Text classification using concept maps

Bachelor-Thesis von David Marlon Gengenbach aus Gemmingen-Stebbach
Tag der Einreichung:

1. Gutachten: Prof. Dr. Iryna Gurevych
2. Gutachten: Prof. Dr. Kristian Kersting

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
UKP Lab

Text classification using concept maps

Vorgelegte Bachelor-Thesis von David Marlon Gengenbach aus Gemmingen-Stebbach

1. Gutachten: Prof. Dr. Iryna Gurevych
2. Gutachten: Prof. Dr. Kristian Kersting

Tag der Einreichung:

Thesis Statement

Pursuant to §23 paragraph 7 of APB TU Darmstadt, I herewith formally declare that I, David Marlon Gengenbach, have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once. In the submitted thesis the written copies and the electronic version for archiving are identical in content.

Darmstadt, 18th of January 2018

(David Marlon Gengenbach)

Contents

Abstract

Text classification is an important natural language processing task. Here, finding appropriate representations for text content is crucial and directly influences the success in solving the task. While state-of-the-art count-based representations like Bag-Of-Words perform well on numerous tasks, they often go hand-in-hand with structural information loss of their underlying text, eg. the semantic or syntax. While one can augment these count-based representations to partially solve these issues, in this work, we instead explore another, more structured text representation, namely concept maps or concept graphs. Automatically generated concept maps have successfully been employed to summarize text and therefor might capture important concepts and their relations. This work explores whether and how one can harness this structural information in the context of text classification. For this, we utilize different graph kernel-based approaches to extract features from the concept maps and examine their usefulness in conjunction with conventional text classification approaches.

1 Introduction

Text is ubiquitous and arguably one of the most important information mediums to this date. Every day and in numerous domains, new texts are produced. Due to the number of digitally available text, processing texts automatically instead of manually becomes more appealing and in some cases even necessary. However, language itself is an inherently ambiguous information medium, which in turn significantly hardens the task of processing it automatically. Still, numerous approaches have been devised to extract information from text to solve different natural language processing tasks, such as text classification/categorization [MS00, p. 575], summarization [MMS99] or language translation [Wea55], to name but a few.

In this work we concentrate on text classification [MS00, p. 232], or categorization, which entails the task of predicting the class of a text document based on its content. In this context, finding an appropriate representation of text is crucial and can greatly influence the success of solving the task. More conventional text representations are often based on counting the words of a text and representing the text content by these word counts. The *Bag-Of-Word* [MS00, p. 237], or BoW, representation is a widely used example of such a procedure which has also shown great real-world performance on several tasks, not only in text classification. However, simple word-count based text representations come at the expense of losing information about their underlying text, eg. the sentence structure and word order. That being said, there are several other, more sophisticated text representations, for instance co-occurrence graphs [RV13], concept maps [NC08; FG17] or document embeddings [DOL15; LB16]. Each of these representations is also subject to different trade-offs, for instance in the ability to capture appropriate information and its compute time.

In our work we are interested in graph-based text representations, especially concept maps [NC08; FG17], to overcome the aforementioned shortcomings of simple text representations like *BoW*. For this, we first generate concept maps for several text corpora and subsequently devise a number of experiments to gain a better insight into the retained information and structure of these concept maps. Additionally, we capitalize on graph kernels [KGK08] to operate on the concept maps which in turn enables us to perform graph classification. At the core of this work stands the question how we can use graph-based approaches to improve on current, conventional text representations like BoW. Similar ways to use graph-based text representations in the context of text classification have been explored in the literature, namely using co-occurrence graphs [RKV15; Nik+17], DRS graphs [GGQ11] and even concept graphs [GB15]. In Section 2.5 we will both introduce the history of this approach more throughly and explain the differences to our work.

Text Classification

The need for automatic text classification appears in many fields as the number of digitally available texts grows daily and makes manually classifying infeasible if not impossible. There are numerous real-world applications for text classification, ranging from automatically determining whether a given email is spam or not [YX08], to sentiment analysis [Liu12]. Finding ways to train an automatic classifier which can reliably and accurately classify raw texts therefor becomes appealing. There is a rich history of research on text classification and several approaches have been devised to automatically process natural language texts. For an overview and introduction into natural language processing, see [MS00].

Graph Classification

Trees, sequences, networks and other graphs occur in a lot of contexts. Because of their structured nature, graphs are perfect candidates to capture connected data, or datapoints with relations between them. Yet, operating on graphs is often challenging, precisely because of structured and often non-linear traits, since graphs are often of non-fixed size and their structure can vary greatly. Nevertheless, finding ways to automatically process graphs becomes more important as they naturally turn up in many contexts, for instance in social networks or transaction histories to name but a few. Especially the task of automatic graph classification has interesting applications, ranging from determining the toxicity of molecules to predicting friends for users in a social network. In Table 1.1 we gathered some example applications and previous work in graph classification.

Context	Vertices	Edges	Classes	
Chemistry	Atoms	Bonds	Toxicity (binary)	[Mah+05]
Biology	Amino Acids	Spatial Links	Protein Types	[Vaz+03]
Social Networks	Users	Are Friends	Bot Detection (binary)	[Wan+14]

Table 1.1: Graph classification applications.

For our work, we explore the usefulness of graph representations of text, namely co-occurrence graphs and concept maps. In Section 2.2 we will introduce the graphs and approaches to classify them more throughly.

1.1 Hypothesis and Goals

In this work, we evaluate the usefulness of graph representations generated from text in the context of text classification. In particular, we work out how or whether we can harness the structure of concept maps to improve text-based classification performance. The main hypothesis of this work is:

> *Concept maps capture structural information about the underlying text. Leveraging this structural information in the context of text classification leads to improved classification performance.*

There are a great number of approaches for text classification, a lot of them based on counting the words of a text and learning a classifier with these frequencies. Yet, these word-count based approaches often do not take the semantic or syntax of sentences into account, ie. text-based approaches often do not leverage the structure and meaning of sentences. There are partial solutions for this issue, for instance by augmenting single-word counts with n-grams [MS00, p. 191] counts. N-grams are sequences of length n of consecutive words in the text, therefore they can, in principle, capture word dependency and word order. However, their usefulness is limited as they also incur a significant cost by greatly increasing the dimensionality of the resulting BoW feature vectors. That said, count-based text representations are widely used and achieve high performance both in compute time and classification scores.

In our work, as a possible solution to the aforementioned issues in conventional count-based representations, we explore how concept maps, and other graph representations like co-occurrence graphs, could improve upon or augment existing text classification approaches. We choose concept maps as our preferred graph representation since they are specially created to capture important concepts and their relation to each other, therefore might also capture the semantic and other structure of their underlying text. In the next sections we will further explain graph representations for text and why they could prove to be a viable addition to the toolbox in text classification.

1.2 Thesis Structure

In the next section, 2 *Background* we will introduce the concepts used in the rest of this work. We also offer an overview over related work and the history of the field of text- and graph based classification.

In Section 3 *Experimental Setup*, we further describe our hypothesis and outline questions regarding it. This section also covers the methodology and experiments we use to answer these questions.

Next, in Section 4 *Results and Discussion*, we then provide the results to the questions posed in the preceding chapter, interpreting them in the context of our hypothesis. Here we also provide related observations regarding our approach.

Finally, in Section 5 *Conclusions*, we gather the results of previous sections into a more high-level picture. We close with finishing remarks concerning possible further work and also interpret our results in the context of previous work.

2 Background

In this chapter we will introduce the notations and concepts we will use in subsequent chapters.

2.1 Texts

Text is ubiquitous and the task of automatically processing or classifying text has become more important as the volume of available texts grows daily [Joa98]. While text is arguably the most prominent representation for information for humans, designing algorithms to automatically parse, “understand" or classify text bear inherent complexities. Language is a highly complex method to convey information. Text often creates its own context, eg. by mentioning some concept in the beginning of a text and only implicitly relating to it at the end. Or by using different words to represent the same concept. Other difficulties arise due to varying writing styles by different authors. Another obvious problem is the ambiguous nature of language[Bri78], eg. in English the word “bear" can relate to the animal or the verb. Often the meaning of a word is defined by its context. Manually defining rules or algorithms to remove the ambiguity of language have been shown to be labor intensive if not impossible [Wei02, p. 11].

These and other difficulties harden the task of processing natural language text significantly [Cho03; Wei02]. Yet, the task of processing text emerges in important real-world applications. In text classification, for instance, massive amounts of text make manual assignment of labels virtually impossible, so figuring out automatic approaches to classify text becomes appealing. Numerous text processing and classification approaches have been proposed in the literature and have shown good accuracy in real-world scenarios.

Since text is most often of non-fixed size, several fixed-size representations have been proposed and employed to enable machine text processing. One common approach to represent texts is by counting all unique words in a text. The counts of unique words can then be, together with a fixed-size vocabulary, transformed into fixed-size vector representations. With this approach, some information about the text is omitted, namely the sentence structure, word-order, word-dependency to name just a few [Cho03]. So, after transforming a text into this representation, recovering the original text becomes impossible. While this representation is quite simple and does not capture all information of the text, when using it in the context of text-classification, these representations actually perform very well. Count-based text representations provide an easy-to-implement baseline and starting point for further improvements. For example, one extension to this representation does not only gather the simple one-word counts but also word pairs or consecutive word sequences. These sequences are called n-grams, where n stands for the size of such the sequence, eg. 2-grams consist of 2 words and so on. In section 3 we will explain these text-based approaches more throughly.

For a recent survey of the history and challenges of natural language processing, see [Cho03; Cas14].

2.2 Graphs

A labeled graph [BM76, p. 1] is a triple $G = (V, E, \pi)$, where V is the set of vertices or nodes, $E \subseteq V \times V$ are the edges and $\pi : V \to X$ is a labeling function which assigns a label to a given vertex. A graph is called undirected when $\forall (v_1, v_2) \in E : (v_2, v_1) \in E$, otherwise it is called directed. A graph $G' = (V', E', \pi)$ is called a subgraph of graph $G = (V, E, \pi)$ when $V' \subseteq V$, $E' \subseteq E$ and $\forall (v, v') \in E : v \in V' \wedge v' \in V' \wedge (v, v') \in E'$.

The (outgoing) neighbors $n(v)$ of a node v are defined as the set $n(v) = \{v' | (v, v') \in E\}$.

Walks and Paths

A walk on a graph is a (finite) sequence of edges, $w = (e_1, \cdots, e_n)$ with the constraint that $\forall 0 < i < n : \forall e_i = (v_1, v_2) \in w : \exists v_3 : e_{i+1} = (v_2, v_3)$ [Vis+06] . The length of the walk is n. The set of all possible walks is denoted W_G. A random walk [Vis+06] starting from vertex v is a walk where the first element of walk w is $e = (v, v')$ for some v'.

A path is the sequence of the vertices visited on a walk, ie. for the walk $w = ((v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$, the path is $(v_1, v_2, v_3, \ldots, v_n)$. The set of all possible paths is denoted P_G.

The distance $d_{istance}(v, v')$ between two vertices is defined as the length of the shortest path between them.

Connected components

A connected component c of a graph is a set of vertices with the constraint $\forall v, v' \in c : \exists p \in P_G : v \in p \land v' \in p$, ie. there exists a path between every vertex in the connected component. The connected component c_v of a vertex v is the connected component c of a graph where $v \in c$. The set of all connected components of a graph is $c_{all}(G) = \{c_v | v \in V\}$. A graph is called connected when there is only one connected component. Only the empty graph has zero connected components.

Degree

The in-degree d_{in} of a vertex $v \in V$ is $d_{in}(v) = |\{v | (v_1, v_2) \in E \land v_2 = v\}|$. The out-degree d_{out} is defined analogously as $d_{out}(v) = |\{v | (v_1, v_2) \in E \land v_1 = v\}|$. The degree of a vertex v is $d(v) = d_{in}(v) + d_{out}(v)$ for directed graphs, and $d(v) = degree_{in}$ for undirected graphs.

symbol	meaning
G	Graph
E	Set of edges
V	Set of vertices
$\pi(v)$	Vertex labeling function
$n(v)$	Neighbours of vertex v
W	All possible graph walks on graph G
w	Graph walk
P	All possible paths on graph G
p	Path
$d(v)$	Degree of vertex v
$d_{in}(v)$	In-degree of vertex v
$d_{out}(v)$	Out-degree of vertex v
$d_{istance}(v, v')$	Shortest path length between v and v'
$c_{all}(G)$	Set of all connected components of G
$c(v)$	Connected component of v

Figure 2.1: Graph notation.

2.2.1 Concept Map

A concept map [NG84; KHA00; FG17; GB15; Val+08] is a directed graph where the nodes are concepts. In our case, the concept maps are automatically constructed from text, yet one can also create them manually. In our work, concepts are important tokens in the text and consist of one or more words of the underlying text. An edge between two concepts show that these concepts are related to each other. Edges can have labels also. For an example of a concept map, see Figure 2.2.

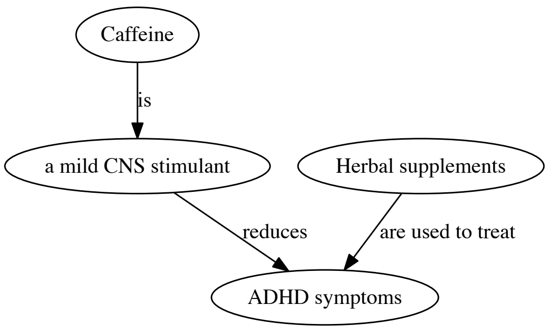


Figure 2.2: Example of a concept map. The nodes are the concepts. The directed edges connect the concepts to each other and have labels. In this example of a concept map, one could recover sentences from this graph, for example "Caffeine *is* a mild CNS stimulant". Taken and adapted from: [FG17]

Concept maps are useful for visualizing concepts and their relation to each other. They can be used to quickly explore a given topic and immediately see connections between concepts. Through the degree of a concept node, one can also infer the relative importance of that concept in the underlying text.

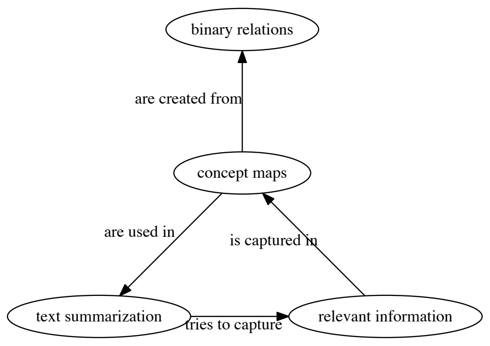
By combining different texts of the same topic, one can also create a concept map for the whole topic. In this case, the concepts are not confined to a single text. The visualization with concept maps of one or more individual texts therefor enables the non-linear exploration of multiple texts of a topic at the same time. This property makes concept maps interesting in the learning context where visual components become crucial for understanding [NG84]. Text or spoken language, on the other hand, is a linear medium [NG84, p. 54], eg. one often has to read the preceding paragraphs and understand its contents to learn about some new concept. Extending text, eg. combining several texts of the same topic, also becomes quite laborious since text is inherently linear, so one has to either manually merge the new texts or add side-notes or references to later text passages. The same task for concept maps can be far easier since one must "only" find relevant concepts in the texts and their relations to each other. After that the graph can easily be constructed. When the same concept appears in two texts, so the idea, they will be mapped to the same node. Additionally, extending a concept map is only a matter of adding new concepts or edges. One can also augment a node with attributes, eg. add a text that is relevant for some node. All these properties enable the iterative construction of concept maps by incurring little to no overhead when extending or augmenting concept maps.

Since the concepts in a concept map often contain only a small subset of the text, concept maps also summarize the underlying text. Only relevant concepts and their relation to each other are captured, making concept maps interesting for (multi-document) text summarization [FG17].

Another interesting property of concept maps with its directed edges is that they can be "unrolled" into text again, enabling conventional text-processing. Figure 2.3 shows an example of such an unrolled concept map.

While text is currently arguably the most common medium for information storage, due to the rise of more interactive information mediums, eg. computers, concept maps might become more important in the future. There are several advantages of concept maps over text, being more structured and visual amongst other things.

In Chapter 3.4 we introduce the concept map extraction implementation we use and further explain the steps in creating the concept maps automatically from given text.



concept maps **are used in** text summarization.
concept maps **are created from** binary relations.
text summarization **tries to capture** relevant information.
relevant information **is captured in** concept maps.

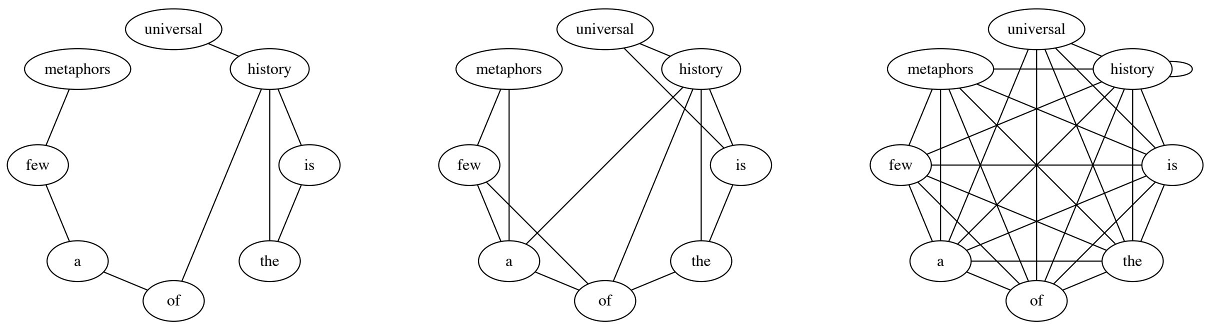
(a) Concept map **(b)** Extracted sentences

Figure 2.3: Example of a concept map with its linearized version. The directed edges with their source- and sink vertices get unrolled into sentences. Note that there are as many sentences as there are edges in the original concept map. Linearizing a concept map in such a way enables conventional text-processing approaches, eg. extracting word counts with *BoW* then classifying with a *SVM*.

2.2.2 Co-Occurrence Graph

A co-occurrence graph [NG84; RKV15; Nik+17], or graph-of-words, is generated from a text by creating a graph with all the words of the underlying text as nodes. In the next step an edge between two nodes is added if the words corresponding to the nodes co-occur in the text. Two words co-occur when the distance between the words in the text is below a given threshold, the window size. Co-occurrence graphs can have edge weights corresponding to the counts of the co-occurrence of the words in the text.

Figure 2.4 shows examples of co-occurrence graphs with different windows sizes.



(a) Window size: 1 **(b)** Window size: 2 **(c)** Window size: 6

Figure 2.4: Example for co-occurrence graphs. Generated from a quote by Jorge Luis Borges[1]: "Universal history is the history of a few metaphors". With increasing window size, the number of edges and thus also connectedness of the co-occurrence graph increases.

Since co-occurrence graphs not only contain the words of the underlying text but also their co-occurrence, they are able to capture structural information about the text. Given a co-occurrence graph, it is not possible to recover the underlying text since all information about word-order or sentence structure is omitted. That said, one could augment co-occurrence graphs by using directed edges and therefor keep the word-order intact. Even with this extension, one can not reconstruct the original text.

Because of their similarity to concept maps, we will use co-occurrence maps as a baseline for the graph classification. It has to be noted that this comparison is not entirely fair since co-occurrence graphs contain all words of the text and there is only a small loss of information. The concept maps we extracted on the other hand have a far greater compression factor as we will see, ie. they summarize the content more. For our comparison, we will also not use the edge weights since concept maps do not have edge weights.

2.3 Classification Task

Classification using machine learning is the task of automatically assigning classes, or labels, to given datapoints. Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \subseteq X \times Y$ of n items where (x_i, y_i) are dataset instances, the supervised single-label classification task is to create a classifier which automatically predicts a label $y' \in Y$ for a given $x \in X$. The labels Y of dataset D are denoted with $Y_D = (y_1, y_2, \dots, y_n)$, the predicted labels are denoted as $Y'_D = (y'_1, y'_2, \dots, y'_n)$. When the number of classes $|Y| = 2$, the classification task is called binary classification. For $|Y| > 2$ the task falls into the multi-class classification domain. In this work we will restrict ourself to single-label classification, in contrast to the multi-label classification task where each document can have one or more labels attached instead of only one as in our case.

In Figure 2.1 we summarize the classification notation. Our notation is similar to the notation used in [Bis06, p. 11].

	Notation	Definition
Datapoint	x	$x \in X$
Label	y	$y \in Y$
Real and predicted label of x_i	y_i, y'_i	$y_i, y'_i \in Y$
Instance	d_i	$(x_i, y_i) \in X \times Y$
Dataset	D	$\{(x_1, y_1), \dots, (x_n, y_n)\} \subseteq X \times Y$

Table 2.1: Classification notation.

To automatically assign the labels in a supervised way, a learning algorithm is used to train a classifier by providing it datapoints and the corresponding labels. The classifier algorithm then, internally, creates a model of the data and often optimizes internal parameters to fit the data. During prediction, that is when wanting to automatically assign a label to a datapoint x_i, the new datapoint is fed into the classifier which in turn returns the predicted label, y'_i.

2.3.1 Classifiers

In this work, we will mostly use a classification algorithm called *Support Vector Machine* [CV95], or *SVM*. The SVM has shown to be robust in its performance [Joa98] and exhibits state-of-the art performance on several tasks and is the go-to classification algorithm in a great number of previous works in the field [Joa98; VFS12; NB06; KM12; Kor+08].

For an more throughout explanation of the SVM and its advantages especially for the text classification task, please see [Joa98]. We will further explain the SVM and its parameters in Section 3.3.1.

2.3.2 Performance Metrics

The performance or score of a trained classifier is evaluated by comparing the labels it predicts, y_i', for datapoints x_i with the real label y_i in the dataset. There are several metrics for evaluating the performance of a classifier. The classification metrics we use in this work are summarized in Figure 2.5. For an extensive overview of classification metrics, see [For03]. In this work, we will use the term classification performance synonymously with classification score, unless otherwise stated. On the other hand, we will refer to the run-time of a classification algorithm by the classification run-time.

	Notation	Definition	
True positives of y	TP_y	$\lvert\{i \mid y_i' = y, y_i' = y_i\}\rvert$	
False positives of y	FP_y	$\lvert\{i \mid y_i' = y, y_i \neq y\}\rvert$	
True negatives of y	TN_y	$\lvert\{i \mid y_i' \neq y, y_i \neq y\}\rvert$	
False negatives of y	FN_y	$\lvert\{i \mid y_i' \neq y, y_i = y\}\rvert$	
Accuracy	A	$\frac{\sum_{y \in Y} TP_y}{\lvert D\rvert}$	
Precision of y	Pr_y	$\frac{TP_y}{TP_y + FP_y}$	$Pr_{\text{macro}} = \frac{\sum_{y \in Y} Pr_y}{\lvert Y\rvert}$
Recall of y	R_y	$\frac{TP_y}{TP_y + FN_y}$	$R_{\text{macro}} = \frac{\sum_{y \in Y} R_y}{\lvert Y\rvert}$
F1 score of y	$F1_y$	$2 \cdot \frac{Pr_y \cdot R_y}{Pr_y + R_y}$	$F1_{\text{macro}} = \frac{\sum_{y \in Y} F1_y}{\lvert Y\rvert}$

Figure 2.5: Classification performance metric notation. We will use accuracy, macro precision, macro recall and the macro F1 score. The macro version of a metric takes into account how many elements there are in each class which is helpful for skewed datasets, ie. where the number of instances per label is not distributed uniformly per label.

2.4 Kernels

In most cases, texts and graphs are not of fixed size. Most classification algorithms, however, operate on fixed sized vectors. Fortunately, there exist several approaches to overcome this problem, one of them being so-called kernels [KM12][Bis06, p. 295].

A kernel is a function k which returns a measure of similarity between two objects:

$$k : X \times X \to \mathbb{R}$$

A kernel function has to be [KM12]:

- *symmetric*: $k(x, x') = k(x', x)$

- *non-negative definite*: $\forall x, x' : k(x, x') \geq 0$

An interesting property of kernels is that they can be combined easily with symmetric operations [Bis06, p. 296], eg. by adding or multiplying the results of two kernel functions, k_1 and k_2:

$$k_{combined}(x, x') = k_1(x, x') \cdot k_2(x, x')$$

One can easily confirm that, when k_1 and k_2 are valid kernels, k_{combined} is also a valid kernel. This property enables composite kernels to capture multiple means of similarity in one metric.

As an example for a kernel, a simple kernel on two numbers (or vectors) is $k(x, x') = |x \cdot x'|$, where $|\cdot|$ is the absolute value (or a norm). Another example of a kernel between strings is $k(s, s') = \delta(s = s')$, where $\delta(\cdot)$ is the Kronecker delta. This kernel returns 1 if the two given strings are the same, and 0 if they are different. Note that both these kernels are indeed valid. That said, when using kernels in the real world, one can also use non-valid kernels, ie. kernels which are non-symmetric and/or return negative numbers. While such “kernels" are not strictly valid, they can nevertheless perform good in real-world applications.

