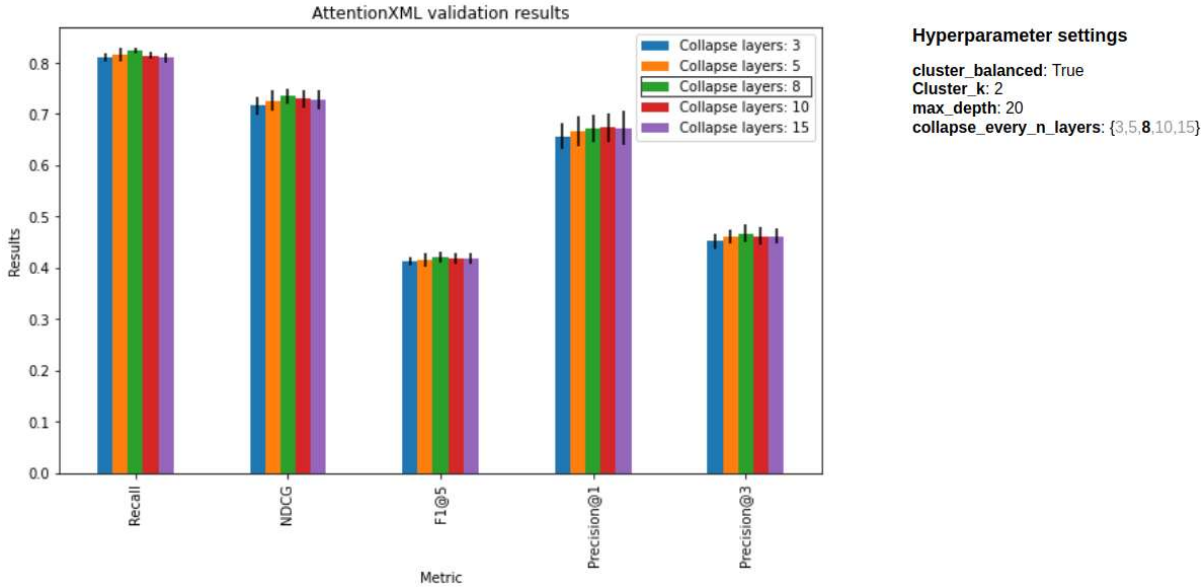

Figure 31 AttentionXML validation results. For AttentionXML architecture the results indicate a trend where larger amount of layer collapsing generates better results.

Table 12 AttentionXML architecture mean results with 95% confidence interval calculated using a sample size of n=5

95% Confidence interval (with n=5)

	Collapse layers: 3	Collapse layers: 5	Collapse layers: 8	Collapse layers: 10	Collapse layers: 15
Recall	0.80984±0.00756	0.81538±0.01067	0.82282±0.00519	0.81375±0.00613	0.80943±0.00857
NDCG	0.71663±0.01527	0.72561±0.01697	0.73432±0.0134	0.72911±0.01608	0.72774±0.01664
F1@5	0.41244±0.00675	0.415±0.01086	0.42083±0.00838	0.41824±0.01013	0.41789±0.00989
Precision@1	0.65569±0.02222	0.66631±0.02478	0.67211±0.02311	0.6731±0.02481	0.6721±0.02974
Precision@3	0.45228±0.01277	0.46067±0.01227	0.46646±0.01502	0.46131±0.01555	0.46195±0.01325

For AttentionXML the only adjusted parameter was layer collapsing, which leads to a (exponentially) more complex model as the amount is increased. Unlike Parabel and Bonsai, AttentionXML model with 8 collapsing layers seems to outperform the other configurations, as it has higher mean score for all metrics except precision@1 (although based on the confidence interval the results are technically on par) and also the deviation of the results seems to be lower than on the other architectures (figure 31 and table 12). Based on these results the tree with 8 collapsing layers is chosen.

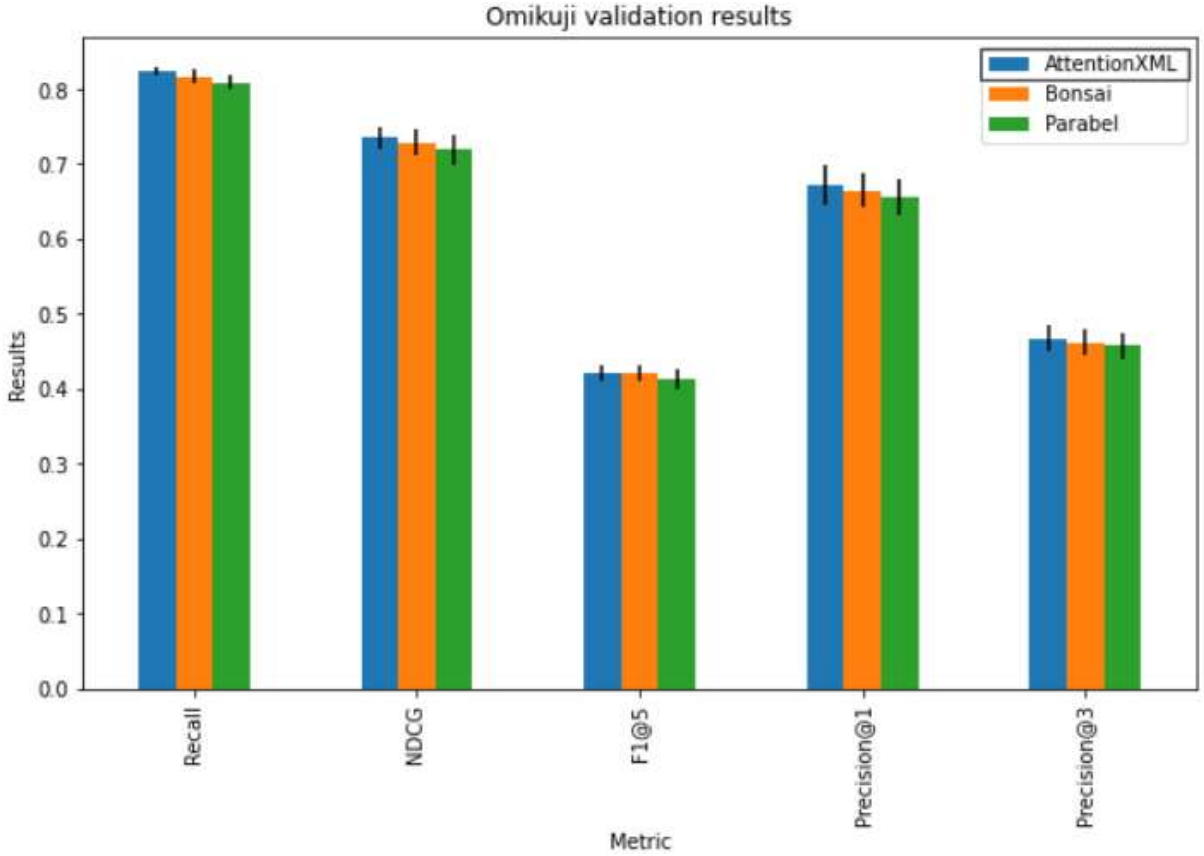

Figure 32 Validation and test results for each architecture type. AttentionXML model outperforms Parabel and Bonsai on both validation and test data sets. However, the performance difference between Bonsai and AttentionXML is minor in the test data set.

Table 13 Validation results with mean and confidence interval of each chosen Omikuji architecture

95% Confidence interval (with n=5)

	AttentionXML	Bonsai	Parabel
Recall	0.82282±0.00519	0.81648±0.00771	0.80874±0.00731
NDCG	0.73432±0.0134	0.72827±0.01489	0.71877±0.01791
F1@5	0.42083±0.00838	0.42045±0.00973	0.41317±0.01128
Precision@1	0.67211±0.02311	0.66438±0.02006	0.6547±0.02067
Precision@3	0.46646±0.01502	0.461±0.01504	0.45679±0.0162

Finally, the selected models - AttentionXML, Parabel and Bonsai - are compared. Of the three models AttentionXML seems to be dominant on all the metrics. AttentionXML outperforms the other models on all metrics, and the deviation of the results is smaller and constantly on a higher level than on the other metrics (Figure 32 and table 13). Based on these results AttentionXML is chosen as the preferred architecture of Omikuji to be evaluated against the other models in the final model selection.

