Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor’s Thesis

Automatic Issue Relation Prediction

Tobias Kässmann

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Steffen Becker
Supervisor:	PD Dr. Roman Klinger, Sandro Speth, M.Sc.
Commenced:	March 1, 2021
Completed:	August 23, 2021

Abstract

Context. When developing or maintaining large component-based software systems (for example, microservices, which are getting more and more popular[1]), software developers are usually divided into multiple developer teams working on different components. Additionally, it is not uncommon to include third-party components in one's software. During the development and maintenance of such a system, issues might occur. Those bugs might not harm the component they arose in but might cause failures on neighbouring ones. Traditionally, this is avoided by manually annotating or linking issues across components to indicate their relationships.

Problem. In large projects, this is hardly feasible due to the lack of knowledge a single developer has over the whole system as well as the other issues/components involved.

Objective. This bachelor's thesis tries to predict issue relations using machine learning (ML), which solves the generalisation of the problem. The objective hereby is to create a data set of annotated issue relations, train ML models on it, as well as measure their performance.

Method. ML models have to be trained on an existing data set of annotated issues and their relations. Those samples of issues and relations are obtained from public GitHub repositories by scraping and inferring their relations. Issue relations can be categorised into five distinct categories, namely:

(1) "issue A depends on issue B"

(2) "issue B depends on issue A"

(3) "issue A and B do not stand in a relation"

(4) "issue A and B have a mutual relation"

(5) "issue A is a duplicate of B"

Afterwards, a baseline model will be created to test other models against it and measure their performance.

Conclusion. The conclusion would be to determine if and to what extent the method presented above will lead to good results.

[1]The global microservice architecture market size will increase from 2019 to 2026 at a compound annual growth rate of 21.37% and therefore reach 3.1 billion by 2026 according to the Cloud Microservices Market Research Report prediction in February 2020 (https://www.instanttechnews.com/technology-news/2020/02/16/cloud-microservices-market-2020-trends-market-share-industry-size-opportunities-analysis-and-forecast-by-2026/)

Kurzfassung

Kontext. Bei der Entwicklung oder Wartung großer komponentenbasierter Softwaresysteme (z. B. Microservices, die immer beliebter werden[2]), werden Softwareentwickler in der Regel in mehrere Entwicklerteams aufgeteilt, die an verschiedenen Komponenten arbeiten. Darüber hinaus ist es nicht unüblich, Komponenten von Drittanbietern in die eigene Software einzubinden. Während der Entwicklung und Wartung eines solchen Systems können Probleme auftreten. Diese Fehler betreffen zwar nicht die Komponente, in der sie entstanden sind, können aber bei benachbarten Komponenten zu Bugs führen. Traditionell wird dies durch manuelle Anmerkungen oder Verknüpfungen von Bugs zwischen den Komponenten vermieden, um ihre Beziehungen zu verdeutlichen.

Problem. Bei großen Projekten ist dies aufgrund des mangelnden Wissens eines einzelnen Entwicklers über das gesamte System und die anderen beteiligten Issues oder Komponenten kaum machbar.

Zielsetzung. In dieser Bachelorarbeit wird versucht, Issue Relations mit Hilfe von maschinellem Lernen vorherzusagen, wodurch die Generalisierung des Problems gelöst wird. Das Ziel besteht darin, einen Datensatz mit annotierten Problembeziehungen zu erstellen, ML-Modelle darauf zu trainieren und ihre Leistung zu messen.

Methodik. Die ML-Modelle müssen auf einem bestehenden Datensatz von annotierten Issues und deren Beziehungen trainiert werden. Diese Stichproben von Issues und Beziehungen werden aus öffentlichen GitHub-Repositories durch Scraping und durch die Ableitung ihrer Beziehungen gewonnen. Issue-Beziehungen können in fünf verschiedene Kategorien eingeteilt werden, nämlich:

(1) “Issue A hängt von Issue B ab”

(2) “Issue B hängt von Issue A ab”

(3) “Issue A und B stehen nicht in einer Beziehung”

(4) “Issue A und B stehen in einer gegenseitigen Beziehung”

(5) “Issue A ist ein Duplikat von B”

Anschließend wird ein Basismodell erstellt, um andere Modelle daran zu testen und deren Leistung zu messen.

Schlussfolgerung. Abschließend wäre zu prüfen, ob und inwieweit die oben vorgestellte Methode zu guten Ergebnissen führt.

[2]Die globale Marktgröße für Microservice-Architekturen wird von 2019 bis 2026 mit einer durchschnittlichen jährlichen Wachstumsrate von 21, 37% wachsen und damit bis 2026 3,1 Milliarden erreichen, so die Prognose des Cloud Microservices Market Research Report im Februar 2020 https://www.instanttechnews.com/technology-news/2020/02/16/cloud-microservices-market-2020-trends-market-share-industry-size-opportunities-analysis-and-forecast-by-2026/

Contents

List of Figures

List of Tables

List of Listings

Acronyms

BOW	Bag of words. 8
CBOW	Continuous Bag-of-Words. 9
fastText	fastText. 9
FN	false negative. 13
FP	false positive. 13
GloVe	Global Vectors for Word Representation. 9
IMS	Issue management systems. 2
IRG	Issue Relation Graph. ix
LDA	Latent Dirichlet Allocation. 12
ML	machine learning. iii
SIF	Smoothed Inverse Frequency. 9
Skip-Gram	Continuous Skip-gram. 9
tf-idf	term frequency–inverse document frequency. 8
TN	true negative. 13
TP	true positive. 13
uSIF	Unsupervised Smoothed Inverse Frequency. 9
Word2Vec	Word2Vec. 9

1 Introduction

Component-based architectures are steadily gaining popularity due to the rise of cloud applications[1] and the increasing popularity of microservice architectures as instances of service-oriented architectures (SOA). However, they come with their own set of challenges when developing them in the large. Usually, the components are developed by different developer teams. So, when a bug occurs in one of the architecture's components, this bug might lead to other bugs or unexpected behaviour in neighbouring (dependent) ones. In small component-based architectures, managed by a small set of developers, this might dependent on the buggy microservice not be a huge deal since every developer might know which components consume other ones and, therefore, they can find easily out if the bug originated in another service and react accordingly. However, in a larger architecture maintained by multiple teams spanning multiple organisations, this provides a whole new set of challenges. A single programmer or eventually the whole developer team usually does not know which other components have bugs, that might influence their service. Therefore, the team maintaining the affected service might have to invest vast amounts of resources in finding the root cause of their bug.

A simple example of such a pitfall is the following scenario described in Figure 1.1: A modification of the interface of one service (the "location/map" service) may adversely affect the functionality of other services dependent on that interface (i.e., "shopping cart" service) if the modification introduces new bugs or regressions and, thus, leading to errors to the affected services. Due to the complex architecture, the development team of the affected service might spend a lot of time finding the bugs' source(s), which leads to higher maintaining costs and possibly a longer downtime.

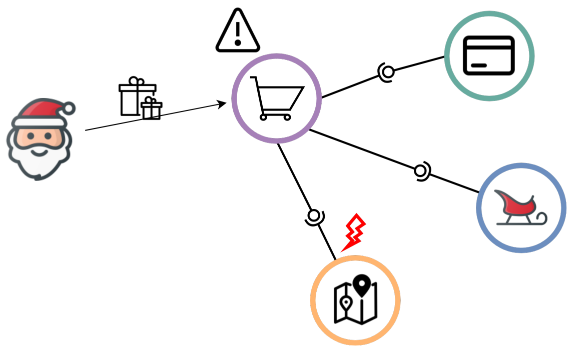


Figure 1.1: Motivation example of a component-based architecture.

[1]https://www.instanttechnews.com/technology-news/2020/02/16/cloud-microservices-market-2020-trends-market-share-industry-size-opportunities-analysis-and-forecast-by-2026/

1.1 State-of-the-Art

The state-of-the-art of annotating issue relations is through manual labour, like using the issue dependency model provided by Jira [20a]. Another possibility is by linking issues (GitHub [20b], Redmine [18]) using their ID and a description of the relationship in the issue's body. However, this method assumes that the person annotating issues understands the system and knows the current bugs. That is often not the case, especially if one included other peoples' code in the form of libraries or by developing software by independent teams, which is common with service-oriented architectures.

Another way of helping the users on solving those issues was provided by Speth et al. [SBB20] through the work of Gropius. Gropius provides the possibility of managing issues from other Issue management systems (IMS) such as GitHub, Jira, Redmine etc. in one system, hence a person annotating issues is able to see the issues of the other teams, even if they are using a different issue management system. Furthermore, Gropius allows it to annotate cross-component issues [SBB21], which are issues that affect multiple components and is, therefore, the first of its kind. Nevertheless, it is still not capable of automatically detecting those issue relations and, therefore, still relying on a manual annotation by the developers.

1.2 Open Challenges

The traditional way of manually annotating or linking issues to others from other components to emphasise their relationship is no longer feasible. Also, this problem might not be an SOA problem - using an automatic algorithm to find bug relations might also help finding faulty libraries during development or even issue relations on the same component.

This bachelor's thesis aims to automate those processes, hence developing a way of finding and annotating related issues by using the information provided by the issues' title, body texts and comments. This leads to the research question:

RQ

"How can relationships between issues that possibly target independently managed software projects be detected?"

This proposal is going to tackle this problem by automatically categorising issue relations using machine learning into the following five categories “A depends on B ($\leftsquigarrow$), B depends on A ($\rightsquigarrow$), A and B do not have a relation ($\nleftrightsquigarrow$), A and B have a mutual relation ($\leftrightsquigarrow$), A and B are duplicates (duplicate)”, where A and B are the issues provided.

Automatically categorising and detecting issue relations might ease the work done by many developers with less misspend time and energy.
Our contributions are:

- A scraped and manually annotated dataset for all four classes consisting out of over $2k$ relations, validated by two annotators.
- Nine different classifier models, tested on different vectorisation strategies and their comparison.
- A Docker-compose and Kubernetes (K8s) microservice architecture utilising our best model for ease of use.

Thesis Structure

This is a brief overview over the structure.

Chapter 2 – Foundations and Related Work: We provide an overview about the foundations as well as the previous research conducted in this and related area(s).

Chapter 3 – Data The data (training, validation and testing), corpus generation as well as its composition are focused on.

Chapter 4 – Classifier: We define the models used in this thesis.

Chapter 5 – Implementation as a Microservice We describe our implementation of the process, and the architecture utilised in the Docker-compose and Kubernetes architectures.

Chapter 6 – Evaluation The results of the different classifiers are presented, analysed and discussed.

Chapter 7 – Conclusion We conclude our thesis with an outlook as well as by presenting our final results.

2 Foundations and Related Work

2.1 Issues and Management Systems

In the scope of our work, issues are pieces of information for documenting characteristics of components, the source code, or the project in general. A characteristic can be anything like a bug, feature, error, enhancement, or change request. In the case of bugs, they document crucial information for the developers such as a detailed description of the failure [BJS+08]. Issues can be parts of other issues or related to other ones. To organise them and keep track of them, they usually are managed using IMS such as Jira [20a], Redmine [18], GitHub [20b] and Gropius [SBB20]. IMS also provide the possibility to add, edit, and resolve issues. In addition, they often offer the user a possibility to annotate or tag related issues even though their implementation differs. Jira, for example, provides the user with the choice of annotating whether an issue is connected to another one within this projects (and within its environment). Redmine implements this by letting the user annotate related tasks. In GitHub, users can "link" issues over the comments or the issue description together using GitHub mentions. Gropius provides all of the features above by combining those issues into one place. Moreover Gropius is able to link issues of different components together, which are lying on different issue management system providers. Further, IMS can be connected to repository systems or even be an instance of one.

2.2 Component-based Architecture

A Component-based software architecture consists of multiple independent and loosely coupled components. The components, their relationships, and their interactions define the architecture of a so-called component-based architecture. Reussner et al. define a (software) component as a "*contractually specified building block for software*" [RBH+16]. We are using the terms microservice architecture and component-based architecture interchangeably because components can be deployed as services. Therefore, they do not differ from an architectural point of view [FL15]. Also, components/services can be combined to form composite services/components [SGM02]. So, a microservice architecture functionally decomposes an application into small (modular) services, which communicate over API calls [New21; Ric17].

2.3 Text Classification

Text classification is the task of automatically assigning one or more classes C to a given document D by learning a function f (2.1) [Joa+99; Man08], as depicted in Equation (2.1).

$$f : D \to C \tag{2.1}$$

The algorithm learns f by adjusting its weights using previous examples. This procedure is called supervised learning because the algorithm sees the "solutions" after making a guess [CCD08].

The task of assigning one or more classes to a document can be divided further into the problems binary-, multiclass-, and multilabel- classification as stated by Hossin et al. [HS15]. Binary classification stands for the task of assigning the document one of two classes, multiclass classification classifies a document into one of three or more classes, and multilabel classification is the task of assigning n of m classes (or also called labels) to a given document.

The task of text classification is usually split into multiple parts, Minonczuk et al. [MP18] breaks it down into the following five steps:

1. Data acquisition to create an unlabelled data set
2. Data analysis and labelling to create a labelled data set from the unlabelled data set.
3. Feature construction and weighting of said features to create data representations from a labelled data set.
4. Feature projection, selection, and training of a classification model to create a classifier from data representations.
5. Solution evaluation of the classifiers.

After the acquisition step, it is common practice to divide the data randomly into three distinct sets: a training-, validation- and test-set. Splitting the data like this ensures that the model's bias towards the training data can be kept as low as possible.

The training data is, as the name suggests, used for the model training. Using the examples in the training data, the model adjusts its weights accordingly - also called fitting.

After the model was fit using the training data, the validation data can estimate the prediction error for the model selection. Moreover, the validation data can evaluate the model's accuracy after changing its hyperparameters and, therefore, simulate the models' performance on previously unseen data.

The test set is the final data set the model gets to see. On the test set, the generalisation of the final model gets tested. Ideally, the test set should only be used at the end of the data analysis to hinder bias towards the best test set performance [HTF09].

2.3.1 Creating a Labelled Data Set

In order to obtain an annotated data set, the labels need to be constructed by one of several methods like crowdsourcing using Amazon's Mechanical Turk[1], CrowdFlower[2] or using an expert annotation as described by Klinger et al. [Kli+18].

[1]https://www.mturk.com/2

[2]https://www.crowdflower.com/

Landis and Koch	
< 0	no agreement
$0 - 0.20$	slight agreement
$0.21 - 0.40$	fair agreement
$0.41 - 0.60$	moderate agreement
$0.61 - 0.80$	substantial agreement
$0.81 - 1$	almost perfect agreement

Fleiss	
< 0.40	poor agreement
$0.40 - 0.60$	fair agreement
$0.60 - 0.75$	good agreement
> 0.75	excellent agreement

Table 2.1: Cohen's kappa guidelines.

For this work, we used expert annotation. For the expert annotation, several experts need to be asked to annotate the data. However, this will almost unavoidably lead to different results because humans are not able to consistently report a gold standard of relevance [Man08]. Therefore, to validate the results, the expert agreement has to be measured to conclude whether or not the results can be considered a gold standard. We used Cohen's kappa coefficient κ [Coh60], a common measure designed to judge the agreement rate corrected by the chance of a chance agreement for categorical judgements [Man08] for the measurement even though there are several other methods to measure the agreement, like Krippendorff's alpha or Fleiss' kappa.

The kappa κ (2.2), itself is calculated using p_0 (2.3) the agreement percentage, and p_e (2.4), the probability of an agreement by chance.

$$\kappa = \frac{p_0 - p_e}{1 - p_e} \tag{2.2}$$

$$p_0 = \frac{1}{N} \sum_{c \in C} h_{ii} \tag{2.3}$$

$$p_e = \sum_{c \in C} p(c|a_1)p(c|a_2) \tag{2.4}$$

The interpretation of the kappa value has no theoretical foundation, and there are no universally accepted guidelines on this topic because they all were chosen arbitrarily by their creators. Here the guidelines from Landis and Koch [LK77] and guidelines created by Fleiss, which have been stated by Wentura et al. [GW97], are tabulated to illustrate the general agreement.

2.3.2 Feature Construction and Weighting

Before the data can be used to train the classifier, it has to be first converted into a suitable form because it consists primarily out of strings of characters [Joa+99].

A popular method to transform documents into machine-readable representations is the so-called Bag of words (BOW). BOW represents a document d (2.5) as a vector on basis of the tokens[3] of a vocabulary V appearing in it.

$$d = (f_{t_0,d}, ..., f_{t_{|V|},d}); \qquad V \subseteq \{t \in d \mid d \in D\} \tag{2.5}$$

$f_{t,d}$ indicates how often the term t appears in the document d. As shown in the formula, each document vector has the length $n = |V|$. This results in mostly large sparse vectors because it is very unlikely that most of the tokens will occur in a given document. To reduce the length of those vectors, Joachims [Joa98] considers a token a feature (and thus part of the vocabulary) if it occurs more than three times in the data set and if it is not a stop word such as “the” or “and”. Also, the words are often converted to lower case to reduce unnecessary entries in the vocabulary [HM21]. Moreover, there are other methods such as stemming [Lov68] or lemmatisation [MWC19] to further reduce the words in the vocabulary, into which details we will not go here. In addition to that, the words occurring in the BOW vector representation can even be weighted with measures such as the binary representation[4] or measures such as the term frequency–inverse document frequency (tf-idf) score to encode the importance of a word [Joa+99; Ull11]. tf-idf is defined as the product of the term frequency tf (2.6) and the inverse document frequency idf (2.7).

$$tf = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.6}$$

$$idf = log(\frac{|D|}{|\{d \in D \mid t \in d\}|}) \tag{2.7}$$

It measures the importance of a word by weighting the number of its occurrences in the document (tf) and the inverse frequency of how often it occurs in all documents (idf).

However, all the extensions mentioned still only capture the presence or absence in a given corpus. For example, sentences like “A implies B” and “B implies A” both would have the same document representation even though the meaning is different. Therefore it is still a basic “bag of words”. Nevertheless, Joachims et al. [Joa98] stated that losing the ordering of the words is of minor importance.

Even so, if one wants to get some of the contexts back, one can extend the BOW model by using N-grams. The “N” in N-grams describes how many neighbouring words are packed together in one tuple[5]. Single words can be regarded as unigrams, and pairs of words as bigrams. A bigram for “A implies B” could look like “A implies, implies B”. They can also be mixed by appending bigrams onto unigrams “Uni+Bigrams”, bigrams onto trigrams “Bi+Trigrams” or uni- onto, bi- and trigrams “Uni+Bi+Trigrams”. However, this unavoidably leads to larger document representations [HM21].