While it is not immediately clear, how one might use kernels instead of a vector representations, we will see that kernels indeed can be utilized to enable operating on non-vector based data, such as graphs or strings. For this, we will introduce several so-called kernel methods, a class of algorithms that can be used for a variety of tasks by using kernels at their core. Prominent examples of kernel methods include the *SVM*, kernelized *Principal Component Analysis* [SSM98] and *Gaussian Processes* [Bis06, p. 303]. In Section 2.4.1, we further explain kernel methods.

For a subset of kernels, one can also create explicit feature vectors, or feature maps, for a given object. In this case, the kernel can be rewritten as an inner product on two fixed-size vectors:

$$k(X, X') = \langle \phi(X), \phi(X') \rangle$$

Here, $\phi(X)$ returns a fixed-size vector representation of the object X and is the main part of the kernel. This $\phi(X)$ can then be used in all vector-based classification algorithms, not only kernel methods. An example for such a kernel is the Bag-of-Words kernel on text. Here, the kernel counts the frequency of words in a given text, then creates a vector with these counts. The inner product of two of the vector representations of two texts then creates a kernel. Since word counts are always ≥ 0, the inner product on the vector presentations ϕ indeed results in a valid kernel.

For an overview over kernels, we recommend [Bis06, p. 293].

Gram matrix

A gram matrix [Bis06, p. 293], or kernel matrix, for a given kernel is a matrix A for a set of objects, $D = (x_1, x_2, \dots, x_n)$ where the entries are the pairwise similarities between all these objects in the dataset:

$$A_{i,j} = k(x_i, x_j)$$

So, the entry in the i-th row and j-th column signifies the similarity of x_i and x_j under the given kernel.

When the explicit feature map $\phi(x)$ for each x is given, one can also construct the gram matrix by vertically stacking these feature vectors into a matrix M_ϕ and calculating the product on the transpose

$$A = M_\phi M_\phi^T$$

The gram matrix is symmetric by definition since the underlying kernel is also symmetric, ie. $A_{i,j} = A_{j,i}$.

2.4.1 Kernel Methods and Kernel trick

We have seen that a kernel can be used to calculate a measure of similarity between two objects, eg. graphs. Also, some kernels can be expressed as an inner product $k(X,X') = \langle \phi(X), \phi(X') \rangle$ on two vector representations of the objects X and X'. In this case, the vector representations $\phi(X)$ can be used in conventional classification algorithms. Yet, calculating these representations is not always possible or feasible since the dimension of such a representation is quite high or even infinite, making the calculation infeasible. There is another approach for classification and other tasks, the aforementioned kernel methods. Kernel methods [Bis06, p. 291] operate directly with the kernel and do not use the explicit feature representations $\phi(X)$. This feature also enables kernel methods to operate on arbitrary objects, given that there exists a kernel operating on these objects.

One popular example for a kernel method is the Support Vector Machine, or SVM. The SVM is able to use both vector representations of elements in the dataset or the pairwise similarities, the gram matrix, between them. Some SVM implementations also allow to define the kernel directly, so the gram matrix does not have to be provided. At training time, the gram matrix A of all objects in the training set, $D_{train} = \{X_1, X_2, \ldots, X_n\}$, is calculated. This gram matrix A and the corresponding classes then get fed into the SVM which fits its coefficients to them. After training, to predict the label for a given instance X_{test}, the pairwise similarities of X_{test} and all training examples, $\{X_1, X_2, \ldots, X_n\}$, are calculated, resulting in a vector $v = (k(X_{test}, X_1), k(X_{test}, X_2), \ldots, k(X_{test}, X_n))$. The vector v contains the similarities of X_{test} to all training instances. This vector v is then fed into the SVM which predicts the label for X_{test}. Note that this kernel method approach does not use explicit feature maps of the instances, only their pairwise similarities. This feature of not needing to calculate the explicit feature map ϕ is called *kernel trick* [Bis06, p. 292]. The kernel trick also enables linear algorithms, such as the *SVM*, to solve non-linear problems, since the kernel can operate in higher-dimensional spaces. The kernel trick can be applied to any algorithm where the core of the algorithm uses vector representations to calculate an inner product. These inner products on the vectors can be replaced with kernels, resulting in a kernelized version of the algorithm [WC]. Apart from the kernelized *SVM*, the class of kernel methods also includes kernelized *Principal Components Analysis* [SSM98], kernelized *k-Means* [Jai10], *Gaussian Processes* [Bis06, p. 303] and others.

In our work, we will use both the kernelized version of the *SVM*, ie. learning with the gram matrix, as well as the version using explicit feature maps ϕ.

2.4.2 Graph Kernels

A graph kernel is a kernel that operates on graphs. Graph kernels that measure the similarity of two graphs are related to the isomorphism test [BM76, p. 4] between two graphs. The (structural) graph isomorphism test returns whether two graphs are structurally the same. In this test, the labels of the vertices and edges are ignored. Otherwise the task would be far easier since one could simply compare the labels and edges of the graphs - when they are both exactly the same, the graphs are the same [BM76, p. 4]. Instead, in the graph isomorphism test one has to find two mappings: one bijective mapping $\psi_V : V_1 \mapsto V_2$ from the vertices of the first graph to the vertices of the second graph and another bijective mapping $\psi_E : E_1 \mapsto E_2$ with the constraint:

$$\forall e = (v_s, v_t) \in E_1 : \exists e' = (v_s', v_t') \in E_2 : \psi_E(e) = e' \Rightarrow \psi_V(v_s) = v_s' \wedge \psi_V(v_t) = v_t'$$

If such a mapping exists, the two graphs are called isomorphic. While the complexity of the graph isomorphism test is not known yet, it is suspected to lie in P or it may also be NP-complete [HK09; KGK08].

One could use the graph isomorphism test, or a variant thereof, to create a graph kernel $k_{\text{isomorphism}}$ by returning 1 if the two graphs are isomorphic and 0 otherwise. Yet, as we noted before, the runtime of the graph isomorphism test is high and also, in the presented version, does not take edge or node labels into account. We will see other examples of graph kernels in the next paragraphs. Note that each graph kernel has different strengths and weaknesses, often being subject to a trade-off between accuracy versus compute time. So, when choosing a suitable kernel, the importance of each aspect of the graphs, eg. labels or structure, has to be assessed and the kernel constructed or chosen accordingly.

Graph Kernel Categories

Several graph kernels have been proposed in the literature. Categories of graph kernels include kernels based on shortest-paths [HJW15; BK05; Nik+17], random-walks [NB06] and subtree patterns [SB09; Dou11; Ker+13], to name just a few.

A great number of graph kernels have in common that they first create a decomposition of the graph into sub-structures, for example into subtrees, and then count the occurrences of these sub-structures in the graphs. The kernels of this kind are sometimes called R-Convolution graph kernels [Hau65]. In the next section, we introduce some examples of graph kernels to get a feeling for the possibilities and trade-offs different approaches have. As mentioned before, any set of valid kernels can be combined easily to form a composite kernel [Bis06, p. 296]. So, any of the algorithms we will introduce in the next section can be combined and extended using other kernels. As we will also see in the section, graph kernels can be quite diverse and capture different information of the objects they are operating on. The scope of a graph kernel can therefor also vary widely from only capturing narrow aspects of the graph to actually incorporating the complete graph structure and content.

For a more detailed overview on graph kernels, [Gär03] contains a survey of current graph kernel approaches.

2.4.3 Examples

Simple Equality Graph Kernel

This graph kernel simply compares the nodes and edges of the two graphs and returns whether they are exactly the same:

$$k(G_1, G_2) = \delta(V_{G_1} = V_{G_2}) \cdot \delta(E_{G_1} = E_{G_2})$$

This kernel takes both structure and content of the graphs into account. Note that there is no partial similarity. The co-domain of k is $\{0, 1\}$, returning 1 when the graphs are exactly the same and 0 otherwise. While this is an exact, simple and fast algorithm, in most cases it is not of great use since it depends on exact matches.

Simple Non-Structural Graph Kernel

Following is an example of a simple kernel which actually discards all structural information about the graph. This graph kernel only operates on the labels of the vertices:

$$k_{\text{common\_label}}(G_1, G_2) = |l(G_1) \cap l(G_2)|$$

where $l(G)$ returns the set of labels for graph G. Therefor, this graph kernel counts the number of common labels of the two graphs. Under this kernel, two graphs are the same when they have the same node labels, completely ignoring the edges. While this kernel only compares vertex labels, edge labels could quite easily be added to the kernel definition.

An extension to this graph kernel is using the Jaccard coefficient instead of simply counting the common node labels:

$$k_{\text{common\_label\_jaccard}}(G_1, G_2) = \frac{|l(G_1) \cap l(G_2)|}{|l(G_1) \cup l(G_2)|}$$

This extension normalizes the function and confines the codomain of k to $[0, 1]$.

Simple Structure-Only Graph Kernel

An example for a graph kernel which only operates on the structure of the graph, is the following:

$$k_{\text{simple\_structure}}(G_1, G_2) = \langle \phi(G_1), \phi(G_1) \rangle$$

with $\phi(G) = (n_{\text{triangle}}(G), n_{\text{rectangle}}(G))^T$. Here, n_{triangle} is the number of the triangle subgraphs in G and $n_{\text{rectangle}}$ the number of rectangle subgraphs in G. This graph kernel is an example of a kernel where the explicit feature map $\phi(G)$ can be computed directly and completely independent of other graphs. It is also an example for a R-convolution kernel.

The graphlet kernel [She+09] is quite similar to this example kernel $k_{\text{simple\_structure}}$: instead of only counting the number of triangle and rectangle subgraphs, the graphlet graph kernel counts so-called graphlets [She+09].

Random Walk Graph Kernel

This kernel does random walks on both graphs and returns the number of matching random walks between the two graphs:

$$k(G_1, G_2) = |r(G_1, n, l) \cap r(G_2, n, l)|$$

where $r(G, n, l)$ returns a set of n random walks of length l on graph G. This graph kernel takes both structure and content into account.

Weisfeiler-Lehman Graph Kernel

The Weisfeiler-Lehman graph kernel is also a kernel that works by counting common sub-structures on two graphs. Here, the sub-structures are subtrees starting from each vertex. In each iteration i with $0 < i < h$ for a given number of iterations h , or until convergence, the WL graph kernel relabels the vertices of the two given graphs. This relabeling is also called recoloring or color refinement [Ker+13].

The WL graph kernel relabels a vertex v by concatenating two components:

1. the current label of the vertex: $\pi_{i-1}(v)$
2. the sorted labels of the neighborhood: $\{\pi_{i-1}(v') | v' \in n(v)\}$

where π_i is the labeling function for iteration i with $\pi_0 = \pi$. The resulting new label of v is the sequence $(\pi_{i-1}(v), \pi_{i-1}(v_{neighbour,1}), \ldots, \pi_{i-1}(v_{neighbour,n}))$. After each iteration, a new labeling function is created accordingly: $\pi_i(v)$ now returns to the new label of v. So, for each iteration, a node gets the same label as another node if they have the same label and neighborhood. For the first iteration, the labels of a node consist of the immediate neighborhood. In the next iteration, the label of a node then encodes the neighborhood of the neighborhood, and so on. The Weisfeiler-Lehman graph kernel is said to have converged in iteration $h = i$ when the number of used labels does not increase in the next iteration, so

$$\max(\{x | x = \pi_i(v), v \in V\}) = \max(\{x | x = \pi_{i+1}(v), v \in V\})$$

The Weisfeiler-Lehman graph kernel has been shown to achieve state-of-the-art performance on several datasets and applications. It takes into account both content and structure, while also performing good in terms of compute-time.

The runtime complexity of the Weisfeiler-Lehman algorithm varies with different extensions. Reported complexities of the WL algorithm range from (1) $\mathcal{O}(hm)$ in [SB09] where h is the number of iterations and m stands for the number of edges in a graph and (2) the quasi-linear $\mathcal{O}((m+n)\log(n))$ in [Ker+13] where n stands for the number of edges.

It has to be noted that the Weisfeiler-Lehman does not use edge labels by default, but can be extended to do so. There are also several other extensions to the Weisfeiler-Lehman kernel in the literature. One example of such an extension reduces the space requirement of the algorithm by “compressing" the labels after each iterations, ie. by giving each composite label a unique and shorter label.

In Figure 2.6 we show an example of one WL iteration with label compression.

In this work we will use a variant of the WL graph kernel called *fast_wl* which was introduced in [Ker+13]. This variant improves the run-time of the plain Weisfeiler-Lehman kernel by cleverly creating a hash of the neighborhood to create the new coloring for the graph, resulting in the aforementioned $\mathcal{O}((m+n)\log(n))$ complexity. The results for this *fast_wl* variant are exactly the same as with the plain WL kernel, yet the calculation of the new labels is significantly faster. Our implementation also uses label compression, explained in Figure 2.6, to further speed up the calculation and assigning integer indices to the new labels, therefore enabling the creation of explicit feature maps $\phi(G)$.

2.5 Graph Kernel Based Text Classification

The idea of converting the text classification task into a graph classification task has been explored before. In this section, we will briefly recap the history of this approach by explaining several papers related to our work. While most of the graph classification approaches are used in contexts where graphs are the default representation of the data at hand, here we mostly explore papers which also derive graphs from existing text.

2.5.1 Related Work

"Shortest-Path Graph Kernels for Document Similarity" [Nik17]

The most similar and recent work to our approach that we found was [Nik17]. In this work, the authors first create co-occurrence graphs out of the text. Next, they create the gram matrix for the graphs with their own graph kernel and use a SVM to classify them.

Their graph kernel is a combination of two sub-kernels:

1. *Simple label matching*: the number of matching node labels are compared between the two graphs
2. *Shortest path*: generate a shortest-path graph, then compare the edges

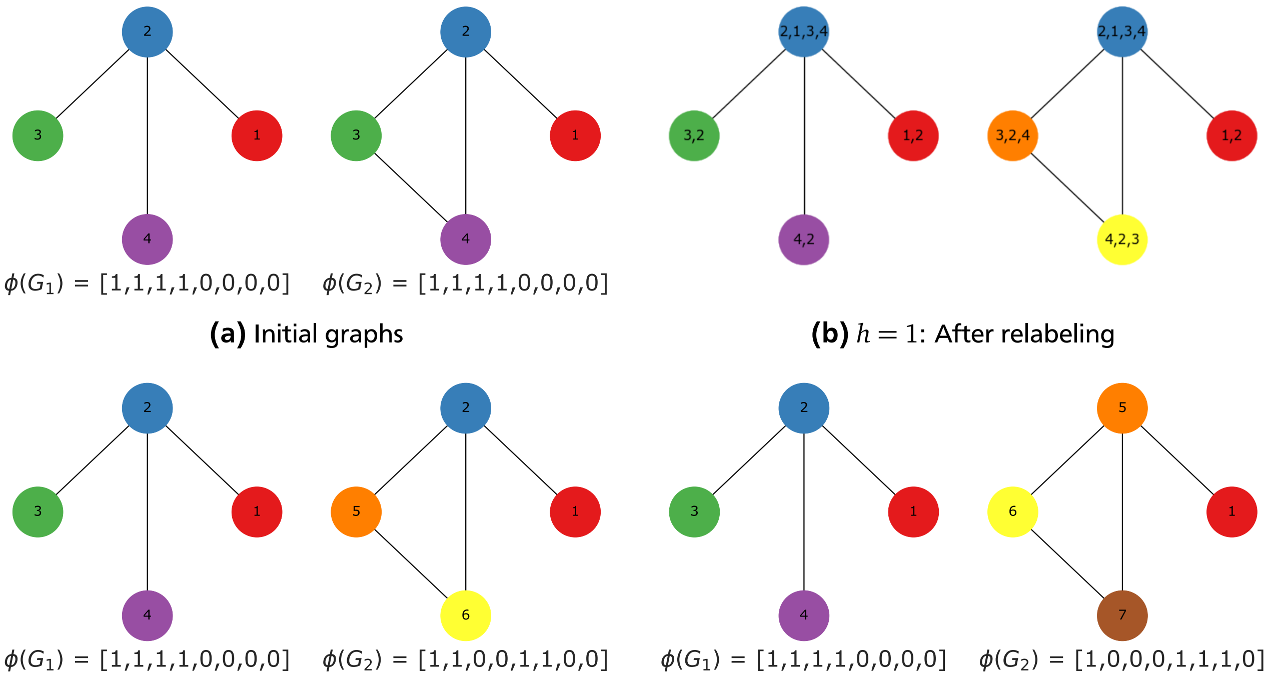
Note that the simple label matching sub-kernel does not take the structure into account, while the second sub-kernel actually uses both. Both sub-kernels return a number which is added to produce the final value of the kernel.

The shortest path sub-kernel works as follows:

1. Generate shortest-path graphs for the two given graphs: The shortest path graph of a given graph has the same nodes. The edges are added by calculating the shortest paths between all pairs of nodes in the original graph. An edge is added between two nodes in the shortest path graph if the shortest path length between these nodes in the original graph is below a given parameter d. The edge also gets a weight assigned, which is $\frac{1}{d}$
2. Count the number of same edges in the both graphs: when two edges are the same, add the product of their edge weights to the similarity

One thing to note about this work is that they used a combined kernel, yet they did not evaluate the performance or importance of each sub-kernel for the classification, ie. they did not report results for each sub-kernel. The simple-set-matching algorithm by itself has a high performance and most likely has the highest contribution to the classification results. We performed simple checks on the same datasets with only the simple-set-matching sub-kernel without their proposed shortest-path extension and got similar scores. A more thorougher analysis would be helpful to get a better insight.

That said, this approach is quite similar to ours. In their approach, they also convert the texts into graphs and then classify the graphs. One difference to our approach is, besides that we use different kernels and more datasets, that we evaluate the performance of concept maps instead of only co-occurrence graphs.


$\phi(G_1)$ = [1,1,1,1,0,0,0,0]
$\phi(G_2)$ = [1,1,1,1,0,0,0,0]
$\phi(G_1)$ = [1,1,1,1,0,0,0,0]
$\phi(G_2)$ = [1,1,0,0,1,1,0,0]
$\phi(G_1)$ = [1,1,1,1,0,0,0,0]
$\phi(G_2)$ = [1,0,0,0,1,1,1,0]

(a) Initial graphs **(b)** $h = 1$: After relabeling

(c) $h = 1$: After label compression **(d)** $h = 2$: After label compression

Figure 2.6: Weisfeiler Lehman algorithm example. This examples shows one WL iteration with two graphs, G_1 and G_2. The initial graphs are shown in **(a)**. The first step of a WL iteration is relabeling the graph vertices by concatenating each node label with its sorted neighborhood labels. The finished relabeled graph is shown in **(b)**. As an extension to the plain Weisfeiler-Lehman algorithm, we also show the label compression step where the long labels, consisting of the node label and its neighborhood labels, get compressed into a smaller label by assigning each unique label a new number. Same multi-label labels in different graphs get the same new label. The result of this step is shown in **(c)**. Under **(a)**, **(b)** and **(d)** we also added the feature maps $\phi(G)$ for both graphs. Each entry in the vector corresponds to a label. If a vector entry is 1, the corresponding label is present in the graph. The inner product on the vectors of the graphs is the value of the Weisfeiler-Lehman graph kernel for these two graphs. In this case, the similarity before the iteration would be $\langle \phi(G_1), \phi(G_2) \rangle = \langle [1,1,1,1,0,0,0,0], [1,1,1,1,0,0,0,0] \rangle = 4$. After the relabeling, the value of the inner product is 2. Before doing the WL iteration, the inner product on the feature maps simply returns the number of common labels. After the first iteration, the neighborhood of each node and therefor the structure of the graph is also reflected in the feature map - resulting in a lower similarity in out example. Notice that before the relabeling, ie. in **(a)**, the labels for the left-most node was the same for both graphs. After the first iterations, in **(b)** and **(c)**, they get different colors. The same happens for the top-most node: after the second iteration, in **(d)**, the labels are also different since in iteration $h = 2$ not only the immediate neighborhood has to be the same for a match, also the neighborhood of its neighborhood.

"Text Categorization as a Graph Classification Problem" [RKV15]

In this paper, the authors generate co-occurrence graphs out of different text classification datasets. To classify the graphs, they introduce their own graph kernel which is also an instance of the aforementioned kernels that count the number of occurrences of sub-structures. In this case, the sub-structures this kernel counts are frequent subgraphs, ie. subgraphs that occur frequently in the graphs of the dataset. To get a set of most-frequent subgraphs in the dataset, they develop their own approach, using depth-first search and a labeling function for graphs that enables finding frequent subgraphs efficiently. After extracting the sub-graphs from the dataset this way, they only retain subgraphs which occur more often in the dataset than some defined parameter, the support value. This significantly reduces the number of considered sub-graphs since most subgraphs occur quite infrequently. This extension is indeed needed since the number of possible sub-graphs grows super-linear with the number of nodes/edges and can render the subsequent counting of the sub-graphs in the dataset infeasible. Also, considering all possible subgraphs in all graphs in the dataset would not only require significant computational resources but would also result in sparse feature vectors since most sub-graphs occur infrequently. This, in turn, would harden the classifier training problem.

As another measure to lower the runtime complexity of their algorithm, they also first extract the main core of each co-occurrence graph and only extract sub-graphs from these, not from the whole graphs. A k-core is a subgraph G_k of graph G where all vertices have a degree greater or equal to k. The main core of a graph is the k-core with the highest possible k. Thus the main core is the most connected subgraph of graph G, which includes only the most “important" vertices and their edges to each other.

So, after extracting the main-core of each co-occurrence graph and determining the set of most-frequent sub-graphs in the dataset, they count the occurrences of the most-frequent sub-graphs for each graph in the dataset. This results in feature maps ϕ_i which contain the counts of each most-frequent sub-graph per co-occurrence graph. These feature vectors together with the labels are then used to train a SVM and are used to subsequently predict new instances by creating their feature maps and then predicting the label with this trained SVM.

The authors evaluated the classification performance of their approach on several datasets, namely the *WebKB* corpus, a subset of the *reuters-21578* corpus (named R8), the *ling-spam* dataset and a corpus of *Amazon reviews*. In our work, we also use these and other datasets. In later sections, we will introduce then more thoroughly.

In the paper, the authors argue that only retaining the main-core does not greatly damage the classification performance, yet it significantly reduces the compute time. They also stress the importance of choosing an adequate support value for the sub-graph mining, mentioning the trade-off between performance and considered features. The process of choosing an adequate support value can be done unsupervised and gets treated similar to hyperparameter tuning of, for example, SVM hyperparameters.

This paper is quite similar to our approach since it also poses the text classification task as a graph classification task. Yet, they consider only one type of graph, the co-occurrence graph. Also, they do not evaluate the effect of combining text- and graph features. That being said, they provide interesting comparisons of text- and graph based approaches, pointing out the similarity of n-grams with the relationships which are captured in co-occurrence graphs.

“Concept Graph Preserving Semantic Relationship for Biomedical Text Categorization" [GB15]

The paper also explores the usefulness of using automatically generated concept graphs for text classification. They first extract a concept graph for each of the articles in a medical dataset. Constructing the concept graph was done by extracting noun phrases with a *Part-of-Speech* tagger, short *POS*. Next, these extracted noun phrases are then looked up in the medical database called *UMLS*. When a noun phrase is found in this database, it becomes a node for the concept graph that is generated from the text. Other noun phrases are ignored. This additional filtering is done to find important concepts in a controlled vocabulary which can be further enriched by relationships which are also stored in the *UMLS* database. The relationships between concepts in the *UMLS* database are semantic, eg. whether two concepts are synonyms for each other. These relationships are then used as the edges between the concepts in the generated concept graph. In the next step, they gather node specific metrics, eg. the frequency of concepts in the underlying text, and sum them up to retrieve a weight w_v for each node v. Next, they obtain edge weights for each edge $e = (v, v')$ between two nodes v and v' by adding up the weights of both nodes, ie. $w_e = d_v + d_{v'}$. Then they remove edges which have a weight below some given threshold. After generating the concept graphs in this manner, the authors use a simple edge matching kernel to calculate the gram matrix for concept graphs. The kernel they use is similar to the graph kernel we introduced before, namely the *Simple Non-Structural Graph Kernel*: instead of counting the number of common nodes, as in our example, they instead match the edges. As a final step, they train a kernelized *SVM* and a kernelized *k-Nearest-Neighbor* classifier with the gram matrix for the concept graphs and the labels as input.