6.3.2.2 Simple ensemble

The only hyperparameters for simple ensemble are the individual weights that are used in the mean score vector calculation. For example, if a certain algorithm is given zero weight, its predictions are not included in the calculation, if it is given a weight of two, the score it predicts carries twice as much weight compared to the other algorithms (i.e. is considered more important or more accurate prediction). Using Annif, optimizing these parameters can be done automatically via a random search, where the weights are assigned randomly for each backend project (i.e. indexing algorithm). First the optimized metric and amount of trials is specified, after which the grid-search is performed automatically storing results from all different trials and finally printing out the optimal configuration of weights for each algorithm. For the simple ensemble the results are optimized based on F1@5 and NDCG-metrics and the optimization process is run separately for each metric. Based on the results an optimal configuration is chosen.

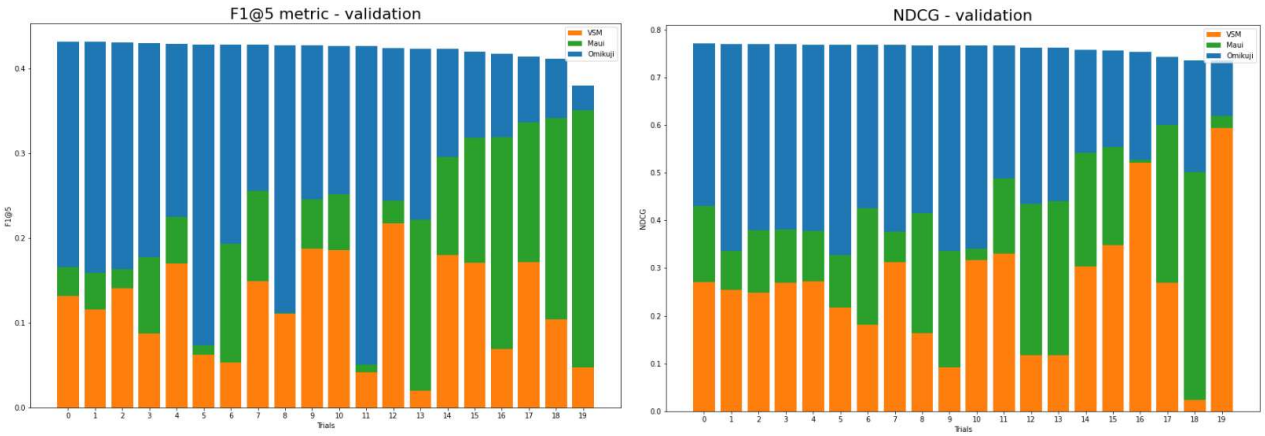

Figure 33 Validation results of simple ensemble from hyperparameter optimization. The results seem to indicate that Omikuji (marked with blue), should carry relatively higher weight than Maui (green) and VSM (orange).

In figure 33 the hyperparameter optimizations results are introduced using a stack column chart. The results are sorted so that the best results are displayed on the left (trial 0) and worst results on the right (trial 19). Each color represents one algorithm and one stack displays the proportional importance of each algorithm in gaining the result (e.g. if a column would be 0.5 high and only have one color - say blue (Omikuji) - it would represent a configuration where Omikuji would have weight one and other algorithms weight zero and the end result on that specific trial/metric was 0.5).

The results on both metrics are quite similar and seem to display a trend where Omikuji (plotted with blue) outperforms VSM (orange) and Maui (green). Quite evidently the stronger results have a relatively higher weight for Omikuji and again for the weaker results both VSM and Maui are given higher weights. Illustrative example of this is the fact that for both metrics the weakest result (trial 19) has the lowest weight for Omikuji. Based on the validation results the best configuration - trial 1 configuration where Omikuji is given relatively higher weight than Maui and VSM - is chosen for testing.

6.3.2.3 Neural Network Ensemble

Neural network ensemble (NN-ensemble) works in a similar manner to simple ensemble, but instead of calculating a simple weighted mean of the subject prediction scores, a neural network is used to train a mapping from subject suggestions to predictions. Essentially the neural network operates as a fine-tuning method to better match the manually labeled data. (Suominen, 2020) NN-ensemble supports four different hyperparameters of which two are explored via a grid search in model validation -phase. The controlled parameters are nodes (amount of nodes in the hidden layer) and epochs (amount of passes over the training data), the two other parameters - where default values are used - are amount of dropout

regularization and the used optimizer. The amount of dropout regularization can be controlled to avoid overfitting the model, but since different model configurations are trained - both simple and complex (i.e. models with few hidden neurons and epochs to more complex configurations) - validation over different regularization parameters would be unnecessary. Further, Adam (Kingma and Ba, 2015) is used as the default optimizer since it works sufficiently fast in training the models.

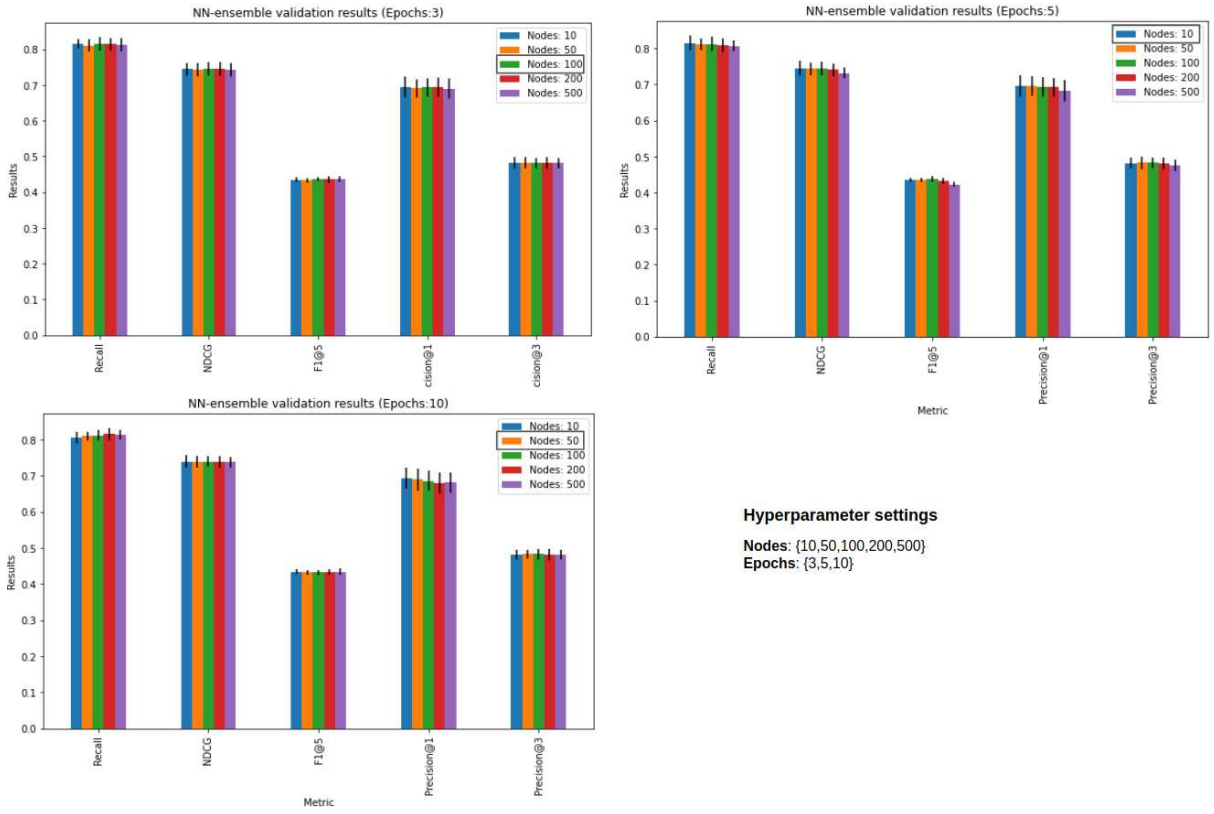

Figure 34 Validation results for different hyperparameter configurations. Best model is chosen from each epoch configuration. The chosen model for each epic is squared in the plot's legend.