A problem of this representation is that the words themselves are reduced to a single number without any meaning. Embeddings solve that issue by grouping words that are similar in context together. Therefore, the meaning of a word is kept, and similar words are forming clusters in

[3]A token can be a word, emoji, or number.

[4]If the token does not appear in the document, the binary representation corresponds to a value of zero, otherwise it takes on a non-zero positive value.

[5]Instead of using token N-grams, one can also use N-grams at the character level [CT+94]

the embedding space. Moreover, embedding representations are denser than word occurrence representations. There are many methods and algorithms to produce embeddings like Word2Vec (Word2Vec) by Milkov et al. [MCCD13], Global Vectors for Word Representation (GloVe) by Pennington et al. [PSM14], fastText (fastText) by Bojanowski et al. [BGJM17] and even transformer architectures like BERT by Devlin et al. [DCLT19].

Two different model-architectures to create the word representations were introduced by Milkov et al. in their publication [MCCD13]. The first model was the Continuous Bag-of-Words (CBOW) model, which predicts a word based on the context words. The context consists of a few words surrounding the current (middle) word. This model is called BOW model because the order of the words in the context is not important.

The second model is the Continuous Skip-gram (Skip-Gram) model, which predicts words within a certain range around the current word in the same sentence. The quality of the Skip-Gram model was further improved by Mircov et al. in their second publication, in which they also proposed a method of creating word vectors out of phrases [MSC+13].

GloVe [PSM14] created by Pennington et al. is an unsupervised learning algorithm for obtaining vector representations for words. They showed that GloVe is able to outperform Word2Vec while having just half the amount of training data available. GloVe itself is trained on the non-zero entries of a global word-word co-occurrence matrix, which lists how frequently words co-occur with one another. Using this matrix, GloVe creates token embeddings.

FastText extends the Skip-Gram model of Word2Vec. By generating character-level N-grams (3- to 6-grams) of the centre word (in angle brackets like $\langle cat \rangle$ for cat), which then get hashed (the hash function uses integers between $[1; 2, 10^6]$) and averaged. The context words are directly taken from the embedding table without using the N-grams and they also used for each positive example five negatives at random with a probability to the square root of the uni-gram frequency [BGJM17]. Using this, it is able to surpass models without morphological analysis and it is able to deal with out of vocabulary words due to their sub-word N-gram averaging method.

The word embeddings of any of those methods can then be averaged or tf-idf weighed to generalise for documents and sentences [BMCG15; CMS17]. Alternatively, they can be weighted using different measures such as the Smoothed Inverse Frequency (SIF) [ALM16] or even Unsupervised Smoothed Inverse Frequency (uSIF) [Eth18]. SIF embeddings [ALM16] are computed by the weighted average of the smoothed inverse frequencies $\tilde{c_s}$ and applying the common component removal on it, to retrieve the discourse vector of the document or sentence c_s. The weighted average is computed by $\tilde{c_s}$ (2.8), where s denotes the sentence or document, $p(w)$ the word frequency, $\overrightarrow{w}$ the word vector of the embeddings and a is a scalar, which needs tuning. The common component removal is computed by c_s (2.9).

$$\tilde{c_s} = \frac{1}{|s|} \sum_{w \in s} \frac{a}{p(w) + a} \cdot \overrightarrow{w} \tag{2.8}$$

$$c_s = \tilde{c_s} - proj_{c_0} \tilde{c_s} \tag{2.9}$$

SIF embeddings are not completely unsupervised because of the scalar α. Additionally, vector lengths can have a confounding effect on the SIF embedding [Eth18]. uSIF solves those problems by using an angular distance based walk model instead of the log-linear one. The embeddings are retrieved by using the unsupervised smoothed inverse frequency $\tilde{c_s}$ as shown in 2.10

$$\tilde{c_s} = \frac{1}{|s|} \sum_{w \in s} \frac{a}{p(w) + \frac{1}{2}a} \cdot \overrightarrow{w} \tag{2.10}$$

$$c_s = \tilde{c_s} - \sum_{i=1}^{m} \lambda_i proj_{c'_i} \tag{2.11}$$

and the partial common component removal cs (2.11).

λ_i are weights on the common discourse vectors [Eth18]. uSIF can estimate a by using $p(w)$, the vocabulary size and the average sentence length.

Other, often better, approaches to generate sentence embeddings are Doc2Vec by Quoc et al. [LM14], Sentence-BERT by Reimers et al. [RG19], InferSent by Conneau et al. [CKS+17], or the universal sentence encoder by Cer et al. [CYK+18].

Doc2Vec is based on Word2Vec, and there are again two algorithms, the PV-DM (Distributed Memory version of Paragraph Vector), which is basically the same procedure as the CBOW except for the fact that it concatenates a paragraph vector onto the averages of the word vectors before passing it into the output layer.

Furthermore, a PV-DOBW (Distributed Bag of Words version of Paragraph Vector) model was presented, which is similar to the “skip-gram” version. This version takes in a document ID and tries to return randomly sampled words from the document.

The SentenceBERT is a siamese network based on BERT/RoBERTa based architectures, from which the obtained embeddings are passed through a pooling layer to derive a fixed-sized sentence embedding. They evaluated their performance for Semantic Textual Similarity tasks using the cosine similarity for their embeddings.

InferSent is another model trained on the SNLI data set, They trained their sentence embeddings on 12 different transfer tasks and showed that supervised learning is better than unsupervised for generating embeddings.

Furthermore, the universal sentence encoder from Cer et al. also consists of two different architectures with a different design challenge. The first architecture is based on a transformer [VSP+17] and build for maximum accuracy at the cost of having a high complexity, while the second architecture is based on a deep averaging network (DAN) (Iyyer et al. [IMBD15]) and build targeting inference. However, it lacks some accuracy. For the DAN architecture, the input embeddings for words and bigrams are averaged together and then fed into a feedforward deep neural network (fully connected layers with a dropout before passing them into a softmax function) to produce sentence embeddings. An advantage of the DAN over the transformer is its asymptotic complexity, which lies in $O(n)$ for the sentence length compared to the transformer architecture of $O(n^2)$.

Because embedding and BOW vectors with a similar meaning are grouped closely together, they can be measured using methods such as the cosine similarity, which is the standard way of quantifying the similarity between two documents [Man08; MRS08]. The cosine similarity uses the angle between two vectors (document representations) and ignores the length of the vector to create a distance metric. That leads to the advantage that longer documents are not negatively biased. The cosine similarity gets calculated by Equation (2.12):

$$cos(\theta) = \frac{A \cdot B}{||A|| ||B||} \tag{2.12}$$

Because the vector representation of the texts is always positive, the cosine value is consequently also always positive and lies in the interval $[0, 1]$.

2.3.3 Feature Projection, Selection and Training of a Classification Model

Several models for text classification include k-nearest neighbours, a random forest classifier, support-vector machines, multinomial naive Bayes or neural networks. Neural networks are designed to roughly mimic how the human brain works, i.e., as a collection of interconnected neurons. A neuron itself is defined as the function (2.13).

$$y = f\left(\sum_i w_i x_i\right) \tag{2.13}$$

on a vector input x [Kri05]. The weights are adjusted by learning, and f is the activation function. Some common activation functions are the $reLU$ (2.14) and $softmax$ (2.15) functions.

$$reLU(x) = max(0, x) \tag{2.14}$$

$$softmax(x) = \frac{exp(x)}{\sum_n exp(x_n)} \tag{2.15}$$

Normally, for the output function, a softmax function is used as an activation function to create unary vectors out of the inputs. For the hidden units (units within the hidden layers of the network), $reLUs$ are often used because their advantage lies in learning high numbers. The network itself consists of multiple neurons that are connected to each other[6].

Another neural network structure is the siamese network (illustrated in Figure 2.1). This network works with input pairs. For both elements of all pairs, the same operations are conducted before they are combined into a single element by a distance-/merging-/loss- layer.

Siamese networks are a type of architecture which lead to groundbreaking results in tasks like image comparison or duplicate comment matching [IUA+20] due to their ability to learn similarity measures. Moreover, they have desirable properties such as symmetry, which helps them not be distracted by the arrangement of the inputs. Siamese networks were first introduced by Bromly et al., who used them in their publication [BGL+93] for comparing signatures.

[6]A network is dense if each neuron is connected to its previous neuron(s).

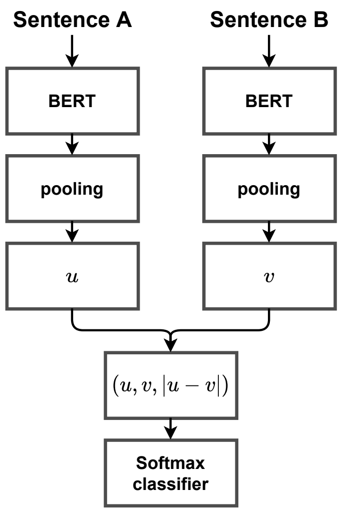


Figure 2.1: Example Siamese Network architecture modelled after Sentence-BERT from Reimers et al. [RG19].

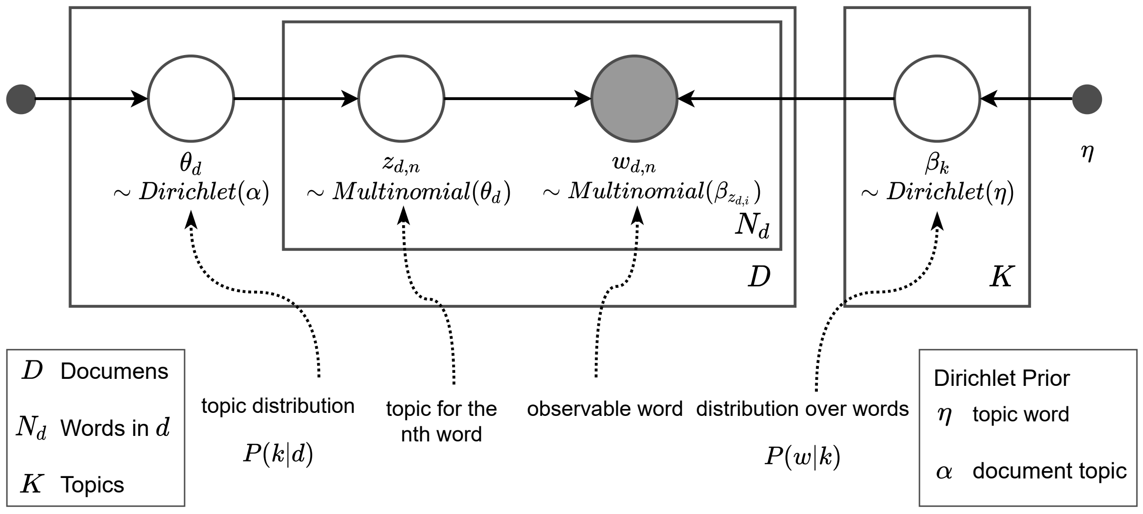


Figure 2.2: Graphical model of LDA after Hoffman [HBWP13].

2.3.4 Topic Modelling

For retrieving topic distributions of documents, we use Latent Dirichlet Allocation (LDA), which is a generative model for discovering abstract topics from a collection of documents.

Generative means that instead of discriminating or classifying and thus providing a probability of the target y given an observation x $P(y|X = x)$, it provides the prediction of an observable X given the target y $P(X|Y = y)$ and therefore generates the outcome.

LDA assumes that each document exhibits K topics with different proportions, which it needs to determine.

Listing 2.1 LDA Generative process

```
Draw topics $\beta_k \sim Dirichlet(\eta)$ for $k \in K$;
For each $d \in D$:
  Draw topic proportions $\theta \sim Dirichlet(\alpha)$;
    For each $w \in N$:
      Draw topic assignment $z_{dn} \sim Multinomial(\theta_d)$;
      Draw (observed) word $w_{dn} \sim Multinomial(\beta_{z dn})$;
```

Figure 2.2 illustrates the LDA as a graphical three-level generative model. As shown in the graph, each node is a random variable and is used for the generating process. The greyed-out node symbolises that it is observable. The other nodes are latent (hidden) variables. The LDA uses the observed words to infer the hidden abstract topic structure by adjusting the weights of α and θ. For values $x \gg 1$ the values generated by the $Dirichlet(x)$ are likely to contain a mixture of most (topics/words), the values for $x \ll 1$ generated by it are likely to contain a mixture of only a few (topics/words), and for $a == 1$ it is evenly distributed.

Using the generative process of the model described by Listing 2.1 one is able to retrieve the matrices $\Theta_{k\times d} = p(k|d)$ and $\Phi_{w\times k} = p(w|k)$.

For these, the following holds: $p(w|d) \approx \Theta \times \Phi$ because $p(w|d) = \sum_{t\in T} p(w|k,d)p(k|d) \overset{I}{\Longrightarrow} \sum_{t\in T} p(w|k)p(k|d)$. I stands for “conditional independence”. However, first acquired topics are probably not perfect. In order to retrieve better topics, the priors α and η have to be adjusted. There are many methods to approximate them, such as Markov chain Monte Carlo methods, expectation propagation and variational inference [HBWP13; Tom18].

2.4 Evaluation Metrics

Evaluation metrics are essential to evaluating and comparing models and their performances. The most common metrics are precision, recall, and the $f1$ score.

Those measurements are based on the properties of predicted instances, the true positive (TP), true negative (TN), false positive (FP), and false negative (FN), which distinguish right from wrong predictions from the view of a single class.

From this view, true predictions can be of one of two kinds. Instances that are correctly classified to be part of this class (TP) as well as instances correctly classified not to be a part of this class (TN). For wrong predictions, there are also two kinds of errors. These are instances predicted to be in the class, which are no a part of the class (FP), and instances predicted to be not a part of the class even though they are a part of it (FN), as explained by Hossin et al. [HS15].

Distinguishing those cases leads to the ability to tell how much *precision* a classifier has for a class by using the formula (2.16).

$$P = \frac{TP}{TP + FP} \tag{2.16}$$

Moreover, one can also calculate the *recall* of one class, which measures how many issues from this class are predicted to be in this class using the formula (2.17)

$$R = \frac{TP}{TP + FN} \tag{2.17}$$

Both precision and recall are measuring how good a given prediction is. Furthermore, they provide valuable insights into what the strengths and weaknesses of a prediction are. The F-measure (2.18) combines those two measurements to generate a score that weights the importance of both measurements to provide a unified measurement unit in the interval $[0, 1]$.

$$F = (1 + \beta^2)\frac{P * R}{\beta^2 * P + R} \tag{2.18}$$

β is a constant which can be chosen to weigh the recall, which is β times as important as the precision. A common approach is to use $\beta = 1$, which is the harmonic mean between precision and recall and results in the $F1$ (henceforth $f1$) score. The F-score can also be enhanced to measure multiple classes by using macro or micro averaging. The macro $f1$[7] is defined as the averaged $f1$ score of all the $f1$ scores of the classes (2.19).

$$macro\ f1 = \frac{\sum_{i=0}^{N} f1_i}{N} \tag{2.19}$$

The micro $f1$ is calculated by aggregating the TP, FP, and FN for each class and using those values to calculate P, R, $f1$. However, the micro $f1$ has the undesirable property of washing out less occurring labels as stated by Lipton et al. [LEN14].

2.5 Related work

This section focuses on previous work and their accomplishments. It is split into two parts, the first part presents work in the field of detecting duplicate issues, and the second part presents work in the field of language inference. For each field, we present several bodies of research and compare them. In addition, we also describe briefly why they are not able to solve our objective.

2.5.1 Literature Research Methodology

We used Google Scholar for finding publications about this research topic to see, what research already has been conducted because of the wide variety of papers it provides. In addition to that we also used Google, because it yields more promising search results than DuckDuckGo from our experience.

Research keywords: issue relation prediction, automatic issue relation prediction, duplicate issue detection, issue deduplication, siamese networks, topic modelling, topic prediction, scikit-learn, tensorflow, nlp, embeddings, entailments, document embeddings, nli, natural language inference, multiclass classification, keras, kubernetes microservice, autotuning, duplicate detection

[7] There are two variants of the $f1$ score, here we used the more robust variant, the "averaged F1" according to Opitz et al. [OB19].

2.5.2 Duplicate Issue Detection

There has been much research in the area of duplicate issue detection (deduplication). Here, we introduce and compare some of the works like the works of Nguyen et al. [NNN+12], Alipour et al. [AHS13], Klein et al. [KCK14] as well as Hindle et al. [HAS16]. Nguyen et al. [NNN+12] used a topic model-based approach using LDA, and combined it with information retrieval as well as categorical features to identify duplicate bugs.

Alipour et al. [AHS13] also used this technique. However, they also improved it by using contextual information based on their prior knowledge about software quality, software architecture, and system-development Latent Dirichlet Allocation topics to improve the bug deduplication.

Klein et al. [KCK14] further refined the technique by introducing metrics based on the topic distribution of the issue reports, relying only on data taken directly from bug reports, like using the time difference or the difference in the number of words. Moreover, they introduced a novel metric that measures the first shared topic between two topic-document distributions.

Hindle et al. [HAS16] used the ideas from their previous work Alipour et al. [AHS13] by evaluating their approach in the context of the candidate-duplicate ranking, instead of the duplicate-or-not decision used previously. Candidate-duplicates is a list of duplicates sorted by their probability of being a duplicate. Moreover, instead of just testing their model on the Android repository, they also tested their model on the Eclipse, Mozilla, and OpenOffice software systems repositories to allow a more direct comparison to other previous works.

All those works made achievements on the task of deduplication. However, our objective is not to focused on deduplication alone but also on predicting the relations between the issues of which duplication plays a part in. Also, our focus is a more general model for different repositories. None of the previous works is able to satisfy this objective of predicting whether or not an issue resulted in another one, or if they are related.

2.5.3 Inference

Another related research field is that of Natural language Inference (also Recognising Textual Entailments), which is one of the problems in natural language understanding [DDMR10]. This research field aims to determine the inference relation between two texts fragments (a text and a hypothesis). The relation can be one of three kinds: "Entailment" meaning the hypothesis can be deduced from the text, "Contradiction" the text can deduce a contradiction to the hypothesis, or "Neutral" meaning the hypothesis is neither true nor false.

There have also been many publications in this research area like the ones of Parikh et al. [PTDU16], or Chen et al. [CZL+17].

Parikh et al. presented an attention-based approach for natural language inference [PTDU16]. For their data, they used the SNLI corpus of Bowman et al. [BAPM15] consisting of 570K human-written English sentence pairs. They used attention to split their problem into smaller sub-problems that can be solved in parallel. Moreover, their model achieved a state-of-the-art accuracy without relying on word-order information. Besides, they also tested adding a minimum amount of order into account by using intra-sentence attention to improve their model further.