Unfortunately, the authors do not report their classification scores or provide the means to obtain the used dataset. Therefore an analysis of the claim that their approach shows statistically significant improvement over conventional, text-based approaches is hardly testable. That said, their approach is quite similar to our approach: they also use noun phrases extracted from a text to subsequently train a kernelized *SVM*. Yet, while they use semantic relationships obtained from a database as edges, we obtain the edges directly from the text. Also, we use different kernels to leverage multiple aspects of the structure and content of the concept maps. The authors, on the other hand, only evaluate one kernel capitalizing on the edges between concepts.

"Text classification using Semantic Information and Graph Kernels" [GGQ11]

In this paper, the text classification task is also converted into a graph classification task. The authors create a graph of the underlying text by extracting so-called Discourse Representation Structures, or DRS, which capture the semantic relationships in the sentences of the underlying text. A DRS consists of two parts, namely a set discourse referents, or entities, and conditions on these entities. The example given in the paper constructs the DRS for the sentence "He throws the ball", which results in the DRS with the entities $\{\text{He}, \text{ball}, \text{throws}\}$ which get renamed to $\{x1, x2, x3\}$. The conditions of the DRS then define the meaning or relationship between entities. For the mentioned example, the DRS conditions are $\{\text{male}(x1), \text{ball}(x2), \text{throws}(x3), \text{event}(x3), \text{agent}(x3, x1), \text{patient}(x3, x2)\}$. To construct a graph from the extracted DRSs, the entities become vertices and the conditions become edges between them. A unary condition on a vertex, for instance $\text{male}(x1)$, becomes a loop for the node $x1$. Conditions on two entities become directed edges, going from the entity of the first argument to the second condition argument, for example the condition $\text{agent}(x3, x1)$ creates a node from $x3$ to $x1$. This results in a graph which captures the semantic relationship between the entities of the underlying graph. Note that edge node labels, or entities, are not as important as the edges , or conditions, between them. This is accounted for in the kernel the authors use in the next step. Also note, that the process of creating a DRS graph for a given text is independent of other texts and can easily be parallized.

After constructing DRS graphs for all texts in the dataset, the authors use a variant of a random-walk kernel, customized to capture the importance of the edges instead of the nodes. In the plain random-walk kernel, random walks are done on the two graphs, G and G' and the counts of equal random walks are counted, resulting in a measure of similarity between these two graphs. The higher the count of equal random walks, the higher the similarity. The equality of two walks is determined by comparing the labels of the visited nodes. In the case of DRS graphs, this would not yield a good kernel since the labels of the nodes are placeholders, ie. two vertices with the same label, for example $x1$, can correspond to totally different entities. So, the standard definition of walk equality can not be used in the case of DRS graphs. Instead, the authors introduce an approach which, for a given node v_1 of graph G, returns an "equal" node v_2 in graph G' that has similar edge **labels**. This enables the random-walk graph kernel to "merge" nodes with different node labels, that is placeholders like $x1$, on the two graphs, thus enabling random walks with meaningful node equality in the case of the DRS graphs. After extracting the gram matrix of the DRS graphs with this extended random-walk graph kernel, the authors then use a SVM to evaluate their approach on the *reuters-21578* dataset. They use only a subset of 50 texts in the five most frequent classes of the dataset as their corpus. Then they create the DRS graphs for these texts and randomly select 25 of each class to create a gram matrix. This gram matrix then gets used to train a one-versus-one SVM classifier. So, they do binary classification and present the results of the performance of the pairs of the 5 classes they considered. The authors explain the reason for this small number of considered documents and classes by mentioning the runtime complexity of their approach, most notably the runtime complexity of the random-walk kernel. They also acknowledge that they do not provide baseline results or comparisons between different approaches to text-classification or results with other graph kernels.

That said, this work is also similar to our approach in the sense that they also use graphs representations of texts to classify.

“Deep Graph Kernels" [YV15]

In this paper, the authors introduce an extension for existing kernels for which the explicit feature map ϕ can be calculated. When calculating the kernel

$$k(G, G') = \phi^T(G) \cdot \phi(G')$$

they add a weighting matrix M

$$k(G, G') = \phi^T(G) \cdot M \cdot \phi(G').$$

The intuition here is that each vector entry of $\phi(G)$ corresponds to the count of some substructure of G and that these substructures are not independent of each other, ie. they can be similar to each other. Yet when calculating the plain kernel only exact matches of substructures in the two graphs are considered. For instance, suppose substructure g_1 is similar to another substructure g_2 and substructure g_1 is only present in G and not in G', conversely substructure g_2 is only present in G' and not in G. Suppose some kernel k would construct $\phi_k(G)$ by counting the occurrences of g_1 and g_2, so $\phi_k(G) = (n_{g_1}(G), n_{g_2}(G))$ where $n_{g_i}(G)$ counts the occurrence of substructure g_i in G. In the above example, $\phi_k(G_1) = (1, 0)$ and $\phi_k(G_2) = (0, 1)$. When calculating $k(G, G') = \phi_k^T(G) \cdot \phi_k(G')$ the similarity under the kernel would equal 0, therefore ignoring the similarity of substructure g_1 and g_2. The proposed extension to the kernel aims to address this issue by using embeddings for the substructures to identify the similarity of substructures and then allowing “partial" instead of only exact matches. In the above example, the new $\phi_{new}(G)$ would not be $(1, 0)$ but $(1, i)$ and $\phi_{new}(G') = (i, 1)$ with some $i > 0$. In our example, the aforementioned matrix M would be

$$M = \begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$$

In this case, when calculating the kernel for these two graphs, the similarity would be $(1, 0)^T \cdot M \cdot (0, 1) = (1, 0)^T \cdot (i, 1) = i$ instead of $(0, 1)^T \cdot (1, 0) = 0$. So, the augmented kernel between these graphs would actually capture the similarity of substructures instead of only using exact matches of them.

The authors then introduce two ways to create the matrix M for a kernel, namely by creating it by hand using a measure of similarity between the substructures or by learning the embeddings with an approach similar to the *word2vec* algorithm. In both cases, the matrix M aims to encode the similarities of the substructures.

The authors report an improvement of classification scores for several real-world datasets, comparing the scores of the plain version of a kernel with their augmented version.

2.5.2 Summary

As we see, several graph representations for text have been studied in the context of text classification. Yet, our approach differs in some crucial ways from the previous work, namely that (1) we capitalize on concept maps, a graph representation which is known for its summarization capabilities, resulting in smaller graphs. Also (2) we explicitly compare our results with conventional text-based approaches that achieve state-of-the-art performance. And while others mainly used graph kernels which are not able to be easily combined with text-based approaches, we (3) capitalize on graph kernels where the explicit feature map ϕ can be generated. This, in turn, enables us to explore an approach where we combine conventional text features, eg. *BoW*, with features obtained from these graph kernels.

3 Experimental Setup

In order to test our hypothesis, we first derive a number of questions. These questions will help us getting an insight into the usefulness of concept maps for the text classification task. We will also often compare the results of these questions to co-occurrence graphs to find differences. While the comparison to co-occurrence graphs is interesting by itself, it is not entirely fair since co-occurrence graphs have far more content than concept maps.

In the next chapter, Chapter 4, we will report the results to the experiments we devised for our questions. We will also provide a discussion of the results, bundling the results to a judgment for the hypothesis.

3.1 Experiments

In this section, we introduce our methodology and explain the experiments we conduct to test the hypothesis.

Following are the questions we devised in order to understand and test our hypothesis which is:

> *Concept maps capture structural information about the underlying text. Leveraging this structural information in the context of text classification leads to improved classification performance.*

We will provide a summary for each of the questions in the next chapter, revisiting the significance of the individual questions for our hypothesis.

3.1.1 Questions

Question 1: How useful are features obtained from concept maps combined with text features in the context of text classification?
This question aims to directly test our hypothesis. When concept maps indeed have a structure that is useful for classification and when this structure is not captured by text-only approaches, the classification performance could increase when combining our graph- and text based approaches. So, here we test the performance of combined text- and graph features by concatenating the vectors of both approaches and then training a classifier on these vectors. We also compare these results with both text-only and graph-only approaches.

Question 2: How diverse is the structure of concept maps?
When the structure of the graphs is homogeneous or the structural differences are not distinct between graphs of different classes, the structure by itself will most likely not contribute to the classification performance. When comparing the graph similarity under some graph kernel, the graphs of a given class should be distinguishable from the graphs of the other classes.

To understand the diversity in the structure of concept maps, we look at different metrics, eg. a histogram of the connected components per graph or the average node/edge ratio for each graph. All these metrics aim to find out the diversity of the connectedness of concept maps. There a lot of other metrics, but as we will see later on ,these metrics suffice to get a good picture about the structure of the concept maps.

Question 3: How important is the structure of concept maps compared to the content?

Another important question is the relative importance of the structure compared to the content, ie. the node and edge labels. For this, we compare the results of different graph kernels that use **(a)** only content and **(b)** only structure and **(c)** both content and structure.

For the **content-only kernel**, we will simply count the number of occurrences of labels in the graphs and essentially create a bag-of-words representation of the graph. The kernel then creates the similarity score by calculating the inner product on these vector representations, effectively counting the number of common labels.

For the **structure-only kernel**, we will use a modified version of the Weisfeiler-Lehman kernel: before executing the WL kernel on the graphs, we first discard all labels on the graph. All nodes of all graphs get the same label. This way, only the structure gets used for the similarity score.

For the **structure-and-content kernel**, we use the plain Weisfeiler-Lehman graph kernel.

The difference in the results will give an additional insight into the structure of concept maps and their usefulness for classification.

Question 4: How useful are multi-word labels in concept maps?

Concept map node labels often consist of more than one word which constitutes one of the differences to co-occurrence graphs and other graph text-representations. Yet, our default approach, ie. the Weisfeiler-Lehman graph kernel, does not leverage these multi-word labels. So, for this questions we evaluate other approaches which make use of the multi-word labels and evaluate the usefulness of them. This approach will later on also be used for the combined classification of text and graph features.

Question 5: How diverse and useful are edge labels in concept maps?

Another difference of concept maps to co-occurrence graphs is that the former have edge labels. For this question we evaluate the importance and role of these edge labels in graph classification. To make use of the edge label, we will also further introduce the linearized, or un-rolled, graph, ie. a graph converted into text.

Question 6: Does removing infrequent node labels from the concept map improve classification performance?

As with text, there are often a great number of words which only occur once in the whole dataset. The same applies to concept maps. In text-based classification approaches, infrequent words are often removed from the text for two reasons: (1) the resulting classifier has less parameters to be trained, thus lower complexity, (2) removing infrequent words greatly reduces the size of feature vectors which, in some cases, can prevent overfitting to too specific features. Another WL specific problem of infrequent words is that when a concept only occurs once and it is in the neighborhood of another node, matches in higher iterations of WL are impossible since the concept only occurs once and can not be in any other neighborhood apart from the only occurrence. For this question, we will evaluate whether removing infrequent node labels will result in better scores.

Question 7: Does merging infrequent concepts increase the classification score?

To overcome the issue of infrequent concepts in concept maps, we merge infrequent concepts with more frequent concepts.

Question 8: How does the performance of using the directed edges in concept maps compare to undirected edges?

Concept maps, in contrast to co-occurrence graphs, also have directed edges. To test the effect of using directed versus un-directed edges, we evaluate the classification performance for both cases.

Question 9: How does the size of concept maps relate to classification performance?

The size of a concept map, ie. the number of nodes and edges, is correlated to the size of its underlying text. Also, the number of occurrences of a single concept in a given text is important for the degree of the corresponding node, ie. when a concept occurs only once in a given text, the corresponding node in the concept map will have a low degree. On the other hand, when a concept occurs multiple times in a text, the degree of this concept will most likely increase. These observations give rise to the question whether the size of the concept maps is correlated with the classification performance. In this experiment we look at the classification scores of concept maps created from texts with varying length.

Question 10: How does the classification results of co-occurrence graphs compare to concept maps?

For this question, we compare the classification results for concept maps and co-occurrence graphs. As noted before, this comparison is slightly unfair since co-occurrence graphs have far more content than concept maps. Nevertheless, it is an interesting baseline for graph-based text-classification.

Question 11: How does the classification performance with concept maps compare to non-structural, text-based approaches?

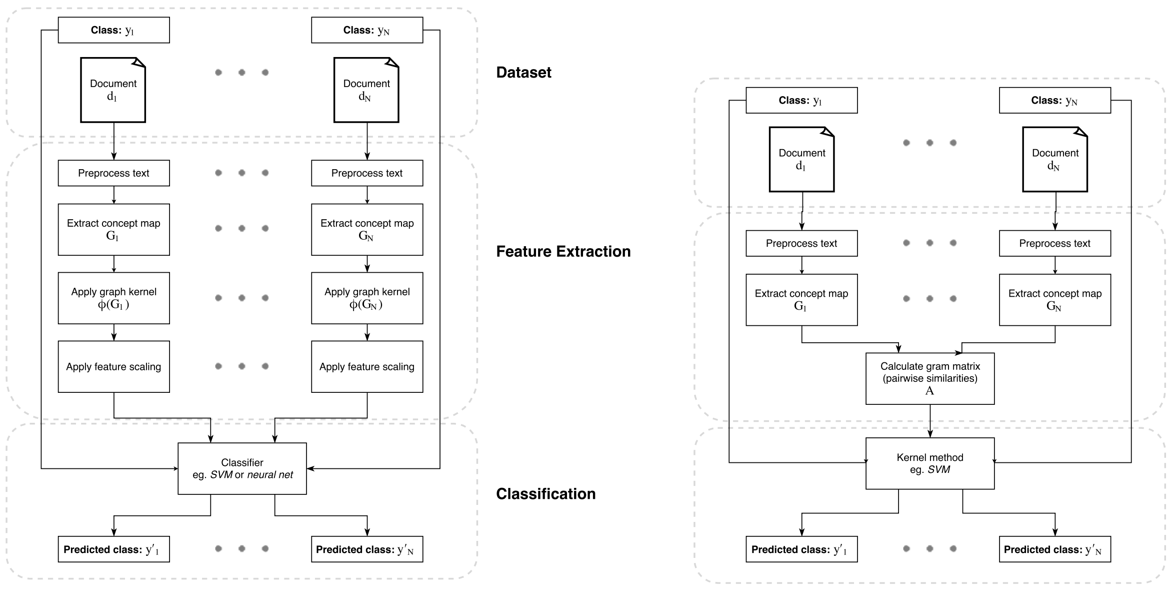
Here we test the classification performance of graph-based classification directly with the performance of text-only approaches. This experiment will serve as a baseline in later experiments.

Question 12: How does the compute time of graph-based- compare to text-based approaches?

Another important question concerns the complexity or run-time of our approach. While graph kernels like the Weisfeiler-Lehman or the Random-Walk kernels haves seen a rise in efficiency and improvement in runtime, there is still also an overhead associated when generating concept maps.

Summary

These questions and the associated experiments all aim to achieve a better insight into the usefulness of concept maps for the classification task. After conducting the experiments and gathering the results, we will further explain the importance of each individual question for our hypothesis.



(a) Feature map ϕ based **(b)** Gram matrix A based

Figure 3.1: Graph kernel based classification pipeline.

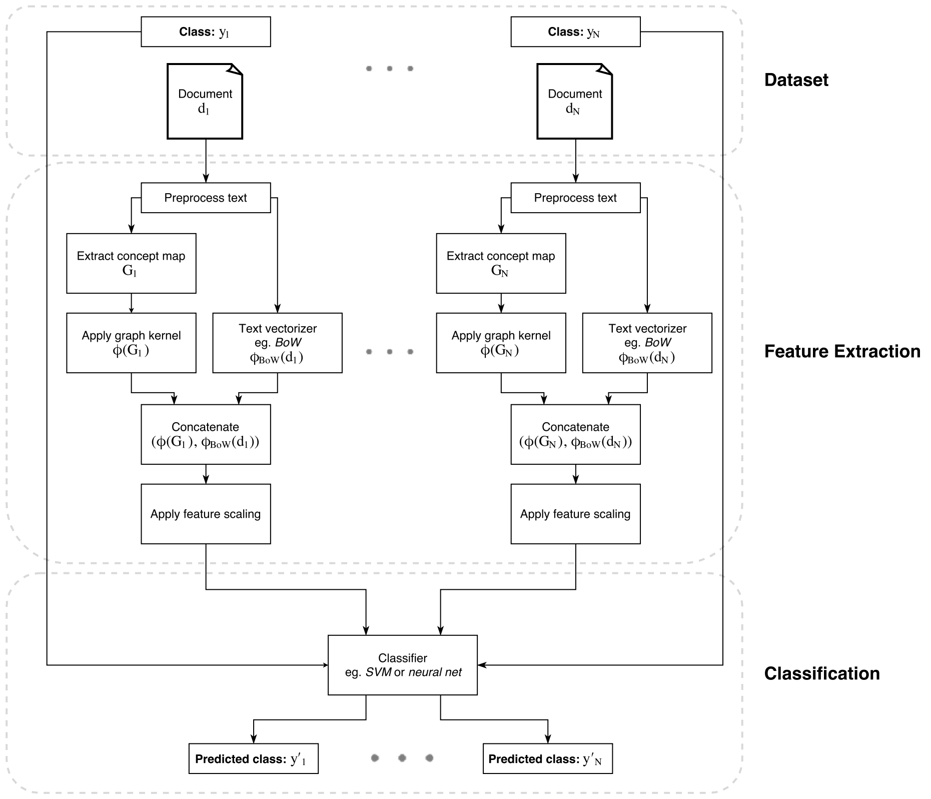

Class: y_1
Class: y_N
Document d_1
Document d_N
Dataset
Preprocess text
Extract concept map G_1
Extract concept map G_N
Apply graph kernel $\phi(G_1)$
Text vectorizer eg. BoW $\phi_{BoW}(d_1)$
Apply graph kernel $\phi(G_N)$
Text vectorizer eg. BoW $\phi_{BoW}(d_N)$
Feature Extraction
Concatenate $(\phi(G_1), \phi_{BoW}(d_1))$
Concatenate $(\phi(G_N), \phi_{BoW}(d_N))$
Apply feature scaling
Classifier eg. SVM or neural net
Classification
Predicted class: y'_1
Predicted class: y'_N

Figure 3.2: Combined graph and text classification pipeline.

3.1.2 Baselines

Preprocessing

Before creating the vector representations of the text documents or the creation of the co-occurrence graphs and concept maps, we first pre-process the plain text by

- lower-casing the text,
- removing non-printable characters,
- replacing numbers with *NUMBER* placeholders,
- replacing tabs and newlines with a space,
- and normalizing the whitespace (eg. replacing multiple spaces with a single space)

These pre-processing steps are similar to the pre-processing done in [Cac07]. An example of pre-processing can be seen in Figure 3.3.

For the co-occurrence graphs, we also optionally filtered out the non-nouns to increase the compression and thus achieve more comparability to concept maps.

Text-based representations

For the text classification pipeline we used two *Bag-Of-Words*, or *BoW*, based text vectorization algorithms, namely

- *Word Frequency* (Wf): this algorithm simply gathers all words in the corpus and creates a mapping between words and consecutive indices. Then it creates a vector representation for each text so that the i-th vector component is the count of the corresponding word in the text. Ie. i is the index of the word in the mapping.

> I've heard *unconfirmed* rumours that there is a new Integra being released for '94.
>
> Does anybody have any info on this?
>
> The local sales people know as much as I can throw them.
>
> –Parms.

(a) Before

> i've heard unconfirmed rumours that there is a new integra being released for ' NUMBER. does anybody have any info on this? the local sales people know as much as i can throw them. parms.

(b) After

Figure 3.3: Pre-processing example. Text taken from *ng20* dataset.

	F1 macro	
	$min_{wf} = 1$	$min_{wf} = 2$
ling-spam	0.983	0.986
ng20	0.788	0.784
nyt_200	0.890	0.878
r8	0.923	0.919
review_polarity	0.867	0.875
rotten_imdb	0.888	0.886
ted_talks	0.453	0.466

Table 3.1: Text classification results for BoW with (1) all words and (2) only words that occur more than once.

- *Term-Frequency-Inverse-Document-Frequency* (TfIdf): this approach is an extension to the *Word Frequency BoW* approach. Instead of using only the counts of a word in the text, this approach also incorporates the term frequency and the inverse document frequency of the words into the vector representation.

Both approaches can also be extended by not only utilizing single words, or unigrams, but n-grams. A word n-gram consists of n words that appear consecutively in the text. For example, the sentence “This is a sentence." has the following 2-grams, or bigrams: $\{(This, is), (is, a), (a, sentence)\}$. Note that word n-grams do not take word inversion into account, ie. the bi-gram (a, b) is not the same as (b, a). Also note that adding n-grams to BoW feature vectors increases their dimensionality which in turn could lead to bigger classifier models and possibly over-fitting to these more specific n-gram features.

For our purposes, we test only uni-gram and bi-gram frequencies since our classification performance seldom profited from adding higher n-gram ranges. Also, we filter out words which occur only once in the training set [Hea+17]. This has multiple advantages: (1) the dimension of resulting feature vectors is significantly lower, especially when also applying the same principle to bi-grams where there are far more combinations which occur infrequently; (2) this in turn reduces both the fit and transform time; (3) the size of models, eg. number of weights of the classifier, is far smaller; (4) overfitting to too specific features *might* be lessened. We also performed experiments to quantify the difference in performance when omitting infrequent words from the BoW approach and actually saw only insignificant differences. We report the results in Table 3.1.

Graph-based representations

To compare the performance of concept maps with other graphs, we generated co-occurrence graphs with window sizes $w \in \{1, 3\}$. We evaluated the performance of co-occurrence graphs where only nouns are retained to mimic the compression factor of concept maps.

For the extraction of the concept maps from the text, we used the implementation introduced in [FG17]. An explanation of the steps to create the concept maps is given in Section 3.4.

3.2 Datasets

We evaluate our approaches and experiments on a number of datasets, ranging from informal texts written by internet users, eg. the *ng20* internet forum corpus, to more structured texts like the *nyt* corpus. The texts of the corpora are also of varying length, enabling us to evaluate the effect of varying concept map sizes.

We provide download links for all the datasets except for the commercial *nyt* dataset. A script to download these datasets is also provided alongside our other code.

ling-spam

The Ling-Spam dataset was created and introduced in [And+00]. The corpus contains email texts which are categorized as “spam" and “no spam". One thing to note is that the classification scores with standard methods are quite high by default, so most likely no substantial increase in performance is to be expected.

We obtained a copy of the corpus from here [1].

ng20

The 20 Newsgroup corpus consists of posts from an internet forum and was introduced in [Lan]. Each post is labelled with one of 20 different classes, corresponding to the topic it have been posted on. The texts are mostly informal and consist of discussions between users of the forum. For this dataset, as an additional pre-processing step, we remove the headers and footers from the documents.

While the classes are nearly evenly distributed, some classes are highly correlated[2], ie. instances of one class A are very similar to the texts from another class B. This adds an additional difficulty to the task.

We obtained the ng20 corpus from here[3].

nyt_200

The documents in this dataset are articles published in the *New York Times* newspaper in the time between 1987 and 2007. In total it contains about 1,8 million articles, covering a great number of topics. Each article has different attributes, examples ranging from the publish date, the author or the section where the article was posted on the *New York Times* website. As labels we used the online sections an article has been posted to.

We found this dataset when searching for a corpus consisting of long documents. For our purposes we only used articles with more than 3000 words and from the six most frequent labels. Since the extraction of concept maps for documents of this document size takes a long time, we randomly selected 200 documents for each of the six labels, resulting in a dataset of 1200 articles. Besides the long texts, the other reason we chose this dataset is that it contains high-quality texts. The texts of most of other datasets we considered are gathered from posts by internet users and often lack basic punctuation or contain misspellings. This missing structure makes the extraction of concept maps harder since the

[1] `http://csmining.org/index.php/ling-spam-datasets.html`

[2] `http://qwone.com/~jason/20Newsgroups/`

[3] `http://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html#sklearn.datasets.fetch_20newsgroups`

concept map extraction relies on reliable part-of-speech tags which in turn profit from correct spelling and syntax.

We obtained our copy of the dataset from here[4].

reuters-21578

This dataset consists of news articles collected and published by Carnegie Group and Reuters. The class distribution of the *reuters-21578* dataset is highly skewed, ie. the number of instances per class is not the same for all classes. Some of the articles have multiple classes assigned. For our purposes, that is single-label classification, we only used articles with one class. We also only use documents which consist of more than 20 words, so that meaningful concept maps can be created. Since the *reuters-21578* dataset is quite big and also contains multi-label documents, ie. documents which have more than one class assigned, we do not actually use this dataset but use a subset which is described below.

We obtained the *reuters-21578* dataset from here[5].

r8

This dataset is a subset of the *reuters-21578* dataset. It consists of the 8 most frequent classes of the *reuters-21578* dataset, ie. the 8 classes with the most documents.

review_polarity

The *review_polarity v2* dataset consists of positive and negative reviews for movies by users. One thing to note is that these reviews are often quite short and informal.