Table 14 NN-ensemble mean values with 95% confidence interval for hyperparameter configurations

95% Confidence interval (with n=5)

Epochs 3

	Nodes: 10	Nodes: 50	Nodes: 100	Nodes: 200	Nodes: 500
Recall	0.81528±0.01103	0.81129±0.01502	0.815±0.01668	0.81416±0.01544	0.81347±0.01694
NDCG	0.74498±0.0157	0.74223±0.01666	0.74561±0.01657	0.74497±0.01612	0.74336±0.01644
F1@5	0.4349±0.00579	0.43318±0.00537	0.43684±0.0057	0.43594±0.00808	0.43631±0.00701
Precision@1	0.69435±0.02506	0.69145±0.02256	0.69339±0.02277	0.69532±0.02372	0.69048±0.02452
Precision@3	0.48162±0.01369	0.48162±0.01378	0.4813±0.01221	0.4813±0.01396	0.48162±0.01326

Epochs 5

	Nodes: 10	Nodes: 50	Nodes: 100	Nodes: 200	Nodes: 500
Recall	0.81528±0.0179	0.81177±0.01373	0.81268±0.01848	0.80886±0.01657	0.80747±0.01218
NDCG	0.74555±0.01705	0.74322±0.01504	0.74405±0.01715	0.74093±0.01529	0.73121±0.01305
F1@5	0.43503±0.00546	0.43485±0.0048	0.43648±0.00716	0.43282±0.00684	0.42279±0.00662
Precision@1	0.69629±0.02511	0.69533±0.02411	0.69339±0.02271	0.69243±0.0228	0.68277±0.02509
Precision@3	0.48162±0.01271	0.48259±0.0148	0.48259±0.01269	0.48066±0.01402	0.47487±0.01464

Epochs 10

	Nodes: 10	Nodes: 50	Nodes: 100	Nodes: 200	Nodes: 500
Recall	0.80449±0.01344	0.80977±0.01069	0.8114±0.01268	0.81501±0.01463	0.8131±0.01273
NDCG	0.73929±0.01614	0.73923±0.01416	0.7392±0.0128	0.73799±0.01373	0.73704±0.01282
F1@5	0.43375±0.00639	0.43197±0.00631	0.43164±0.00621	0.43255±0.00654	0.43326±0.0085
Precision@1	0.69339±0.02609	0.68953±0.0267	0.68567±0.0246	0.67986±0.02607	0.68083±0.02466
Precision@3	0.48098±0.01111	0.4826±0.01109	0.4826±0.012	0.48196±0.01411	0.48131±0.01217

In figure 34 different validation results are plotted - with varying epoch and hidden node configurations - and a model is chosen for each epoch-stage (e.g. 3 epochs, 5 epochs, etc.). The performance differences between different node configurations (on matching epoch-stage) are small and often inconclusive, as there is no single configuration that would outperform all the other models on all the metrics (table 14). Since the performance differences were very limited, the overall best model for each epoch-stage was chosen based on a sum of all the individual metrics, where the architecture yielding the highest result is chosen. The chosen models are highlighted with a square in the legend for each epoch-stage in figure 34 (similarly for table 14).

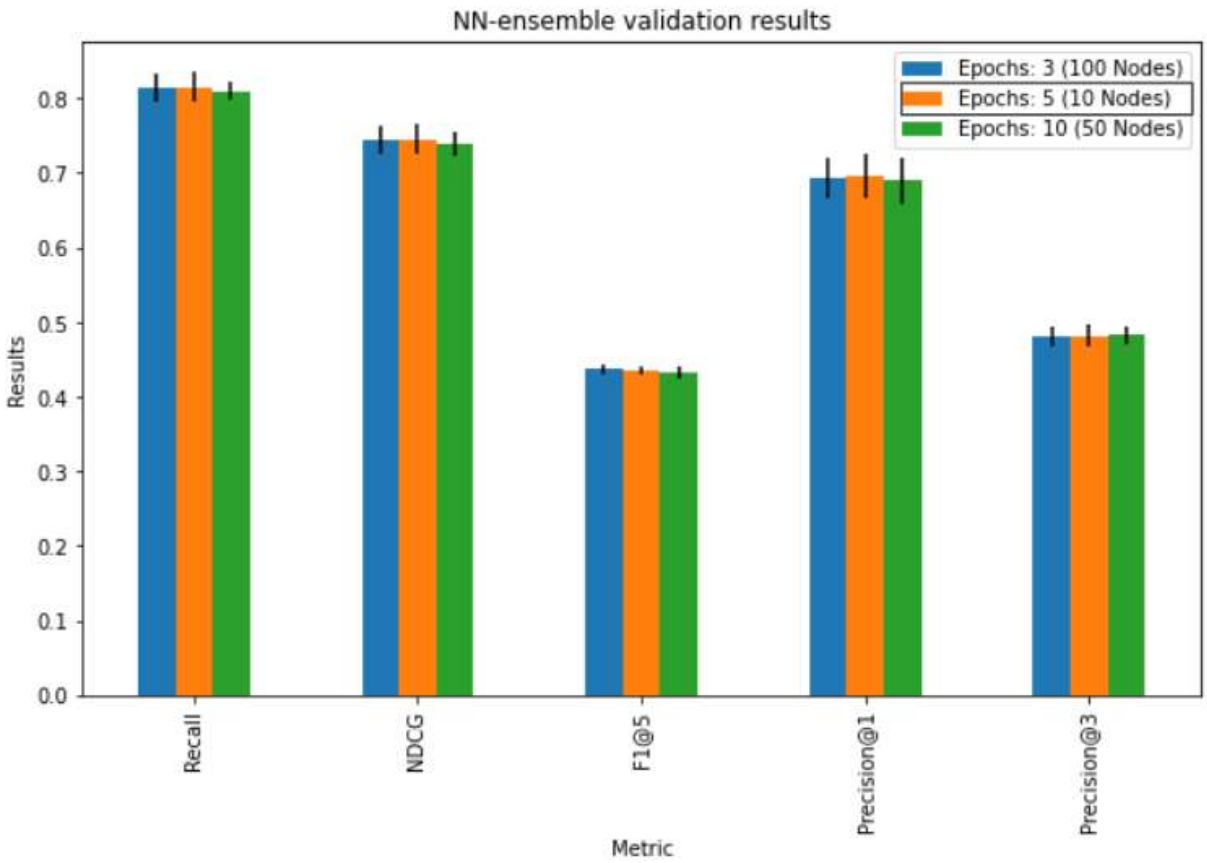

Figure 35 NN-ensemble validation results. Comparing the different epoch-stage models and choosing the overall best model configuration.

Table 15 Comparing different epoch-stage architectures mean values with 95% confidence interval

	Epochs: 3 (100 Nodes)	Epochs: 5 (10 nodes)	Epochs: 10 (50 Nodes)
Recall	0.815±0.01668	0.81528±0.0179	0.80977±0.01069
NDCG	0.74561±0.01657	0.74555±0.01705	0.73923±0.01416
F1@5	0.43684±0.0057	0.43503±0.00546	0.43197±0.00631
Precision@1	0.69339±0.02277	0.69629±0.02511	0.68953±0.0267
Precision@3	0.4813±0.01221	0.48162±0.01271	0.4826±0.01109

Figure 35 brings together the different epoch-stage models chosen in the previous step. Even though the performance differences are somewhat more noticeable here than between the different epoch-stage models, all the models are still within 1-2 % from each other and there isn't a single dominant model. With little difference on the performance, the model is selected using a sum of all the metrics and choosing the architecture with the highest score. With this scoring method, the "Epochs 5"-model is selected.

6.3.3 Model selection

Model selection is the last phase in determining the final model configuration to use in the subject indexing task. In the previous part - model validation - validation data was used for hyperparameter optimization. Model validation was performed on three indexing algorithms - Omikuji, simple ensemble and NN-ensemble - and optimal architectures were chosen for each model. In this phase - model selection - all the models (including both the models with hyperparameters (i.e. the validated models) and models without hyperparameters) are trained using train and validation data and evaluated using test data and their performance is measured using the same metrics as in the validation phase. From these results the dominant model is chosen.