Chen et al. [CZL+17] demonstrated that carefully designed sequential inference models based on LSTMs were able to outperform all previous models. Moreover, incorporating syntactic parsing information resulted in their best result. For this matter, they presented two different architectures, a sequential model named ESIM that outperforms all previous models and a better, more sophisticated network incorporating parsing information using tree LSTM's.

Like Parikh et al., they used the SNLI corpus of Bowman et al. [BAPM15]. Moreover, they tested using some of the findings of Parikh et al. in their work to improve their models. They used a concatenation of average and max pooling before passing it into the classifier, which stands in contrast to Parikh et al. work that utilises summation. They considered that summation could be sensitive to the sequence length, which makes it less robust. Nevertheless, they also tested summation but showed that pooling instead of summation leads to significantly better results. In addition to that, their base model (ESIM) outperformed the work of Parikh et al., their vanilla decomposable attention model, and their approach using intra-sentence attention.

3 Data

This chapter focuses on the gathering and generation of the data. Moreover, it also focuses on the statistics of the data and analyses the relations the issues are standing in. It is split into four sections. The first section is about gathering the data, the second about generating unrelated issues, the third about generalising the corpus and presenting the relation distribution, and the last one presents the statistics regarding the corpus and its relations.

3.1 Data Gathering

Before a classifier can be trained on data, the data has to be first gathered. First, a formal description is defined, which is used throughout the thesis. Afterwards, the methods for the selection process of the data are presented, followed by information about the participants' inquiries, which was used to ensure the validity of the annotations to create a central label of reference. Lastly, we present some insights and implementation details about the data collected. Figure 3.1 illustrates the process of the data gathering from collecting the issues from GitHub to validating the annotations. The steps of this picture describe the process described in the subsections (the numbers at the bottom of each element)

3.1.1 Formal Description

Let I be the set of all issues, then a relation $R \in C$ between two issues is defined as $a\ R\ b := (a, b)_R$ for $a, b \in I$. A relation can be one of the following five classes $\{\rightsquigarrow;\ \leftsquigarrow;\ \leftrightsquigarrow;\ \not\leftrightsquigarrow;\ \text{duplicate}\}$. The five classes identified are:

1. "Issue B depends on A" := $A \rightsquigarrow B$
2. "Issue A depends on B" := $A \leftsquigarrow B$
3. "Issue A and issue B do not have a relation" := $A \not\leftrightsquigarrow B$
4. "Issue A and issue B have a mutual relation[1]" := $A \leftrightsquigarrow B$
5. "Issue A is a duplicate of issue B" := $A\ dup\ B$

Therefore, the whole scraped document relations DR are defined as $\{(a,\ b,\ c) \mid a, b \in I,\ c \in C\}$.

[1]It can mean, that an issue C is causing the other two hence from an unknown source or that the direction was just unclear to the persons commenting in GitHub

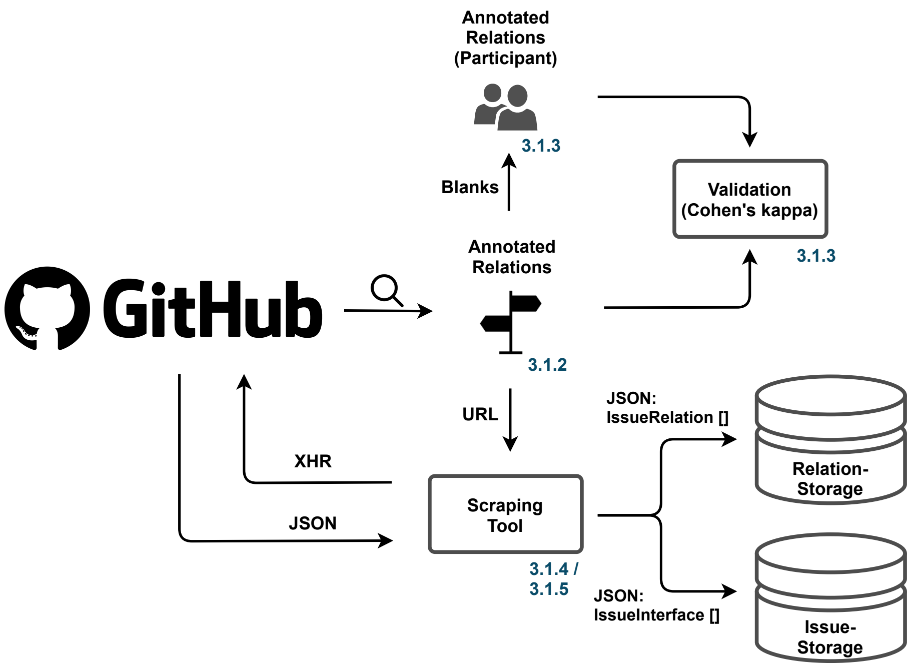


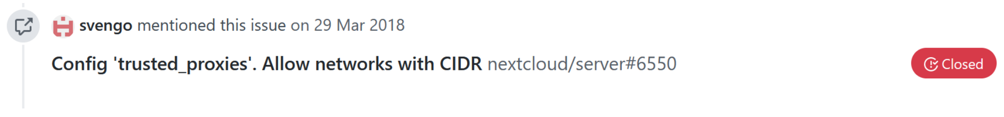
Figure 3.1: Elements of the gathering.

3.1.2 Selecting Issues

The issues were selected from GitHub, which has a lot of easily accessible, popular open-source projects. First, repositories were searched by going through popular projects in GitHub (as illustrated in Figure 3.1 by the magnifying glass). Those projects are used and maintained by a vast user base due to their popularity. As such, it tends to be the case that these repositories contain more issue reports than smaller ones. Furthermore, the big user base potentially leads to a lot of coding issues (issues based on software or source code errors), and most importantly, the issues are reviewed and commented on by many users to identify the problem in which they originated. Another factor for choosing the repositories was that they should consist of a coding project and not just an assembly of resources such as the popular projects "free-programming-books"[2] or "papers we love"[3], because the issues of those projects are most certainly not coding issues. We tabulated the final repositories from which issues were selected in the appendix section Table A.19. Moreover, the issues should not be "pull requests" because they are utilised for fixing issues. Lastly, we did not refrain from closed issues because the resolution of an issue was not of any importance to us.

Also, we obtained issues from multiple repositories to generate a bit of diversity because repositories like Kubernetes[4] have around 40.000 issues, and we did not want the classifier to be fixated on specific issue templates.

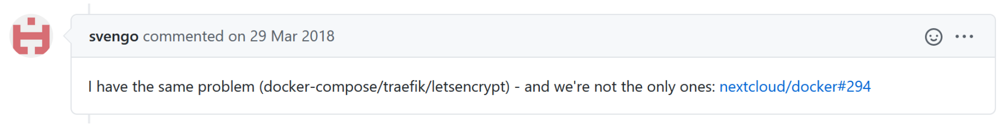
[2]https://github.com/EbookFoundation/free-programming-books
[3]https://github.com/papers-we-love/papers-we-love
[4]https://github.com/Kubernetes/Kubernetes/issues?q=is%3Aissue


svengo mentioned this issue on 29 Mar 2018

Config 'trusted_proxies'. Allow networks with CIDR nextcloud/server#6550

(a) GitHub mention in nextcloud/docker/issues/294[a].

[a]https://github.com/nextcloud/docker/issues/294


svengo commented on 29 Mar 2018

I have the same problem (docker-compose/traefik/letsencrypt) - and we're not the only ones: nextcloud/docker#294

(b) GitHub mention in nextcloud/server/issues/6650[a].

[a]https://github.com/nextcloud/server/issues/6550

Figure 3.2: GitHub mention viewed from both issues.

```
https://github.com/bitcoin/bitcoin/issues/5668 <dupl.> https://github.com/bitcoin/bitcoin/issues/5850
https://github.com/bitcoin/bitcoin/issues/5668 <=> https://github.com/PIVX-Project/PIVX/issues/69
https://github.com/bitcoin/bitcoin/issues/5668 => https://github.com/firoorg/firo/issues/208
https://github.com/npm/npm/issues/17858 <=> https://github.com/npm/npm/issues/17987
https://github.com/npm/npm/issues/17858 => https://github.com/cleverbeagle/pup/issues/29
https://github.com/npm/npm/issues/17858 => https://github.com/dotnet-architecture/eShopOnContainers/issues/268
https://github.com/npm/npm/issues/17858 => https://github.com/JedWatson/classnames/issues/138
https://github.com/npm/npm/issues/17858 <dupl.> https://github.com/npm/npm/issues/18042
```

(a) Sample of self annotated relations.

```
https://github.com/bitcoin/bitcoin/issues/5668  [  ]  https://github.com/bitcoin/bitcoin/issues/5850
https://github.com/bitcoin/bitcoin/issues/5668  [  ]  https://github.com/PIVX-Project/PIVX/issues/69
https://github.com/bitcoin/bitcoin/issues/5668  [  ]  https://github.com/firoorg/firo/issues/208
https://github.com/npm/npm/issues/17858  [  ]  https://github.com/npm/npm/issues/17987
https://github.com/npm/npm/issues/17858  [  ]  https://github.com/cleverbeagle/pup/issues/29
```

(b) Blank relations to be filled out by the participant.

```
https://github.com/bitcoin/bitcoin/issues/5668  [dup]  https://github.com/bitcoin/bitcoin/issues/5850
https://github.com/bitcoin/bitcoin/issues/5668  [<=>]  https://github.com/PIVX-Project/PIVX/issues/69
https://github.com/bitcoin/bitcoin/issues/5668  [<=>]  https://github.com/firoorg/firo/issues/208
https://github.com/npm/npm/issues/17858  [<=>]  https://github.com/npm/npm/issues/17987
https://github.com/npm/npm/issues/17858  [=>]  https://github.com/cleverbeagle/pup/issues/29
https://github.com/npm/npm/issues/17858  [=>]  https://github.com/dotnet-architecture/eShopOnContainers/issues/268
```

(c) Sample of annotated relations by the participant.

Figure 3.3: Different types of relation annotations.

After an issue from a repository was selected, we manually searched for GitHub mentions[5] (such as the one seen in Figure 3.2), or URLs linking two issues. The GitHub comments of both issues were read by us to validate that the reference between the issues is correct. Moreover, through reading through the GitHub comments, the class most fitting to the issue relation was obtained.

Both issue URLs and the relation (in the form of an arrow) were documented in a separate text document (illustrated in the form of a way-point in Figure 3.1) when we found an issue relation class. For simplicity, the arrows representing the classes were changed to a simpler variant in the text document, as depicted in Figure 3.3a to document the findings.

[5]https://guides.github.com/features/issues/notifications — GitHub mentions are a feature provided by GitHub to connect two issues together.

3.1.3 Relation Validation by Participants

We chose to use the expert annotation over crowdsourcing methods as described by Klinger et al. [Kli+18]. Therefore, the documented findings were then cut into smaller text documents (henceforth work packages) containing about 100 issues each. Furthermore, the arrows in the documents handed to the participant were replaced by blanks, as illustrated in Figure 3.3b.

The participant is male and 21 years old. He has fundamental programming knowledge and has used GitHub in the past. After briefing him on filling out the blanks with the relation class most fitting to the relation, he received every second day one of the work package documents containing about 100 issues.

We undertook the practice of splitting the data into multiple smaller work packages and handing them out in this particular manner to reduce the “carry over stress” of the participant.

A piece of a document filled out by the participant (with the relationships he saw most fitted) can be seen in Figure 3.3c. To validate the participant’s reports, both findings of our own annotation and the participant’s annotation were compared using Cohen’s kappa. The reason for this is that humans are not able to consistently report a gold standard of relevance [Man08].

The corresponding k value for the first 1460 issues of the two different annotators for the data set was $k = 0.7041$. This kappa value is considered a “Substantial Agreement” according to Landis and Koch [LK77], and a “fair to good” agreement according to Fleiss [GW97] Table 2.1 which means there is a broad mutual agreement. Therefore, there was no need to inquire further participants to review the disagreements of the class choice on some of the issues.

3.1.4 Collecting the Issues

After the validity was verified, data of the issue relations had to be collected. In each issue relation, two vital elements were essential; The issues themselves, and the relation they are standing in.

The relation itself consists of two issues and a relation date. This date is a part of the GitHub mention and can be seen from both issues as shown in Figure 3.2. It corresponds to when one of either issues (Figure 3.2a) was mentioned in the other’s comments (Figure 3.2b). Note that there is not always a date attached in the relation file, which can happen due to an outdated URL or if the issue was mentioned in the issue’s body. The “mention” date is used to ensure that the classifier is not getting all the information handed to it by reading through the comments. Instead, the classifier can see all the dates and comments leading up to the mention to get the most background information. No comments are provided to the classifier if there was no date (an undefined one) provided in the relation.

For the issues, the data kept is the issue title, the issue description (body), the comment bodies, and their dates, as explained previously.

Listing 3.1 Relation and issue interfaces

```
/**
 * This interface is used for documenting the
 * relations between two issues
 */
export interface IssueRelation {
    urlIssueA: string;
    urlIssueB: string;
    relation: Relation;
    dateMentioned: Date
}

/**
 * An issue can have several comments
 */
export interface Comment {
    createdAt: Date;
    updatedAt: Date;
    body: string;
}

/**
* The IssueInterface is the interface used for issues
**/
export interface IssueInterface {
    id: string;
    issueID: number;
    repository: string;
    user: string;
    title: string;
    body: string;
    comments: Comment[];
    url: string;
}
```

3.1.5 Implementation

To collect the data from GitHub, we developed a scraper. This scraper takes as input the text file of the annotated relations as depicted in Figure 3.3a whose lines serve to document the relations between two issues, and uses the relations and URLs to create relation JSONs using the “IssueRelation” interface shown in Listing 3.1.

Additionally, it takes the URLs of the GitHub issues and creates REST API requests by taking the user-/repository name and the issue number from the URLs. More precisely, it creates two API requests per issue to query the GitHub issue body and title with the first request and the comments of a given issue with the second one. Using this information, issue JSON objects are being constructed using the “IssueInterface” shown in Listing 3.1.

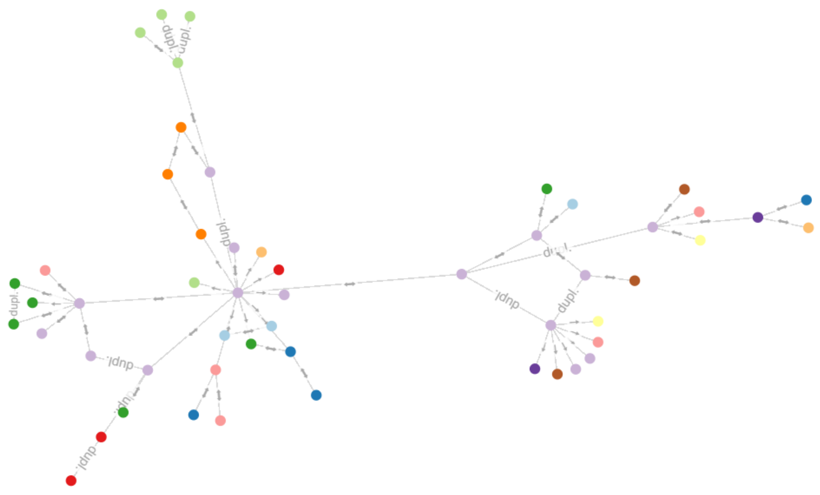


Figure 3.4: IRG sub-graph.

3.2 Generating unrelated Issues

Figure 3.4 depicts a sub-graph of the IRG built of our collected data. It visualises the relations between some issues (here denoted as dots). The edges between two nodes have a label attached to them, which shows the relation class the two issues are standing in (visualised in the same simple notation as in the Figure 3.3a) and the direction of the relation (if there is one).

3.2.1 Choosing Issue Pairs

The whole IRG is a disconnected graph. It consists of many connected sub-graphs like the one in Figure 3.4. We utilised this property to generate unrelated issues by randomly picking two distinct disconnected graphs. From each graph, we selected one issue at random. Then, this newly created unrelated issue was appended to the unrelated issue data set iff. the newly constructed unrelated issue was not already present there. Like the other data sets, this data set also got shuffled.

3.2.2 Risk

We did not use two issues from the same connected sub-graphs because they can stand in a transitive relation. For example, three of the four green nodes at the top in Figure 3.4 are probably standing in such a transitive relation. The reason for this is that the first node is a duplicate of the second one, and the second node is a duplicate of the third. Therefore, the first node may also be a duplicate of the third. Hence, there is a risk involved that the GitHub users did not find all the relations or did not see the need of annotating more issues (and their relation) as duplicates. To counteract this problem, GitHub issues from different sub-graphs were taken to minimise the risk of them standing in a relation.

	$\rightsquigarrow$	$\leftsquigarrow$	$\leftrightsquigarrow$	$duplicate$	$\not\leftrightsquigarrow$	Total
relations	492	56	898	558	899	2004 (2903)*

Table 3.1: Scraped class distribution. The “*” in the “total” column of the table denotes the total relations, including the added unrelated issues.

3.2.3 Test Data Set

The test data set and validation data set were generated by randomly cutting out sub-graphs of the IRG so that the overall sum of the nodes of the sub-graphs in the test and validation data sets consisted at least of 20% each of all the nodes. Using this number, the percentage of data used for each validation and testing lies in the typical range of 20% to 25% [HTF09]. The physical separation ensures that the issues provided in the test and validation sets were not previously seen by the classifier. Furthermore, the unrelated issues in the test and validation sets are created only from issues within the set because, otherwise, the same problem might occur.

3.3 Generalisation

For the text corpus over 2000 issue relations were scraped as depicted in Table 3.1 by the method described in Section 3.1. The issue relations are comprised of a total of 1952 distinct issues. The difference in number between the amount of relation and issues is due to the sub-graphs not always being sparse graphs. For example, an issue is not always having a relation with a previously unseen issue. Moreover, at least two issues are standing in a relation, and there can be no issue without relation. Besides, the relations can form dense sub-graphs.

The added unrelated issues have been separated because they were not scraped but generated using the procedure seen in Section 3.2.

Ensuring Symmetry

From these scraped relations, equivalence classes for the classifier can be derived by rearranging them. This can be done, because iff. $\forall a, b \in I\ \forall c' \in \{\leftrightsquigarrow, \not\leftrightsquigarrow, dup\} : (a, b)_{c'} \Leftrightarrow (b, a)_{c'}$ for the other classes $\in \{\leftsquigarrow, \rightsquigarrow\}$, $b \in I$: $(a, b)_c \Leftrightarrow (b, a)_c$ holds.