The dataset was introduced in [PL04]. We obtained our copy from the author's website [6].

rotten_imdb

This dataset consists of sentences, each labeled with one of two classes: *subjective* or *objective*. Note that this dataset consists of short texts and can therefore be used to evaluate the performance of small concept maps.

The dataset w as introduced in [PL04]. We obtained our copy from the author's website[7].

ted_talks

This corpus consists of video transcripts from TED talks[8] released under the Creative Commons License[9]. We obtained the transcripts from Kaggle[10]. The transcripts had no labels attached, only an URL to the corresponding video. Each TED talk video has one or more tags attached, which we automatically crawled from the video URL. Since there are possible multiple tags per video, we had to filter out specific tags since our task entails multi-class-, not multi-label classification. To filter out the tags, we first looked at the most-frequent tags and create a correlation, or co-occurrence, matrix of these tags to find out how often these tags co-occur. We then selected a subset of four tags, namely $Y = (\text{economics}, \text{environment}, \text{brain}, \text{entertainment})$ which were loosely correlated. We discarded all transcripts with tags y which contain more than one of the tags Y, ie. $|Y \cap y| > 1$. All tags besides Y were ignored. The resulting datasets contains 682 documents, each consisting a large number of words per document.

[4] https://catalog.ldc.upenn.edu/LDC2008T19 (This is a commercial dataset and has to be bought.)
[5] http://www.nltk.org/book/ch02.html#reuters-corpus
[6] http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz
[7] http://www.cs.cornell.edu/people/pabo/movie-review-data/rotten_imdb.tar.gz
[8] https://www.ted.com/
[9] https://www.ted.com/about/our-organization/our-policies-terms/ted-talks-usage-policy#h2--ted-talks-under-creative-commons-license
[10] https://www.kaggle.com/rounakbanik/ted-talks

	# classes	# docs	# words	median #words/doc	#uniq. words/#words
ling-spam	2	2.893	1.303k	277	0.10
ng20	20	18.846	3.570k	79	0.06
nyt_200	6	1.200	4.735k	3397	0.07
r8	8	7.288	855k	82	0.06
review_polarity	2	2.000	1.248k	584	0.07
rotten_imdb	2	10.000	204k	19	0.12
ted_talks	4	682	1.353k	2027	0.09

Table 3.2: Dataset statistics. *# words* in thousands.

3.3 Methods

3.3.1 Cross-Validation and Model Selection

Cross-Validation

For all the classification tasks we use train-/validation- and test set splits. The split is done as follows: 80% for train- and validation set together and 20% for the test set. We then use stratified 4-fold cross-validation to further split the train- and validation set. The stratification ensures that the class distribution in the different sets is *almost* the same as in the class distribution of the complete dataset.

Model Selection

Hyperparameter tuning, or model selection, is done using cross-validation on the train-/ validation set, ie. we first train classifiers for all hyperparameter choices on all k train-/validation splits, and after finding the best hyper-parameters on these sets, we retrain the best classifier on the whole train-/validation set and then evaluate the performance on the test set **once**. So, the test set was **not** used for immediate evaluation or parameter tuning. The classification performance is only evaluated once on the test set to retrieve the final results we report here.

For our experiments, we use the *SVM*[CV95] as classifier unless stated otherwise. The parameters for the *SVM* are gathered in Table 3.3.

SVM parameter	Value
C	$\{0.01, 0.1, 1\}$
Regularization	L2
Kernel	linear
Optimization problem	dual
Stopping criteria tolerance	0.0005
Class weight	balanced

Table 3.3: *SVM* parameters. The *C* parameter is tuned using cross-validation. A more detailed explanation is available online[11] and in [CV95]

We choose the *SVM* as our classifier algorithm since it can be used for both classical, vector-based classification as well as for gram-based classification, making it an ideal candidate for our experiments. That said, the *SVM* also has shown state-of-the-art performance in (text) classification and other domains. In contrast to other learning algorithms, eg. a neural net, SVMs are also more robust in their convergence [Joa98].

To increase the reproducibility of our results, we utilized the same random generator seed, 42, for all experiments, eg. for “randomly" splitting the dataset in the train/validation/test splits or “randomly" initializing the coefficients for a *SVM*.

We also evaluated results obtained with nested cross-validation to further de-bias our results [VS06]. Yet, when inspecting the results from tests with nested cross-validation we saw that the standard deviations of the results are relatively low. This observation and the fact that nested cross-validation also increases the compute-time significantly, led us to abandon nested cross-validation for our experiments. However, to be able comparing the results of two different models, we consistently used the permutation significance test.

3.3.2 Metrics

As a classification metric, we mostly focus on the F1 macro score since it captures the overall performance of classification algorithms by merging two other metrics, namely precision and recall. For an overview and definition of the used metrics, please see Section 2.3. We will provide all our results online[12] with additional metrics, like accuracy, precision and recall. We do not report standard deviations for our experiments since we, as mentioned before, do not use nested cross-validation, so we only have results on a single static test set. See the previous Section 3.3.1 for a more throughout explanation.

To analyze the significance of a difference between two models, we use significance tests. An introduction to and an example for significance tests are given in Section 3.3.4.

3.3.3 Feature Scaling and Centering

Classifiers, especially *SVMs*, profit from scaled and centered features [GB01], ie. features which have a mean of 0 and are in the range of $[-1, 1]$.

Scaling

For our purposes, we scaled the feature vectors by using feature-wise scaling[13] [Bis06, p. 567]. Given the feature vectors $\phi(d_i) = (f_1, f_2, \dots, f_n)$ for each of the N dataset instances d_i, we first find the maximum for each vector entry f_x across the whole dataset. For instance, the maximum of first feature vector entry, f_1, is

$$f_1^{\max} = \max(\{\phi(d_i)_1 | i = \{1, 2, \dots, N\}\})$$

where $\phi(d_i)_1$ stands for the first entry of $\phi(d_i)$. After finding the maximum for each feature f_x, we then scale this feature by the absolute value of the maximum, eg. $|f_x^{\max}|$. The new, scaled feature vector is thus

$$\phi_{\text{scaled}}(d_i) = (f_1/|f_1^{\max}|, f_2/|f_2^{\max}|, \dots, f_n/|f_n^{\max}|)$$

For WL, we also tried scaling the feature vectors to unit length. Yet we saw the same or even worse results than without unit-length scaling, so we did not apply it. Unit-length scaling is applied to single feature vectors, ϕ, so that its length $||\phi||_{norm} = 1$ under some norm, eg. L1 or L2.

[12] See `https://github.com/davidgengenbach/bachelor-thesis`

[13] Implementation: `http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler`

Centering

Centering a vector means subtracting the mean of each feature entry [Bis06, p. 567], therefor effectively centering the mean of the feature vector to 0, ie. the average of each feature vector index is 0. Since we use only highly sparse and high-dimensional vectors, centering the feature vectors is not feasible since it would destroy the sparsity which in turn would make the subsequent computation too memory consuming if not impossible. This is due to the fact that centering a vector often means adding or subtracting another vector from it which creates non-zero entries for all entries. Only when the vector is already centered, the centering would not affect the sparsity.

Fortunately, since we have very sparse features, the mean for each feature is very close to zero. This is due to the fact that most features occur very infrequently. Together with the high number of elements in the dataset, the mean of each feature will be close to zero since the average, $\hat{f}_x$, for each feature f_x is calculated by taking the sum of the feature for all elements and dividing it by the number of elements in the dataset

$$\hat{f}_x = \frac{\sum_{d_n}(\phi(d_n)_x)}{N}$$

In our cases, $N > \sum_{d_n}(\phi(d_n)_x)$ is true for most of the features since most features occur quite infrequently and therefor the sum is also low.

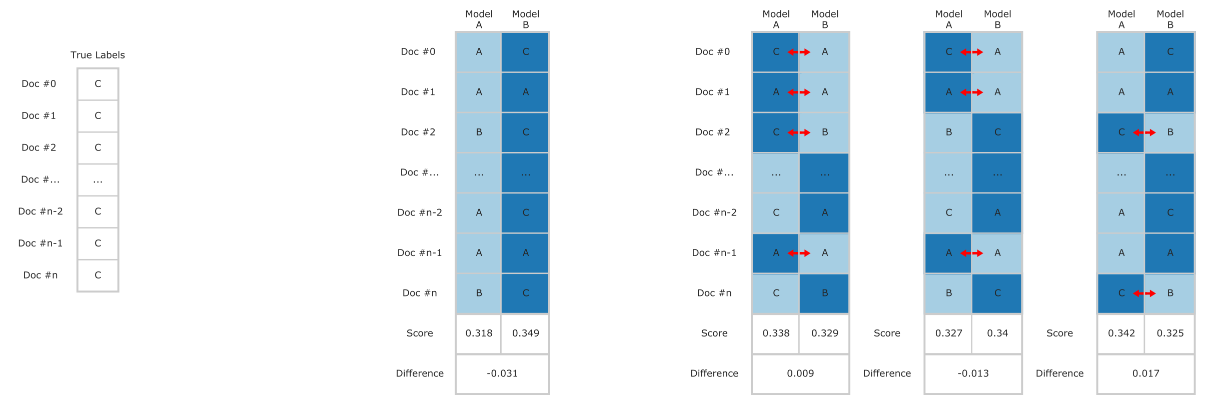
3.3.4 Significance tests

When comparing two models we use the permutation test [Fis25], or exact test, to test the significance of the difference in performance of these two models. The permutation test tests whether the observed difference of the performances of the two approaches is a result of chance or really signifies a more fundamental difference without assuming an underlying distribution of the differences that can be observed. The test only returns a probability for observing a given performance difference. The test does not give a definite answer whether the difference was due to chance. When the probability of observing the difference by chance is below a given threshold, the confidence p, we say that the test indicates that the approaches significantly differ in their performance. For our purposes, ie. to test whether one classification approach is a significant improvement over another approach, we use the two-tailed version [Koc15] of the permutation test. We also provide the p value for our tests. We say that a model is significantly better when the observed performance difference has a frequency of $p < 0.05$ using the permutation test with a sample size of 10000.

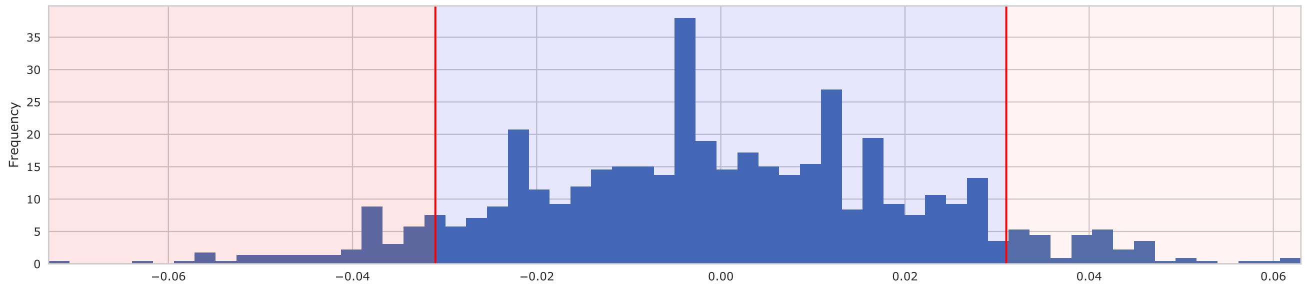
Example

In this example we will use the two-tailed permutation test to test whether the performances of two given classifiers, Model A and B, differ in a non-random way. In Figure 3.4 we depicted the steps in the permutation test. In our example, the hypothesis is that Model A has a lower score than Model B. For this example, we choose accuracy as the score. To calculate the accuracy, we need the true labels of the dataset, seen in Figure 3.4 . In Figure 3.4 **a** we see the predicted labels of the two models and the accuracy as a score beneath them. We also see the difference in the scores. In Figure 3.4 **a** we also see that Model A has a lower score than Model B. This lower score could be due to chance and not because Model B is fundamentally better than Model A. To test our hypothesis more throughly we now execute the permutation test. In the first step, Figure 3.4 **b**, we generate n samples by interchanging the predictions of the two models randomly. Here, we depicted three randomly selected samples from the n generated samples. To generate a sample, we switch the predictions of Model A and B for every document with a probability of 0.5. The switch of two predictions is marked with a red arrow in our depiction. In the optimal case, one could generate all possible permutations of the predictions, ie. generate all possible samples. Unfortunately this often is not an option due to the sheer number of possibilities. That said, generating a large number of samples also gives a good basis for judgment.

In the next step, we calculate the accuracy scores for both models for each of the n samples and also calculate the difference between these scores. In Figure 3.4 **c** we see the histogram of score differences in the samples. The red lines mark the initially observed difference between the models A and B (also it's negative). In the last step, we gather these differences and count the number of samples, n_{higher}, where the absolute difference between the models is smaller than the score difference of the original predictions of Model A and B. n_{higher} divided by the total number of generated samples, n, is then the frequency of samples, $f_{higher} = \frac{n_{higher}}{n}$, where the score difference was higher when randomly exchanging the predictions compared to the original observed score difference. f_{higher} gives an intuition about the likelihood of observing the difference between the scores of Model A and B. In the histogram of observed differences Figure 3.4 **c** this f_{higher} corresponds to the ratio between the area of elements in the blue surface to total area of the histogram. In our example, f_{higher} is 0.135, meaning that the probability to observe the score difference we see between the two models is 13.5% when randomly exchanging the predictions of the two models. This is far too high to accept the hypothesis on ground of these predictions.



(a) True labels **(b)** Predictions **(c)** Samples



(d) Observed differences in sample scores

Figure 3.4: Permutation test example. A significance test is most often used to test whether an observed difference is a product of chance or indeed is significant. Significance tests only provide a probability, not a definite answer.

3.4 Implementation

We mostly used Python[14] to implement the code required to run the experiments. The code and instructions on how set-up the experiments can be found on GitHub [15]. Our work heavily relies on open source software and in this form would not be possible without it. In particular, we implemented most

[14] https://www.python.org/
[15] https://github.com/davidgengenbach/bachelor-thesis

of the experiments using *scikit-learn* [Ped+12], *networkx* [HSS08] and *pandas* [McK10], *numpy/scipy* [WCV11], *SymPy* [Meu+17] and several other open-source projects.

In the core of our implementation are *sci-kit learn* pipelines, classifiers, eg. *SVMs*, and transformers, eg. *TfidfVectorizer* or a *StandardScaler*. Pipelines contain a number of transformers and can also consist of pipelines themselves. Each step in the pipeline retrieves the output of the previous step as input. The first transformer in a pipeline gets the user-provided input, eg. texts or graphs, as input. At the end of a pipeline, in the last step, is a classifier which also retrieves the output from the step before, that is vector representations of the user-provided input. We both use out-of-the-box transformers provided by the *scikit-learn*, eg. the *TfidfVectorizer* and the *SVM*, and implement our own transformers, eg. our implementation of the Weisfeiler Lehman graph kernel.

Each step in a pipeline can be parametrized, eg. the C parameter of the SVM classifier can be provided. This in turn enables easy-to-implement parameter grid search which in turn can be easily parallelized. We augmented this existing framework provided by *scikit-learn* with several additions. One of them providing the ability to define experiments, or test runs, with experiment definitions. Here, we can provide all pipeline parameters, eg. the Weisfeiler-Lehman iterations h or the C for the SVM, in a single file and also define the used transformers in a pipeline. Also, meta-parameters, for instance the number of cross-validation folds or the wanted classifier metrics should be evaluated on the test set, can be specified in these experiment definitions. We can also do model-selection with these experiment definitions by specifying multiple parameter combinations that should be evaluated with cross-validation. After finding the best parameter combination, these parameters are then evaluated once on a test set and the predictions results also saved to disk. This enables us to quickly explore new parameter combinations and clearly define our experiments in a single file.

All experiment definitions with all parameters are provided online[16] alongside the code.

Concept Map Extraction

We used the code provided by Falke[17] to create concept maps for our datasets. In this section we will briefly describe the steps the code performs. For a more detailed explanation, see [**Falke2017**].

The input to the concept map extraction algorithm is a single text document. The output is a single concept map for this text document.

Extraction: First, the algorithm extracts concepts and their relations to another. This is done by extracting binary relations from the text. A binary relation consists of three parts: two concepts and the relation between them. An example for a binary relation is “David likes something", here “David" and “something“ are the concepts and “likes" the relation them. Note that the binary relation is directed, eg. “Something likes David" is not the same binary relation as “David likes something." Also, a concept can consist of multiple words, eg. for “David likes something else.", the concepts would be “David" and “something else". The extracted concepts are later used as node labels, and the relation between them as the edges between concepts.

Filtering: Next, the extracted relations get filtered so that only concepts are kept which contain at least one noun and which consist of fewer words than some a given threshold, in our case 10 words. This filtering is done to ensure the brevity and practical usefulness of the concepts when visualizing the graphs.

Grouping: In the next step, related concepts are merged to reduce the redundancy of node labels. Here, the idea is to group similar concepts together. Often concepts are referenced by a synonym or implicitly mentioned through co-references, eg. by using “this" without explicitly specifying what is referenced. To find concepts which should be merged, the authors propose a solution based on pairwise classification. Here, for each two given concept mentions, a feature vector $\phi(x, x')$ is calculated which is subsequently used to train a one-layer neural net to classify whether two given concept mentions refer

[16] `https://github.com/davidgengenbach/bachelor-thesis/tree/master/code/configs/experiments`
[17] `https://github.com/UKPLab/ijcnlp2017-cmaps`

to the same concept. The feature vector ϕ for two concept mentions are created using several metrics, eg. the cosine similarity of word embeddings and the normalized Levenshtein distance.

After these steps, we can create and export the concept maps.

Summarization: The original implementation also summarizes the resulting graphs by finding the most relevant sub-graph of the concept map. This is done to further summarize the graph and remove un-connected components from the graph, eg. connected components consisting of only two nodes. We skipped this step in our concept map creation since this additional filtering would result in much smaller graphs which would in turn harden our classification task further.

4 Results and Discussion

Answer to Question 1: How useful are features obtained from concept maps combined with text features in the context of text classification?

In this section we investigate how and whether the graph-based approaches improve the classification score when combined with text-based approaches. This is arguably the most interesting question since it will show the usefulness of concept maps in text classification.

For the text features we vectorized the text with Bag-Of-Word approaches, namely a simple term frequency count and augmented by Tfidf. Here, we evaluated both uni-grams and bi-grams by cross-validation, then we chose the best performing variant to evaluate on the test set. For the graph features we used the Weisfeiler-Lehman algorithm since it is able to create feature map $\phi(G)$ for the graphs. These feature maps can easily be concatenated with the text features. For other graph kernels where an explicit feature map $\phi(G)$ can not be calculated, in order to combine graph- and text features, we would have needed to create a combined kernel that takes both text- and graph features into account and calculate a gram matrix. The interpretability of this approach would be far lower since both graph- and text-features would be merged together in a single number which inhibits to distinguish the relative importance of the text- and graph features after creating the gram matrix. In Table 4.1 we see the classification scores for the combined features for both co-occurrence graphs and concept maps. We also report the scores of different Weisfeiler-Lehman extensions and graph pre-processing steps, eg. splitting multi-word labels. The details of these approaches will we introduced in the answers for the next questions.

In the results we can see that the classification performances of the text-only and and combined approaches are comparable. On some datasets, the combined approach is slightly better than the text-only approach. Yet, under the permutation test, these differences are not significant. Interestingly, the co-occurrence graphs with their relatively simple structure often perform the best when combined with text features. However, on other datasets, their performance is not as good as the scores of text-only- or combined with concept maps approaches. Another interesting observation concerns the different WL extensions and graph-preprocessing steps we devised for concept maps. Here, we see only minor improvement or even lower performance than with the combined approach. When doing graph-only classification, on the other hand, we see high improvements over the plain version, ie. plain WL and no graph pre-processing. For instance, one approach, which linearizes the graph into text and subsequently uses uni-gram Tfdif Bag-Of-Words to vectorize the text, consistently results in lower classification performance when combined with the conventional text approach. On the other hand, when doing graph-only classification, linearizing the graph into text actually consistently resulted in the best scores. So, while ignoring the structure in graph-only classification has shown the best performance, on the combined approach, it performed worse than other extensions or even plain WL. This observation can also be seen with other extensions, yet not as distinguished. All these observations highlight the importance of carefully selecting the used graph kernels. Using different variants of the Weisfeiler-Lehman graph kernel, for example, resulted in greatly varying performance on different datasets. Also, the performances on graph-only classification differ from the performance when combining text- and graph features.

To better understand the classification performance when combining text- and graph features, we also train a one-layer neural network[1] with the combined features using stochastic gradient descent. After training we investigate the weights. The weights, or coefficients, for each input feature are an indicator for the importance of that input feature. Thus we can look at weights for the text features and compare

[1] Implementation: `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html`

them to weights for graph features. This gives us an insight into the relative importance of text- and graph features. In Figure 4.1 we show an example of an one-layer neural net. For our purposes, we do not look at single features, or input neurons, but on a range of features, namely the text and graph feature ranges since, in our case, the text- and graph features are concatenated.

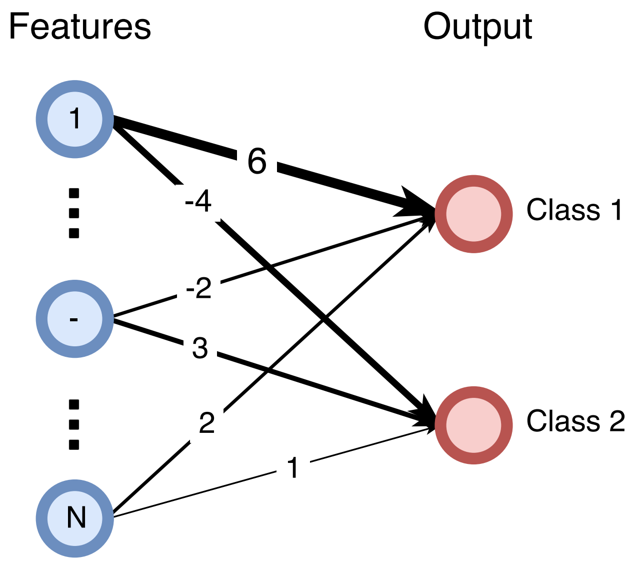


Figure 4.1: Example of an one-layer neural net. The numbers on the line from input features to the output are weights. We examine these weights for single features to gain an insight into the importance the neural net assigned to a given feature. Since a feature contributes to all output neurons, we look at the sum of its absolute weights. Eg. for *Feature 1* the sum of the absolute weights would be $|6| + |-4| = 10$ and for *Feature N* it is $|2| + |1| = 3$, which gives us the hint that *Feature 1* might be more important for the classification result than *Feature N*. This analysis presupposes that the feature magnitudes, ie. the values of features, are normalized or - roughly speaking - in the same range on average, eg. *Feature 1* does not take values between $[100, 200]$ while *Feature N* takes values from $[0, 1]$, instead both feature values are roughly in the same range.

Figure 4.2 then shows an example of a weight analysis for the *ng20* dataset and for two regularizations, namely *L1* and *L2*.

Both regularization terms are used to discourage “big" weights, effectively forcing the training algorithm to choose important features to generalize the data instead of over-fitting the data. While both *L1* and *L2* regularization discourage big weights, *L1* penalizes small weights harder than *L2*, leading to more zero coefficients [HTF09, p. 13]. Therefore, the selection of important features is enforced more with *L1* regularization. We leverage this phenomenon to analyze how important the features, graph and text, are for classification. When looking at the absolute sum of the weights of the trained one-layer neural net, we discover that using *L1* regularization results in the text features becoming more important than graph features for classification. While the difference of absolute sums of the coefficients does not seem especially high, it has to be noted that graph features are approximately only half as dense as text features. Thus the contribution by graph features to the end result is even lower than the neural net coefficients indicate.