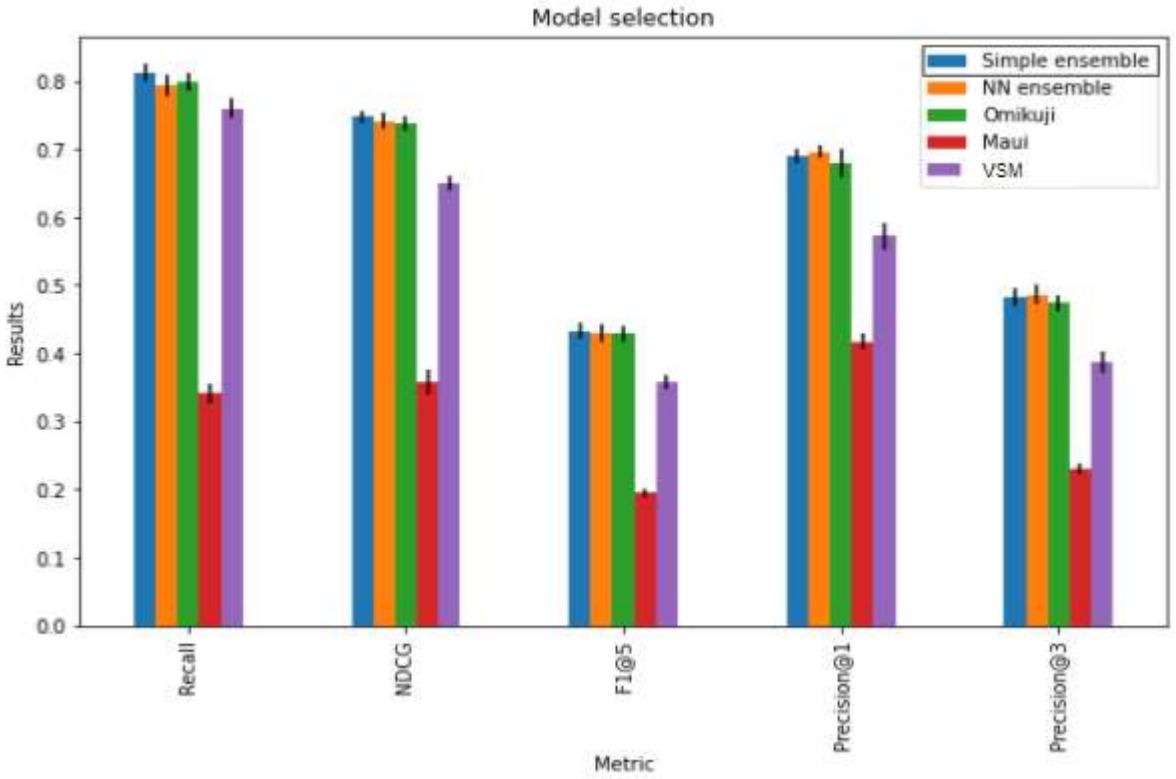

Figure 36 Model selection: performance metrics are compared, and the overall best model is chosen.

Table 16 Model selection mean scores with 95% confidence interval using test data

95% Confidence interval (with n=5)

	Simple ensemble	NN ensemble	Omikuji	Maui	VSM
Recall	0.81216±0.01147	0.79266±0.01415	0.79945±0.0115	0.34142±0.013	0.75997±0.01229
NDCG	0.74706±0.00864	0.74059±0.01049	0.73677±0.00933	0.35678±0.01645	0.64956±0.00985
F1@5	0.43219±0.01048	0.42976±0.01196	0.42886±0.01059	0.19442±0.0065	0.35809±0.00859
Precision@1	0.6898±0.00969	0.69621±0.00856	0.6785±0.0186	0.41753±0.01082	0.57257±0.0176
Precision@3	0.48351±0.01128	0.48648±0.01222	0.47398±0.01106	0.2305±0.00663	0.38701±0.01348

Model selection results are visualized in figure 36 and mean scores with 95% confidence intervals can be seen in table 16. From the results we see that:

- Ensemble models are outperforming individual models on most metrics
- Out of the individual indexing algorithms Omikuji's performance is notably better than VSM or Maui
- Maui's results are not on par with literature, which indicates that the training data and/or indexed data is not optimal

First, the test results seem to indicate that combining suggestions from different models and 'voting' for the top predictions (i.e. calculating a weighted mean or using a neural network) is beneficial. Out of the five models, Simple Ensemble outperforms the other models on all but precision (where NN-ensemble is better) and is the most accurate model in the group. Somewhat surprisingly NN-ensembles performance is weaker than expected. Even though the neural network is specifically trained on PTV-data, it fails to create a general model that would generate better predictions than a simple weighted mean of the suggestions. Specific reasons why this happens should be further investigated. For example examining the ensemble training data to see what indexing terms it contains and how they are distributed to try to understand better if the neural network is overfitting to some specific indexing choices that are not present (or at least have a different distribution) in the testing data.

Other than the ensemble models, Omikuji's performance is clearly better than VSM's or Maui's and it even outperforms NN-ensemble on recall. Out of the individual algorithms Omikuji is the superior model. VSM's performance is roughly as expected, as it is quite a simple model - using only a TF-IDF presentation and cosine similarity to identify possible indexing terms. Maui's performance on the other hand was worse than expected, as it is not on par with the experimental results seen in literature (see for example evaluation results from Suominen, 2019). Possible explanation for these sub-par results comes from both the

fact that the PTV-data used for both training and testing the algorithm is not optimally suited for Maui's indexing style. Since Maui is a lexical indexer (i.e. it uses dictionary mapping in order to prune out candidate terms from the input text and then uses a set of heuristic features (table 2) to filter and predict most likely subjects) it will try to find suitable indexing terms directly from the description it is given. Given that in the PTV-data most often the input consists of a few short sentences that describe the service in question, it is likely that the description does not include all necessary indexing terms in verbatim, meaning that the candidate terms Maui generates will not include the correct terms. To examine this further the PTV-data would need to be investigated to see if it contains necessary terms for indexing. It should be noted though, that getting a low score on these metrics does not directly indicate that the predicted indexing terms are of bad quality, it only means that the predicted terms are different from the manually labeled ones. To assess the *quality* of the indexing terms a manual inspection should be performed, since there exists numerous other indexing words that are of good quality (i.e. describe the service well) but are, for some reason, not included as an index label in the test data.

After reviewing the test results for each indexing algorithm, based on performance on the set metrics the simple ensemble should be considered as the best alternative. To achieve better results, both NN-ensemble's and Maui's performance should be investigated further to see if there are some modifications that might lead to better performance. Improving Maui's performance could potentially also improve both NN-ensemble and simple ensemble performance, as they are dependent on the suggestions from the individual algorithms.

7 Conclusions and summary

This thesis started with three research questions that pondered on the possibility of finding and evaluating different methods for semantic annotation. To summarize and conclude the findings of this thesis, it seems fitting to yet again present the same questions, provide some answers and also reflect on the questions that have developed along the way and are still, to some degree, left unanswered. So we'll start with the research questions:

1) What techniques/algorithms can be used for semantic annotation?
2) How to evaluate the results between different algorithmic approaches?
3) How to implement a working solution for semantic annotation?

First question inquired about different techniques and algorithms for semantic annotation. Although at first the idea of semantic annotation (i.e. assigning the main topics of a

document using a controlled vocabulary) seems fairly niche, it turns out there is a multitude of different approaches and openly available algorithms for semantic annotation and this area of research is in the limelight of both industry and academia. Reason for this is more in the framing of the problem than in the pure interest towards semantic annotation. By looking at semantic annotation as a supervised learning problem where we have a large target set of labels (e.g. vocabulary) and each training example (e.g. document) is labeled using a tiny subset of relevant labels (e.g. topics) taken from the target set, it is somewhat trivial to see that solving semantic annotation efficiently also solves a host of other similarly framed supervised learning tasks. More specifically these types of problems refer to a branch of machine learning (ML) called *extreme multi-label classification* (XMC).