By this procedure, the amount of issues suitable for training gets doubled. This step is essential to generate more data and generalise it so that the relations of the data set do not bias the classifier. Hence, the prediction would be less influenced by the issue order handed to the classifier.

Issue Distribution

After ensuring the symmetry by reordering the issues, the relations in the resulting relation classes were shuffled and cropped. Removing some relations ensured that the amount of relations in each class is the same. The final relation distribution can be seen in Table 3.2.

	$\rightsquigarrow$	$\leftsquigarrow$	$\leftrightsquigarrow$	*duplicate*	$\not\leftrightsquigarrow$	Total
train	238	238	238	238	238	1190
validation	134	134	134	134	134	670
test	176	176	176	176	176	880
Total	548	548	548	548	548	2740

Table 3.2: Class distribution between the data sets.

The training data was used for the classifier's training. The validation data was used to validate the classifier and the effect of the hyperparameters, such as finding the best vectorisation strategy. The test data was used explicitly for testing and comparing the different classifier architectures by presenting them with previously unseen data. All the data sets are distinct data set. Hence they do not share issues with each other.

3.4 Corpus Statistics

Here the document corpus gets analysed, consisting of the issue titles and bodies of the GitHub issues. The comments are not present in these documents because the length of an issue is influenced by the date in the GitHub mention as can be seen in Figure 3.2 and, hence, this would not be an accurate representation of the real issues' lengths. Additionally, due to the length of an issue being influenced by the comments and the respective relation date, the same issue has different lengths for different relations (as we want to preserve the most information possible for the classification).

3.4.1 Amount of Comments per Issue

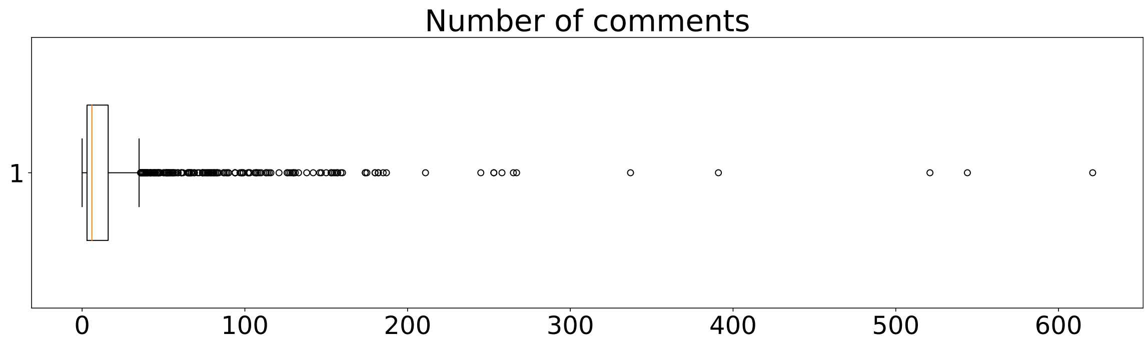
The number of comments in an issue varies widely; some have none, and others have over 500, as can be seen, in Section 3.4.1. The median number of comments is five as depicted in Figure 3.5b without outliers.

3.4.2 Token Amount

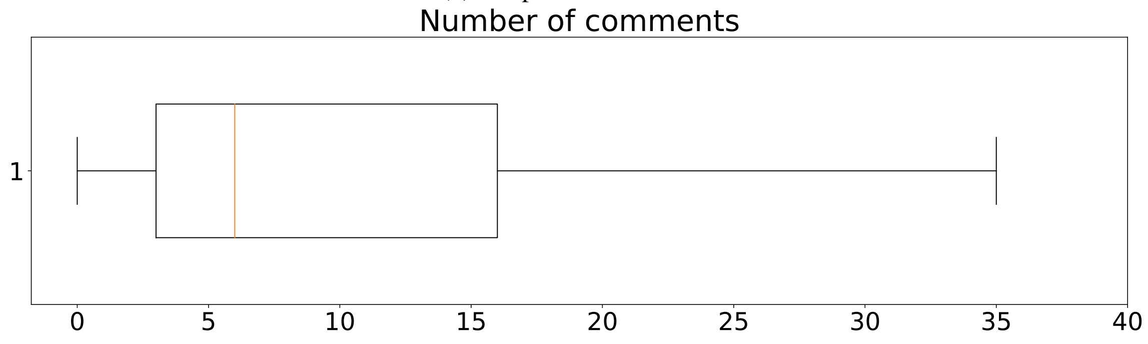
Without comments and (English) stop words, the issues have an average length of around 130 tokens, as depicted in Figure 3.6. Furthermore, both box plots in this figure are representing the same data, except for the fact that the outliers have been removed for the second one (Figure 3.6b) to enhance the readability.

3.4.3 30 Most Occurring Words

Also, the top 30 tokens of the corpus were plotted to analyse abnormalities illustrated in Figure 3.7. Another observation is that there are many terms referring to software systems. Several names such as "npm", "Docker", "minikube", or "Kubernetes", are among the 30 most occurring tokens. Therefore, it seems, that the data set is heavily influenced by the technologies the repositories



(a) Boxplot with outliers.



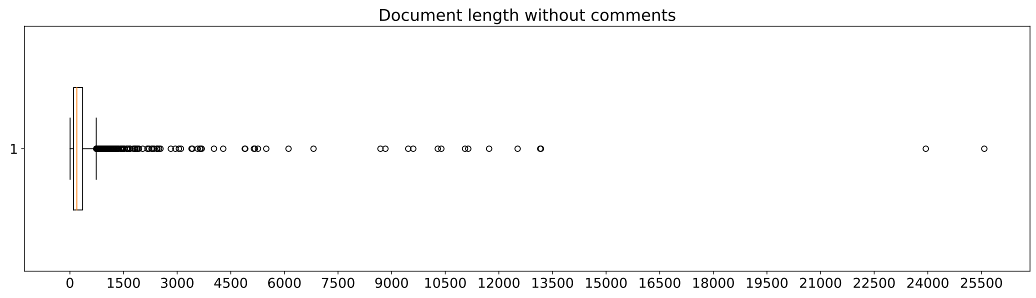
(b) Boxplot without outliers.

Figure 3.5: Number of comments in the issue documents.

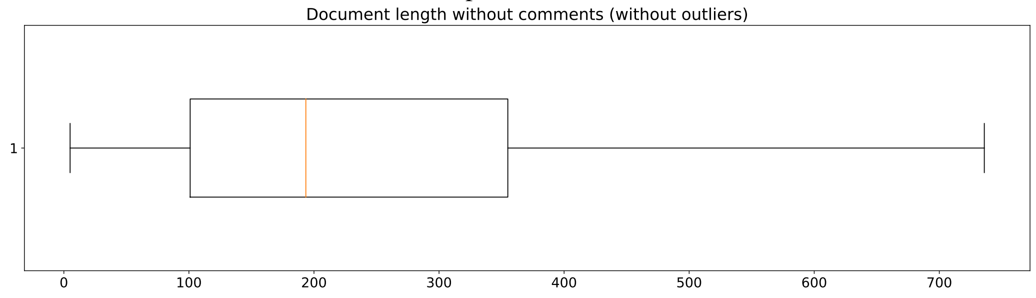
(Table A.19 tabulates the repository names) used. Nevertheless, there are still the usual stopwords like "to" (as shown in Figure 3.7b). Another observation is that there are plenty of different numbers amongst the top 30 tokens, which might be due to the versioning of the different frameworks and libraries.

3.4.4 Representation of the Issues for the Classifier

The issues given to the classifier consist of an issue title, issue body, and the comment's bodies up to the date found in the relation. They are all concatenated by using a line break character and represented in the form of a string. The idea behind this was that the classifier is not influenced by unique tokens between title, body and comments. Moreover, the classifier has the maximum amount of text in its corpus without knowing the relation class determined by the annotator in the comment. If no date was found in the relation JSON, our script appended none of the issue comments onto the issue. An exception to this rule are the randomly generated comments generated from within the specific set (the train, validation or test set). For those, all the comments were appended to simulate two not labelled issues.

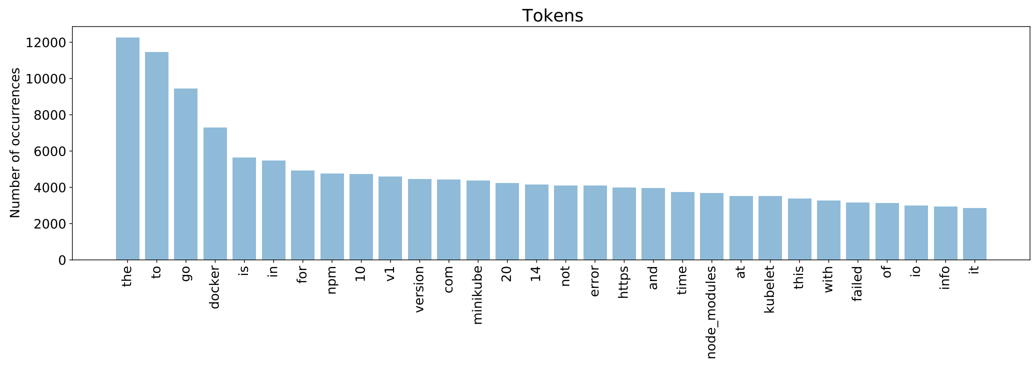


(a) Boxplot with outliers.

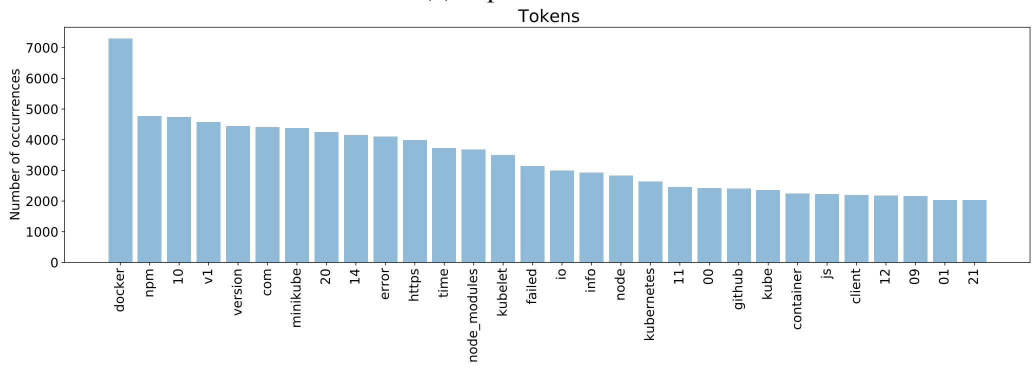


(b) Boxplot without outliers.

Figure 3.6: Number of tokens in issue documents.



(a) Top 30 tokens.



(b) Top 30 tokens without English stop words.

Figure 3.7: 30 most occurring tokens.

4 Classifier

This chapter presents the experiment setups for the different siamese models for distinguishing the relations of a given text.

The "basic" classifier architecture of the models consisting out of a vectoriser (with additional topics in case of the "Topic" model 4.9) is depicted in Figure 4.1. As shown in this image, the models themselves can be split into two parts; The vectorisation part, where the text is converted into feature vectors, and the classification part, where the vectors are merged into one to encapsulate their information. This single vector is afterwards used to compute the output using a dense (softmax) layer. To enable a direct comparison of each model, the batch size was fixed to 25, and the number of epochs to 25. Moreover, a word was considered part of the vocabulary if it occurred at least two times.

We regarded finding the optimal vectorisation method as a hyperparameter to be tuned for the different models. Also, by using multiple vectorisation methods, their ability to transform large documents into vector representations can be tested. We used the standard BOW vectoriser, a vectoriser which uses word embeddings to obtain the document embeddings and a encoder specialised to obtain embeddings from word sequences.

For the BOW vectorisation strategies, different vocabulary sizes were tested, those are 512, 1024, 8192 and 16384 words. In addition, we tested for all those vocabulary sizes unigrams and uni+bigrams, as well as having tf-idf as a weighting. Before the vectorisation, the English stopwords were removed, and all the words were transformed to lower case to save words in the vocabulary. Moreover, a token was included in the vocabulary if it appeared more than two times.

For the embedding-based solutions, we tested averaged and uSIF fastText vectors and averaged and uSIF pretrained glove vectors[1]. Both embedding variations generated embeddings of a length of 100. The window size for the fastText embeddings was 3. Besides those, we also tested the impact of the standard 512 length universal sentence encoder[2].

4.1 Sub. Model

The "Sub." model is our baseline model to compare against. It is inspired by the siamese networks of the Sentence Bert [RG19] from Reimers et al. that utilised a concatenation $(u, v, |u - v|)$ of the single vectors u, v as well as the element-wise difference $|u - v|$ to merge both document vectors.

[1]Based on Wikipedia 2014 and Gigaword, with 5.6B uncased tokens, https://github.com/RaRe-Technologies/gensim-data/releases/tag/glove-wiki-gigaword-100

[2]https://tfhub.dev/google/universal-sentence-encoder/4

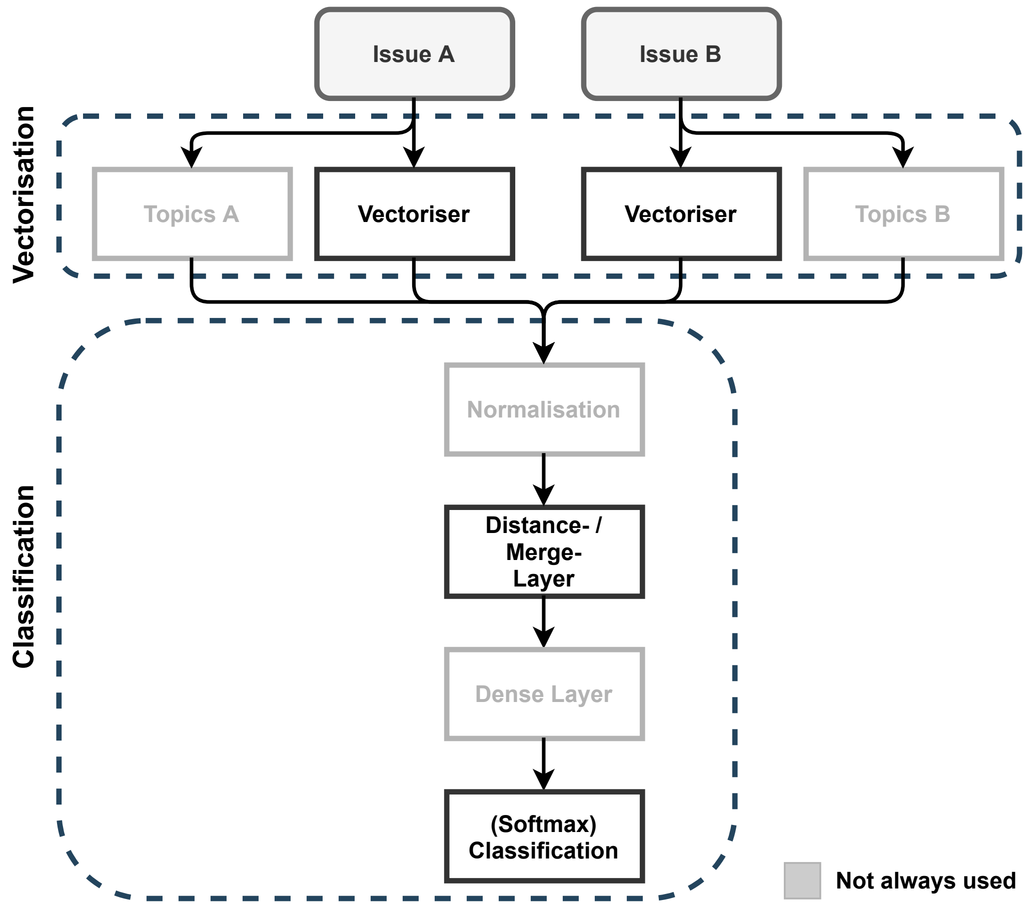


Figure 4.1: Basic classifier architecture.

And the InferSent model architecture [CKS+17] from Conneau et al. that used the concatenation $(u, v, |u - v|, u * v)$ of both single vectors, the element-wise difference in addition to their dot wise product for their baseline.

Our model consists of 3 layers. It takes two document strings of varying length as input and outputs a vector representation u, v. The vector representations for the strings differ in length due to the different vectorisation processes. Therefore, the length of the vector also does not stay the same.

After the vector representations of the documents were created, the input is fed into a merging layer. This merging layer performs an element-wise subtraction of both vectors to create one vector containing all the information. Then the merged vector gets fed into a dense softmax layer for prediction.

Using the subtraction instead of an absolute element-wise difference allows the model to distinguish whether an input token came from the left or right input (for an input, which is not present in both documents), while the property of minimising the impact of overlapping input tokens gets preserved.

4.2 Mult. Model

The “Mult.” model is built on the basis that unrelated and duplicate issues have a low and high overlap, respectively. Therefore, using an element-wise multiplication for the merge should improve the models’ abilities to identify if the issues are either duplicated, related or unrelated.

The mode architecture for this model is essentially the same as the one for the baseline “Sub.” model Section 4.1. The only difference is that an element-wise multiplication is used for the merge instead of an element-wise subtraction.

4.3 Avg. Model

By using the average of the tokens from both vectors, tokens, which are present in one but not both documents, are still taken into account, while the ones that are present in both issues do not lead to overly high numbers. Therefore, this property stands in contrast to the “Mult.” model, which “removes” tokens that do not occur in both documents. Therefore, it should lead to slightly better results than the “Mult.” model because it should better understand what words do occur in the documents. Moreover, the values would not get as big as for the pure multiplication layer, which can help the model adjusting its weights accordingly.

This model also utilises the same basic architecture as the previous ones except for the element-wise average for the merging process.

4.4 Concat. Model

This model utilises all the previous techniques and merges them into one big vector representation. Therefore, this model is able to access all the information the previous model have and should, in theory, be able to make a better prediction based on the input vector representation. The model architecture differs from the previous models because the merging layer now consists out of a concatenation of the previous techniques. Hence, it looks like $(a - b, a * b, (a + b)/2)$ and, thus, produces a longer output vector for the two given vectors a, b from the vectorisation process. Nevertheless, the output any input layers are still the same.

4.5 CosConcat. Model

BOW vector representations for similar documents are relatively close in vector space. The same holds for similar (sentence) embeddings, which are also grouped closely in space. Therefore, this model should have a better understanding of “duplicate” documents and “unrelated” ones due to their similarity or dissimilarity in the problem they describe. The merged vector is defined by $(a - b, a * b, (a + b)/2, \theta)$ for the two document vectors a, b. θ is the value of the cosine similarity of both input vectors. Other than that, the model consists out of the two input vectors and one dense $softmax$ output vector.