Another interesting result of this weight analysis is the importance the neural net assigns to different Weisfeiler-Lehman iterations. In our example, we chose $h = 5$ for our WL iterations. The higher the iteration, the bigger the neighborhood that is considered by WL. When looking at the importance of the first iteration and *L1* regularization, *Graph 1* in Figure 4.2, we see that while the neural net nearly discourages all higher WL iterations by assigning low weights, it assigns higher importance to the first iteration of WL. The first iteration of WL takes only the immediate neighborhood into account, therefore

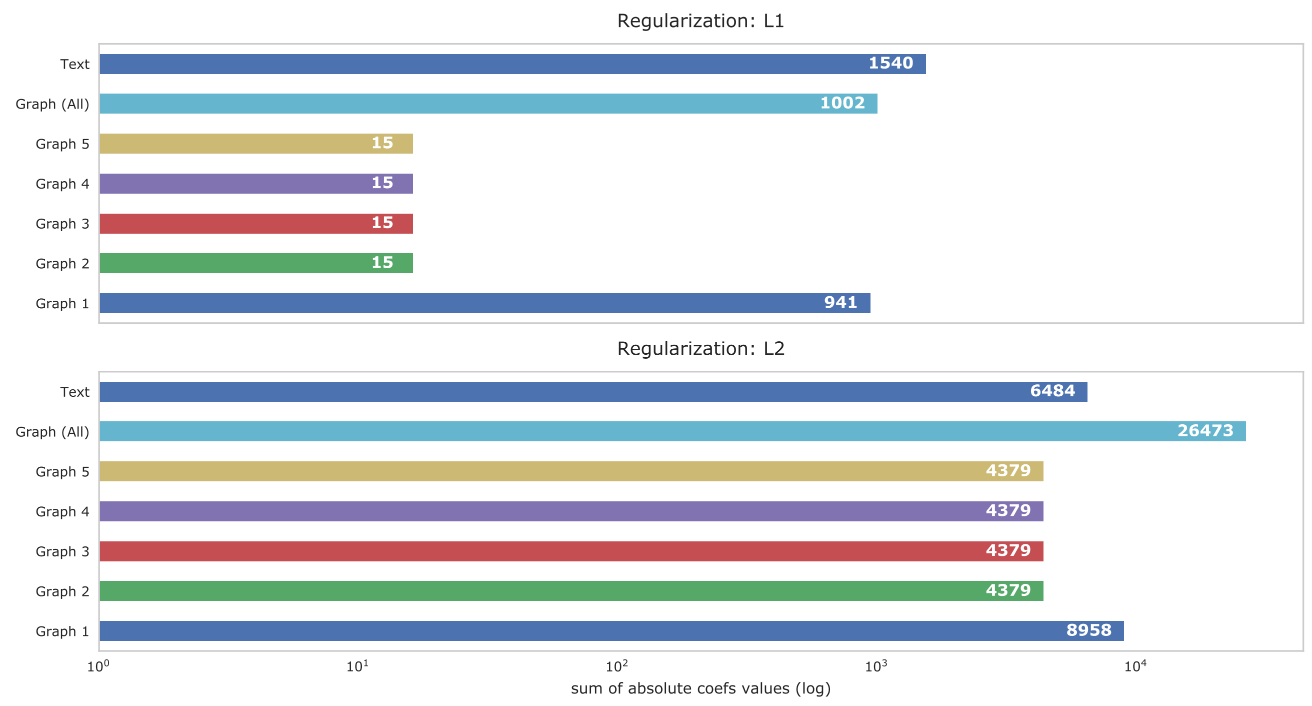


Figure 4.2: Histogram of the trained one-layer neural net weights. The neural net was trained on the concatenated graph- and text features. Higher values indicate higher importance. *Graph (All)* stands for the sum of all graph feature weights, while the *Graph N* stand for the individual Weisfeiler-Lehman iterations. Higher iterations of WL take a bigger neighborhood into account. Dataset: *ng20*.

matches are far more probable than matches in higher iterations. In our example in Figure 4.2, *L1*-outperformed *L2* regularization by a high margin, approximately 7% difference in the *F1* macro score. However, it has to be noted that no hyper-parameter tuning was performed and only a simple train-/test split was used.

Answer Summary: When combining the graph- and text features we do not achieve significantly higher classification scores and even observe a degradation in performance on some datasets. Examining the weights of one-layer neural net shows us that, when training on combined text- and graph features, graph features get lower absolute weights assigned. A possible explanation is that the graph features are not as important for the combined text- and graph feature classification.

Answer to Question 2: How diverse is the structure of concept maps?

To investigate whether concept maps are useful for text classification, we first look at the structure of concept maps. Our hypothesis especially highlights the structure of the concept maps since it is one of the main differences to other text representations, eg. *BoW*. Finding out whether the structure of concept maps is heterogeneous therefore is a first crucial step.

In Table 4.2, we gathered metrics about the connectedness of concept maps, also providing the metrics for co-occurrence graphs with window size 1 for comparison. One thing that immediately becomes apparent is the compression factor of the concept maps: on average, only approximately 7% of the original text-content is retained in the concept maps. This compression is useful for tasks like text summarization where only important concepts should be kept from the underlying text. As a downside,

		F1 macro					
		Concept Map				Co-Occurrence	Text
		WL	Split + WL	To-Text	Infreq. + WL	WL	
ling-spam	Single	0.816	0.913	0.934	0.803	0.987	**0.990**
	Combined	**0.990**	0.986	0.980	**0.990**	0.984	
ng20	Single	0.419	0.574	0.622	0.357	0.593	**0.781**
	Combined	**0.791**	0.737	0.756	0.781	0.672	
nyt_200	Single	0.744	0.846	0.899	0.774	0.881	**0.921**
	Combined	**0.917**	0.891	0.904	0.888	0.875	
r8	Single	0.677	0.860	0.892	0.633	0.890	**0.921**
	Combined	0.921	0.929	**0.931**	0.929	0.914	
review_polarity	Single	0.609	0.715	0.737	0.607	0.785	**0.877**
	Combined	0.870	0.837	0.850	**0.880**	0.855	
rotten_imdb	Single	0.635	0.780	0.828	0.551	0.825	**0.886**
	Combined	0.885	0.877	0.883	0.890	**0.910**	
ted_talks	Single	0.244	0.413	0.359	0.364	0.443	**0.464**
	Combined	**0.481**	0.433	0.436	0.468	0.442	

Table 4.1: Classification results for combined graph- and text features classification. *Split* stands for the WL extension where the multi-word labels are split, see answer to Question 4. *To-Text* converted the graph back to text, then a classifical Tfidf-BoW approach was used, see answer to Question 3. *Infreq.* is the WL extension where infrequent nodes are ignored, see answer to Question 7. No significant improvement of the combined approach was found upon the text-only features, under the permutation test.

one consequence of the compression can be seen when looking at the connectedness of the concept maps. On average, concept maps only have 0.69 edges per node. The minimum number of edges per node is 0.5 since are no unconnected nodes, ie. nodes without an in- or out going edge, so there has to be at least one edge between two nodes, thus 0.5.

So, concept maps do not only have few nodes, because of the compression, but also a relatively low number of edges between them. This is most likely due to the small length of the underlying texts in most datasets. The shortness of the text results in a lower number of occurrences of some concept in the text, ie. most of the concepts occur only once in a text. The *nyt* dataset on the other hand contains text of much higher length which also is reflected in the number of nodes per graph, see Table 4.2. Nevertheless, we also see the un-connectedness in the *nyt* dataset, even if the concept maps are relatively speaking more connected than the concept maps in other datasets. Another reason for the un-connected concept maps, apart from the shortness of the underlying text, is that we do not use the summarizing step in the concept map creation: to create more connected concept maps, in the code by [FG17], only the main core of the graph is retained, ie. the most-connected subgraph. We do not use this additional subgraph extraction step as this would further decrease the number of nodes, thus increasing the compression factor even more.

Another hint for the un-connectedness of concept maps is the number of connected components. In Table 4.3 we can see that most of the graphs have more than one connected component. Together with the observation of the low number of nodes per graph, this also implies the low connectedness of concept maps. In this table we also report the percentage of connected components of size $\{2, 3, 4\}$ to the total number of connected components. The results in this table show that the bulk of concept maps consist

	#nodes/#words		#edges/#nodes		#nodes/graph	
	cmap	coo	cmap	coo	cmap	coo
ling-spam	0.05	0.44	0.67	1.80	22.57	198.89
ng20	0.06	0.49	0.68	1.66	10.08	89.95
nyt_200	0.06	0.30	0.83	2.43	246.19	1191.03
r8	0.09	0.52	0.72	1.55	10.52	59.29
review_polarity	0.07	0.51	0.66	1.77	42.91	321.32
rotten_imdb	0.08	0.87	0.60	1.03	1.74	17.98
ted_talks	0.04	0.29	0.71	2.65	84.73	582.35
Ø	0.07	0.49	0.69	1.84	59.82	351.54

Table 4.2: Graph statistics per dataset. *#words* is the number of words in the whole text dataset (not unique). The metrics for co-occurrence graphs are obtained using a window size of $w = 1$ and with all words, not only nouns. The *#nodes/#words* metric indicates a compression factor. Note that, on average, the concept maps have a compression factor of 7% compared to the 29% of co-occurrence graphs. This means that co-occurrence graphs have approximately four times more nodes than concept maps. The *#edges/#nodes* metric is an indicator for the connectedness of the graphs. Because we remove nodes which have no edge to other nodes, the *#edges/#nodes* metric captures the average degree of the nodes. Looking at that metric we also see that, on average, the co-occurrence graphs with window size 1 have roughly more than twice as much edges per node as concept maps.

of small connected components and are generally quite un-connected. Connected components of size 2 and 3, ie. containing 2 or 3 nodes, make up well over 80 % of all connected components.

In Figure 4.3 we also plotted an histogram of the number of connected components per graph for *ng20*. In Figure 4.5 shows examples of concept maps and co-occurrence graphs with different window sizes.

	$\|s_c\| = 2$	$\|s_c\| = 3$	$\|s_c\| = 4$	$\|s_{all}\| > 1$
ling-spam	63.8	19.5	6.9	89.6
ng20	65.2	19.4	6.9	72.2
nyt_200	61.4	19.8	7.7	100.0
r8	54.3	21.2	9.0	77.8
review_polarity	62.0	20.4	7.5	100.0
rotten_imdb	66.9	23.6	6.9	19.1
tagmynews	54.2	28.3	11.2	31.7
ted_talks	62.6	18.4	6.9	99.4
Ø	% 61.3	% 21.3	% 7.9	% 73.7

Table 4.3: Percentage of connected component size of concept maps per dataset. $|s_c| = n$ corresponds to the size of the connected component and the percentage signifies how many connected component in the whole dataset have this size, eg. when $|s_c| = 2$ has a 50% share it means that 50% of all connected components in all graphs have the size 2. $|s_{all}| > 1$ stands for the percentage of graphs that consist of more than one connected component, eg. when $|s_{all}| > 1$ is 50% it means that 50% of all graphs have more than one connected component.

The co-occurrence graphs also have a simple structure. Co-occurrence graphs are always connected, ie. the number of connected components is 1, or 0 in the case of an empty graph. When the window size

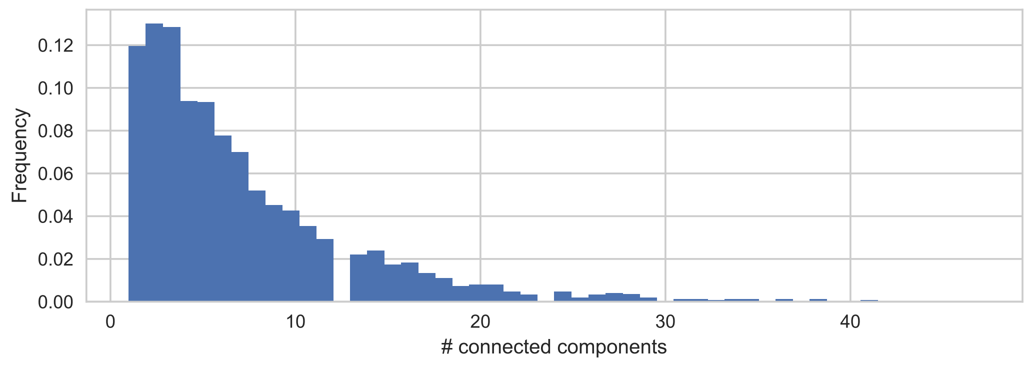


Figure 4.3: Histogram of connected components per concept map. Dataset: *ling-spam*.

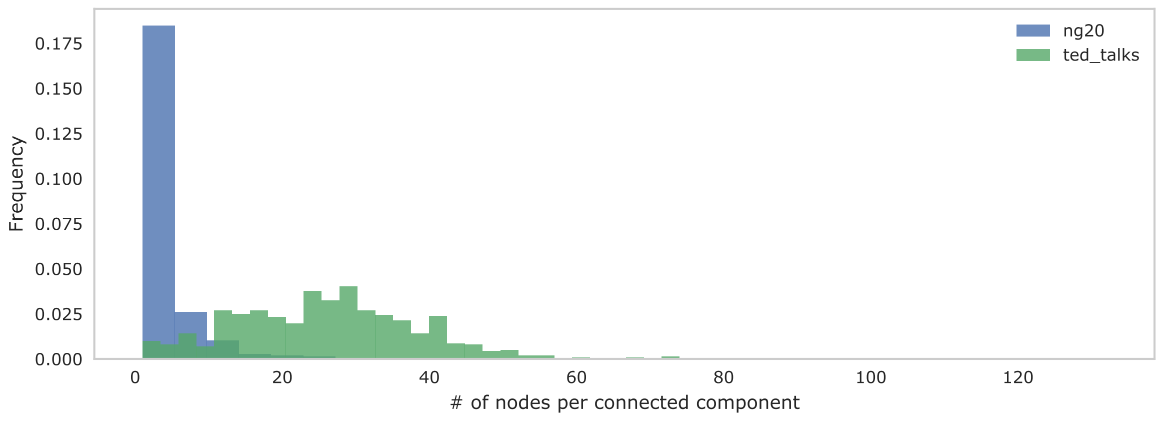


Figure 4.4: Histogram of the size of connected components, ie. the number of nodes per connected component, of concept maps for two datasets. The distribution of the size of connected components varies greatly between different datasets. In this instance, the *ng20* dataset has mostly small connected components while the *ted_talks* dataset has bigger connected components. This is most likely due to the fact that the *ted_talks* dataset consists of longer, more coherent texts while the *ng20* texts are shorter as well as more informal, leading to fewer recurring concepts.

is 1, the graph is similar to a path, meaning that most of the nodes have a degree < 2. With increasing window size, the graph also gets more connected.

Answer Summary: Concept maps have a relatively homogeneous structure. The bulk of its nodes have only a small number of neighbors. The degrees of nodes are quite low, also hinting that there are few recurring concepts.

Answer to Question 3: How important is the structure of concept maps compared to the content?

After investigating the structure of concept maps, we now aim to quantify the importance of the structure compared to the content. The content, that is the node and edge labels, of concept maps are also captured in co-occurrence graphs and with conventional text-based approaches. So, the next interesting question about concept maps is, how much or whether the structure adds to the classification performance. For this, we compare the results of using graph kernels which use **(a)** only the content, **(b)** only the structure and **(c)** both content and structure.

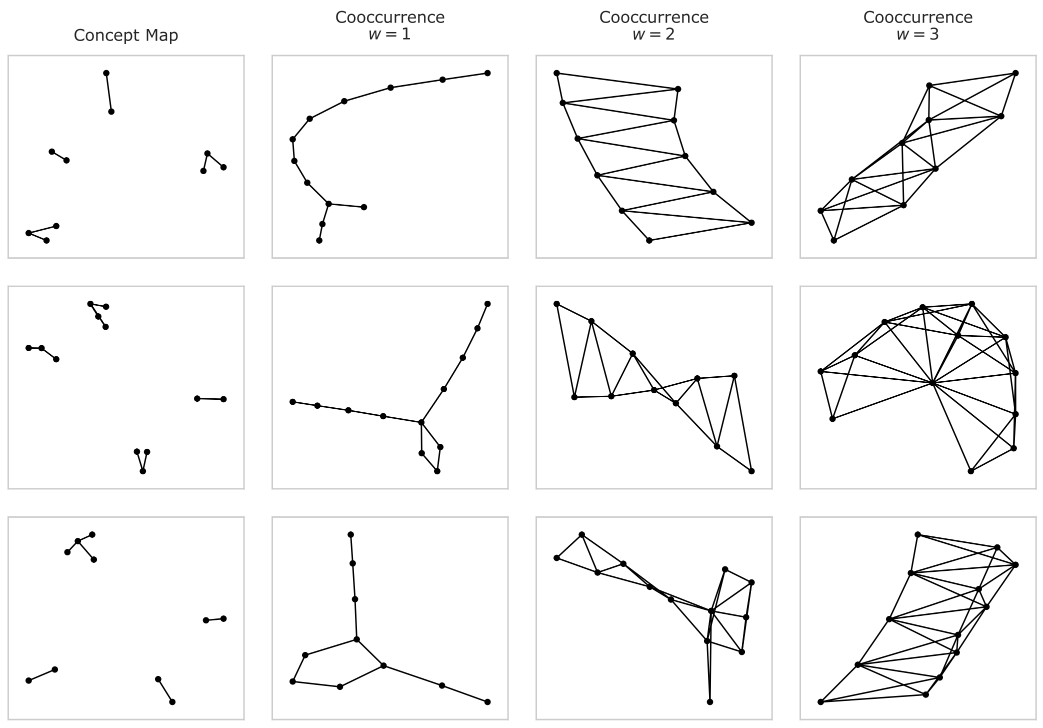


Figure 4.5: Graph examples per type. Three random examples are shown per type. w stands for the window size of the co-occurrence graph. The concept map examples all have more than one connected component, while all co-occurrence graphs have only one. The shown graphs were randomly chosen from all graphs with $5 \leq |V| \leq 10$.Dataset: *ng20*

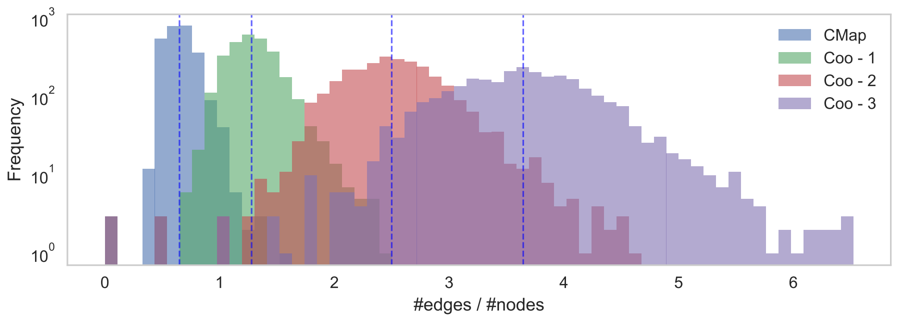


Figure 4.6: Histogram of the number of edges divided by the number of nodes. Per graph type. The lines correspond to the median value.

For **(a)** (content only), we use a kernel that discards all edges and uses only the labels of nodes and edges. Next, we create a bag-of-words vector representation out of the labels and edges. In this step, we also evaluated using not only single words and counting them, but also using word n-grams of size 2, or bi-grams. For this, we create pairs of words by joining node labels together that have an edge between them. The resulting vector representations of the graph then get fed into a conventional classifier, in our case a SVM.

For **(b)** (structure only), we use a modified version of the Weisfeiler-Lehman graph kernel. Before applying the actual WL kernel, we discard all node labels and give every node in all graphs the same label, effectively ridding the graphs of content. Next, we apply the WL graph kernel. This variant of WL only takes the structure of the graph into account. After executing WL on the graphs, we obtain the feature maps which get subsequently get fed into a SVM also.

For **(c)** (structure and content combined), we use the Weisfeiler-Lehman that takes both structure and content into account.

In Table 4.4 we report the results obtained from these experiments.

		F1 macro			
		(a) content only	(b) structure only	(c) both	Dummy
ling-spam	Concept Map	0.934	0.544	0.816	0.421
	Cooccurrence	0.997	0.620	0.987	
ng20	Concept Map	0.622	0.057	0.419	0.051
	Cooccurrence	0.627	0.068	0.593	
nyt_200	Concept Map	0.899	0.324	0.744	0.170
	Cooccurrence	0.891	0.422	0.881	
r8	Concept Map	0.892	0.184	0.677	0.087
	Cooccurrence	0.890	0.300	0.890	
review_polarity	Concept Map	0.737	0.575	0.609	0.502
	Cooccurrence	0.762	0.526	0.785	
rotten_imdb	Concept Map	0.828	0.579	0.635	0.504
	Cooccurrence	0.831	0.561	0.825	
ted_talks	Concept Map	0.359	0.279	0.244	0.236
	Cooccurrence	0.461	0.141	0.443	

Table 4.4: Results for linearized graphs. *Combined* are features generated using plain WL, using both structure and content. With *Structure-only*, all node labels are omitted, then also plain WL. *Content-only* linearizes the graph into text, then does conventional BoW feature extraction.

Here, we see that the content-only approach works the best for both concept maps and cooccurrence graphs. The structure-only approach performs far worse, nearly as worse as the dummy classifier which only predicts the most-common label. The combined approach, ie. plain WL, performs better than the structure-only but worse than the content-only approach.

While it is no surprise that removing the labels from the graphs results in a far lower classification score, the extent in which co-occurrence graphs still perform better than concept maps has to be noted. This could indicate that the structure of co-occurrence graphs, while relatively simple, might also be useful for classification.

Answer Summary: The content-only graph kernel, which un-rolls the graph into text and then uses a conventional uni-gram *BoW* vectorizer approach, performs the best for both concept maps and co-occurrence graphs. Ignoring the labels and then running WL results in low classification scores comparable to the most-frequent dummy classifier. When combining both, content and structure, ie. plain WL, the score is lower than content-only. This all indicates that the content of the graphs, for graph-only classification, has a greater importance than the structure since the structure gets completely discarded by the content-only approach and still performs the best.

Answer to Question 4: How useful are multi-word labels in concept maps?

In Figure 4.7 we see, that most concept map node labels consist of more than one word.

However, with the normal approach, like counting the node labels, these multi-word labels are treated as single labels, effectively discarding important information. In the experiments of the last question,

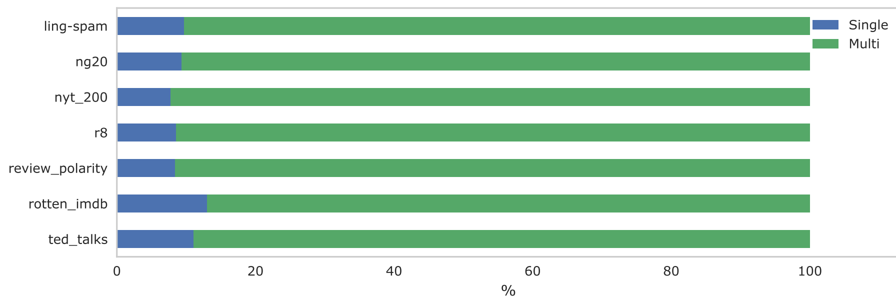
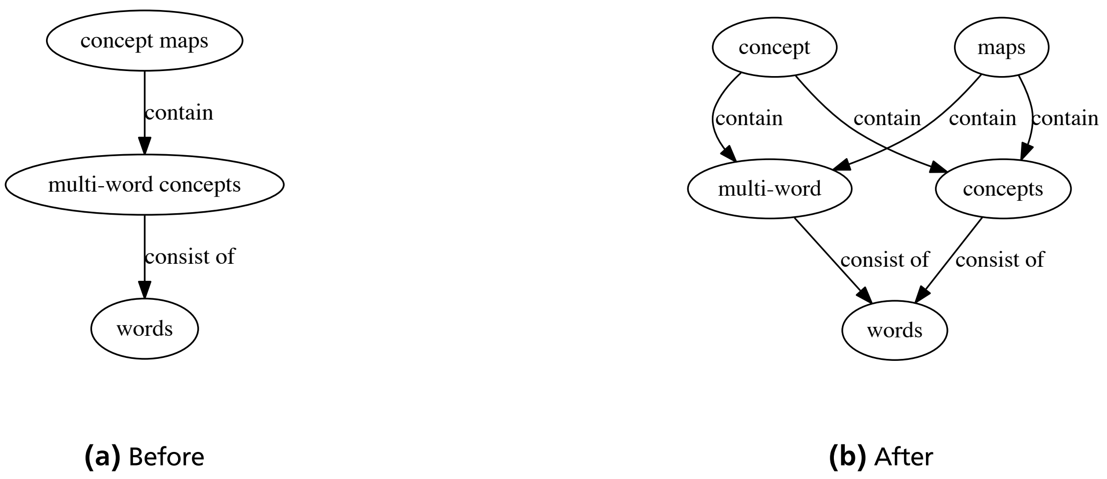


Figure 4.7: Percentage of multi-word to single-word node labels per dataset.

Question 3, we actually implicitly used the multi-word labels since we un-rolled the graph into text again and then used conventional text vectorizer, ie. BoW. In these experiments, we saw an improvement over the graph based approach, possibly due to the multi-word labels being split. So, for our next graph-based approach, we also split nodes with multi-word labels into multiple single-word nodes. This is done by splitting the node labels into individual words, than creating nodes with these single words and adding all the (directed) edges of the original node. When a resulting single-word label is a stopword, the corresponding node is removed from the graph. See Figure 4.8 for an example.

Additionally, we also evaluate the stemming or lemmatizing [p. 4][MS00] the split single-words.