In addition to semantic annotation, the XMC-framework can be used to address problems arising in recommendation systems, ranking and web-advertising (Babbar and Schölkopf, 2019). For example Agrawal et al. (2013) demonstrated that similar framework can be used to automatically recommend bid phrases based on a specific ad-landing page, where the target label set consisted of 10 million queries - accounting for large majority of the generated revenue of the search engine - and training examples of different ad-landing pages labeled with relevant queries. Similarly, e-stores can leverage the framework by recommending a subset of relevant items from the e-store's large collection based on previous actions - for example buying or browsing history - of similar users. (Babbar and Schölkopf, 2019) With such a broad problem-space, the applicative interest towards XMC is high and well-organized with easily available[37] benchmark datasets, metrics, results and code for different approaches.

In this thesis a set of three different term assignment algorithms were evaluated. In addition to the three individual models, two ensemble models - term assignment models combining multiple different algorithms - were also tested and evaluated. Individual term assignment models included *Omikuji, Maui* and *Vector Space Model* (VMS) and the evaluated ensemble models were *simple ensemble* and *Neural Network ensemble* (nn-ensemble). Out of the individual models, Omikuji is the most up-to-date ML-algorithm with state-of-the-art performance on different XMC benchmark datasets (Prabhu et al., 2018). Maui (Medelyan, 2009) is mainly a lexical indexer, using dictionary mapping and other text features in order to map documents to specific labels and VSM is purely a representational model using vectors to represent labels and map them to documents using cosine similarity as a defining metric. Ensemble models on the other hand combine results from the three individual models and

[37] See for example: http://manikvarma.org/downloads/XC/XMLRepository.html

predict which label suggestions should be included in the final prediction. As each individual model suffers from some type of inaccuracies, combining the results can help diminish these individual errors and increase the overall prediction accuracy - which can be seen analogous to random forest where multiple decision trees are combined.

In summary to the first research question, the problem of semantic annotation can be framed as a (extreme) multi-class machine learning problem where we have a large set of target labels and each training/test example should be indexed with a small subset of relevant labels. This type of problem framing is analogous to many types of recommendation, ranking and web-advertising schemes and solving these efficiently is at the focus of both industry and academia. In addition to machine learning based algorithms there also exists algorithms that use lexical approaches where dictionary mapping and different textual features are used to map correct labels to given text inputs. However, these approaches fall a bit short from the current state-of-the-art machine learning based methods, but as was seen in this thesis, combining different types of algorithms - both associative and lexical - by using ensemble models can yield slightly better results than any algorithm individually.

The second research question addressed the problem of evaluating term assignment results. To this question there are two different ways of answering, one that is straightforward and other that is a bit more nuanced. The straightforward way of answering this question is to say that there exists multiple different metrics that can be used to measure the proficiency of any term assignment model. In this thesis four different metrics were used, of which three use the results from the confusion matrix to calculate statistics of how the many of the desired labels are recalled or how precise the predictions are and finally combining both these metrics to get an overall score - using recall, precision and F1-score respectively. In addition to metrics from the confusion matrix *Normalized Discounted Cumulative Gain* (NDCG) -metric was used to evaluate the ranking quality of the algorithm and assess if the algorithm provides the most relevant labels in correct order. In this thesis these four metrics were used to assess all the models and decide on the ranking of the different algorithms. Although using these metrics is a sufficient way of studying the relative fitness of each model, it still leaves room for improvement. A more nuanced way of evaluating different algorithms should also consider the true output of each model and reason from there if the predictions are appropriate to the given text input. In this thesis the vocabulary consisted of over 40 000 individual terms, and yet, most of the training/test examples had between 2 to 5 correct labels attached to them. It is unreasonable to think that only the terms used to index the train/test examples are correct terms. This means that there could also exist other - possibly even more suitable - indexing terms that some other (human

or machine) indexer might use. However, using the mentioned statistics, these terms would register as false predictions and would degrade the final results. For this reason, a pure statistical analysis of indexing performance can give a partly wrong impression of the overall quality. To combat this usually some type of manual inspection of the indexing results is necessary.

Last, a question about implementation. Even though the final goal for creating and evaluating different term assignment algorithms is eventually aimed at deploying the trained models online, the scope of this thesis was to find possible models for term assignment, train and evaluate those models and finally decide on the best alternative. Deployment to an online platform and the continued maintenance plans necessary for a web-service were deliberately left out, in order to focus solely on the issues and possible solutions around semantic annotation. From this point of view the question of implementation comes down to finding suitable algorithms, vocabularies and training data to build the models, then evaluating the different approaches and finally packaging everything so that taking the next step of actual deployment is possible. In this regard the work was successful. An open source tool called Annif[38] was used to train and evaluate various term assignment models. Training data and vocabularies were fetched from the internet using different APIs and then modified programmatically to the desired form, so that they could be used in building the term assignment models. Using Annif also mitigates the deployment process, as it is built to be used as a microservice-style REST API. This means that the implemented models are already packaged in a way that supports the deployment efforts and are aligned with the ultimate goal of providing Finnish employment and business services semi-automated metadata generation.

Overall, the efforts taken in this thesis work seem to have paid off. The trained term assignment models are working sufficiently well and even though the models are not perfect they do provide added value in generating index term suggestions, which will potentially have a positive net effect on the amount of metadata gathered from employment services. This in turn should make it easier for the customer to navigate and find suitable services. Further down the road, quality metadata could also aid in matching user profiles to recommended or beneficial employment services that could potentially help them in finding interesting work or ways to improve and renew their own expertise.

[38] http://annif.org/

8 References

Alashwal, H., Halaby, M., Crouse, J., Abdalla, A. and Moustafa, A., 2019. The Application Of Unsupervised Clustering Methods To Alzheimer'S Disease. [online] Available at: <https://www.frontiersin.org/articles/10.3389/fncom.2019.00031/full> [Accessed 5 July 2020].

Allahyari, M., Pouriyehv, S., Assef, M., Safaei, S., Trippe, E., Gutierrez, J. and Kochut, K., 2017. Text Summarization Techniques: A Brief Survey. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1707.02268.pdf> [Accessed 25 April 2020].

Baker, F., 1962. Information Retrieval Based Upon Latent Class Analysis. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/321138.321148> [Accessed 1 May 2020].

Baker, F., Veal, D. and Wyatt, B., 1972. TOWARDS AUTOMATIC PROFILE CONSTRUCTION. [online] Emerald.com. Available at: <https://www.emerald.com/insight/content/doi/10.1108/eb026529/full/html> [Accessed 2 May 2020].

Barber, S., Barraclough, E. and Gray, A., 1973. On-Line Information Retrieval As A Scientists Tool. [online] ScienceDirect. Available at: <https://www.sciencedirect.com/science/article/abs/pii/0020027173900934> [Accessed 2 May 2020].

Berger, M., 2015. Large Scale Multi-Label Text Classification With Semantic Word Vectors. [online] Cs224d.stanford.edu. Available at: <https://cs224d.stanford.edu/reports/BergerMark.pdf> [Accessed 3 May 2020].

Borko, H. and Bernick, M., 1963. Automatic Document Classification. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/321160.321165> [Accessed 1 May 2020].

Borko, H., 1964. Research In Automatic Generation Of Classification Systems | Proceedings Of The April 21-23, 1964, Spring Joint Computer Conference. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/1464122.1464173> [Accessed 1 May 2020].

Brickley, D. and Guha, R., 2014. RDF Schema 1.1. [online] W3.org. Available at: <https://www.w3.org/TR/rdf-schema/> [Accessed 3 May 2020].

Brownlee, J., 2016. What Is A Confusion Matrix In Machine Learning. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/confusion-matrix-machine-learning/> [Accessed 28 June 2020].

Chadha, N., Gangwar, R. and Bedi, R., 2015. Current Challenges And Application Of Speech Recognition Process Using Natural Language Processing: A Survey. [online] Available at: <https://www.researchgate.net/publication/291042346_Current_Challenges_and_Application_of_Speech_Recognition_Process_using_Natural_Language_Processing_A_Survey> [Accessed 25 April 2020].

Chéragui, M., 2012. Theoretical Overview Of Machine Translation. [online] Ceur-ws.org. Available at: <http://ceur-ws.org/Vol-867/Paper17.pdf> [Accessed 25 April 2020].