4.6 UniCosConcat. Model

This architecture takes into consideration, that the cosine values lie in the interval $[0, 1]$. In contrast, the other values of the merged vectors can take any value in $\mathbb{Z}$ for the standard BOW vectorisation. Therefore, the absolute values of the merged vectors (except for the averaged vector) that are not zero will most likely be larger than the cosine similarity value. Taking this into account, this model uses the Euclidean / $l2$ norm to produce unit vectors $\widehat{a}, \widehat{b}$ out of the given vectors a, b by appending a so-called lambda layer in front of the merging layer. The lambda layer is used to apply a function onto all elements of the vector. In this case, it takes the document vectors a, b as input and applies the formula $\widehat{u} = u/\|u\|$ on it. So the merging layer looks as follows $(\widehat{a} - \widehat{b}, \widehat{a} * \widehat{b}, (\widehat{a} + \widehat{b})/2, \theta)$. This should decrease the effect of common non-stop words that might have a significant impact due to their high occurrences in some documents in the context of BOW. The sentence encoder already deals with that problem, and the tf-idf weighting also reduces the impact of those words due to the log normalisation (if a token occurs a lot in a given document). Besides, the cosine similarity θ will not be influenced by the unit vectors because the direction of the stays the same.

4.7 SumConcat. Model

Unary vectors might negatively influence the embeddings in the embedding space because they pull all the vectors closer together. Moreover, small changes in the weights of the model will probably have a considerable impact in dealing with smaller numbers. Therefore, this model appends the weights of the vectors onto the concatenation instead of creating unit vectors. So this model tests the hypothesis of whether or not it is advantageous to provide the model with the length information of the vectors so that it can adjust its weights accordingly and therefore reach a better macro $f1$ score. The extended merging layer looks as follows: $(x,\ \|x\|,\ y,\ \|y\|,\ z,\ \|z\|); \quad x = a - b \quad y = a * b \quad z = (a + b)/2.$

4.8 Issue-vector Model

Both publications, the Sentence Bert [RG19] from Reimers et al. as well as the InferSent model architecture from [CKS+17] from Conneau et al. appended the document vectors onto their distance metric $(a, b, |a - b|)$ and $(a, b, |a - b|, a * b)$. Therefore, this architecture tests if the appending of the issue vectors themselves can help the model to improve on finding the source-issue. The merging layer of this model looks like $(a, b, a - b, a * b, (a + b)/2)$.

4.9 Topic Model

Nguyen et al. used topic modelling in their work for deduplication. They found out that increasing the number of topics increased their accuracy up to a specific range, in which it becomes stable. After this range with more than 380 topics, their accuracy began to decline due to nuanced topics, which may lead topics to a semantically overlap with each other. [NNN+12] Moreover, they stated that different repositories have different stable ranges. Besides, all the ranges started between 100 – 140 topics. Nevertheless, our issues are from multiple projects and not single ones. Hence we can deduce that the lower bound has to be larger than 100. Therefore, we used the topic similarity measured by the cosine distance of the topic predictions to enhance the deduplication of this model. The model's merging layer has four inputs, two for the vectorised issue documents a, b and two for the vectorised topics x, y. Those are getting combined to $(a - b, a * b, (a + b)/2, \theta_t)$, where θ_t stands for the cosine similarity of the topic and not the document vectors.

To decrease the number of numbers in the training set, we created a special count vectoriser for the topics, which replaces numbers by "num" and transforms words to lower case. Also, the English stop words were removed, and the vocabulary was capped to 20000. Also, the topic words have to appear at least 5 times in the data set, and they should not appear in more than 50% of the documents

5 Implementation as a Microservice

This chapter discusses the implementation of the processes “vectorisation” and “classification” into a microservice-based solution. Section 5.1 explores the architecture of the microservice. Section 5.2 discusses the details of the *Vectorisation Service* which acts as a pre-processor for the *Classification Service*. Section 5.3 details the classification microservice and its functionality. Last but not least, Section 5.4 outlines how the microservice is deployed into a Kubernetes cluster.

5.1 Architecture

The architecture consists of two services; The *classifier Service* and the *Vectorisation Service*. This is shown in Figure 5.1. Through the decoupling of both services, the issues can be independently vectorised and classified. The advantage is that the classification process is much faster than the vectorisation process. Moreover, as a result of decoupling both services, the vectorisation phase can make use of vertical scaling. In addition, the data can also be stored in a database for later use. Both services are written in `python:3.8` and make use of the `tiangolo/uvicorn-gunicorn-fastapi` Docker image[1]. This image creates an `Uvicorn` ASGI server managed by `Gunicorn` to run the `FastAPI` web framework.
The microservice itself gets created using Docker compose and is connected using a bridge network[2].

5.2 Vectorisation Service

At the beginning, the universal sentence encoder[3] (which is used for the vectorisation process) gets loaded by the *Vectorisation Service*. After the sentence encoder has been fully loaded, the service exposes two routes/endpoints: The *vectorise_issue* endpoint, and the *vectorise_and_classify_issues* endpoint. Both expect JSON arrays of the form outlined in Listing 5.1 as input. These JSON arrays provide the vectorisation service with a list of issues.

After computing the vectorised issues, the service responds with “Vectorise Issue Response JSON” (cf. Listing 5.2) and “Issue Relation Response” (cf. Listing 5.3) in the case of a request to the *vectorise_issue* endpoint and the *vectorise_and_classify_issues* endpoint, respectively. For the classification of the issues, it sends an HTTP request to the *Classification Service* using HTTPX[4], an asynchronous HTTP client.

[1]https://hub.docker.com/r/tiangolo/uvicorn-gunicorn-fastapi/
[2]The name is “mybridge”
[3]https://tfhub.dev/google/universal-sentence-encoder/4
[4]https://www.python-httpx.org/

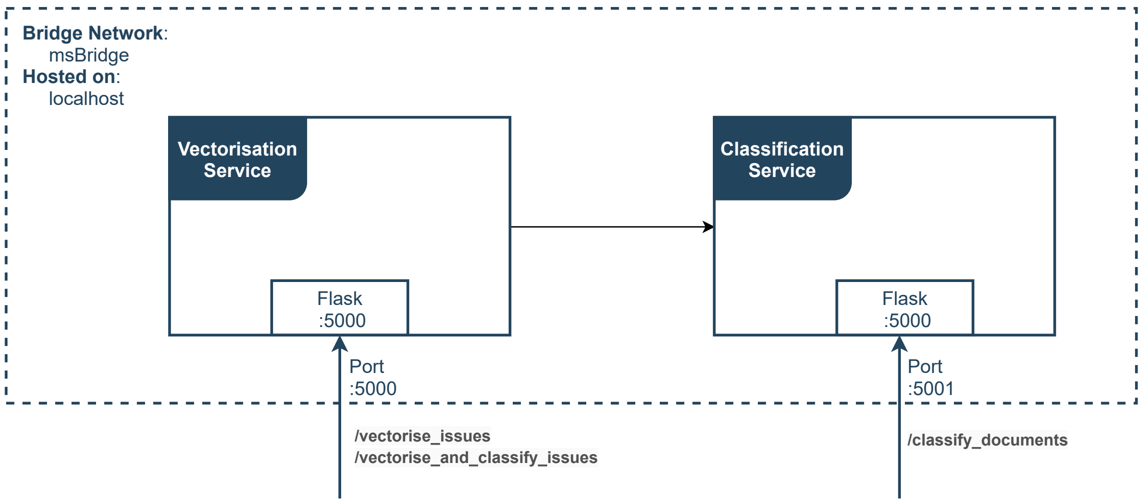


Figure 5.1: Microservice architecture.

Listing 5.1 Issue Request JSON

```
{
"issueTexts": ["fuu", "bar", ...]
}
```

5.3 Classification Service

The classification service makes use of the “UniCosConcat.” model introduced in Section 4.6. The model (included as a tar file in the folder) gets loaded for fast deployment during service startup.

Listing 5.2 Vectorise Issue Response JSON

```
//response from vectorise_issue
{
    "issueTexts": ["fuu", "bar", ...]
    "vectorisedTexts": [
        [0.1232, 0.2123, ...],
        ...
]}
```

Listing 5.3 Issue Relation Response JSON

```
//response from vectorise_and_classify_issues
{
    "issueTexts": ["fuu", "bar", ...]
    "relations": [
        {"issueA": 0, "issueB": 1, "relation": 0},
        {"issueA": 0, "issueB": 2, "relation": 2},
        ...
]}
```

This service expects JSON objects of the form of the *vectorise_issue response* Listing 5.2 and creates an "Issue Relation Response" Listing 5.3. The response of this service encodes the relation using two integers for the issue indexes ("issueA", and "issueB") and one for the relation "relation", which reduces the space requirements of the response. In addition, not all issue pairs are classified; just the upper half of the adjacency matrix of the dependency graph is considered, which is enough to encode the whole IRG.

5.4 Kubernetes

For Kubernetes, the Docker images have to be first built using Docker Compose or using the Dockerfile's. Those images are then used in the Kubernetes deployment file[5]. Afterwards, it creates two Kubernetes services: The *Vectorisation Service*, and the *Classification Service* with one pod each containing the images created previously. Both services are of the type *Load Balancer* and exposed using an external IP address outside of the cluster.
The whole configuration `YAML` for the deployments, services and network policy as well as the code for building the images can be found with the other code in our GitHub repository[6].

[5]If the images should be pulled from a registry, the "imagePullPolicy" has to be changed
[6]https://github.com/T0B1K/Issue_Relations

6 Evaluation

This chapter presents the results of our findings divided into three sections; results, discussion, and threats to validity. The first section presents the findings themselves by discussing the results. The second section discusses our findings and argues about the acceptance/rejection of the research hypotheses. In the last section, we discuss situations and conditions that could threaten the validity of the research.

6.1 Results

This section presents the general results of our experiment and compares the results of the different models. All models have been trained using the ADAM optimiser [KB17] and a Crossentropy loss. Moreover, for each model, several vectorisation methods were evaluated, as stated in Chapter 4.

Those vectorisation methods were only tested on the validation data set. The reason, for that is that the vectorisation step was regarded as a hyperparameter to tune. The best performing vectorisation process according to the macro $f1$ score on the validation data set was used on the test data set with the corresponding model. This step was undertaken to mitigate bias. Nevertheless, all the different vectorisation methods for the different models are tabulated in the appendix and sorted by the model.

The results of the models for the experiment are shown in Table 6.1 and Table 6.2. Table 6.2 lists the $f1$ scores as well as the macro $f1$ score for the models and the issue relation classes. The Table 6.1 further divides the $f1$ scores of the classes into their respective precision and recall values.

Both tables, as well as the tables in the appendix section encode the names of the models and their vectorisation strategy in the following manner:

```
model_name-(tfidf | count | sentence | ft | glove)_{(vocabulary_size [,Un+Bigram]? | avg | sif)}
```

The only two variables in this regular expression are the "model_name" and the "vocabulary_size". So "$tfIdf_{1024;Un+Bigram}$" stands for a tf-idf vectoriser with a vocabulary size of 1024 using un- and bigrams. And "ft_{avg}" stands for an vectoriser using fastText word level embeddings and a average weighting for the document embeddings, and "sentence" stands for the embeddings obtained using the universal sentence encoder.

The experiment results in Table 6.2 show that all of the models reached a macro $f1$ score of over 0.2, which is the expected value for a model, which makes uniformly distributed random guesses for all five classes. Nevertheless, the results also show that none of the models was able to reach a macro $f1$ score of over 0.5. Moreover, it can be seen that there is no universal model that reached the best score for each of the five classes of relations, and each model has its own strengths and weaknesses for the different classes. For the classes "$\rightsquigarrow$", "$\leftrightsquigarrow$","unrelated" and "duplicate", the

Model	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
sub-tfIdf$_{1024}$	0.3545	0.6648	0.375	0.6136	0.2353	0.1136	0.2522	0.1648	0.2097	0.0739
mult-tfIdf$_{1024,Un+Bigram}$	0.2308	0.0682	0.2254	0.1818	**0.2351**	**0.4489**	0.5596	0.6932	0.3333	0.25
avg-count$_{1024,Un+Bigram}$	0.2185	0.1875	0.2453	0.0739	0.2061	0.1932	0.2612	0.5284	0.1806	0.1591
concat-count$_{512}$	0.3388	0.233	0.4468	0.2386	0.2438	0.2216	0.3411	0.5852	0.2906	0.3352
issue_vector-count$_{512}$	0.3985	0.3011	0.3971	0.3068	0.2564	0.1705	0.3436	0.5682	0.2709	0.3125
cosConcat-count$_{8192}$	0.4264	0.3125	0.5385	0.3977	0.2486	0.2557	0.4263	0.7727	0.2562	0.1761
uniCosConcat-sentence$_{512}$	**0.5062**	**0.4659**	**0.5058**	**0.4943**	0.2523	0.1591	**0.7124**	**0.6193**	**0.2908**	**0.4659**
sumConcat-count$_{8192}$	0.4694	0.2614	0.4889	0.375	0.2619	0.1875	0.394	0.75	0.2688	0.2841
topic-count$_{1024,Un+Bigram}$	0.3401	0.3807	0.4423	0.2614	0.229	0.1705	0.3946	0.6591	0.2078	0.1818

Table 6.1: Precision and Recall measures for the different models.

Model	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
sub-tfIdf$_{1024}$	0.4625	0.4655	0.1533	0.1993	0.1092	0.278
mult-tfIdf$_{1024,Un+Bigram}$	0.1053	0.2013	**0.3086**	0.6193	0.2857	0.304
avg-count$_{1024,Un+Bigram}$	0.2018	0.1135	0.1994	0.3496	0.1692	0.2067
concat-count$_{512}$	0.2761	0.3111	0.2321	0.431	0.3113	0.3123
issue_vector-count$_{512}$	0.343	0.3462	0.2048	0.4283	0.2902	0.3225
cosConcat-count$_{8192}$	0.3607	0.4575	0.2521	0.5495	0.2088	0.3657
uniCosConcat-sentence$_{512}$	**0.4852**	**0.5**	0.1951	**0.6626**	**0.3581**	**0.4402**
sumConcat-count$_{8192}$	0.3358	0.4244	0.2185	0.5166	0.2762	0.3543
topic-count$_{1024,Un+Bigram}$	0.3592	0.3286	0.1954	0.4936	0.1939	0.3141

Table 6.2: Precision and Recall measures for the different models.

"UniCosConcat." model (introduced in Section 4.6) reached the best $f1$ scores with scores of about 0.49, 0.5, 0.66, 0.36 respectively, while the "mult." model (introduced in Section 4.2) reached the best $f1$ score for the class "$\leftrightsquigarrow$" with a score of about 0.31.

The results of the models listed in Table 6.2 yield that almost every model reached a relatively low $f1$ score for the $\leftrightsquigarrow$ class. As stated previously, the highest $f1$ score was reached by the "Mult." model with an $f1$ score of about 0.31, which is a considerable amount higher than the score for the second-best model, the "CosConcat." model with an $f1$ score of about 0.25. However, examining the precision and recall creating the $f1$ score of the "Mult." model shows that the reason for its comparably high $f1$ score is a relatively high recall value of 0.4459. Its precision lags behind and is even lower than the precision of the "CosConcat." model with a score of 0.2351 compared to 0.2486. Moreover, both the "CosConcat." and the "Mult." model are the only models with lower precision than recall.

The highest precision in this class was reached by the "SumConcat." model, with a value of about 0.26. However, the recall of the "SumConcat." model is with about 0.19, significantly lower than its precision. Therefore, it only reached the fourth-highest $f1$ score of this class. It got surpassed by the "Concat." model with an $f1$ score of about 0.23. Nevertheless, the rest of the models, with the exception of the "issue" vector model with an $f1$ score of 0.2048, lie below 0.2.

On the "duplicate" class the models reached on average a slightly higher $f1$ score compared to the $\leftrightsquigarrow$ class. The "duplicate" class can be split into two categories: Models with an $f1$ score under 0.21 and models with an $f1$ score over 0.27. For the models under 0.21, the best model is the

"CosConcat." model, with an $f1$ score of 0.2088, closely followed by the "Topic" model with an $f1$ score of 0.19 lower by about 0.02. About 0.2 lower than that lies with an $f1$ score of about 0.17 the "Avg." model, followed by the "Sub." model with an $f1$ score of about 0.11. For all those models, their recall is lower than their precision. While the recall is only about 0.02 lower than the precision for the "Avg." model, the recall for the "Topic", "CosConcat." and "Sub." models are lower by about 0.03, 0.08 and 0.14 respectively.

For the other half of the models with an $f1$ score of over 0.27, the worst performing models are the "SumConcat." model with an $f1$ score of 0.2762 followed by the "Mult." model with an $f1$ score of about 0.29 and the "Issue-vector" model with an $f1$ score of about 0.29 for the "duplicate" class. The best two models for this class are the "Concat." model and the "UniCosConcat." model with $f1$ scores of about 0.31 and 0.36 respectively. As can be seen in Table 6.1, for this models, their precision for this class is lower than their recall.

For the models of this half, their precision is lower than their recall. For the "SumConcat." model, the precision is lower by about 0.01, which is the least difference compared to the other models in this category. The other models have differences of about 0.04 for the "Issue-vector" model, 0.05 for the "Concat." model and even 0.18 for the "UniCosConcat." model between their precision and recall. The only model in this category, for which this does not hold, is the "Mult." model with a about 0.08 higher precision than recall.

For the other three classes, $\rightsquigarrow$, $\leftarrowtail\!\!\leftsquigarrow$ and especially the "unrelated" class, the models performed on average better than for the previous classes. Moreover, with the exception of the "Sub." model with an $f1$ score of about 0.2, which is worse than the ones for the classes $\leftsquigarrow$ and $\rightsquigarrow$, all other models reached their highest $f1$ score for the "unrelated" class. The second lowest $f1$ score was reached by the "Avg." model for this class, followed by the"Issue-vector" and the "Concat." model with $f1$ scores of about 0.43. The other models performed a lot better with $f1$ scores of about 0.49 for the "Topic" model, 0.52 for the "SumConcat." model and 0.55 for the "CosConcat." model. Nevertheless, those models are still far worse than the "UniCosConcat." model with an $f1$ score of 0.6626 for the "unrelated" class. Moreover, with the exception of the "Sub." model with a relatively low $f1$ score, the "UniCosConcat." model is the only model, which has a higher precision than recall, with a precision slightly higher than 0.7 and a recall of about 0.62. The other models have a far higher recall compared to their precision for this class.