(a) Before **(b)** After

Figure 4.8: Example for multi-word label splits. Here, both the “concept maps" and “multi-word concepts" node labels are split.

In Figure 4.5 we report the results when doing graph classification with and without multi-word label splitting. For this experiment, we use the Weisfeiler-Lehman graph kernel to extract the feature maps. As we can see, the splitting actually improves the performance quite a lot, consistently outperforming the non-split version on all datasets. On some datasets, the additional stemming of single-word labels after splitting also further improves the score.

The classification scores for this multi-word splitting nearly performs as well as the linearized, BoW approach from the previous Question 3, ranging from 1% to 3% difference in F1 macro. However, it has to be noted that splitting the multi-word node labels, and subsequently extracting features with WL, produces higher dimensional feature vectors than the un-split approach.

	F1 macro			
	Un-Split	Un-Stemmed	Stemmed	p-Value
ling-spam	0.8160	0.8835	**0.9131**	0.0008
ng20	0.4188	0.5680	**0.5741**	0.0001
nyt_200	0.7436	**0.8464**	0.8294	0.0001
r8	0.6772	**0.8599**	0.8441	0.0001
review_polarity	0.6094	**0.7150**	0.6699	0.0036
rotten_imdb	0.6346	**0.7805**	0.7775	0.0001
ted_talks	0.2436	0.3934	**0.4126**	0.0035

Table 4.5: Results for multi-label node label split. The p-value is calculated with the best split-word approach, ie. stemmed and un-stemmed, versus the plain version.

Answer Summary: Splitting the multi-word labels shows the greatest performance improvement for WL, ranging from a raise of 10% to 24% better F1 macro score over the un-split approach. Splitting the node labels and then using WL performs nearly as good as the best graph kernel we saw, the linearized content-only approach from Question 3.

Answer to Question 5: How diverse and useful are edge labels in concept maps?

To evaluate how important the edge labels are, we first look at the occurrences of edge labels, ie. how often a unique edge label occurs in the concept maps of a whole dataset. In Table 4.6 we see that, on average, 76% of all unique edge labels occur only once per dataset. For comparison, the percentage of unique words which only occur once in a text, on average, is often well below 50% for our datasets.

dataset	$\%_{unique}$	$\%_{all}$
ling-spam	69	27
ng20	73	30
nyt_middle	73	32
nyt_small	78	43
review_polarity	80	39
rotten_imdb	80	47
tagmynews	75	39
ted_talks	80	39
Ø	76	37

Table 4.6: Percentage of concept map edge labels occurring only once in the whole dataset. $\%_{unique}$ corresponds to the percentage of edge labels which only occur once to the number of unique labels in the whole dataset, eg. $\%_{unique} = 50\%$ would mean that 50% of all unique edge labels occur only once. $\%_{all}$ stands for the percentage of edge labels which occur only once to all labels (with duplicates), eg. $\%_{all} = 50\%$ would mean that 50% of all edge labels occur only once in the dataset.

When examining the most occurring edge labels per dataset, most of these most-frequent edge labels consist of stopwords or non-descriptive words like “is", “has", “are". These most-frequent edge labels, together with the edge labels which occur only once, form the bulk of all edge labels.

As a test for the importance of edge labels, we evaluate **(a)** a graph kernel which uses the edge labels against **(b)** a graph kernel which does not use edge labels. For both, **(a)** and **(b)**, we use a graph kernel which first converts the graphs into a text and then vectorizes the text with BoW, introduced in Question 3. For **(a)** we use the edge labels for the text, for **(b)** we omit them.

	Edge Labels		p-value
	without	with	
ling-spam	**0.898**	0.896	0.830
ng20	0.620	**0.624**	0.502
nyt_200	***0.900**	0.875	0.031
r8	0.882	**0.883**	0.959
review_polarity	0.700	**0.727**	0.271
rotten_imdb	0.817	**0.823**	0.533
ted_talks	**0.410**	0.363	0.164

Table 4.7: Classification results concept maps using a graph kernel with and without edge labels. The star * signifies results where the model is significantly better under the permutation test ($p = 0.05$)

As we can see in Table 4.7, omitting or using the edge labels results in non-significant differences in classification performance. For some datasets, omitting the edge label actually even improves the classification score. These facts all indicate that edge labels, while crucial and useful for text-summarization and the subsequent use by a human, seem to have limited use in graph classification.

Answer Summary: The bulk of all edge labels either occurs only once per dataset or consists of stopwords. In our experiments, omitting or using the edge labels actually leads to comparable performances.

Answer to Question 6: Does removing infrequent node labels from the concept map improve classification performance?

In Figure 4.9 we can see that, depending on the dataset, between 75% to 90% of all node labels occur only once per dataset. This is also common in texts where most words also only occur once per dataset.

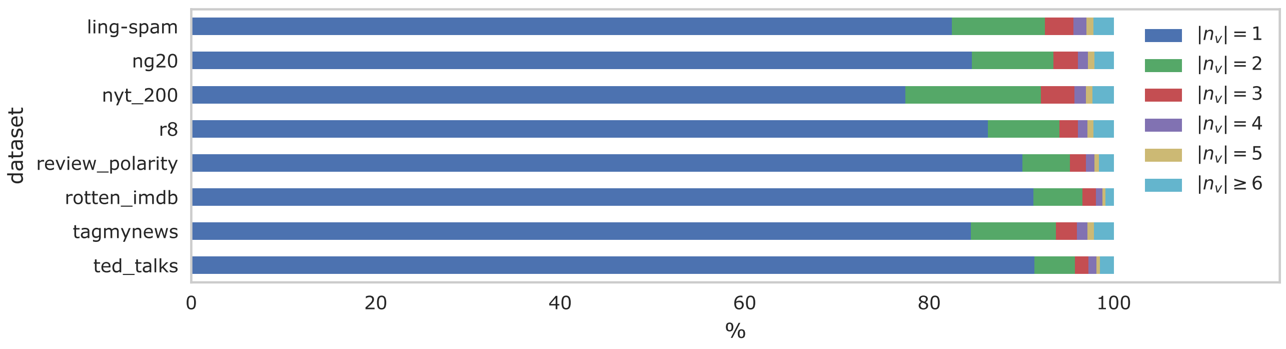


Figure 4.9: Concept map node label occurrences per dataset. $|n_v| = i$ stands for the percentage of labels with i occurrences in the dataset, eg. when $|n_v| = 1$ has 50%, it would mean that 50% of all unique concepts only occur once per dataset.

In text-based approaches, infrequent words are either ignored or filtered out to reduce the vocabulary and subsequently the dimension of the feature vector Yet, for our approach, infrequent node labels might pose a far greater problem. As noted before, in our work, we capitalize on the Weisfeiler-Lehman graph kernel to extract useful features for subsequent classification. In the context of WL, infrequent node labels might become a problem since a match becomes less likely with fewer occurring words, or a match even becomes impossible as in the case of node labels which occur only once. When simply creating a feature vector by counting the node labels in each graph, infrequent node labels would not pose a problem. WL on the other hand relies on exact matches of neighborhoods. Thus, a node label which only occurs once would “taint" its neighborhood, effectively making matches in its neighborhoods impossible.

In Table 4.8 we see the results of removing infrequent labels.

	F1 macro		
	Plain	Removed	p-value
ling-spam	**0.8160**	0.8035	0.3215
ng20	***0.4188**	0.3565	0.0001
nyt_200	0.7436	**0.7741**	0.1089
r8	***0.6772**	0.6327	0.0113
review_polarity	**0.6094**	0.6068	0.9349
rotten_imdb	***0.6346**	0.5514	0.0001
ted_talks	0.2436	***0.3637**	0.0092

Table 4.8: Classification results with infrequent nodes removed.

As we can see, removing infrequent node labels results in a lower score for most datasets, except for the *ted_talks* corpus.

Answer Summary: Removing infrequent node labels results in lower scores, apart on the *ted_talks* dataset. Interestingly, as we have seen in the answer for Question 1, when removing the infrequent labels and then combining the resulting WL features with text features, removing the infrequent labels actually resulted in the highest scores on two datasets. This once more shows that observations made on graph-only classification might not hold when combining the graph- with text features.

Answer to Question 7: Does merging infrequent concepts increase the classification score?

In the previous question, we looked at infrequent node labels and removed them. For this question, we actually merge infrequent nodes label with other, more frequent nodes labels. To do this, we have to find a measure of similarity between two node labels to determine whether they should be merged or not. There are a great number of approaches to define similarity between two word labels, eg. for example the edit distance between word sequences. For our purpose, we use word embeddings [Mik+13; PSM; GL14] to obtain a measure of similarity between multi-word labels. We leverage the pre-trained word embeddings introduced and provided in [PSM]. Each word w in the vocabulary of the pre-trained word embedding has a relatively low-dimensional vector v_w assigned, the embedding. Roughly speaking, the intuition of word embeddings is that when an word w is semantically similar to another word w', the distance $||v_w - v'_w||$ between their embeddings is low, while dissimilar words have higher distances. Pre-trained word embeddings, as a downside, have a fixed vocabulary. So, we actually have two problems when using word embeddings for our multi-word node labels, namely (1) we have multi-word labels, while our pre-trained word embeddings only contain single single-word embeddings, (2) some node labels will not be in the vocabulary of the pre-trained vocabulary. Preliminary tests show us that both problems, (1) and (2), are very common in our datasets and therefor must be addressed.

Multi-Word and Missing Node Label Lookup

Our idea to solve these issues was to first create a Word2Vec [Mik+13] word embedding *(TrainedEmbedding)* from the texts in the datasets. In the next step, we resolve (1) multi-word labels and (2) labels which are not in the vocabulary of the pre-trained word embedding *(Pretrained Embedding)*. For each node label n, (a) we split n into single words if it is a multi-word label and then (b) obtain the embeddings for the single words from the *(TrainedEmbedding)*, then (c) create the average of the found embeddings. If the node label contains only one word, we need not average the found embeddings. So, in this stage we have obtained a (multi-) word embedding v_n for the node label n from the *(TrainedEmbedding)*. Next, we search for similar words ws to v_n with the constraint that each word in ws also has to be in the *(Pretrained Embedding)*. We obtain the similar words by using the multi-word embedding v_n to search for similar word embeddings in the *(TrainedEmbedding)*. The similar words ws also contain the similarity between the nodes, ie. their distance. We only keep the top n similar words in ws. In our case, $n = 10$.

So, at this stage, for each (multi-word) node label n in the concept maps, we have similar words ws_n which are both in the *(Pretrained Embedding)* and the *(TrainedEmbedding)*. Now, we solved both problems, (1) the multi-word labels and (2) words which are not in the vocabulary of the *(Pretrained Embedding)*.

Node Label Clustering

In the next step, we have to find node label sets which should be merged. For this, we greedily merge labels into clusters with unique identifiers, eg. consecutive numbers, where each label n in a cluster have a similarity, or distance, greater than a given threshold, t, to *any* node label in the cluster. After this step, each node label in the concept maps has a assigned cluster with one or more node labels in it. The intuition of these clusters is that a cluster only contains similar labels. This also means that we could merge the labels in some cluster.

Infrequent Node Label Merging

So now, we actually can replace infrequent node labels with the cluster identifier. Assuming that both the *(TrainedEmbedding)* and the *(Pretrained Embedding)* actually capture semantical similarities, this new label will be semantically similar to the original, but infrequent node label.

Discussion

There are several assumptions made here for this approach to be useful, namely that a semantically meaningful word embedding for infrequent node labels can be obtained. When a node label only occurs once in the whole dataset, creating a meaningful word embedding for this single occurrence can be quite difficult since word embeddings actually work by defining a word by its context. Yet, in this instance, we have only one context since the node label occurs only once. That said, this caveat only applies partially since most of the node labels we merge actually are multi-word labels and the individual words in them occur, most likely, more than once. Another important caveat to keep in mind is that we used the complete text corpus to train our *(TrainedEmbedding)*. This means that we did no trainings- and validation split nor cross-validation at all. This was done to reduce the already quite high compute time of training the embeddings and merging them. The results are most likely tainted by the omission of the clear separation between train- and test data.

Results

After relabeling the infrequent labels as explained above, we then use our default approach, applying WL to the relabeled concept maps to obtain feature maps, then classifying them. As a baseline, we also apply the same approach to un-relabeled concept maps. In Table 4.9 we report our results. For the experiments, we tested different thresholds $t \in \{0.5, 0.7, 0.9\}$ and $n = 10$, that is we only considered the top-10 words when creating the new embeddings, see above.

Threshold t	F1 Macro 0.5	0.9	Plain	p-value
ling-spam	**0.831**	0.817	0.816	0.515
ng20	0.377	0.383	*__0.419__	0.000
nyt_200	0.739	0.727	**0.744**	0.885
r8	**0.705**	0.693	0.677	0.268
review_polarity	**0.616**	0.606	0.609	0.780
rotten_imdb	0.577	0.605	**0.635**	0.194
ted_talks	0.281	**0.317**	0.244	0.188

Table 4.9: Classification results for relabeled concept maps. The p-value is obtained by a significance test between the plain and the best relabeled version, eg. 0.5 or 0.9. Plain corresponds to the un-relabeled approach.

As we can see, on some datasets, relabeling the nodes can improve the classification score, while others do not profit from this additional pre-processing step.

Answer Summary: After devising and applying an approach to merge (multi-word) node labels of concept maps, we classify them. In most datasets, this results in a great improvement in classification performance. Nevertheless, there are also big caveats to this approach, for instance the additional overhead of computing word embeddings for the whole dataset.

Answer to Question 8: How does the performance of using the directed edges in concept maps compare to undirected edges?

The concept maps we extracted have directed edges. In this question, we evaluate the usefulness of these two cases. We compare the classification performance of using the directed versus un-directed case with the Weisfeiler-Lehman graph kernel.

When using WL with undirected edges, the neighbors n_v of a node v are all nodes v' that are connected by an edge to v, or $n_v = \{v' | (v, v') \in E \lor (v', v) \in E\}$. With directed edges, on the other hand, the neighbors n_v of a node v are only nodes v' where there exists a directed edge from v to v', or $n_v = \{v' | (v, v') \in E\}$. So, the size of neighborhoods per node are expected to decrease, since there is an additional constraint to what constitutes a neighbor.

In Table 4.10 we see the results of using WL with directed and un-directed edges. On all datasets, the directed edges outperform the un-directed ones.

	F1 macro un-directed	directed	p-value
ling-spam	0.7482	*__0.8160__	0.0001
ng20	0.4165	**0.4188**	0.6944
nyt_200	0.7430	**0.7436**	0.9784
r8	0.5764	*__0.6772__	0.0001
review_polarity	0.5807	**0.6094**	0.1326
rotten_imdb	0.6328	**0.6346**	0.7433
ted_talks	0.2189	**0.2436**	0.3107

Table 4.10: Classification results when using directed versus un-directed edges with WL.

This is most likely due to the aforementioned property that the neighborhoods are smaller with directed edges, making exact matches of neighborhoods more probable. When the order of words/nodes is fixed in one direction, vertices with no outgoing edges do not have a neighborhood. Thus they are, for all WL iterations, considered to be the same label, ie. they get no new color assigned.

As a result of these better classification results with directed edges, we will use the directed version of WL. Later on, we will also evaluate whether or how using (un-) directed edges affects the score when combining graph and text features.

Answer Summary: Using directed instead of un-directed edges results in (significantly) better performance with the WL graph kernel. A possible explanation for this better performance is that neighborhood matches with directed edges are far easier since the neighborhood of a vertex is only defined through its outgoing edges, and not the in-going as well. The smaller neighborhood in turn makes exact matches of neighborhoods in different graphs more probable.

Answer to Question 9: How does the size of concept maps relate to classification performance?

In most datasets, longer texts often result in higher classification performance. So, an interesting question is whether the size of a concept map also correlates with its classification performance, ie. whether "bigger" graphs in terms of the number of nodes and edges are easier to classify. To test this, we classify the graphs with our standard approach, using the Weisfeiler-Lehman graph kernel, then a SVM to classify the feature maps. For text, we extract the features with Tfidf-BoW, then also classify the features with a SVM.

In Table 4.11 we report the Spearman rank and Pearson correlations [HK11] between the accuracy, ie. whether the label for the document/graph was predicted right, and the size of the document/graph. Here, we also see a (weak) correlation for the graph/text sizes in some datasets. In the *ling-spam* and *ng20* dataset, the accuracy correlates far more with the document/graph size than in other datasets. It has to be noted that there is no partial similarity in accuracy, only 1 if the label was predicted right and 0 otherwise. This might also explain that the relatively low correlation between the graph/document size and the accuracy, even in *ng20*. Interestingly, on some datasets, the accuracy correlated far stronger with the concept map size than with the size of the texts, eg. *rotten_imdb* or *nyt_200*.

These results also highlight another finding: one observation for some dataset might not be consistent with observations from another dataset, eg. for our results the correlations are sometimes weak, other times practically not present at all.

We also tested other metrics to find correlations, for instance the number of edges or the number of connected components, yet we could not see relevant correlations.

Answer Summary: In some datasets, we detect a correlation between the size of graph/documents and prediction accuracy. For other datasets, this correlation can not be measured distinctly. The results vary widely across datasets and between graphs/texts.

Answer to Question 10: How does the classification results of co-occurrence graphs compare to concept maps?

As a baseline, we also look at another graph representation for text, namely co-occurrence graphs. The comparison between concept maps and co-occurrence graphs is somewhat unfair since co-occurrence graphs retain much more of the content of its underlying text. While both concept maps and co-occurrence graph are graph representations, they have several differences. While concept maps have

	Graph		Text	
	Spearman	Pearson	Spearman	Pearson
ling-spam	0.2135	0.1294	0.1490	0.1073
ng20	0.2699	0.1554	0.2415	0.1395
nyt_200	-0.1986	-0.1692	0.0077	0.0230
r8	-0.0212	-0.0455	-0.0499	-0.0699
review_polarity	-0.0311	-0.0334	-0.0286	-0.0301
rotten_imdb	0.1680	0.1629	0.0313	0.0513
ted_talks	-0.0232	-0.0586	-0.0991	-0.1337

Table 4.11: Spearman rank correlations and Pearson correlation coefficient between the accuracy and graph/text size. Here, the graph size is the number of nodes in the graph, text size the number of words. The higher the absolute value of the Spearman/Pearson coefficient is to 1, the higher the correlation. A positive correlation value between A and B means that if A increases, B also increases and vice versa. A negative correlation means that when A increases, B decreases and vice versa.

directed edges with labels, co-occurrence edges are, in the our version, undirected and have no edge labels. The node labels for concept maps also contain multiple words while co-occurrence graphs have single-word labels per definition.

We use the comparison of co-occurrence graphs to concept maps to establish a baseline for graph representations. Since our main hypothesis assumes that concept maps contain useful structural information, the co-occurrence graph is a good candidate for comparison, since its structure is quite simple. For window size $w = 1$, the co-occurrence graphs have a mainly linear structure. With higher window sizes, the co-occurrence graphs get more connected until they eventually become fully connected.

In Table 4.12 we report our results.

	F1 macro		
	Concept	Co-Occurrence	Difference
ling-spam	0.816	**0.987**	0.171
ng20	0.419	**0.593**	0.174
nyt_200	0.744	**0.881**	0.138
r8	0.677	**0.890**	0.213
review_polarity	0.609	**0.785**	0.175
rotten_imdb	0.635	**0.825**	0.191
ted_talks	0.244	**0.443**	0.199

Table 4.12: Classification scores for co-occurrence graphs and concept maps.

As we can see, graph classification using co-occurrence graphs performs significantly better than the concept maps. Co-occurrence graphs also perform nearly as good as the conventional, text-based approach. That said, co-occurrence graphs contain far more information than concepts, as we have seen in Table 4.2. Concept maps on the other hand summarize the text by filtering out only important concepts. Conversely, co-occurrence graphs actually contain all words of its underlying text. For this comparison, we used co-occurrence graphs where we only keep the nouns of the text to mimic the summarization of concept maps. Apart from that, since co-occurrence graphs contain more information, creating the feature maps with WL also incurs more compute time. Interestingly, for the combined graph- and text feature approach, we have actually seen that co-occurrence graphs perform worse than concept maps.

This is most likely due to the high dimensionality of the co-occurrence graphs which subsequently leads to overfitting.

Answer Summary: Using WL, co-occurrence graphs perform far better than concept maps in graph-only classification. One explanation is that co-occurrence graphs actually retain more information about the text, ie. all words and their co-occurrence, while concept maps have a far higher compression factor, even though the co-occurrence graphs were created from nouns-only. However, this increased number of nodes/edges in co-occurrence graphs also increases the runtime and memory footprint when extracting WL features. Also, when combining graph- with text features, co-occurrence graphs perform not as good as concept maps, as seen in the answer for Question 1.

Answer to Question 11: How does the classification performance with concept maps compare to non-structural, text-based approaches?

As one can see in Figure 4.13, the text-only approach outperforms graph-only approaches by a high margin. This is most likely due to the high compression factor of both concept maps and co-occurrence graphs.

	F1 macro			
type	Concept	Co-Occurrence	Text (Count)	Text (Tfidf)
ling-spam	0.816	0.987	0.986	0.990
ng20	0.419	0.593	0.754	0.781
nyt_200	0.744	0.881	0.921	0.912
r8	0.677	0.890	0.921	0.919
review_polarity	0.609	0.785	0.862	0.877
rotten_imdb	0.635	0.825	0.881	0.886
ted_talks	0.244	0.443	0.443	0.464

Table 4.13: Results for graph- and text-based classification

So, we see that graph-only approaches seem to perform worse than text-only approaches by default. When linearizing the graphs into text again and apply conventional text vectorizers, eg. BoW with Tfidf, we get the best results, also hinting to the fact that the structure is far harder to leverage than simple node counts. That said, we are interested in whether graph representations are useful in text classification, therefor we will also test the performance of graph-based approaches when combining them with conventional text approaches.

Answer Summary: In our experiments, the text-only approach performs better than both our graph-only approaches, namely WL with concept maps and co-occurrence graphs. One possible explanation can be found in the compression factor of both co-occurrence and concept maps.

Answer to Question 12: How does the compute time of graph-based- compare to text-based approaches?

Until now, we only looked at the classification scores of our classification approaches. Another aspect of great real-world importance is the runtime and memory consumption of these approaches. In table 4.14, we report empirical data on both the runtime[2] and final classifier size of the used pipelines for each

[2] Machine specification. CPU: *Intel(R) Core(TM) i7-2675QM CPU @ 2.20GHz, 4-Core, RAM: 8GB*

dataset. For this data, we use simple approaches for both graphs and texts: for the **texts** we vectorize the text documents with BoW with Tfidf. For **graphs**, we vectorize the graphs with the plain Weisfeiler-Lehman graph kernel with $h = 4$ iterations. For both the text- and graph approach, we then train a SVM on the whole dataset, then predict the labels for the whole dataset. Afterwards we save the models, ie. the internal coefficients of the SVM, and record the size of the saved model. We also record the runtime of both approaches.

As we can see, the graph kernel creates higher dimensional feature vectors for the objects than the text-based BoW approach. This observation both give possible explanations for both the higher classifier model size and the run-time. One could circumvent the higher dimensionality of features extracted by WL by applying dimensionality reduction. Yet, this would also incur an additional compute overhead and add complexity to the approach. Preliminary tests with truncated SVD [Hal+09] and PCA [Jol02] to reduce the dimensionality of the graph features resulted in out-of-memory exceptions.

Note, that the reported runtime and memory consumption for the graph-based pipeline does not incorporate the creation of the concept maps, only the creation of the feature maps and subsequent training/prediction. The creation of concept maps from text with the code provided in [FG17] takes well over a day for our datasets. For instance, the concept map creation for the documents in one of the middle-sized datasets, *ling-spam* with 1.2 million words in total and less than 2900 documents, took over 25 hours and had a peak memory usage of 83 Gigabyte[3]. The runtime of the concept map creation could most likely be reduced by parallelizing parts of its pipeline. For example, the extraction of relevant concepts and their relation from a text is independent from the same extraction of another text, so this stage has neither data- nor functional dependencies.

	Classifier Size		Runtime		Feature Runtime		# Features	
	Graph	Text	Graph	Text	Graph	Text	Graph	Text
ling-spam	11	4	5	2	3	2	246	61
ng20	143	29	115	10	15	5	750	134
nyt_200	90	9	59	8	7	6	1061	88
r8	28	3	19	2	6	1	282	25
review_polarity	17	3	7	2	3	2	369	40
rotten_imdb	4	1	5	0	4	0	80	21
ted_talks	17	3	2	2	1	2	254	34

Table 4.14: Runtime, classifier size and number of features of text- and graph based classification. The feature runtime corresponds to the feature extraction runtime, both for WL for graphs and Tfidf for texts. The runtime is reported in *seconds*, classifier size in *Megabytes*, *# Features* in thousands.