Chinchor, N., 1992. MUC-4 EVALUATION METRICS. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/pdf/10.3115/1072064.1072067> [Accessed 30 June 2020].

CHUNG, C. and PENNEBAKER, J., 2007. The Psychological Functions Of Function Words. [online] The Psychological Functions of Function Words. Available at:

<https://www.researchgate.net/publication/237378690_The_Psychological_Functions_of_Function_Words> [Accessed 20 April 2020].

Cleverdon, C., 1967. The CRANFIELD TESTS ON INDEX LANGUAGE DEVICES. [online] Emerald.com. Available at: <https://www.emerald.com/insight/content/doi/10.1108/eb050097/full/html> [Accessed 2 May 2020].

Cybenko, G., 1989. Approximation By Superpositions Of A Sigmoidal Function. [online] Citeseerx.ist.psu.edu. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.441.7873&rep=rep1&type=pdf> [Accessed 20 July 2020].

Davydova, O., 2018. Text Preprocessing In Python: Steps, Tools, And Examples. [online] Medium. Available at: <https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908> [Accessed 12 April 2020].

Dhillon, I., Guan, Y. and Kogan, J., 2002. Refining Clusters In High Dimensional Text Data. [online] Grid.cs.gsu.edu. Available at: <https://grid.cs.gsu.edu/~wkim/index_files/papers/refinehd.pdf> [Accessed 5 July 2020].

Dictionary.cambridge.org. 2020. Function Word. [online] Available at: <https://dictionary.cambridge.org/dictionary/english/function-word> [Accessed 20 April 2020].

Digi- ja väestötietovirasto. 2020. Palvelutietovaranto. [online] Available at: <https://dvv.fi/palvelutietovaranto> [Accessed 30 July 2020].

Finna.fi. 2020. Finna.Fi. [online] Available at: <https://www.finna.fi/Content/about?lng=en-gb> [Accessed 29 July 2020].

Finto.fi. 2020. Finto. [online] Available at: <https://finto.fi/en/> [Accessed 29 July 2020].

Fox, C., 1989. A Stop List For General Text. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/378881.378888> [Accessed 21 April 2020].

Fuhr, N. and Knortz, G., 2020. Retrieval Test Evaluation Of A Rule Based Automatic Indexing (AIR/PHYS). [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.5555/636805.636831> [Accessed 2 May 2020].

Galke, L., Mai, F., Schelten, A., Brunsch, D. and Scherp, A., 2017. Using Titles Vs. Full-Text As Source For Automated Semantic Document Annotation | Proceedings Of The Knowledge Capture Conference. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/3148011.3148039> [Accessed 3 May 2020].

GitHub. 2020. Omikuji. [online] Available at: <https://github.com/tomtung/omikuji> [Accessed 1 August 2020].

Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep Learning. [online] Deeplearningbook.org. Available at: <https://www.deeplearningbook.org/contents/mlp.html> [Accessed 20 July 2020].

Google Developers. 2020. Classification: Accuracy | Machine Learning Crash Course. [online] Available at: <https://developers.google.com/machine-learning/crash-course/classification/accuracy> [Accessed 28 June 2020].

Google Developers. 2020. Classification: Precision And Recall | Machine Learning Crash Course. [online] Available at: <https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall> [Accessed 28 June 2020].

Grosse, R., 2018. Lecture 9: Generalization. [online] Cs.toronto.edu. Available at: <https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L09%20Generalization.pdf> [Accessed 23 June 2020].

Gruber Thomas R., A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5(2):199-220, 1993

Gruber, T., 2009. Ontology. [online] Available at: <http://tomgruber.org/writing/ontology-definition-2007.htmone> [Accessed 1 April 2020].

Gutierrez-Osuna, R., 2001. Three-Way Data Splits. [online] Research.cs.tamu.edu. Available at: <http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf> [Accessed 22 June 2020].

Harping, P., 2010. Introduction To Controlled Vocabularies. Los Angeles: Getty Research Institute, pp.24 - 47.

Hayak, S., 2005. Neural Networks; A Comprehensive Foundation. [online] Cdn.preterhuman.net. Available at: <https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/Neural%20Networks%20-%20A%20Comprehensive%20Foundation%20-%20Simon%20Haykin.pdf> [Accessed 18 July 2020].

Hayes, P. and Weinstein, S., 1990. Construe-TIS: A System For Content-Based Indexing Of A Database Of News Stories. [online] Aaai.org. Available at: <https://www.aaai.org/Papers/IAAI/1990/IAAI90-006.pdf> [Accessed 2 May 2020].

Hornik, K., 1991. Approximation Capabilities Of Multilayer Feedforward Networks. [online] Available at: <https://doi.org/10.1016/0893-6080(91)90009-T> [Accessed 20 July 2020].

Hornik, K., Feinerer, I., Kober, M. and Buchta, C., 2012. Spherical K-Means Clustering. [online] Available at: <https://www.researchgate.net/publication/283428234_Spherical_k-Means_Clustering> [Accessed 5 July 2020]

Jain, H., Balasubramanian, V., Chundur, B. and Varma, M., 2019. Slice: Scalable Linear Extreme Classifiers Trained On 100 Million Labels For Related Searches. [online] Manikvarma.org. Available at: <https://dl.acm.org/doi/10.1145/3289600.3290979> [Accessed 7 June 2020].

Janocha, K. and Czarnecki, W., 2017. On Loss Functions For Deep Neural Networks In Classification. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1702.05659.pdf> [Accessed 21 July 2020].

Järvelin, K. and Kekäläinen, J., 2002. Cumulated Gain-Based Evaluation Of IR Techniques. [online] Cc.gatech.edu. Available at: <https://www.cc.gatech.edu/~zha/CS8803WST/dcg.pdf> [Accessed 30 June 2020].

Johnson, M., Schuster, M., Le, Q., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M. and Dean, J., 2017. Google'S Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. [online] Association for Computational Linguistics. Available at: <https://www.aclweb.org/anthology/Q17-1024.pdf> [Accessed 25 April 2020].

Jones, K., 1972. A Statistical Interpretation Of Term Specificity And Its Application In Retrieval. [online] Semanticscholar.org. Available at: <https://www.semanticscholar.org/paper/A-statistical-interpretation-of-term-specificity-in-Jones/4f09e6ec1b7d4390d23881852fd7240994abeb58> [Accessed 1 May 2020].

Jones, K., 2001. Natural Language Processing: A Historical Review. [online] Cl.cam.ac.uk. Available at: <https://www.cl.cam.ac.uk/archive/ksj21/histdw4.pdf> [Accessed 5 April 2020].

Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T., 2017. Bag Of Tricks For Efficient Text Classification. [online] ACL Anthology. Available at: <https://www.aclweb.org/anthology/E17-2068/> [Accessed 3 May 2020].

Jurafsky, D. and Martin, J., 2019. Speech And Language Processing. [online] Web.stanford.edu. Available at: <https://web.stanford.edu/~jurafsky/slp3/edbook_oct162019.pdf> [Accessed 15 April 2020].

Jurafsky, D., 2012. Introduction To NLP - What Is Natural Language Processing?. [online] Slideplayer.com. Available at: <https://slideplayer.com/slide/7759122/> [Accessed 5 April 2020].

Jyx.jyu.fi. 2020. JYX. [online] Available at: <https://jyx.jyu.fi/?locale-attribute=en> [Accessed 30 July 2020].

Keen, M., 1973. THE ABERYSTWYTH INDEX LANGUAGES TEST. [online] Emerald.com. Available at: <https://www.emerald.com/insight/content/doi/10.1108/eb026547/full/html> [Accessed 2 May 2020].

Khandagale, S., Xiao, H. and Babbar, R., 2019. Bonsai - Diverse And Shallow Trees For Extreme Multi-Label Classification. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1904.08249.pdf> [Accessed 2 August 2020].

Kim, D., 2015. Group-Theoretical Vector Space Model. [online] Available at: <https://www.researchgate.net/publication/272119920_Group-theoretical_vector_space_model> [Accessed 21 May 2020].