Comparing the model's yields, that for the $\rightsquigarrow$ class, the best models are the "UniCosConcat." model and the "Sub." model. Both models reached an $f1$ score of over 0.4 with an $f1$ score of about 0.46 for the "Sub." model and an $f1$ score of about 0.49 for the "UniCosConcat." model. Those models performed by far better than the subsequent best models (the "CosConcat." model and the "Topic" model), which reached $f1$ scores of about 0.36. In addition to that, the "Issue-vector" model and the "SumConcat." model are also not too far off with $f1$ scores of about 0.34. The worst performing models for this class are the "Concat." model with an $f1$ score of about 0.28 followed by the "Mult." model and the "Avg." model with scores of about 0.2 and 0.11 respectively. For this class, with the exception of the "Sub.", and the "Topic" model, all models have a better precision than recall.

For the $\leftsquigarrow$ class, the same holds, with the exception of the "Topic" model, which has this time a considerate higher precision. Besides the best models for this class are again the "UniCosConcat." and the "Sub." model with $f1$ scores of about 0.5 and 0.47 respectively. Moreover, for this class, the "CosConcat." and the "sumCount" model both reached a comparably high $f1$ score with an

$f1$ score of about 0.46 for the “CosConcat.” and one of about 0.42 for the “SumConcat.” model. Those scores are considerably higher than the ones of the next models, the “Issue-vector” model, the “Topic” model and the “Concat.” model with values of about 0.35, 0.33 and 0.31 respectively. The worst models are the “Mult.” model with a score of about 0.2 and the “Avg.” model at around 0.11. For the classes $\leftsquigarrow$ and $\rightsquigarrow$ roughly the half of the models reached similar results for their $f1$ scores which only differ by about 0.03 (the “UniCosConcat.”, “Sub.”, “Concat.”, “Issue-vector” and the “Topic” model).

6.2 Discussion

This section analyses the results presented in section Section 6.1. First, the research question posed in the introduction and its acceptance/rejection gets discussed. Afterwards, the different models and their vectorisation methods get analysed in detail.

6.2.1 Discussion of the Research Question

The research question posed in the introduction was “How can relationships between issues that possibly target independently managed software projects be detected?” is solved through most of the models presented in Chapter 4 (excluded the average mode, which is producing more or less random results). Therefore, using a siamese-like network that classifies the relation for two given issues can detect issues, which can be even from different repositories. Nevertheless, as can be seen in Section 6.1 different models have different strengths and weaknesses. Our best model, according to our findings, is the “UniCosConcat.” model using the Universal Sentence Encoder to create document embeddings. Even though it is the best model by a relatively large margin, it is still not able to detect every relationship between the issues, and it is also not able to reach a truly high performance. In addition, it was outperformed for the class $\leftrightsquigarrow$ by the “Mult.” model. Moreover, it can be seen that the classification task for the “duplicate” class and especially the $\leftrightsquigarrow$ class seems to be the most challenging one for the models. This is indicated by the better models having a considerable higher recall than precision for this class. Therefore, there is still room for improvement.

6.2.2 Model Comparison

The element-wise subtraction had a profound impact on the models’ performance to predict issues of the classes $\leftsquigarrow$ and $\rightsquigarrow$. That can be seen by all of the models using an element-wise subtraction as part of the merging layer having a better $f1$ score for those classes than the “Mult.” or “Avg.” model. Nevertheless, the high $f1$ score for those classes for the “Sub.” model can be traced back to a comparably high recall. At the same time, the precision of the “Sub.” model for predicting the classes $\leftsquigarrow$, and $\rightsquigarrow$ is comparably poor, which indicates that the model was just focused on predicting one of those two classes. Therefore, it is beneficial to use a slightly more sophisticated model for this task, like the ones using the concatenation of the element-wise subtraction, average and multiplication in addition to either the topic distribution, cosine similarity or even the issue vectors themselves.

For detecting “unrelated” issues, our results show that an element-wise multiplication has a profound impact. This can be seen by comparing the models “Avg.” and “Sub.” with the rest, which uses element-wise multiplication within their merging layer. All of the classes using an element-wise multiplication have a precision of over 0.3 for the “unrelated” class, while the ones not using it reached a precision under 0.27. The “Avg.” class reached a comparably high recall for this class with about 0.53. Nevertheless, the recall for all classes using the element-wise multiplication was higher than 0.56 (only the “Concat.” and “Issue-vector” model were under 0.6, and some models reached even a recall over 0.7). The element-wise multiplication even has an slight advantage in predicting “duplicates”. This advantage, however, is minimal because the precision of the “Topic” model has a worse precision than 0.25 while the other models have a higher precision than 0.25 for identifying “dupicate” issues and are thus about 0.05 better than the “Avg.” and “Sub.” model. Also, except for the “CosConcat.” and the “Topic” model, the recall for those models is also significantly higher by being higher than 0.28 while the other classes are below 0.2. The cosine values without weighting might be responsible for the lower values because the “CosConcat.” and the “Topic” model are the only two models with an unweighted cosine value.

The cosine similarity of the topics seems to have a relatively low impact. Comparing the “concat.” model and the “Topic” model shows that using the topic similarity mostly leads to a better recall for the ⬿ and $\rightsquigarrow$ class. Also, it has a profound impact on identifying unrelated issues, as can be seen by the higher precision and recall compared to the “Concat.” model. Nevertheless, it is at the expense of the “duplicate” class and the “$\leftrightsquigarrow$” class. Here, the problem may be issue relations, where issues may be classified as “duplicate” or “$\leftrightsquigarrow$” (the two largest classes) while actually not being members of them as stated in Section 6.3.

The cosine impact of the document vectors themselves, on the other hand, had a profound impact on the classification. Both models, the “CosConcat.” one and the “UniCosConcat.” one reached the best macro $f1$ scores of all models. Comparing them to the other models shows that using the cosine similarity of the document vectors leads to a drastic increase in precision for the classes ⬿, $\rightsquigarrow$ and “unrelated”. Nevertheless, comparing the unweighted cosine similarity of the “CosConcat.” model with the “Concat.” model reveals that the “duplicate” class suffers under the unweighted cosine similarity due to its low recall. However, weighting the model (as with the “UniCosConcat.”) normalises the precision and enhances the recall. However, this is again at the cost of the recall of the $\leftrightsquigarrow$ class. Nevertheless, the precision of unweighted cosine similarity is still worse than with the plain multiplication. The reason for those issues is probably that the weighting of the cosine similarity can hardly adjust according to the document length. Therefore, the impact even after the weighting is either too big or too small depending on the often occurring words in a document.

Besides, a weighting of sorts also has a relatively positive impact on the classification of issue relations. This can be seen by the “SumConcat.” in comparison to the “Concat.” model. Except for a slight decrease in precision for the “duplicate” class, the precision increased for every class. Nevertheless, this does not seem to hold for the recall, which was worse for the “duplicate” as well as for the “$\leftrightsquigarrow$” class.

The unary vectors resulted of the “UniCosConcat.” model resulted in an overall better $f1$ score. However, the $\leftrightsquigarrow$ class suffered under it, as can be seen in comparison to the “CosConcat.” one. So applying a weighting of sorts always seems to have a negative impact on this class, which may be due to the reasons mentioned in Section 6.3.

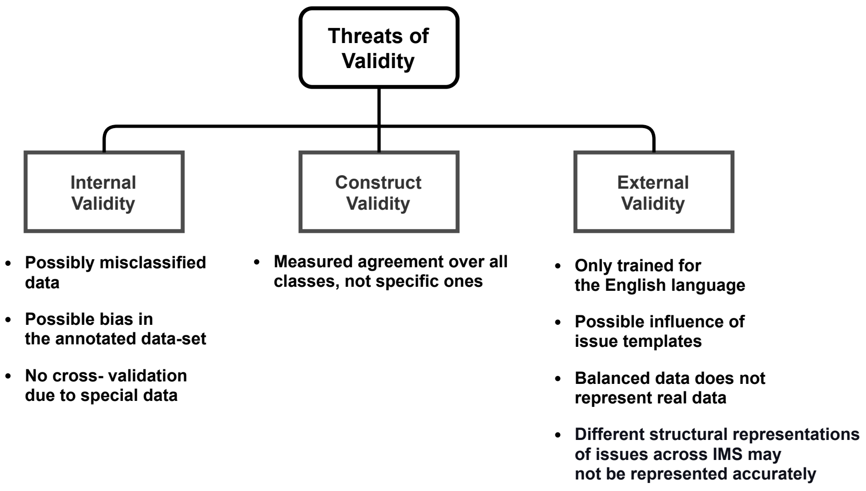

Threats of Validity
Internal Validity
Construct Validity
External Validity
• Possibly misclassified data
• Possible bias in the annotated data-set
• No cross- validation due to special data
• Measured agreement over all classes, not specific ones
• Only trained for the English language
• Possible influence of issue templates
• Balanced data does not represent real data
• Different structural representations of issues across IMS may not be represented accurately

Figure 6.1: Overview of threats to validity of this thesis' concept.

Also worth mentioning is that the simpler models like the "Sub." and the "Mult." model still outperformed the more sophisticated models in the classes at which they were good. Therefore, a more complex model does not necessarily lead to better $f1$ scores, as can be seen in the case of the $\leftrightsquigarrow$ class, for which all models have trouble getting a high $f1$ score.

6.2.3 Vectorisation Methods

From the vectorisation results presented in the results section, it can be deduced that the BOW methods, as well as the sentence encoder, lead to the best embeddings for this task. While the BOW vectorisation strategies work for many of our classifiers, the sentence encoder works best when the vectors have been previously transformed to euclidean vectors. Besides, the worst vectorisation results were the ones of the averaged embeddings as well as the uSIF ones. Nevertheless, the uSIF method was able to beat the average one constantly. In addition, we did not find significant differences in using pretrained GloVe embeddings over the fastText ones. The poor performance of those embeddings can be traced back to the relatively long document length, which presents a problem for most sentence embeddings obtained by word embeddings [CKS+17]. This was probably also a challenging task for the sentence encoder due to the document length.

6.3 Threats to Validity

This section discusses what might threaten the validity of our results. It is divided into three parts: The internal validity discussed in Section 6.3.1, the external validity discussed in Section 6.3.2, and the construct validity in Section 6.3.3. Figure 6.1 depicts and summarises the threats to the validity of this thesis' concept.

6.3.1 Internal Validity

One threat is that the annotators in GitHub presumably put a lot of effort into identifying bugs and duplicates, but the other relations did not receive much attention. That might have led to issues lying in the “$\leftrightsquigarrow$” class (where the direction is not known), which are probably members of the “$\leftsquigarrow$” or “$\rightsquigarrow$” classes, which in turn could have influenced the classifiers’ predictions (the same holds for the “duplicate” class).

In addition, the training, testing, and validation data sets are completely distinct from one another, and the “unrelated” issues are created from them. Therefore, we did not use cross-validation or a stratified split to train the models. However, this in itself could lead to the bias of us just seeing one result for the test set, which might not represent the data as a whole. We attempt to lessen the bias by using only the best vectoriser for each model on the test set to mitigate this. Nevertheless, this might not have nullified the bias.

Another threat to the internal validity is our own and the participant’s biases. In order to validate the data, two annotated data sets were created, which reached a substantial agreement according to Landis and Koch [LK77]. Nevertheless, the data might be influenced by opinion and interpretation. Also, we did not inquire experts of the repository, and therefore the relations extracted might not be the actual relations.

6.3.2 External Validity

A threat to the external validity is the use of issue templates within the issues scraped. We tried to reduce the bias posed by issue templates by using many different repositories. However, the risk might not be fully mitigated and therefore a generalisation for all issues irrespective of the utilised issue template is not possible. In addition, the issues were collected from GitHub, and because of its open-source nature, the comments of the users might not live up to the quality of the comments of closed repositories such as the ones in IMS such as Jira etc.

Another threat to the external validity is the use of the English language. For our issues, we collected issues that were written in English. Therefore, our models will unavoidably lead to worse results for other languages or ones using multilingual phrases.

A different threat is that the models were trained on the same amount of data samples for each class, so it is as “fair” as possible for each class. However, this leads to the model not reflecting the real world distribution, such as “unrelated” issues being more common than “duplicated” ones, when creating an IRG for a whole repository.

The last external validity threat we identified is the use of the GitHub issues themselves. GitHub issues may be structured differently across different IMS such as Jira, Redmine etc. Therefore, they may not be represented in our models the right way.

6.3.3 Construct Validity

For measuring the agreement with respect to the annotation between ours and the participant's, we used Cohen's kappa. However, we did not look further into which classes were mismatched in detail, so there might be a higher miss-match of one class over the others, which could have influenced our data set and is therefore not taken into account by the kappa value.

7 Conclusion

This chapter presents a summary of this thesis. In addition, it lists the advantages and limitations of our findings. Lastly, it lists the lessons learned and provides an outlook for the future work.

7.1 Summary

The research question this thesis aims to solve is "How can relationships between issues that possibly target independently managed software projects be detected?".

To solve this research question, a data set of issue relations had to be created. Those issue relations were gathered from multiple GitHub repositories listed in Table A.19. GitHub was chosen because of the ease of accessing issues as well as the sheer amount of different repositories publicly available. The relations themselves were extracted by searching for GitHub mentions within the comments of an issue. The class was extracted from the comments of the issues and the comment of the annotator linking those issues (the other comments sometimes disagreed with the annotator of the comment). Using multiple repositories and cross-component/cross-repository issues resulted in a data set of over 2000 issues relations, which target independently managed software projects. Moreover, we conducted an expert survey to evaluate the accuracy of the classes, which resulted in a kappa value of 0.7041, representing a "substantial agreement" according to Landis and Koch [LK77] and a "fair to good" agreement according to Fleiss. Using the issues and relations, we created the data by using the issue title, body and the comment bodies up to the specific mention, whose string representations we concatenated using line-breaks. This ensures that the relation type is not directly stated in the data while the data contains the most information possible. Moreover, we constructed unrelated issues to provide five possible categories a issue pair of the data can belong to ("$A \rightsquigarrow B$, $A \leftsquigarrow B$, $A \not\leftrightsquigarrow B$, $A \leftrightsquigarrow B$, $A\ duplicate\ B$" [1], where A and B are the issues provided.)

The models for the classification are split into two parts; A vectorisation part, a the classification model part. This leads to the ability of testing different vectorisation strategies for the different classifiers, The vectorisation strategies can be categorised into the BOW vectoriser, the word embedding vectoriser and the sentence embedding vectoriser. For the BOW vectoriser, different vocabulary lengths (512, 1024, 8192, 16384), the use of bigrams in addition to unigrams and the usage of a tf-idf weighting in comparison to a "count" vectoriser were tested. For the word embeddings, we tested two kinds of embeddings: pre-trained GloVe embeddings based $5.6B$ uncased tokens from Wikipedia 2014 and Gigaword[2], and fasttext embeddings. The word embeddings were

[1] "A depends on B" := $A \rightsquigarrow B$ "B depends on A" := $A \leftsquigarrow B$ "A and B do not have a relation" := $A \not\leftrightsquigarrow B$ "A and B have a mutual relation" := $A \leftrightsquigarrow B$ "A and B are duplicates" := $AduplicateB$

[2] https://github.com/RaRe-Technologies/gensim-data/releases/tag/glove-wiki-gigaword-100

then combined to form sentence embeddings using average weighting or uSIF. The last vectorisation method was the universal sentence encoder[3] which is trained and optimised for greater-than-word length texts.

The classification models have a siamese network-like structure. They take in two[4] inputs and then merge both vectors in the merging layer together to form a vector representation of both vectors. This vector representation representing both documents then gets passed into a fully connected dense softmax layer, which predicts the class the two issues belong to. For those classification models, we created nine different merging strategies and evaluated them against each other.

The baseline model was a model with an element-wise subtraction as a merging layer to combine both vectors. The second model tested was using an element-wise multiplication and the third an element-wise averaging. The fourth model uses a concatenation of the previous merging techniques for the merge. The fifth model follows this approach by using the cosine similarity in addition to the concatenation. The sixth model uses unary weighted vectors in addition to the cosine similarity and the concatenation. The seventh model is also based on the concatenation one but appends three new dimensions onto the final vector, the length of the element-wise subtraction, average and multiplication vectors. The eighth model appends the issue vectors themselves onto the concatenation. The ninth model has four inputs and uses the topic label predictions using a LDA topic model in addition to the vectorised documents. Then, it calculates the cosine distance between the topics and appends this onto the concatenation of the element-wise average, multiplication, subtraction.

Our results show that the BOW vectorisation strategies are able to outperform the sentence embeddings constructed by the word embeddings for every model. The sentence/document embeddings obtained by the universal sentence encoder were almost as good as the BOW approach, except for the unary weighted vectors, in which it outperformed BOW. The different merging strategies showed that it is indeed possible to detect related issues. Moreover, we concluded, that using an element wise subtraction allows the model to predict dependent relations. Further, using multiplication allows the model to distinguish related from unrelated issues, and using concatenation of the different models combined their capabilities into one model. However, the "specialised" models are better suited for their respective tasks. Using LDA- generated labels yields only a small advantage for classifying unrelated issues, and appending the issue vectors onto the concatenation helps it to find the direction of dependent issues. Using the summation and the cosine similarity helped the model to find the direction of a dependent relation and helps in distinguishing duplicates at the cost of the duplicate detection. Using unit vectors in addition to the cosine similarity and the concatenation led to the best results with a macro $f1$ score of about 0.44 whose only disadvantage is the classification of $\leftrightsquigarrow$.

Lastly, we created a microservice architecture consisting of the vectorisation service and the classification service to build a scalable solution for identifying the issue relations for multiple issues.

[3]https://tfhub.dev/google/universal-sentence-encoder/4
[4]except for the LDA one

7.2 Benefits

Using our software, software developers are able to find possibly related issues in a system and, therefore, are able to detect the origins of issues. Moreover, by splitting the software up into vectorisation and classification, the resource-heavy vectorisation process can be scaled up. This reduces the time to generate vector representations out of the issues before passing them into the classifier for comparison. Even though the results are not yet fully matured, it is still a massive leap over not having cross-component issues/issue origins identified and marked as such.

7.3 Limitations

A limitation of our model is that they are not good at the task of plain deduplication. In addition to that, our models were trained using English data. Therefore it does not support multiple languages or multilingual issues/data. Another limitation is that our models have been trained on GitHub issues and, therefore, are not necessarily representing issues from other IMS.

7.4 Lessons Learned

The task of identifying issue relations is a complex task for humans and machines alike, and there is still a lot of potential in this area of research. Also, in most GitHub repositories, the issues are just marked as “duplicated” or “related”, but the direction of the relationship is often unclear, which led to a bunch of relations we could not use. This could have been different with issues from different IMS such as Redmine or Jira, which also support multiple relation types. Also, it was surprising to see that a lot of the models have a comparably low effort on detecting dependent issues of the classes $\leftsquigarrow$ and $\rightsquigarrow$.