Answer Summary: Graph-based classification using WL incurs both a runtime and memory consumption overhead to text-based classification. A possible explanation is the higher dimensionality of the feature vectors created by WL. Applying dimensionality reduction might be a solution to circumvent these issues. However, it has to be noted that the feature vectors are very sparse and are most likely not normalized, so *PCA* is not an option. The creation of concept maps from text also incurs a high runtime and memory consumption overhead.

[3] Machine specification. CPU: *2 × Intel(R) Xeon(R) CPU X5650 2.67GHz, 6-Core*, RAM: *192GB*

4.0.1 Summary

As we have observed in several of the answers, transforming the text classification task into graph classification and using concept maps is possible. While we see near state-to-the-art performance with the graph-based approaches, the graph-only performance still lags behind text-only performance as we have seen in the results of Question 11. We provided possible explanations in Question 2, for example the low connectivity and the high compression factor of concept maps to name just a few.

Questions 3, 4, 5, 6, 7 and 8 all contain approaches to improve the graph-only performance and create more meaningful graph features. While we, as said before, came close to the classification performance of the text-only approach, there most likely is only so much one can do to improve the classificatio score of concept maps. In some of these questions, we explored the particularities of concept maps, namely (1) the multi-word node labels, (2) the directed edges and (3) the edge labels.

For (1), we tried splitting the multi-word concepts of the concept maps into single-word nodes. This resulted in a high increase in classification score. This was done in Question 4.

In Question 3, we also implicitly split the multi-word concepts. Since we linearized the graph into text again, ie. un-rolling the edges into sentences. Afterwards we used conventional, text-based approaches like uni-gram BoW extracted from the resulting text. This approach resulted in the best classification performance we achieved using the concept maps, except when combining the graph features with text features. This observation alludes to the explanation that the structure of concept maps is either not that important for classification or the structure could not be captured usefully using the graph-based approaches we tried.

For (2), the directed edges, we compared the classification results of both directed and un-directed version of the concept maps. Using the directed edges outperformed the un-directed edges consistently and by a high margin. We also provided a possible explanation, namely that the neighborhood of a node are far smaller when using directed edges. Smaller neighborhoods in turn increase the likelihood for matched neighborhoods.

For (3), the edge labels, in Question 5 we first analyzed the distribution of occurrences of edge labels. Here, we observed that most edge labels occur either only once or very often. The top edge labels, ie. the edge labels which had the most occurrences in the concept maps of a dataset, were almost exclusively non-topic related words, like “are" or “is". Next, we linearized the concept map into text with (a) all words and (b) no edge labels. Finally, we used a Tfidf-BoW to create features and classify them. Using or omitting the edge labels resulted in comparable performances, hinting to the observation that edge labels are not as important for the graph-based approach.

Finally, for the last Question 12, we report the classification times and classifier model size of our graph- and text based approaches. Here, we observe that the runtime for our graph-based approaches is higher than that for the conventional text-based classification. We also saw the additional overhead produced by the need to generate the concept maps from the text.

One thing to note, and this is common to all the questions and observations in this work, is that we evaluated our questions on multiple datasets. Often times, the observations varied quite heavily from dataset to dataset. For instance, the performance of one WL extensions might show great improvement on one dataset, while showing a low performance on another dataset. Especially when trying to find correlations and explanations for the performance of our approaches, the differing datasets proved to harden the task. This further highlights the importance of using model selection. Arguably more than in other domains, graph-kernel based classification performance relies heavily on the used graphs, structure, graph kernels and particularities of the domain. We evaluated several other graph kernels, for instance shortest-path- or random-walk based graph kernels, and observed poorer performance than with our default graph kernel, Weisfeiler-Lehman. Yet, even with the Weisfeiler-Lehman kernel with our extensions and several graph pre-processing approaches, we did not outperform the linearized graph approach, ie. where we un-roll the concept map into text then use a conventional BoW-based approach. A possible explanation for this observation is, that the concept maps are generated from the text by extracting multi-word concepts and their relation to each other from the text. In order for this approach to create structurally interesting concept maps, concepts have to occur multiple times. If that is not the case, that is when the concepts occur only once, the resulting concept maps consist of a number of connected components which contain only two nodes and one edge. These two-node connected components are like tri-grams, with multi-concepts and edge label instead of single-words. A similar observation was done in Question 3.

In the next Section 5 we will further summarize our results.

4.1 Related And Intermediate Observations

In this section we report observations which are not directly related to our hypothesis, but have provided us valuable insights into our analysis.

4.1.1 Weisfeiler-Lehman ϕ Feature Map Visualization

When debugging our Weisfeiler-Lehman implementation and extensions, we often wanted to see the effect on the resulting feature vectors. For this, we plotted the feature vector $\phi(G)$ for each graph G for several WL iterations. In Figure 4.10 we see an example of such a ϕ distribution plot.

It has to be noted that we implemented WL where the labels are assigned when they are first encountered. That means that when we create the feature map $\phi(G_i)$ for graph G_i, we relabel the graph by assigning it a new multi-label by taking the neighborhood into account (*recoloring*). In the next step, we compress the new multi-label by assigning it a new number if it has not been encountered before and save it in a multi-label-lookup. If the multi-label has been encountered before, ie. the multi-label is in the multi-label-lookup, we assign it the number it has been assigned before. This is the *label compression* step. So, we iterate over the graphs one by one and create feature maps ϕ. When the first processed graph, G_1, has the new compressed labels $\{1, 2, \ldots, n\}$, the next processed graph, G_2, will have labels that are either **(1)** new, in which case they get a new, compressed label that is higher than n or **(2)** they were already encountered in G_1, so they get the label from the lookup. This way we get incrementing labels. When we also sort the graphs by their assigned class, ie. all graphs of class y_i are processed after another, then one can look at the labels that are assigned to each class and also see the nice pattern which arises in Figure 4.10.

This plot also gives an insight about the convergence of WL. The highest encountered ϕ index in iteration $h = 1$ must be higher or equal than the highest ϕ index encountered in $h = 0$. This is directly due to the fact that more colors are needed to color the graph nodes in higher iterations - if WL has not converged. The number of labels, or colors, $|L_i|$ assigned in iteration $h = i$ must be smaller or equal than the labels for iteration $h = i + 1$, ie. $|L_i| \leq |L_{i+1}|$. When $|L_i| = |L_{i+1}|$, WL is said to have converged, ie. the number of colors will not change anymore.

In this figure, there is also a highest horizontal, green line. This line signifies the highest ϕ index where there was a non-zero entry. It also serves as a marker of the last compressed label number which was assigned in this dataset.

While this kind of plot is useful for analyzing the whole datasets at once, it becomes especially interesting when splitting the dataset into sets, eg. by splitting it into a stratified train- and test set. When creating the feature maps ϕ for the graphs in the train set, for each WL iteration h we save the multi-label-lookup which we is then used to create the feature maps for the test set. The multi-label lookup therefor acts as a kind of continuation, saving the state of the WL iteration with all encountered labels. Then, when creating the feature maps for the test set, we use the multi-label lookups to generate the feature maps in the same way, now for the graphs in the test set. When plotting the train- and test set feature maps separated we can get interesting insights into the dataset and the concept maps. For an example, see Figure 4.11.

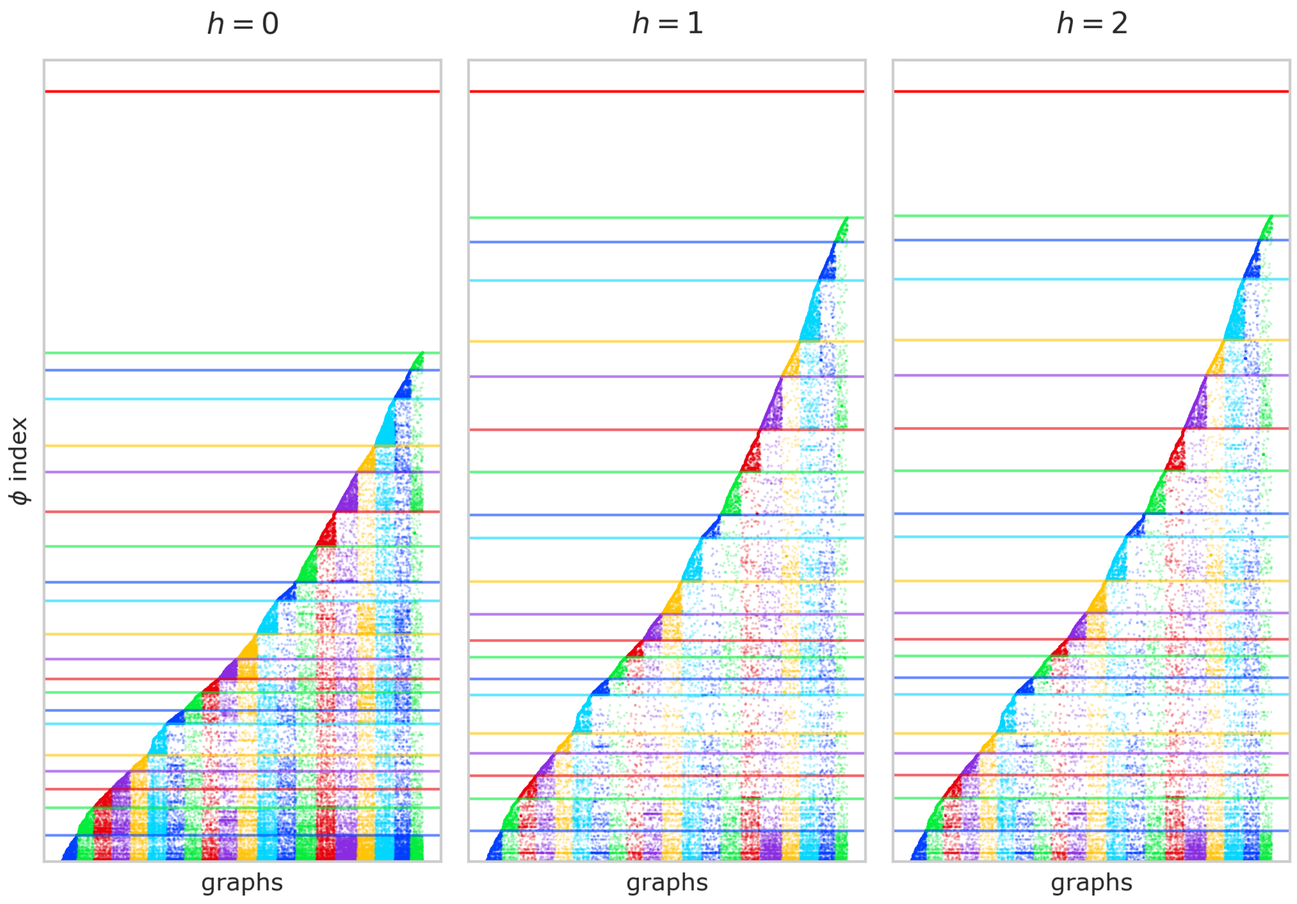
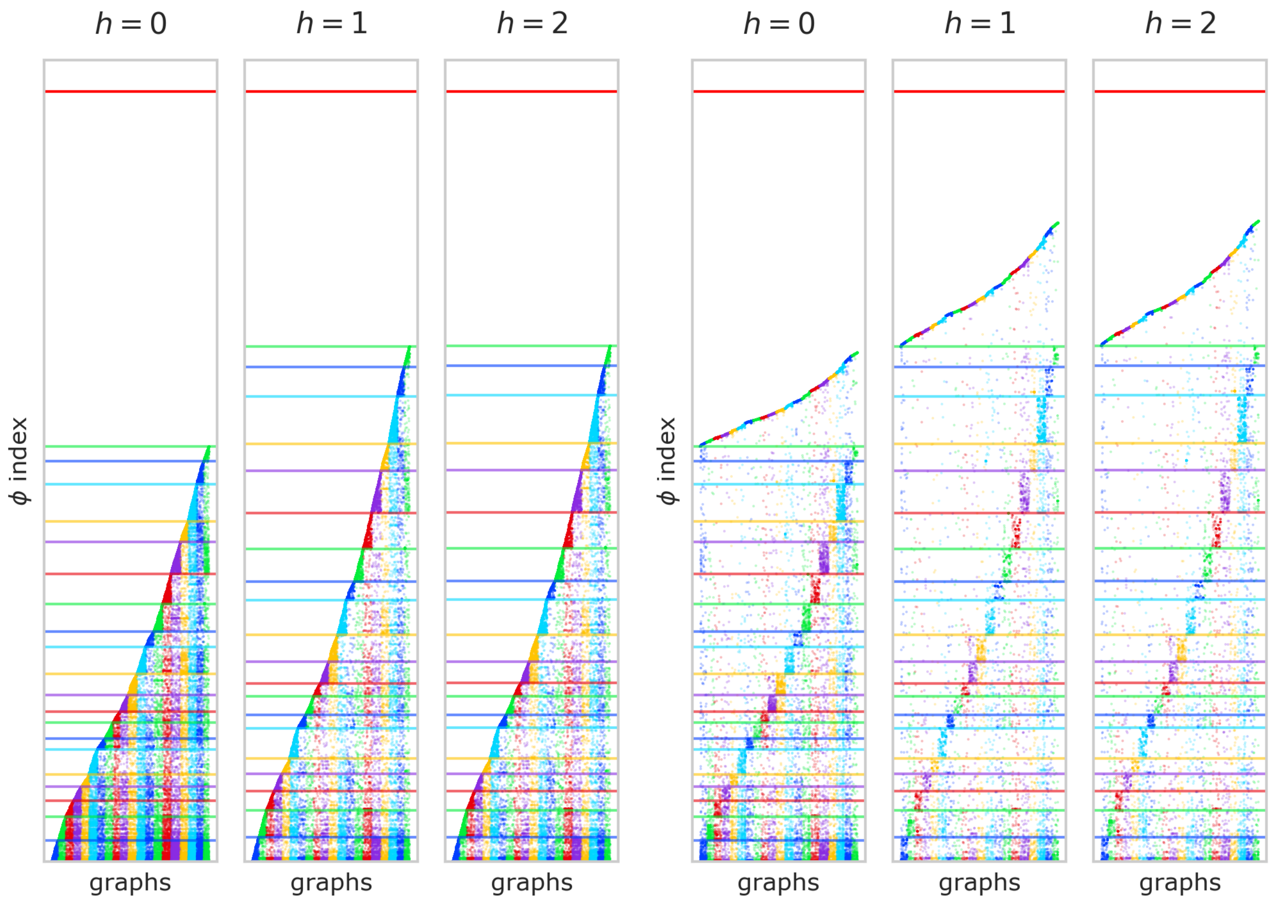


Figure 4.10: Example of a WL ϕ distribution plot. For each WL iteration h, the horizontal x-axis corresponds to the concept maps in the datasets, while the vertical y-axis corresponds to the indices of $\phi(G)$ which are non-zero, ie. when there is a point at $x = 0$ and $y = 10$, the graph G_0 has the label 10. There are as many points in this plot as there are vertices in the dataset. The colors of a point corresponds to the class of the graph, in this case the 20 classes of *ng20*. The red line marks the number of vertices in the datasets which is also the dimension of the feature map ϕ. Dataset: *ng20*. *Note: embedding this plot in a vector format was not possible due to the sheer number of points to be drawn. The rasterized image seen here interpolates the individual points.*

While the train set looks nearly the same as if plotting the simple dataset without the split, the test set differs quite a bit. Notice how, in the test set, there are "clusters" around some regions for each of the classes. Keep in mind that the y-position of the point signifies a non-zero entry at index i in $\phi(G)$ for that graph. Because of the aforementioned implementation, we now are able to see a pattern for each of the classes, namely the clusters. For the test set, we can also see how many new labels have been found in a given iteration. As we said before, the horizontal green line, in both plots, signifies the last label that has been encountered. In the case of the split sets, it also marks the last label that has been encountered in the train set. All points, that is labels, above this line are new in the test set and do not occur in the train set.

Plotting the ϕ distribution like so, we gain a better insight into the graph connectedness and the similarity or uniqueness of neighborhoods. When WL converges at iteration h, so when the height of the green line does not change from iteration $h-1$ to h, this means that all neighborhoods that are different,



(a) Train **(b)** Test

Figure 4.11: WL ϕ distribution with a stratified train/test split.

have been marked as so and got a unique label. If the number of labels is equal to the number of vertices, all neighborhoods are different.

4.1.2 Weisfeiler-Lehman Node Weighting Extension

With increasing iterations h of the Weisfeiler-Lehman algorithm, exact matches of subtrees of height h become more difficult. In iteration 0, WL only counts the number of node labels in the graphs, for iteration 1 it takes the immediate neighborhood of the nodes into account, for iteration 2 is takes the neighborhood of the neighborhood into account and so on. As a result, **(1)** with higher iterations, the probability of having an exact match decreases. Another difficulty arises for nodes with a high number of neighbors. **(2)** The probability of an exact match also decreases with a higher degree.

These two difficulties, **(1)** the increased difficulty of a match with higher iterations and **(2)** the increased difficulty of a match for nodes with higher degrees, are not addressed when using Weisfeiler-Lehman in its plain version. Both issues are encoded neither in the features maps nor are they considered when creating the gram matrix.

As a possible solution for these issues, we propose an extension to the Weisfeiler-Lehman algorithm. Our extension augments the WL algorithm by adding node weights. For each of the nodes n in all graphs, we first calculate node weight n_w which encodes the importance of the nodes or the difficulty of getting an exact match in WL. An example of such node weights are the node degrees or node weights extracted by the *PageRank* algorithm [Pag+98]. The node degree is an interesting metric for node weights since it actually encodes the size of the immediate neighborhood of a node and therefore could possibly address both issues, **(1)** and **(2)**.

One could see our extension as similar to binary *BoW*, ie. only using whether a word has occurred or not, and the extension that also uses the term frequency where the number of occurrences is taken into account, too.

When using this extension, one has to be careful with feature scaling. As we use the node weights to scale individual features, using a separate feature-wise scaler would effectively revert our weighting.

Experiment

To test our extension, we compare results of using WL feature maps with and without our extension. We classify our concept maps datasets as well as a number of other graph benchmark datasets, obtained from [Ker+16]. For statistics about the additional datasets, consult the author's website[4]. We only used datasets from the website consisting of more than 1000 graphs and which contain node labels.

	F1 macro		*p*-value
	Plain	Node-Weights	
ling-spam	**0.8160**	0.7760	0.0542
ng20	**0.4188**	0.4127	0.2340
nyt_200	**0.7436**	0.7150	0.3421
r8	**0.6772**	0.6467	0.1270
review_polarity	**0.6094**	0.6045	0.8406
rotten_imdb	**0.6346**	0.6338	0.8728
ted_talks	0.2436	**0.2984**	0.2605
AIDS	0.9560	***0.9841**	0.0146
DD	0.6602	***0.7733**	0.0002
Mutagenicity	**0.7702**	0.7625	0.3007
NCI1	**0.8425**	0.8334	0.2821
NCI109	**0.7929**	0.7877	0.4971
PROTEINS	0.7266	**0.7317**	0.8554
Tox21_AHR	0.7009	**0.7056**	0.6705
Tox21_AR	**0.8147**	0.7978	0.1172
Tox21_AR-LBD	***0.8477**	0.8284	0.2318
Tox21_ARE	0.6722	**0.6917**	0.1378
Tox21_ATAD5	**0.7456**	0.7426	0.7449
Tox21_ER	**0.6860**	0.6747	0.0868
Tox21_ER_LBD	**0.7335**	0.7180	0.0648
Tox21_HSE	0.6156	***0.6499**	0.0384
Tox21_MMP	0.7663	**0.7682**	0.7874
Tox21_PPAR-gamma	**0.7006**	0.6910	0.7970
Tox21_aromatase	0.6571	**0.6876**	0.1642
Tox21_p53	0.7214	**0.7375**	0.3403

Table 4.15: F1 macro classification results for the node weight extension.

As we can see, the classification performance of our extension varies greatly per dataset. Especially on the concept maps, our WL extension does not perform as good as plain WL. On some of the graph benchmark datasets, on the other hand, we can observe significant improvements. While these results

[4] https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

are interesting, a more throughout analysis of the performance is needed to assert the usefulness of this extension. Especially adding further normalization might prove to be effective.

5 Conclusions

In our work, we conduct several experiments to test the usefulness of graph-representations of text, especially concept maps, for text classification. Our experiments ranged from gathering information about the structure of the graphs to actually performing the classification task with different approaches. For our graph-based classification, we mostly capitalized on graph kernels. Through all these experiments, we aimed to leverage the particular properties of concept maps, especially the structure.

When directly comparing the performance of graph-based and conventional text-based classification, we see that the text-only features outperform the graph-based approach by a high margin, for both co-occurrence graphs and concept maps. So, our next approach is to combine text- and graph features and classify them together. This combined approach, while having a far higher runtime due to the increased dimension of the feature vectors, presents no significant improvement of classification scores upon the text-only approach. On some datasets, the classification score even is a little lower than the text-only approach. This is also most likely due to the high dimensionality of the combined features and the subsequent risk of overfitting to too specific features.

However, after re-evaluating and extending our graph-based approach, we see significant improvement in our graph-only classification performance. Especially splitting the multi-word node labels of the concept maps into single-word nodes results in significant improvement on all datasets. As another extension, we pre-process the graphs by removing infrequent nodes to further reduce the dimensionality of the resulting feature vectors. Since we capitalize on the Weisfeiler-Lehman graph kernel to extract our graph features, our guess is that the removal of infrequent words could improve the quality of the features since WL, roughly speaking, counts the matching neighborhoods of nodes. So, a node label which only occurs infrequently could “taint" its neighborhood, making an exact match of that neighborhood less likely. However, when classifying the graphs with the infrequent labels removed, we actually see mixed results, ranging from great improvements on some datasets to lower scores on others. Another approach we evaluate is merging infrequent nodes by creating embeddings for each (multi-word) node label using both pre-trained embeddings and creating our own word2vec embeddings from the text. In the next step, we merge node labels with a similarity above some given threshold. We then assign identifiers to the resulting label clusters. Finally, we relabel each node in the concept with the identifier of its cluster. On most datasets, this approach improves the graph-only classification scores.

Besides all these Weisfeiler-Lehman specific extensions and improvements, we also evaluate “linearizing" the concept maps into text. By un-rolling the graph into text, we are able to perform text-based classification on them, which - surprisingly - provides the highest classification score we achieve using graphs, for both co-occurrence graphs and concept maps. For this “graph kernel" we discard all structural information since we only use uni-grams, ie. single word frequencies. In the case of co-occurrence graphs, the performance is nearly as good as the text-only approach, which is not entirely surprising since co-occurrence graphs capture *nearly* all information beside the word order which we do not use with our uni-gram *BoW* approach anyway. For concept maps, the classification scores with this simple linearized graph kernel are also the best, yet still approximately 5-10% lower in the F1 macro score than both co-occurrence graphs and the text-only approach.

We subsequently repeat the experiments of text- and graph combined features with our extensions, eg. the multi-word label splitting or the relabeling. Here, interestingly, the classification performance is lower than without these extensions, ie. using the plain WL graph kernel. This observation is somewhat surprising since these extensions all improved the classification score when only classifying the graphs. One possible explanation is, that all these WL extensions aim to transform the concept maps into graphs where the neighborhoods are equalized, ie. more similar to each other. For instance, the relabeling extension aims to merge infrequent with frequent labels, in order to increase the likelihood of same neighborhoods. On the other hand, doing this also removes (structural) information from the concept maps, which in turn can not be used for classification. For the multi-word label splitting extension, one possible explanation for the lower classification performance when combining the resulting features with text features, is, that it significantly increases the dimensionality of the feature vectors created by WL. This, in turn, might have lead to overfitting.

Apart the graph kernel customizations we devised for concept maps, eg. splitting the labels, we also propose our own extension to WL, namely node weighting. Here, our intuition was taken from the fact that WL matches of big neighborhoods are far more difficult than with low-size neighborhoods. Since this issue is not explicitly encoded in the feature maps of WL, we propose to scale each feature map entry, corresponding to the label of a node, with a node weight, eg. the degree of that node. That way, the importance, or frequency, of a node is also encoded in the feature map. Although this results in lower classification scores for our concept map datasets, we see significant improvement on datasets obtained from [Ker+16]. However, a more throughout analysis would be appropriate to confirm the usefulness of our WL extension.

Beyond all the different approaches and experiments, the importance of evaluating the classification scores on different datasets became clear immediately. One approach leading to a great improvement on some datasets, might actually lead to far lower scores on other datasets. This once more highlights the importance of appropriate model-selection per dataset, especially when using a graph-based approach. Understanding the trade-offs of different graph kernels is crucial to achieve higher performance. The information graph kernels capture varies widely, ranging from simple counts of node labels to more sophisticated structural information. Therefore, knowing the structure and particularities of the processed graphs is essential to achieve higher classification performance. In our case, the graph kernel that performed best actually translated the graphs back into text.