Kim, S., & Beck, H. W. (2006). A Practical Comparison Between Thesaurus and Ontology Techniques As a Basis for Search Improvement. Journal of Agricultural & Food Information, 7(4), 23–42. doi:10.1300/j108v07n04_04

Kim, Y., 2014. Convolutional Neural Networks For Sentence Classification. [online] arXiv.org. Available at: <https://arxiv.org/abs/1408.5882> [Accessed 3 May 2020].

Kingma, D. and Ba, J., 2014. Adam: A Method For Stochastic Optimization. [online] arXiv.org. Available at: <https://arxiv.org/abs/1412.6980> [Accessed 17 June 2020].

Kingma, D. and Ba, J., 2015. Adam: A Method For Stochastic Optimization. [online] arXiv.org. Available at: <https://arxiv.org/abs/1412.6980> [Accessed 4 August 2020].

Kodinariya, T. and Makwana, P., 2013. Review On Determining Of Cluster In K-Means Clustering. [online] Research gate. Available at: <https://www.researchgate.net/publication/313554124_Review_on_Determining_of_Cluster_in_K-means_Clustering> [Accessed 3 May 2020].l

Koehrsen, W., 2018. Beyond Accuracy: Precision And Recall. [online] Medium. Available at: <https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c> [Accessed 29 June 2020].

Kowsari, K., Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L. and Brown, D., 2019. Text Classification Algorithms: A Survey. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1904.08067.pdf> [Accessed 1 May 2020].

Larkey, L., 1998. Automatic Essay Grading Using Text Categorization Techniques. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/290941.290965> [Accessed 3 May 2020].

Larkey, L., 1999. A Patent Search And Classification System. [online] Citeseerx.ist.psu.edu. Available at: <https://www.semanticscholar.org/paper/A-patent-search-and-classification-system-Larkey/25360587f36bfa025542cfb3484fb08c93b96926> [Accessed 3 May 2020].

Lassila, O. and McGuinness, D., 2001. The Role Of A Frame-Based Representation On The Semantic Web. [online] Ksl.stanford.edu. Available at: <http://www.ksl.stanford.edu/people/dlm/etai/lassila-mcguinness-fbr-sw.html> [Accessed 1 April 2020].

Lewis, D. and Ringuette, M., 1994. A Comparison Of Two Learning Algorithms For Text Categorization. [online] Pdfs.semanticscholar.org. Available at: <https://pdfs.semanticscholar.org/e9fd/1a7ae0322d417ab2d32017e373dd50efc063.pdf> [Accessed 3 May 2020].

Lo, R., He, B. and Ounis, I., 2005. Automatically Building A Stopword List For An Information Retrieval System. [online] Terrierteam.dcs.gla.ac.uk. Available at: <http://terrierteam.dcs.gla.ac.uk/publications/rtlo_DIRpaper.pdf> [Accessed 20 April 2020].

Luhn, H., 1958. The Automatic Creation Of Literature Abstracts. [online] Courses.ischool.berkeley.edu. Available at: <http://courses.ischool.berkeley.edu/i256/f06/papers/luhn58.pdf> [Accessed 1 May 2020].

Manning, C., Raghavan, P. and Schütze, H., 2009. Introduction To Information Retrieval. [online] Nlp.stanford.edu. Available at: <https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf> [Accessed 23 May 2020].

Marcus, M., Marcinkiewicz, M. and Santorini, B., 1993. Building A Large Annotated Corpus Of English: The Penn Treebank. [online] Available at: <https://www.aclweb.org/anthology/J93-2004.pdf> [Accessed 16 April 2020].

Maron, M., 1961. Automatic Indexing: An Experimental Inquiry. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/321075.321084> [Accessed 29 April 2020].

Martinez, A., Estrada, H. and Hernandez, Y., 2017. Semantic Annotation Of Unstructured Documents Using Concepts Similarity. [ebook] Scientific Programming. Available at: <https://www.hindawi.com/journals/sp/2017/7831897/> [Accessed 25 April 2020].

Medelyan, O., 2009. Human-Competitive Automatic Topic Indexing. [online] Researchcommons.waikato.ac.nz. Available at: <https://researchcommons.waikato.ac.nz/bitstream/handle/10289/3513/thesis.pdf?sequence=1&isAllowed=y> [Accessed 25 April 2020].

Merriam-webster.com. 2020. Definition Of ONTOLOGY. [online] Available at: <https://www.merriam-webster.com/dictionary/ontology> [Accessed 31 March 2020].

Michelbacher, L., 2013. Multi-Word Tokenization For Natural Language Processing. [online] D-nb.info. Available at: <https://d-nb.info/1046313010/34> [Accessed 12 April 2020].

Miller, S., 2013. Http://Dcevents.Dublincore.Org. [online] Available at: <http://dcevents.dublincore.org/IntConf/dc-2013/paper/download/140/105> [Accessed 4 April 2020].

Munteanu, D., 2007. VECTOR SPACE MODEL FOR DOCUMENT REPRESENTATION IN INFORMATION RETRIEVAL. [online] Available at: <https://www.researchgate.net/publication/45087099_Vector_space_model_for_document_representation_in_information_retrieval> [Accessed 22 May 2020].

Nadkarni, P., Ohno-Machado, L. and Chapman, W., 2011. Natural Language Processing: An Introduction. [online] Akaon.com. Available at: <https://akaon.com/blog/wp-content/uploads/2014/08/NLP%20-%20an%20introduction%20-%20Nadkarni.pdf> [Accessed 25 April 2020].

Ng, A. and Katanforoosh, K., 2018. Deep Learning: Neural Networks. [online] Cs229.stanford.edu. Available at: <http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf> [Accessed 18 July 2020].

Nielsen, Michael A., “Neural networks and Deep Learning”, Determination Press, 2015

Nivre, J., de Marneffe, M., Ginter, F., Goldberg, Y., Hajic, J., Manning, C., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R. and Zeman, D., 2016. Universal Dependencies V1: A Multilingual Treebank Collection. [online] Petrovi.de. Available at: <https://www.petrovi.de/data/lrec16.pdf> [Accessed 18 April 2020].

Noy, N. and McGuinness, D., 2001. Ontology Development 101: A Guide To Creating Your First Ontology. [online] Protege.stanford.edu. Available at: <https://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html> [Accessed 10 May 2020].

Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S., 2018. Activation Functions: Comparison Of Trends In Practice And Research For Deep Learning. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1811.03378.pdf> [Accessed 21 July 2020].

Palmer, D., 2000. Tokenization And Sentence Segmentation. [online] S3.amazonaws.com. Available at: <https://s3.amazonaws.com/tm-town-nlp-resources/ch2.pdf> [Accessed 13 April 2020].

Parker, L., 2007. Notes On Multilayer, Feedforward Neural Networks. [online] Web.eecs.utk.edu. Available at: <http://web.eecs.utk.edu/~leparker/Courses/CS594-fall07/handouts/Neural-net-notes.pdf> [Accessed 21 July 2020].

Petrov, S., McDonald, R. and Das, D., 2011. Universal Part-Of-Speech Tagset. [online] Research gate. Available at: <https://www.researchgate.net/publication/51887367_A_Universal_Part-of-Speech_Tagset> [Accessed 15 April 2020].

Porter, M., 1980. An Algorithm For Suffix Stripping. [online] Pdfs.semanticscholar.org. Available at: <https://pdfs.semanticscholar.org/a651/bb7cc7fc68ece0cc66ab921486d163373385.pdf?_ga=2.213658132.1061355828.1587209951-32053718.1586509589> [Accessed 18 April 2020].

Prabhu, Y. and Varma, M., 2014. Fastxml: A Fast, Accurate And Stable Tree-Classifier For Extreme Multi-Label Learning. [online] Manikvarma.org. Available at: <https://dl.acm.org/doi/10.1145/2623330.2623651> [Accessed 3 May 2020].

Prabhu, Y., Kag, A., Harsola, S., Agrawal, R. and Varma, M., 2018. Parabel: Partitioned Label Trees For Extreme Classification With Application To Dynamic Search Advertising. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/3178876.3185998> [Accessed 3 May 2020].

Rijsbergen, C., 1979. INFORMATION RETRIEVAL. [online] Dcs.gla.ac.uk. Available at: <http://www.dcs.gla.ac.uk/Keith/Chapter.2/Ch.2.html> [Accessed 21 April 2020].