7.5 Future Work

In the future, we want to find out if the usernames or issue date difference have a positive impact on the predictions of our model, similar to Klein’s et al. publication for duplicate bug detection [KCK14]. Moreover, we want to try out different non-siamese model architectures to see how they are performing on that task. In addition, there are still other vectorisation methods worth testing, such as using ELMo [PNI+18] embeddings. Furthermore, we want to test our software in a production environment to evaluate the impact of using our microservice on the productivity of software developer teams. Also, we plan to get rid of the influence of certain issue templates by trying to filter them out. Moreover, the amount of issues is still not satisfactory. Therefore we want to try gathering more data.

Bibliography

[18] *RedmineIssues - Redmine*. https://www.redmine.org/projects/redmine/wiki/RedmineIssues. (Accessed on 12/17/2020). 27.12 2018 (cit. on pp. 2, 5).

[20a] *Link an issue | Jira Software Cloud | Atlassian Support*. https://support.atlassian.com/jira-software-cloud/docs/link-an-issue/. (Accessed on 12/17/2020). Nov. 2020 (cit. on pp. 2, 5).

[20b] *Mastering Issues · GitHub Guides*. https://guides.github.com/features/issues/. (Accessed on 12/17/2020). July 2020 (cit. on pp. 2, 5).

[AHS13] A. Alipour, A. Hindle, E. Stroulia. “A contextual approach towards more accurate duplicate bug report detection”. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2013, pp. 183–192 (cit. on p. 15).

[ALM16] S. Arora, Y. Liang, T. Ma. “A simple but tough-to-beat baseline for sentence embeddings”. In: (2016) (cit. on p. 9).

[BAPM15] S. R. Bowman, G. Angeli, C. Potts, C. D. Manning. *A large annotated corpus for learning natural language inference*. 2015. arXiv: 1508.05326 [cs.CL] (cit. on pp. 15, 16).

[BGJM17] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146. ISSN: 2307-387X (cit. on p. 9).

[BGL+93] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, R. Shah. “Signature verification using aßiamese”time delay neural network”. In: *Advances in neural information processing systems* 6 (1993), pp. 737–744 (cit. on p. 11).

[BJS+08] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, T. Zimmermann. “What makes a good bug report?” In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM. 2008, pp. 308–318 (cit. on p. 5).

[BMCG15] Y. Belinkov, M. Mohtarami, S. Cyphers, J. Glass. “VectorSLU: A continuous word vector approach to answer selection in community question answering systems”. In: *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*. 2015, pp. 282–287 (cit. on p. 9).

[CCD08] P. Cunningham, M. Cord, S. J. Delany. “Supervised learning”. In: *Machine learning techniques for multimedia*. Springer, 2008, pp. 21–49 (cit. on p. 6).

[CKS+17] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, A. Bordes. “Supervised learning of universal sentence representations from natural language inference data”. In: *arXiv preprint arXiv:1705.02364* (2017) (cit. on pp. 10, 28, 30, 42).

[CMS17] E. A. Corrêa Júnior, V. Q. Marinho, L. B. dos Santos. "NILC-USP at SemEval-2017 Task 4: A Multi-view Ensemble for Twitter Sentiment Analysis". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 611–615. DOI: 10.18653/v1/S17-2100. URL: https://www.aclweb.org/anthology/S17-2100 (cit. on p. 9).

[Coh60] J. Cohen. "A coefficient of agreement for nominal scales". In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46 (cit. on p. 7).

[CT+94] W. B. Cavnar, J. M. Trenkle, et al. "N-gram-based text categorization". In: *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*. Vol. 161175. Citeseer. 1994 (cit. on p. 8).

[CYK+18] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, R. Kurzweil. *Universal Sentence Encoder*. 2018. arXiv: 1803.11175 [cs.CL] (cit. on p. 10).

[CZL+17] Q. Chen, X. Zhu, Z.-H. Ling, S. Wei, H. Jiang, D. Inkpen. "Enhanced LSTM for Natural Language Inference". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017). DOI: 10.18653/v1/p17-1152. URL: http://dx.doi.org/10.18653/v1/P17-1152 (cit. on pp. 15, 16).

[DCLT19] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL] (cit. on p. 9).

[DDMR10] I. Dagan, B. Dolan, B. Magnini, D. Roth. "Recognizing Textual Entailment: Rational, Evaluation And Approaches". In: *Journal of Natural Language Engineering* 4 (Jan. 2010). URL: https://www.microsoft.com/en-us/research/publication/recognizing-textual-entailment-rational-evaluation-and-approaches/ (cit. on p. 15).

[Eth18] K. Ethayarajh. "Unsupervised random walk sentence embeddings: A strong but simple baseline". In: *Proceedings of The Third Workshop on Representation Learning for NLP*. 2018, pp. 91–100 (cit. on pp. 9, 10).

[FL15] M. Fowler, J. Lewis. "Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr?" In: (2015) (cit. on p. 5).

[GW97] W. Greve, D. Wentura. *Wissenschaftliche Beobachtung: eine Einführung*. Beltz, 1997, p. 111. ISBN: 3-621-27360-3. DOI: http://dx.doi.org/10.22028/D291-27212 (cit. on pp. 7, 20).

[HAS16] A. Hindle, A. Alipour, E. Stroulia. "A contextual approach towards more accurate duplicate bug report detection and ranking". In: *Empirical Software Engineering* 21.2 (2016), pp. 368–410 (cit. on p. 15).

[HBWP13] M. D. Hoffman, D. M. Blei, C. Wang, J. Paisley. "Stochastic variational inference." In: *Journal of Machine Learning Research* 14.5 (2013) (cit. on pp. 12, 13).

[HM21] T. Hasan, A. Matin. “Extract Sentiment from Customer Reviews: A Better Approach of TF-IDF and BOW-Based Text Classification Using N-Gram Technique”. In: *Proceedings of International Joint Conference on Advances in Computational Intelligence*. Ed. by M. S. Uddin, J. C. Bansal. Singapore: Springer Singapore, 2021, pp. 231–244. ISBN: 978-981-16-0586-4 (cit. on p. 8).

[HS15] M. Hossin, M. Sulaiman. “A review on evaluation metrics for data classification evaluations”. In: *International Journal of Data Mining & Knowledge Management Process* 5.2 (2015), p. 1 (cit. on pp. 6, 13).

[HTF09] T. Hastie, R. Tibshirani, J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009 (cit. on pp. 6, 23).

[IMBD15] M. Iyyer, V. Manjunatha, J. Boyd-Graber, H. Daumé III. “Deep unordered composition rivals syntactic methods for text classification”. 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. 1681–1691 (cit. on p. 10).

[IUA+20] Z. Imtiaz, M. Umer, M. Ahmad, S. Ullah, G. S. Choi, A. Mehmood. “Duplicate Questions Pair Detection Using Siamese MaLSTM”. In: *IEEE Access* 8 (2020), pp. 21932–21942. DOI: 10.1109/ACCESS.2020.2969041 (cit. on p. 11).

[Joa+99] T. Joachims et al. “Transductive inference for text classification using support vector machines”. In: *Icml*. Vol. 99. 1999, pp. 200–209 (cit. on pp. 5, 7, 8).

[Joa98] T. Joachims. “Text categorization with support vector machines: Learning with many relevant features”. In: *European conference on machine learning*. Springer. 1998, pp. 137–142 (cit. on p. 8).

[KB17] D. P. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (cit. on p. 37).

[KCK14] N. Klein, C. S. Corley, N. A. Kraft. “New features for duplicate bug detection”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. 2014, pp. 324–327 (cit. on pp. 15, 47).

[Kli+18] R. Klinger et al. “An analysis of annotated corpora for emotion classification in text”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. 2018, pp. 2104–2119 (cit. on pp. 6, 20).

[Kri05] D. Kriesel. *Ein kleiner Überblick über Neuronale Netze*. dkriesel, 2005. URL: http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf (cit. on p. 11).

[LEN14] Z. C. Lipton, C. Elkan, B. Naryanaswamy. “Optimal thresholding of classifiers to maximize F1 measure”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2014, pp. 225–239 (cit. on p. 14).

[LK77] J. R. Landis, G. G. Koch. “The Measurement of Observer Agreement for Categorical Data”. In: *Biometrics* 33.1 (1977), pp. 159–174. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2529310 (cit. on pp. 7, 20, 43, 45).

[LM14] Q. Le, T. Mikolov. “Distributed representations of sentences and documents”. In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196 (cit. on p. 10).

[Lov68] J. B. Lovins. “Development of a stemming algorithm.” In: *Mech. Transl. Comput. Linguistics* 11.1-2 (1968), pp. 22–31 (cit. on p. 8).

[Man08] C. D. Manning. *Introduction to Information Retrieval*. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf(visited 2021-06-03). London: Cambridge University Press, 2008. ISBN: 0521865719 (cit. on pp. 5, 7, 10, 20).

[MCCD13] T. Mikolov, K. Chen, G. Corrado, J. Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL] (cit. on p. 9).

[MP18] M. M. Mirończuk, J. Protasiewicz. “A recent overview of the state-of-the-art elements of text classification”. In: *Expert Systems with Applications* 106 (2018), pp. 36–54. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.03.058. URL: https://www.sciencedirect.com/science/article/pii/S095741741830215X (cit. on p. 6).

[MRS08] C. D. Manning, P. Raghavan, H. Schutze. *Introduction to Information Retrieval*. https://nlp.stanford.edu/IR-book/pdf/06vect.pdf. (Accessed on 11/26/2020). 2008 (cit. on p. 10).

[MSC+13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. “Distributed representations of words and phrases and their compositionality”. In: *arXiv preprint arXiv:1310.4546* (2013) (cit. on p. 9).

[MWC19] C. Malaviya, S. Wu, R. Cotterell. “A simple joint model for improved contextual neural lemmatization”. In: *arXiv preprint arXiv:1904.02306* (2019) (cit. on p. 8).

[New21] S. Newman. *Building microservices*. Ö’Reilly Media, Inc.”, 2021 (cit. on p. 5).

[NNN+12] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, C. Sun. “Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling”. In: (2012) (cit. on pp. 15, 31).

[OB19] J. Opitz, S. Burst. “Macro f1 and macro f1”. In: *arXiv preprint arXiv:1911.03347* (2019) (cit. on p. 14).

[PNI+18] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer. “Deep contextualized word representations”. In: *Proc. of NAACL*. 2018 (cit. on p. 47).

[PSM14] J. Pennington, R. Socher, C. D. Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543 (cit. on p. 9).

[PTDU16] A. P. Parikh, O. Täckström, D. Das, J. Uszkoreit. *A Decomposable Attention Model for Natural Language Inference*. 2016. arXiv: 1606.01933 [cs.CL] (cit. on p. 15).

[RBH+16] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, K. Krogmann. *Modeling and simulating software architectures: The Palladio approach*. MIT Press, 2016 (cit. on p. 5).

[RG19] N. Reimers, I. Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *arXiv preprint arXiv:1908.10084* (2019) (cit. on pp. 10, 12, 27, 30).

[Ric17] C. Richardson. *Microservice Patterns MEAP*. Manning, 2017 (cit. on p. 5).

[SBB20] S. Speth, U. Breitenbücher, S. Becker. "Gropius—A Tool for Managing Cross-component Issues". In: *European Conference on Software Architecture*. Springer. 2020, pp. 82–94 (cit. on pp. 2, 5).

[SBB21] S. Speth, S. Becker, U. Breitenbücher. "Cross-Component Issue Metamodel and Modelling Language." In: *CLOSER*. 2021, pp. 304–311 (cit. on p. 2).

[SGM02] C. Szyperski, D. Gruntz, S. Murer. *Component software: beyond object-oriented programming*. Pearson Education, 2002 (cit. on p. 5).

[Tom18] A. Tomar. *Topic modeling using Latent Dirichlet Allocation(LDA) and Gibbs Sampling explained! | by Ankur Tomar | Analytics Vidhya | Medium*. https://medium.com/analytics-vidhya/topic-modeling-using-lda-and-gibbs-sampling-explained-49d49b3d1045. (Accessed on 07/13/2021). Nov. 2018 (cit. on p. 13).

[Ull11] J. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011 (cit. on p. 8).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017) (cit. on p. 10).

All links were last followed on August 20, 2021.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.246	0.3433	0.3028	0.3209	0.2151	0.1493	0.2086	0.2164	0.1468	0.1194
tfIdf_{512}	0.3524	0.5522	0.4083	0.6642	0.2388	0.1194	0.2308	0.2015	0.2069	0.0896
$\text{count}_{512,Un+Bigram}$	0.1769	0.194	0.3	0.2463	0.2406	0.2388	0.2308	0.2463	0.2482	0.2537
$\text{tfIdf}_{512,Un+Bigram}$	0.3397	0.5299	0.3349	0.5373	0.2167	0.097	0.2358	0.2164	0.2222	0.1045
count_{1024}	0.3282	0.3209	0.2177	0.2388	0.1504	0.1269	0.2416	0.2687	0.2692	0.2612
tfIdf_{1024}	0.3691	0.6418	0.3818	0.6269	0.2063	0.097	0.2727	0.2239	0.3409	0.1119
$\text{count}_{1024,Un+Bigram}$	0.2222	0.2239	0.3642	0.4104	0.156	0.1642	0.2019	0.1567	0.2086	0.2164
$\text{tfIdf}_{1024,Un+Bigram}$	0.3409	0.5597	0.3934	0.6194	0.1739	0.0896	0.2727	0.2239	0.2333	0.1045
count_{8192}	0.268	0.306	0.3442	0.3955	0.2439	0.1493	0.2083	0.2239	0.1898	0.194
tfIdf_{8192}	0.3611	0.6791	0.3957	0.6791	0.275	0.0821	0.2451	0.1866	0.2174	0.0746
$\text{count}_{8192,Un+Bigram}$	0.3099	0.3284	0.1987	0.2239	0.1887	0.1493	0.1824	0.2313	0.2871	0.2164
$\text{tfIdf}_{8192,Un+Bigram}$	0.3438	0.5746	0.3632	0.6045	0.2535	0.1343	0.2642	0.209	0.3043	0.1045
count_{16384}	0.3377	0.3806	0.2714	0.2836	0.2394	0.1269	0.1895	0.2164	0.1484	0.1716
tfIdf_{16384}	0.3684	0.6791	0.3766	0.6493	0.2444	0.0821	0.2	0.1418	0.25	0.097
$\text{count}_{16384,Un+Bigram}$	0.2026	0.2313	0.1697	0.209	0.2121	0.1567	0.2013	0.2313	0.1616	0.1194
$\text{tfIdf}_{16384,Un+Bigram}$	0.362	0.597	0.3923	0.6119	0.1757	0.097	0.2455	0.2015	0.3214	0.1343
ft_{avg}	0.3021	0.4328	0.2723	0.4552	0.1636	0.0672	0.2131	0.194	0.1818	0.1045
glove_{avg}	0.2922	0.4776	0.2675	0.4552	0.1493	0.0746	0.2323	0.1716	0.1754	0.0746
ft_{usif}	0.3081	0.4552	0.2406	0.3358	0.1395	0.0896	0.2358	0.2164	0.25	0.1418
glove_{usif}	0.3037	0.4851	0.2949	0.4776	0.1688	0.097	0.1591	0.1045	0.2162	0.1194
sentence_{512}	0.3911	0.791	0.3961	0.7537	0.2093	0.0672	0.2121	0.1045	0.2286	0.0597

Table A.1: Precision and Recall measures for the different Sub vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.2866	0.3116	0.1762	0.2125	0.1317	0.2237
tfIdf_{512}	0.4302	0.5057	0.1592	0.2151	0.125	0.287
$\text{count}_{512,Un+Bigram}$	0.1851	0.2705	0.2397	0.2383	0.2509	0.2369
$\text{tfIdf}_{512,Un+Bigram}$	0.414	0.4126	0.134	0.2257	0.1421	0.2657
count_{1024}	0.3245	0.2278	0.1377	0.2544	0.2652	0.2419
tfIdf_{1024}	0.4687	0.4746	0.132	0.2459	0.1685	0.2979
$\text{count}_{1024,Un+Bigram}$	0.223	0.386	0.16	0.1765	0.2125	0.2316
$\text{tfIdf}_{1024,Un+Bigram}$	0.4237	0.4812	0.1182	0.2459	0.1443	0.2827
count_{8192}	0.2857	0.3681	0.1852	0.2158	0.1919	0.2493
tfIdf_{8192}	0.4715	0.5	0.1264	0.2119	0.1111	0.2842
$\text{count}_{8192,Un+Bigram}$	0.3188	0.2105	0.1667	0.2039	0.2468	0.2293
$\text{tfIdf}_{8192,Un+Bigram}$	0.4302	0.4538	0.1756	0.2333	0.1556	0.2897
count_{16384}	0.3579	0.2774	0.1659	0.2021	0.1592	0.2325
tfIdf_{16384}	0.4777	0.4767	0.1229	0.1659	0.1398	0.2766
$\text{count}_{16384,Un+Bigram}$	0.216	0.1873	0.1803	0.2153	0.1373	0.1872
$\text{tfIdf}_{16384,Un+Bigram}$	0.4507	0.4781	0.125	0.2213	0.1895	0.2929
ft_{avg}	0.3558	0.3408	0.0952	0.2031	0.1327	0.2255
glove_{avg}	0.3626	0.337	0.0995	0.1974	0.1047	0.2202
ft_{usif}	0.3675	0.2804	0.1091	0.2257	0.181	0.2327
glove_{usif}	0.3736	0.3647	0.1232	0.1261	0.1538	0.2283
sentence_{512}	0.5235	0.5193	0.1017	0.14	0.0947	0.2758