Even though we were not able to augment the text-based classification toolbox by another approach, that is graph-based features, we nonetheless explored a number of extensions to existing graph kernels which could be useful for other graph-based tasks. In our work, we started from text, created graph representations and then classified them to evaluate the usefulness of graph representations in text classification. However and importantly, concept maps and other graphs can also be created from scratch or extracted automatically from non-text sources, eg. knowledge databases. While text is currently arguably the most important information medium, maybe - with the rise of more interactive media - concept maps become of greater importance in the future. There are several advantages of concept maps over text. For instance, to understand a paragraph at the end of a text, one often must have read the preceding text. Relations between concepts in a text are often implicit and have to be inferred from the context. For a reader, a mental picture of the content of the text therefor must carefully be created by the author of the text. Several levels of detail have to be assessed and introduced by the author, thoughtfully connecting concepts. In concept maps, on the other hand, concepts have explicit relations to each other. One can start at every node of a concept map and explore the relationships between concepts by following edges. This enables the non-linear and visual exploration of the topic of a concept map. Apart from that, one might also imagine creating concept maps with different levels of details, therefor adding the possibility to add/remove parts of the graph based on their level of detail. One can think of guided tours through concept maps where parts of the graphs get revealed after each other. The interactive possibilities of concept maps are numerous, the key being their easy modifiability. While adding information to texts can be quite laborious since one must find the appropriate section in the text where to add the new parts, with concept maps, on the other hand, extending the graph is as easy as adding a new node or edge to the graph. Merging multiple concept maps with common concepts is also far easier since one must *only* merge common nodes/concepts. All these properties and possibilities lead to our opinion that concept maps will become an important information medium alongside text. Not only side by side, but also a merging of the two mediums is conceivable. For instance, concept maps could not only have nodes with single labels but assigned texts, images and so on. Several fields could greatly benefit from concept maps, not only in the learning context [NG84].

5.1 Future Work

Written or spoken messages are necessarily linear sequences of concepts and propositions.
— Joseph D. Novak, Bob D. Gowin, “Learning How to Learn" [NG84, p. 54]

If I had to reduce all of educational psychology to just one principle, I would say this: The most important single factor influencing learning is what the learner already knows. Ascertain this and teach him accordingly.
— David Paul Ausubel, “Educational Psychology: A Cognitive View" [Aus68]

There are several aspects which have not been covered in this work. New graph kernels and approaches to graph processing appear constantly. These new approaches could also be explored in their usefulness for text-classification. In our work, we capitalized on the Weisfeiler-Lehman graph kernel, not only because its ability to extract explicit feature maps which in turn could be combined with text features, eg. from BoW. We proposed an extension to the WL kernel where we augment the plain WL with node weights. Here, it would be interesting to further evaluate the usefulness of this extension on other datasets and with additional normalization. Another possible extension to WL to reduce the dimensionality of the resulting explicit feature maps would be to only add “new" labels to the feature map. For example, if a graph G has a connected component with only two nodes, n and n', their labels will not change after the first iteration $h = 1$. So for this connected component, WL will have converged and will not provide new information in higher iterations $h > 1$. Therefor adding ignoring the labels of these nodes in higher iterations would decrease the dimensionality of the resulting feature maps. That being said, there are numerous other graph kernels which might prove to be more appropriate for our task than WL. Applying them with their different extensions to better suit the particularities of concept maps could result in the performance improvement which we were not able to achieve with our approach. As we have seen, selecting an appropriate graph kernel is of high importance when doing kernel-based graph processing.

Exploring the differences of concept maps of different datasets could also provide useful insights and improvements for the creation of concept maps. As we saw, the performance and different graph kernels, and their extensions, resulted in often widely varying classification performance on different datasets, despite fact that the concept maps were all generated by the exact same algorithm. Finding correlations between the classification performance and properties of concept maps might give new ideas for more suited graph kernels for the classification task.

In our work, we mainly explored concept maps as a text representation. However, there are a number of other structured representations, not limited to co-occurrence graphs or (semantic) parsing trees. These other graph representations might also capture structural information about the underlying text and could be interesting candidates for further exploration. Here, the idea is also to specially leverage the particularities and structures of these graph types.

As a more far-fetching idea, one could also research how the modifiability and composability of concept maps can be harnessed more effectively in the learning context to enable more personalized learning experiences. For example, when a person wants to learn about some topic using a concept map, he/she most likely has some previous knowledge about that topic. Providing him/her the same information as everyone else might be the traditional way to go. However, providing the person with a more personalized concept map could prove to be useful in removing a lot of information overhead and enhance the learning experience. Such a personalized concept map could, for example, omit information the user has explored before and has previous knowledge about. Or, add connections to other topics the person already knows as an entry point. Also, learning the learning preferences of a user, or giving him options to easily modify/extend a given concept map (semi-) automatically could benefit concept maps becoming a more important information medium. Observations about the behavior of users of such a learning system in turn could give better insight into possible improvements to automatic concept map creation. For all this, the key difference to text as an information medium is that concept maps are easily modifiable and composable. In this idea we envision here, the concept maps could all be interconnected, effectively creating a big knowledge network in which the user can explore given topics in a non-linear way. Here, a new task arises, namely to classify subgraphs of the big concept map which can provide an entry point for a given topic, maybe also under user-provided constraints, eg. the previous knowledge of the user. Another task would entail further summarizing concept maps, effectively changing the level of detail. For example, given three concepts c_1, c_2, and c_3, where c_1 is connected to c_2 and c_2 to c_3, how can we remove node c_2 and add an edge from c_1 to c_3 in a meaningful way. In this example, if the two concept c_1 and c_3 are important concepts and c_2 is less important by some metric, eg. its node degree or some metric of abstractness/detail, this procedure could lower the level of detail of the concept map. This in turn could enable the interactive exploration of concept maps with varying levels of details. So, instead of approaching the topic of concept maps as yet another way to represent text, this proposed idea would really leverage the particularities and advantages of concept maps and graphs in general. Utilizing these aspects in learning contexts might then improve the usefulness of concept maps in other domains.

Bibliography

[And+00] Ion Androutsopoulos et al. “An Evaluation of Naive Bayesian Anti-Spam Filtering”. In: *European Conference on Machine Learning* (2000), pp. 9–17. ISSN: 01635840. DOI: 10.1109/IAW.2007.381951. arXiv: 0006013 [cs]. URL: http://arxiv.org/abs/cs/0006013.

[Aus68] D P Ausubel. *Educational Psychology: A Cognitive View*. 1968, p. 685. ISBN: 0030899516. DOI: 10.1107/S010827019000508X.

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006. ISBN: 0120588307.

[BK05] K.M. Borgwardt and H. Kriegel. “Shortest-Path Kernels on Graphs”. In: *Fifth IEEE International Conference on Data Mining (ICDM’05)* (2005), pp. 74–81. ISSN: 1550-4786. DOI: 10.1109/ICDM.2005.132.

[BM76] J.A. A Bondy and U.S.R. S R Murty. *Graph theory with applications*. Vol. 290. 1976. ISBN: 0444194517. URL: http://people.math.jussieu.fr/%7B~%7Djabondy/books/gtwa/pdf/preface.pdf.

[Bri78] Bruce K. Britton. “Lexical ambiguity of words used in english text”. In: *Behavior Research Methods & Instrumentation* 10.1 (1978), pp. 1–7. ISSN: 1554351X. DOI: 10.3758/BF03205079.

[Cac07] Ana Margarida De Jesus Cardoso Cachopo. “Improving Methods for Single-label Text Categorization”. In: (2007), p. 167. DOI: 10.1002/cplx. URL: http://web.ist.utl.pt/acardoso/.

[Cas14] Richard Eckart de Castilho. “Natural Language Processing: Integration of Automatic and Manual Analysis”. In: (2014), p. 205.

[Cho03] Gobinda G. Chowdhury. “Natural Language Processing”. In: *Annual Review of Applied Linguistics* 37.1 (2003), pp. 51–89. ISSN: 0267-1905. DOI: 10.1017/S0267190500001446. arXiv: 0812.0143v2.

[CV95] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (1995), pp. 273–297. ISSN: 15730565. DOI: 10.1023/A:1022627411411. arXiv: arXiv:1011.1669v3.

[DOL15] Andrew M. Dai, Christopher Olah, and Quoc V. Le. “Document Embedding with Paragraph Vectors”. In: (2015), pp. 1–8. arXiv: 1507.07998. URL: http://arxiv.org/abs/1507.07998.

[Dou11] Brendan L Douglas. “The Weisfeiler-Lehman Method and Graph Isomorphism Testing”. In: *ArXiv* (2011), pp. 1–43. arXiv: arXiv:1101.5211v1.

[FG17] Tobias Falke and Iryna Gurevych. “Concept-Map-Based Multi-Document Summarization using Concept Co-Reference Resolution and Global Importance Optimization”. In: (2017), pp. 1–10.

[Fis25] Ronald Aylmer Fisher. “Statistical Methods for Research Workers”. In: (1925), p. 362. ISSN: 15334406. DOI: 10.1056/NEJMc061160. arXiv: 0-05-002170-2.

[For03] George Forman. “An Extensive Empirical Study of Feature Selection Metrics for Text Classification”. In: *Journal of Machine Learning Research* 3 (2003), pp. 1289–1305. ISSN: 15324435. DOI: 10.1162/153244303322753670.

[Gär03] Thomas Gärtner. “A survey of kernels for structured data”. In: *ACM SIGKDD Explorations Newsletter* 5.1 (2003), p. 49. ISSN: 19310145. DOI: 10.1145/959242.959248. URL: http://portal.acm.org/citation.cfm?doid=959242.959248.

[GB01] Arnulf B a Graf and Silvio Borer. “Normalization in Support Vector Machines”. In: *Neural Computation* 13 (2001), pp. 277–282. DOI: 10.1007/3-540-45404-7_37. URL: http://www.springerlink.com/index/q3068112g9332857.pdf.

[GB15] Chetna Gulrandhe and Chetan Bawankar. “Concept Graph Preserving Semantic Relationship for Biomedical Text Categorization”. In: *International Journal Of Computer Science And Applications* 8.1 (2015), pp. 9–12.

[GGQ11] Miguel Gaspar, T Gonçalves, and Paulo Quaresma. “Text classification using semantic information and graph kernels”. In: *EPIA-11, 15th Portuguese Conference on Artificial Intelligence* (2011). URL: http://www.di.uevora.pt/%7B~%7Dpq/papers/epia2011a.pdf.

[GL14] Yoav Goldberg and Omer Levy. “word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method”. In: 2 (2014), pp. 1–5. ISSN: 0003-6951. DOI: 10.1162/jmlr.2003.3.4-5.951. arXiv: 1402.3722. URL: http://arxiv.org/abs/1402.3722.

[Hal+09] Halko et al. “Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions”. In: (2009).

[Hau65] David Haussler. “Convolution Kernels on Discrete Structures”. In: (1965), pp. 1–25.

[Hea+17] Bradford Heap et al. “Word Vector Enrichment of Low Frequency Words in the Bag-of-Words Model for Short Text Multi-class Classification Problems”. In: (2017). arXiv: 1709.05778. URL: http://arxiv.org/abs/1709.05778.

[HJW15] Linus Hermansson, Fredrik D. Johansson, and Osamu Watanabe. “Generalized shortest path kernel on graphs”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9356 (2015), pp. 78–85. ISSN: 16113349. DOI: 10.1007/978-3-319-24282-8_8. arXiv: 1510.06492.

[HK09] Shohei Hido and Hisashi Kashima. “A linear-time graph kernel”. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2009, pp. 179–188. ISBN: 9780769538952. DOI: 10.1109/ICDM.2009.30.

[HK11] Jan Hauke and Tomasz Kossowski. “Comparison of values of pearson’s and spearman’s correlation coefficients on the same sets of data”. In: *Quaestiones Geographicae* 30.2 (2011), pp. 87–93. ISSN: 0137477X. DOI: 10.2478/v10117-011-0021-1.

[HSS08] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. “Exploring network structure, dynamics, and function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference (SciPy 2008)* SciPy (2008), pp. 11–15. ISSN: 1540-9295.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. “The Elements of Statistical Learning”. In: 18 (2009), p. 764. ISSN: 00111287. DOI: 10.1007/978-0-387-84858-7. arXiv: 1512.00567. URL: http://statweb.stanford.edu/%7B~%7Dtibs/ElemStatLearn/%7B%5C%%7D5Cnhttp://link.springer.com/10.1007/978-0-387-84858-7.

[Jai10] Anil K. Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern Recognition Letters* 31.8 (2010), pp. 651–666. ISSN: 01678655. DOI: 10.1016/j.patrec.2009.09.011. arXiv: 0402594v3 [arXiv:cond-mat]. URL: http://dx.doi.org/10.1016/j.patrec.2009.09.011.

[Joa98] Thorsten Joachims. “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”. In: *Machine Learning* 1398.LS-8 Report 23 (1998), pp. 137–142. ISSN: 03436993. DOI: 10.1007/BFb0026683. URL: http://www.springerlink.com/index/drhq581108850171.pdf.

[Jol02] I T Jolliffe. “Principal Component Analysis, Second Edition”. In: *Encyclopedia of Statistics in Behavioral Science* 30.3 (2002), p. 487. ISSN: 00401706. DOI: 10.2307/1270093. arXiv: arXiv:1011.1669v3. URL: http://onlinelibrary.wiley.com/doi/10.1002/0470013192.bsa501/full.

[Ker+13] Kristian Kersting et al. “Power Iterated Color Refinement”. In: *28th AAAI Conference on Artificial Intelligence* (2013), pp. 1904–1910.

[Ker+16] Kristian Kersting et al. *Benchmark Data Sets for Graph Kernels*. 2016. URL: http://graphkernels.cs.tu-dortmund.de.

[KGK08] Viveka Kulharia, Arnab Ghosh, and Harish Karnick. “Graph Kernels”. In: (2008), pp. 1201–1242. arXiv: 0807.0093. URL: http://arxiv.org/abs/0807.0093.

[KHA00] Ian M. Kinchin, David B. Hay, and Alan Adams. “How a qualitative approach to concept map analysis can be used to aid learning by illustrating patterns of conceptual development”. In: *Educational Research* 42.1 (2000), pp. 43–57. ISSN: 0013-1881. DOI: 10.1080/001318800363908. URL: http://www.tandfonline.com/doi/abs/10.1080/001318800363908.

[KM12] Nils Kriege and Petra Mutzel. “Subgraph Matching Kernels for Attributed Graphs”. In: (2012).

[Koc15] Ned Kock. “One-tailed or two-tailed P values in PLS-SEM ?” In: *International Journal of e-Collaboration* 11.2 (2015), pp. 1–7. ISSN: 1548-3673. DOI: 10.4018/ijec.2015040101. arXiv: /ehis.ebscohost.com/ [http:].

[Kor+08] Jacek Koronacki et al. *Machine Learning*. Vol. 262. 1. 2008, pp. 1–5. ISBN: 978-3-540-79451-6. DOI: 10.1007/978-3-540-79452-3. URL: http://link.springer.com/10.1007/978-3-642-05177-7%7B%5C%%7D5Cnhttp://link.springer.com/10.1007/978-3-540-79452-3.

[Lan] Ken Lang. “NewsWeeder : Learning to Filter Netnews”. In: ().

[LB16] Jey Han Lau and Timothy Baldwin. “An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation”. In: (2016). DOI: 10.18653/v1/W16-1609. arXiv: 1607.05368. URL: http://arxiv.org/abs/1607.05368.

[Liu12] Bing Liu. “Sentiment Analysis and Opinion Mining”. In: May (2012), pp. 1–108. ISSN: 1947-4040. DOI: 10.2200/S00416ED1V01Y201204HLT016. arXiv: 1003.5699.

[Mah+05] P Mahe et al. “Graph kernels for molecular structure-activity relationship analysis with support vector machines”. In: *Journal of Chemical Information and Modeling* 45.4 (2005), pp. 939–951. ISSN: 1549-9596. DOI: 10.1021/ci050039t. URL: papers2://publication/uuid/B9C49825-DCD7-45FA-9EC1-609077C1C090.

[McK10] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference* 1697900.Scipy (2010), pp. 51–56. ISSN: 0440877763224. URL: http://conference.scipy.org/proceedings/scipy2010/mckinney.html.

[Meu+17] Aaron Meurer et al. “SymPy: symbolic computing in Python”. In: *PeerJ Computer Science* 3 (2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://peerj.com/articles/cs-103.

[Mik+13] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: (2013), pp. 1–9. ISSN: 10495258. DOI: 10.1162/jmlr.2003.3.4-5.951. arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[MMS99] Inderjeet Mani, Mark T Maybury, and Mark Sanderson. “Advances in Automatic Text Summarization”. In: *Computational Linguistics* 26.2 (1999), pp. 280–281.

[MS00] Christopher D Manning and Hinrich Schütze. *Foundations of Natural Language Processing*. 2000, p. 678. ISBN: 1420085921. DOI: 10.1162/coli.2000.26.2.277. arXiv: 0306099 [cs].

[NB06] Michel Neuhaus and Horst Bunke. “A Random Walk Kernel Derived from Graph Edit Distance”. In: *Proc. 11th Int. Workshop on Structural and Syntactic Pattern Recognition,* Im (2006), pp. 91–199. ISSN: 16113349. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.1546.

[NC08] J D Novak and J Canas. “The Theory Underlying Concept Maps and How to Construct and Use Them”. In: *IHMC CmapTools* (2008), pp. 1–36. ISSN: 1809-4398. DOI: TechnicalReportIHMCCmapTools2006-01Rev2008-01. URL: http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf.

[NG84] Joseph D. Novak and D. Bob Gowin. *Learning How To Learn*. 1984, p. 199. ISBN: 9781139173469. DOI: https://doi.org/10.1017/CBO9781139173469.

[Nik+17] Giannis Nikolentzos et al. “Shortest-Path Graph Kernels for Document Similarity”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (2017), pp. 1891–1901. URL: https://www.aclweb.org/anthology/D17-1202.

[Nik17] Giannis Nikolentzos. “Matching Node Embeddings for Graph Similarity”. In: *Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017)* (2017), pp. 2429–2435.

[Pag+98] Lawrence Page et al. “The PageRank Citation Ranking: Bringing Order to the Web”. In: *World Wide Web Internet And Web Information Systems* 54.1999-66 (1998), pp. 1–17. ISSN: 1752-0509. DOI: 10.1.1.31.1768. arXiv: 1111.4503v1. URL: http://ilpubs.stanford.edu:8090/422.

[Ped+12] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2012), pp. 2825–2830. ISSN: 15324435. DOI: 10.1007/s13398-014-0173-7.2. arXiv: 1201.0490. URL: http://dl.acm.org/citation.cfm?id=2078195%7B%5C%%7D5Cnhttp://arxiv.org/abs/1201.0490.

[PL04] Bo Pang and Lillian Lee. “A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts”. In: (2004). ISSN: 1554-0669. DOI: 10.3115/1218955.1218990. arXiv: 0409058 [cs]. URL: http://arxiv.org/abs/cs/0409058.

[PSM] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “GloVe : Global Vectors for Word Representation”. In: (). ISSN: 10495258. DOI: 10.3115/v1/D14-1162. arXiv: 1504.06654. URL: https://nlp.stanford.edu/pubs/glove.pdf.

[RKV15] Francois Rousseau, Emmanouil Kiagias, and Michalis Vazirgiannis. “Text Categorization as a Graph Classification Problem”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (2015), pp. 1702–1712. URL: http://www.aclweb.org/anthology/P15-1164.

[RV13] François Rousseau and Michalis Vazirgiannis. “Graph-of-word and TW-IDF: new approach to ad hoc IR”. In: *22nd ACM international conference on Conference on information & knowledge management* (2013), pp. 59–68. ISSN: 1450322638. DOI: 10.1145/2505515.2505671. URL: http://dl.acm.org/citation.cfm?id=2505671.

[SB09] Nino Shervashidze and Karsten M Borgwardt. “Fast Subtree Kernels on Graphs”. In: *23rd Annual Conference on Neural Information Processing Systems* (2009), pp. 1660–1668.

[She+09] Nino Shervashidze et al. “Efficient Graphlet Kernels for Large Graph Comparison”. In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics* 5 (2009), pp. 488–495. ISSN: 15324435. DOI: 10.1.1.165.7842.

[SSM98] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. “Nonlinear Component Analysis as a Kernel Eigenvalue Problem”. In: *Neural Computation* 10.5 (1998), pp. 1299–1319. ISSN: 0899-7667. DOI: 10.1162/089976698300017467. arXiv: arXiv:1011.1669v3. URL: http://www.mitpressjournals.org/doi/10.1162/089976698300017467.

[Val+08] Alejandro Valerio et al. "Automatic classification of concept maps based on a topological taxonomy and its application to studying features of human-built maps". In: (2008).

[Vaz+03] Alexei Vazquez et al. "Global protein function prediction from protein-protein interaction networks". In: *Nature Biotechnology* 21.6 (2003), pp. 697–700. ISSN: 10870156. DOI: 10.1038/nbt825. arXiv: 0306611 [cond-mat].

[VFS12] Daniele Vitale, Paolo Ferragina, and Ugo Scaiella. "Classification of short texts by deploying topical annotations". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7224 LNCS (2012), pp. 376–387. ISSN: 03029743. DOI: 10.1007/978-3-642-28997-2_32.

[Vis+06] S V N Vishwanathan et al. "Fast computation of graph kernels". In: *Advances in Neural Information Processing Systems* 11 (2006), pp. 1449–1456. ISSN: 1793-5091. URL: http://www.ncbi.nlm.nih.gov/pubmed/17992741%7B%5C%%7D5Cnhttp://eprints.pascal-network.org/archive/00003990/.

[VS06] Sudhir Varma and Richard Simon. "Bias in error estimation when using cross-validation for model selection". In: *BMC Bioinformatics* 7 (2006), pp. 1–8. ISSN: 14712105. DOI: 10.1186/1471-2105-7-91.

[Wan+14] Alex Hai Wang et al. "Detecting Spam Bots in Online Social Networking Sites : A Machine Learning Approach". In: (2014), pp. –8. DOI: 10.1007/978-3-642-13739-6.

[WC] Gang Wu and Edward Y Chang. "Formulating Distance Functions via the Kernel Trick". In: 1 (), pp. 703–709.

[WCV11] Stéfan van der Walt, S Chris Colbert, and Gaël Varoquaux. "The NumPy Array: A Struture for Efficient Numerical Computation". In: *Computing in Science {&} Engeneering* 13 (2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37.

[Wea55] Warren Weaver. "Translation". In: *Machine translation of languages* 14 (1955), pp. 15–23. URL: papers2://publication/uuid/C80EF4E1-CF0D-4B77-946F-007B2AADDD84.

[Wei02] Gerhard Weikum. "Foundations of statistical natural language processing". In: *ACM SIGMOD Record* 31.3 (2002), p. 37. ISSN: 01635808. DOI: 10.1145/601858.601867. arXiv: arXiv:1011.1669v3. URL: http://portal.acm.org/citation.cfm?doid=601858.601867.

[YV15] Pinar Yanardag and S.V.N. Vishwanathan. "Deep Graph Kernels". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*. 2015, pp. 1365–1374. ISBN: 9781450336642. DOI: 10.1145/2783258.2783417. URL: http://dl.acm.org/citation.cfm?doid=2783258.2783417.

[YX08] Bo Yu and Zong-ben Xu. "A comparative study for content-based dynamic spam classification using four machine learning algorithms". In: *Knowledge-Based Systems* 21.4 (2008), pp. 355–362. ISSN: 09507051. DOI: 10.1016/j.knosys.2008.01.001. URL: http://linkinghub.elsevier.com/retrieve/pii/S0950705108000026.

Acknowledgements

I would like to acknowledge several people who helped me along the way. First, I'd like to thank Tobias Falke who gave me great support during the thesis with his answers and help structuring my thesis/work in general. Prof. Dr. Kristian Kersting, who provided me important hints and guided my thesis to a more scientific approach. My family, for their support and good words. Patrick Wieth and Marius Vieth, for awakening my interest in ML. Micha Ober, for the server he so graciously borrowed me for several months. Andreas Frankenberger, for providing a place for the borrowed server where it terrorized his employees with its non-bearable fan noises. And last but not least, the *Neural Lourdes* friends for the interesting - and *sometimes* even productive - discussions about ML. Also, thanks to all the contributers to open-source or free software who provide their work free of charge instead of creating pay-walls which arguably hinder progress in science. This work would not be possible in this form without open-source or free software.

A thousand thanks to all of you!