Robertson, S., 2004. Understanding Inverse Document Frequency: On Theoretical Arguments For IDF. [online] Researchgate. Available at: <https://www.researchgate.net/publication/238123710_Understanding_Inverse_Document_Frequency_On_Theoretical_Arguments_for_IDF> [Accessed 25 April 2020].

Saif, Hassan; Fern´andez, Miriam; He, Yulan and Alani, Harith (2014). On stopwords, filtering and data sparsity for sentiment analysis of Twitter. In: LREC 2014, Ninth International Conference on Language Resources and Evaluation. Proceedings., pp. 810–817.

Salton, G. and McGill, M., 1983. Introduction To Modern Information Retrieval. [online] Sigir.org. Available at: <https://sigir.org/files/museum/introduction_to_modern_information_retrieval/chapter_4.pdf> [Accessed 2 May 2020].

Salton, G., 1971. The SMART Retrieval System—Experiments In Automatic Document Processing. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/book/10.5555/1102022> [Accessed 2 May 2020].

Salton, G., Wong, A. and Yang, C., 1975. A Vector Space Model For Automatic Indexing. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/361219.361220> [Accessed 2 May 2020].

Schwarz, K., 2005. Domain Model Enhanced Search - A Comparison Of Taxonomy, Thesaurus And Ontology. [online] Pdfs.semanticscholar.org. Available at: <https://pdfs.semanticscholar.org/5b63/f9696de8627500ee62e13ae32b5dfe0c857b.pdf> [Accessed 3 April 2020].

Sebastiani, F., 2002. Machine Learning In Automated Text Categorization. [online] Nmis.isti.cnr.it. Available at: <http://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf> [Accessed 5 April 2020].

Siblini, W., Pasqual, C., Lavielle, A. and Cauchois, C., 2019. Multilingual Question Answering From Formatted Text Applied To Conversational Agents. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1910.04659.pdf> [Accessed 25 April 2020].

Silfverberg, M., 2016. Morphological Disambiguation Using Probabilistic Sequence Models. [online] Helda.helsinki.fi. Available at: <https://helda.helsinki.fi/bitstream/handle/10138/167029/morpholo.pdf?sequence=1> [Accessed 19 April 2020].

Sinka, M. and Corne, D., 2003. Stop Word Modernisation: Application To Web Document. [online] Nanopdf.com. Available at: <https://nanopdf.com/download/stop-word-modernisation-application-to-web-document_pdf> [Accessed 25 April 2020].

Snyder, B., Naseem, T., Eisenstein, J. and Barzil, R., 2009. Adding More Languages Improves Unsupervised Multilingual Part-Of-Speech Tagging: A Bayesian Non-Parametric Approach. [online] Available at: <https://www.aclweb.org/anthology/N09-1010.pdf> [Accessed 16 April 2020].

Sparck-Jones, K. and Rijsbergen, C., 1975. REPORT ON THE NEED FOR AND PROVISION OF AN 'IDEAL' INFORMATION RETRIEVAL TEST COLLECTION. [online] Sigir.org. Available at: <https://sigir.org/files/museum/pub-14/pub_14.pdf> [Accessed 2 May 2020].

Stevens, M., 1965. Automatic Indexing: A State-Of-The-Art Report. [online] Govinfo.gov. Available at: <https://www.govinfo.gov/content/pkg/GOVPUB-C13-0f6ec07a727cf9b3330f1039751e37af/pdf/GOVPUB-C13-0f6ec07a727cf9b3330f1039751e37af.pdf> [Accessed 1 May 2020].

Strehl, A., Ghosh, J. and Mooney, R., 2000. Impact Of Similarity Measures On Web-Page Clustering. [online] Citeseerx.ist.psu.edu. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=CBAA6BFF0E0AF395BCCF5DD2CEA6F4F4?doi=10.1.1.41.2110&rep=rep1&type=pdf> [Accessed 5 July 2020].

Suominen, O., 2019. Annif: DIY automated subject indexing using multiple algorithms. LIBER Quarterly, 29(1), pp.1–25. DOI:http://doi.org/10.18352/lq.10285

Suominen, O., 2020. Natlibfi/Annif. [online] GitHub. Available at: <https://github.com/NatLibFi/Annif/wiki/Backend%3A-nn_ensemble> [Accessed 17 June 2020].

Syakur, M., Khotimah, B., Rochman, E. and Satoto, B., 2018. Integration K-Means Clustering Method And Elbow Method For Identification Of The Best Customer Profile Cluster. [online] Research gate. Available at: <https://www.researchgate.net/publication/324553963_Integration_K-Means_Clustering_Method_and_Elbow_Method_For_Identification_of_The_Best_Customer_Profile_Cluster> [Accessed 3 May 2020].

Tagami, Y., 2017. Annexml: Approximate Nearest Neighbor Search For Extreme Multi-Label Classification. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/3097983.3097987> [Accessed 3 May 2020].

Toepfer, M. and Seifert, C., 2018. Fusion Architectures For Automatic Subject Indexing Under Concept Drift. [online] Ris.utwente.nl. Available at: <https://ris.utwente.nl/ws/portalfiles/portal/80439235/Toepfer2018_ijdl_subject_indexing_under_concept_drift_preprint.pdf> [Accessed 14 May 2020].

Toman, M., Tesar, R. and Jezek, K., 2006. Influence Of Word Normalization On Text Classification. [online] Textmining.zcu.cz. Available at: <http://textmining.zcu.cz/publications/inscit20060710.pdf> [Accessed 18 April 2020].

W3.org. 2012. OWL 2 Web Ontology Language Document Overview (Second Edition). [online] Available at: <https://www.w3.org/TR/owl2-overview/> [Accessed 3 May 2020].

W3.org. 2020. RDF 1.1 Concepts And Abstract Syntax. [online] Available at: <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#resources-and-statements> [Accessed 28 March 2020].

W3.org. 2020. RDF 1.1 Primer. [online] Available at: <https://www.w3.org/TR/rdf11-primer/#section-IRI> [Accessed 28 March 2020].

Wilbur, J. and Kim, W., 2003. The Dimensions Of Indexing. [online] PubMed Central (PMC). Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1480214/> [Accessed 3 May 2020].

Willett, P., 2006. The Porter Stemming Algorithm: Then And Now. [online] researchgate. Available at: <https://www.researchgate.net/publication/33038304_The_Porter_stemming_algorithm_Then_and_now> [Accessed 19 April 2020].

Williams, J., 1966. DISCRIMINANT ANALYSIS FOR CONTENT CLASSIFICATION. [online] Apps.dtic.mil. Available at: <https://apps.dtic.mil/docs/citations/AD0630127> [Accessed 1 May 2020].

Xu, Y. and Goodacre, R., 2018. On Splitting Training And Validation Set: A Comparative Study Of Cross-Validation, Bootstrap And Systematic Sampling For Estimating The Generalization Performance Of Supervised Learning. [online] Link.springer.com. Available at: <https://link.springer.com/content/pdf/10.1007%2Fs41664-018-0068-2.pdf> [Accessed 22 June 2020].

Yen, I., Huang, X., Zhong, K., Ravikumar, P. and Dhillon, I., 2016. PD-Sparse : A Primal And Dual Sparse Approach To Extreme Multiclass And Multilabel Classification. [online] Proceedings.mlr.press. Available at: <http://proceedings.mlr.press/v48/yenb16.pdf> [Accessed 3 May 2020].

Zhong, S., 2005. Efficient Online Spherical K-Means Clustering - IEEE Conference Publication. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/1556436/> [Accessed 5 July 2020].

Agrawal, R., Gupta, A., Prabhu, Y. and Varma, M., 2013. *Multi-Label Learning With Millions Of Labels: Recommending Advertiser Bid Phrases For Web Pages*. [online] Manikvarma.org. Available at: <http://manikvarma.org/pubs/agrawal13.pdf> [Accessed 10 August 2020].

Babbar, R., and Schölkopf, B. 2019. Data scarcity, robustness and extreme multi-label classification. Machine Learning, 108(8-9), 1329-1351. https://doi.org/10.1007/s10994-019-05791-5