Table A.2: $f1$ measures for the different Sub vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.1583	0.1418	0.2338	0.1343	0.1967	0.1791	0.3275	0.694	0.2388	0.1194
tfIdf_{512}	0.2899	0.1493	0.275	0.0821	0.2151	0.2761	0.4835	0.6567	0.3961	0.6119
$\text{count}_{512,Un+Bigram}$	0.2576	0.1269	0.2591	0.3731	0.2627	0.2313	0.3833	0.6493	0.2576	0.1269
$\text{tfIdf}_{512,Un+Bigram}$	0.2195	0.0672	0.2973	0.1642	0.2118	0.2687	0.4241	0.709	0.3851	0.4627
count_{1024}	0.1933	0.1716	0.2586	0.1119	0.2222	0.2388	0.3939	0.6791	0.2627	0.2313
tfIdf_{1024}	0.2564	0.1493	0.2708	0.097	0.214	0.3881	0.5673	0.7239	0.3308	0.3209
$\text{count}_{1024,Un+Bigram}$	0.2609	0.0448	0.2201	0.2612	0.2737	0.194	0.3477	0.7239	0.3333	0.2836
$\text{tfIdf}_{1024,Un+Bigram}$	0.25	0.1269	0.2955	0.097	0.2414	0.3134	0.5185	0.7313	0.3795	0.5522
count_{8192}	0.2903	0.2015	0.23	0.1716	0.22	0.2463	0.4206	0.791	0.2667	0.1493
tfIdf_{8192}	0.2154	0.1045	0.3111	0.1045	0.2275	0.4328	0.5538	0.7687	0.3529	0.3134
$\text{count}_{8192,Un+Bigram}$	0.2252	0.1866	0.2466	0.1343	0.2292	0.2463	0.4487	0.8806	0.2785	0.1642
$\text{tfIdf}_{8192,Un+Bigram}$	0.2	0.0373	0.2785	0.1642	0.2159	0.4254	0.5964	0.7388	0.3382	0.3433
count_{16384}	0.3034	0.2015	0.2105	0.1194	0.1872	0.2836	0.5116	0.8209	0.2644	0.1716
tfIdf_{16384}	0.2778	0.0373	0.2444	0.1642	0.2231	0.403	0.5303	0.7836	0.3852	0.3507
$\text{count}_{16384,Un+Bigram}$	0.3514	0.194	0.1839	0.1194	0.2323	0.2687	0.4226	0.8358	0.3596	0.2388
$\text{tfIdf}_{16384,Un+Bigram}$	0.2	0.0075	0.2672	0.2313	0.2211	0.3284	0.5549	0.7537	0.3036	0.3806
ft_{avg}	0.2955	0.097	0.2439	0.0746	0.2034	0.0896	0.272	0.7836	0.2786	0.291
glove_{avg}	0.0	0.0	0.1818	0.0299	0.2041	0.0746	0.27	0.2015	0.1864	0.694
ft_{usif}	0.32	0.0597	0.2255	0.1716	0.2889	0.097	0.2861	0.7836	0.3588	0.3507
glove_{usif}	0.2105	0.0299	0.2	0.0299	0.2742	0.1269	0.402	0.597	0.2378	0.6567
sentence_{512}	0.0	0.0	0.1732	0.2313	0.1719	0.2836	0.8769	0.4254	0.1527	0.2313

Table A.3: Precision and Recall measures for the different Mult vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.1496	0.1706	0.1875	0.445	0.1592	0.2224
tfIdf_{512}	0.197	0.1264	0.2418	0.557	0.4809	0.3206
$\text{count}_{512,Un+Bigram}$	0.17	0.3058	0.246	0.482	0.17	0.2748
$\text{tfIdf}_{512,Un+Bigram}$	0.1029	0.2115	0.2368	0.5307	0.4203	0.3004
count_{1024}	0.1818	0.1563	0.2302	0.4986	0.246	0.2626
tfIdf_{1024}	0.1887	0.1429	0.2759	0.6361	0.3258	0.3139
$\text{count}_{1024,Un+Bigram}$	0.0764	0.2389	0.2271	0.4697	0.3065	0.2637
$\text{tfIdf}_{1024,Un+Bigram}$	0.1683	0.1461	0.2727	0.6068	0.4498	0.3287
count_{8192}	0.2379	0.1966	0.2324	0.5492	0.1914	0.2815
tfIdf_{8192}	0.1407	0.1564	0.2982	0.6438	0.332	0.3142
$\text{count}_{8192,Un+Bigram}$	0.2041	0.1739	0.2374	0.5945	0.2066	0.2833
$\text{tfIdf}_{8192,Un+Bigram}$	0.0629	0.2066	0.2864	0.66	0.3407	0.3113
count_{16384}	0.2422	0.1524	0.2255	0.6304	0.2081	0.2917
tfIdf_{16384}	0.0658	0.1964	0.2872	0.6325	0.3672	0.3098
$\text{count}_{16384,Un+Bigram}$	0.25	0.1448	0.2491	0.5614	0.287	0.2985
$\text{tfIdf}_{16384,Un+Bigram}$	0.0144	0.248	0.2643	0.6392	0.3377	0.3007
ft_{avg}	0.1461	0.1143	0.1244	0.4038	0.2847	0.2147
glove_{avg}	0.0	0.0513	0.1093	0.2308	0.2938	0.137
ft_{usif}	0.1006	0.1949	0.1453	0.4192	0.3547	0.2429
glove_{usif}	0.0523	0.0519	0.1735	0.4805	0.3492	0.2215
sentence_{512}	0.0	0.1981	0.2141	0.5729	0.184	0.2338

Table A.4: $f1$ measures for the different Mult vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.1829	0.1119	0.1538	0.0597	0.1484	0.1418	0.204	0.5299	0.2333	0.1045
tfIdf_{512}	0.0	0.0	0.087	0.0149	0.1633	0.0597	0.2069	0.8507	0.2326	0.0746
$\text{count}_{512,Un+Bigram}$	0.2268	0.1642	0.1683	0.1269	0.1383	0.097	0.2082	0.5299	0.2703	0.0746
$\text{tfIdf}_{512,Un+Bigram}$	0.2308	0.0224	0.1429	0.0075	0.2241	0.097	0.2098	0.8657	0.1538	0.0448
count_{1024}	0.2212	0.1866	0.0	0.0	0.2353	0.1791	0.2252	0.6269	0.3387	0.1567
tfIdf_{1024}	0.0714	0.0075	0.3333	0.0149	0.1525	0.0672	0.2115	0.8806	0.2424	0.0597
$\text{count}_{1024,Un+Bigram}$	0.3265	0.2388	0.1972	0.1045	0.1769	0.194	0.2727	0.5149	0.2277	0.1716
$\text{tfIdf}_{1024,Un+Bigram}$	0.0	0.0	0.0714	0.0075	0.1803	0.0821	0.212	0.8731	0.2439	0.0746
count_{8192}	0.24	0.0896	0.1532	0.1269	0.2034	0.1791	0.231	0.6119	0.1667	0.0448
tfIdf_{8192}	0.1	0.0149	0.1333	0.0149	0.2022	0.1343	0.2058	0.7985	0.1538	0.0299
$\text{count}_{8192,Un+Bigram}$	0.2241	0.097	0.32	0.0597	0.165	0.2463	0.253	0.6194	0.2203	0.097
$\text{tfIdf}_{8192,Un+Bigram}$	0.1176	0.0149	0.0	0.0	0.1959	0.1418	0.2199	0.8731	0.087	0.0149
count_{16384}	0.1647	0.1045	0.1304	0.0224	0.1857	0.097	0.2425	0.7836	0.2034	0.0896
tfIdf_{16384}	0.1724	0.0373	0.1667	0.0299	0.175	0.1045	0.2222	0.8507	0.1667	0.0299
$\text{count}_{16384,Un+Bigram}$	0.1739	0.0299	0.2542	0.1119	0.2	0.1791	0.2356	0.7313	0.2692	0.1045
$\text{tfIdf}_{16384,Un+Bigram}$	0.0	0.0	0.1429	0.0075	0.2449	0.1791	0.2144	0.8657	0.2353	0.0299
ft_{avg}	0.2407	0.097	0.1525	0.0672	0.3333	0.1269	0.1827	0.4254	0.1598	0.2313
glove_{avg}	0.1818	0.0896	0.125	0.0149	0.1531	0.1119	0.2021	0.2836	0.2053	0.4627
ft_{usif}	0.2143	0.0896	0.1875	0.0448	0.2295	0.1045	0.2107	0.5896	0.2123	0.2313
glove_{usif}	0.1429	0.0299	0.2344	0.1119	0.2245	0.1642	0.2359	0.5	0.1939	0.2836
sentence_{512}	0.0	0.0	0.0	0.0	0.2791	0.0896	0.2196	0.8881	0.1571	0.0821

Table A.5: Precision and Recall measures for the different Avg vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.1389	0.086	0.145	0.2946	0.1443	0.1618
tfIdf_{512}	0.0	0.0255	0.0874	0.3328	0.113	0.1117
$\text{count}_{512,Un+Bigram}$	0.1905	0.1447	0.114	0.2989	0.117	0.173
$\text{tfIdf}_{512,Un+Bigram}$	0.0408	0.0142	0.1354	0.3377	0.0694	0.1195
count_{1024}	0.2024	0.0	0.2034	0.3314	0.2143	0.1903
tfIdf_{1024}	0.0135	0.0286	0.0933	0.341	0.0958	0.1144
$\text{count}_{1024,Un+Bigram}$	0.2759	0.1366	0.1851	0.3566	0.1957	0.23
$\text{tfIdf}_{1024,Un+Bigram}$	0.0	0.0135	0.1128	0.3411	0.1143	0.1163
count_{8192}	0.1304	0.1388	0.1905	0.3354	0.0706	0.1731
tfIdf_{8192}	0.026	0.0268	0.1614	0.3272	0.05	0.1183
$\text{count}_{8192,Un+Bigram}$	0.1354	0.1006	0.1976	0.3593	0.1347	0.1855
$\text{tfIdf}_{8192,Un+Bigram}$	0.0265	0.0	0.1645	0.3514	0.0255	0.1136
count_{16384}	0.1279	0.0382	0.1275	0.3704	0.1244	0.1577
tfIdf_{16384}	0.0613	0.0506	0.1308	0.3524	0.0506	0.1291
$\text{count}_{16384,Un+Bigram}$	0.051	0.1554	0.189	0.3564	0.1505	0.1805
$\text{tfIdf}_{16384,Un+Bigram}$	0.0	0.0142	0.2069	0.3437	0.053	0.1236
ft_{avg}	0.1383	0.0933	0.1838	0.2556	0.189	0.172
glove_{avg}	0.12	0.0267	0.1293	0.236	0.2844	0.1593
ft_{usif}	0.1263	0.0723	0.1436	0.3104	0.2214	0.1748
glove_{usif}	0.0494	0.1515	0.1897	0.3206	0.2303	0.1883
sentence_{512}	0.0	0.0	0.1356	0.3521	0.1078	0.1191

Table A.6: $f1$ measures for the different Avg vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count$_{512}$	0.4133	0.2313	0.3858	0.3657	0.2595	0.306	0.3873	0.5896	0.2453	0.194
tfIdf$_{512}$	0.3553	0.2015	0.4235	0.2687	0.26	0.097	0.2225	0.6194	0.2791	0.1791
count$_{512,Un+Bigram}$	0.3804	0.2612	0.3425	0.1866	0.2061	0.2537	0.3476	0.6045	0.2897	0.2313
tfIdf$_{512,Un+Bigram}$	0.3088	0.1567	0.3864	0.2537	0.2143	0.0896	0.2262	0.6567	0.3188	0.1642
count$_{1024}$	0.2889	0.194	0.31	0.2313	0.2407	0.194	0.3443	0.6269	0.2422	0.2313
tfIdf$_{1024}$	0.2897	0.2313	0.3933	0.2612	0.1296	0.0522	0.2066	0.5597	0.1404	0.0597
count$_{1024,Un+Bigram}$	0.3103	0.2687	0.3667	0.1642	0.2199	0.2313	0.331	0.7015	0.3188	0.1642
tfIdf$_{1024,Un+Bigram}$	0.2874	0.1866	0.4318	0.2836	0.1897	0.0821	0.2114	0.5821	0.25	0.1269
count$_{8192}$	0.3458	0.2761	0.4066	0.2761	0.2025	0.2463	0.4241	0.709	0.2235	0.1418
tfIdf$_{8192}$	0.3421	0.291	0.4811	0.3806	0.1724	0.0746	0.1955	0.5224	0.1471	0.0373
count$_{8192,Un+Bigram}$	0.3723	0.2612	0.3273	0.2687	0.2024	0.1269	0.3137	0.7164	0.25	0.1418
tfIdf$_{8192,Un+Bigram}$	0.3039	0.2313	0.3883	0.2985	0.1719	0.0821	0.2039	0.5522	0.2632	0.0746
count$_{16384}$	0.3762	0.2836	0.3659	0.3358	0.1917	0.1716	0.417	0.7313	0.2198	0.1493
tfIdf$_{16384}$	0.375	0.3358	0.463	0.3731	0.1754	0.0746	0.1953	0.5	0.2143	0.0672
count$_{16384,Un+Bigram}$	0.378	0.2313	0.3492	0.3284	0.2162	0.1791	0.4024	0.7388	0.2667	0.209
tfIdf$_{16384,Un+Bigram}$	0.3404	0.2388	0.4815	0.291	0.1463	0.0896	0.2021	0.5672	0.1622	0.0448
ft$_{avg}$	0.2262	0.1418	0.2143	0.1343	0.1951	0.0597	0.2556	0.6866	0.2277	0.1716
glove$_{avg}$	0.3241	0.2612	0.2817	0.2985	0.1304	0.0448	0.2312	0.2985	0.1841	0.2761
ft$_{usif}$	0.2576	0.1269	0.1667	0.0896	0.1154	0.0224	0.2892	0.7985	0.3603	0.3657
glove$_{usif}$	0.3333	0.2612	0.3036	0.2537	0.2128	0.0746	0.2707	0.4627	0.1977	0.2612
sentence$_{512}$	0.424	0.3955	0.431	0.3731	0.3793	0.0821	0.2107	0.5299	0.1905	0.0896

Table A.7: Precision and Recall measures for the different Concat vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count$_{512}$	0.2967	0.3755	0.2808	0.4675	0.2167	0.3274
tfIdf$_{512}$	0.2571	0.3288	0.1413	0.3274	0.2182	0.2546
count$_{512,Un+Bigram}$	0.3097	0.2415	0.2274	0.4414	0.2573	0.2955
tfIdf$_{512,Un+Bigram}$	0.2079	0.3063	0.1263	0.3365	0.2167	0.2387
count$_{1024}$	0.2321	0.265	0.2149	0.4444	0.2366	0.2786
tfIdf$_{1024}$	0.2573	0.3139	0.0745	0.3018	0.0838	0.2063
count$_{1024,Un+Bigram}$	0.288	0.2268	0.2255	0.4498	0.2167	0.2814
tfIdf$_{1024,Un+Bigram}$	0.2262	0.3423	0.1146	0.3101	0.1683	0.2323
count$_{8192}$	0.3071	0.3289	0.2222	0.5307	0.1735	0.3125
tfIdf$_{8192}$	0.3145	0.425	0.1042	0.2846	0.0595	0.2376
count$_{8192,Un+Bigram}$	0.307	0.2951	0.156	0.4364	0.181	0.2751
tfIdf$_{8192,Un+Bigram}$	0.2627	0.3376	0.1111	0.2978	0.1163	0.2251
count$_{16384}$	0.3234	0.3502	0.1811	0.5312	0.1778	0.3127
tfIdf$_{16384}$	0.3543	0.4132	0.1047	0.2809	0.1023	0.2511
count$_{16384,Un+Bigram}$	0.287	0.3385	0.1959	0.5211	0.2343	0.3154
tfIdf$_{16384,Un+Bigram}$	0.2807	0.3628	0.1111	0.298	0.0702	0.2246
ft$_{avg}$	0.1743	0.1651	0.0914	0.3725	0.1957	0.1998
glove$_{avg}$	0.2893	0.2899	0.0667	0.2606	0.2209	0.2255
ft$_{usif}$	0.17	0.1165	0.0375	0.4246	0.363	0.2223
glove$_{usif}$	0.2929	0.2764	0.1105	0.3416	0.2251	0.2493
sentence$_{512}$	0.4093	0.4	0.135	0.3015	0.1218	0.2735

Table A.8: $f1$ measures for the different Concat vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.3426	0.2761	0.338	0.1791	0.2	0.2537	0.2983	0.5299	0.1928	0.1194
tfIdf_{512}	0.3372	0.2164	0.4176	0.2836	0.2449	0.0896	0.275	0.6567	0.3468	0.3209
$\text{count}_{512,Un+Bigram}$	0.4086	0.2836	0.3438	0.2463	0.1948	0.2239	0.354	0.597	0.2178	0.1642
$\text{tfIdf}_{512,Un+Bigram}$	0.3151	0.1716	0.3977	0.2612	0.2281	0.097	0.2578	0.6791	0.3434	0.2537
count_{1024}	0.296	0.2761	0.3415	0.3134	0.2418	0.2761	0.4	0.6418	0.1852	0.0746
tfIdf_{1024}	0.307	0.2612	0.3556	0.2388	0.2206	0.1119	0.2872	0.6343	0.3039	0.2313
$\text{count}_{1024,Un+Bigram}$	0.3368	0.2388	0.3671	0.2164	0.2541	0.3507	0.3532	0.5299	0.3273	0.2687
$\text{tfIdf}_{1024,Un+Bigram}$	0.2841	0.1866	0.382	0.2537	0.2297	0.1269	0.2701	0.6269	0.3148	0.2537
count_{8192}	0.3425	0.3731	0.4211	0.2985	0.2155	0.1866	0.4306	0.694	0.2268	0.1642
tfIdf_{8192}	0.3492	0.3284	0.4825	0.4104	0.2394	0.1269	0.2635	0.5448	0.3293	0.2015
$\text{count}_{8192,Un+Bigram}$	0.396	0.2985	0.3566	0.3433	0.1953	0.1866	0.4163	0.7239	0.2025	0.1194
$\text{tfIdf}_{8192,Un+Bigram}$	0.2897	0.2313	0.375	0.2687	0.2278	0.1343	0.2484	0.5896	0.2571	0.1343
count_{16384}	0.3542	0.3806	0.4684	0.2761	0.2014	0.209	0.3824	0.6791	0.2143	0.1119
tfIdf_{16384}	0.3636	0.3284	0.4464	0.3731	0.2429	0.1269	0.2527	0.5299	0.3488	0.2239
$\text{count}_{16384,Un+Bigram}$	0.3434	0.2537	0.3778	0.2537	0.2772	0.209	0.3591	0.7985	0.2805	0.1716
$\text{tfIdf}_{16384,Un+Bigram}$	0.3333	0.2463	0.4773	0.3134	0.1959	0.1418	0.25	0.5746	0.2949	0.1716
ft_{avg}	0.2125	0.1269	0.2099	0.1269	0.2286	0.0597	0.2548	0.6866	0.2478	0.209
glove_{avg}	0.2857	0.2687	0.2358	0.1866	0.1552	0.0672	0.2535	0.2687	0.1891	0.3358
ft_{usif}	0.2895	0.1642	0.2055	0.1119	0.1786	0.0373	0.2877	0.7687	0.3778	0.3806
glove_{usif}	0.3053	0.2164	0.2871	0.2164	0.2069	0.0896	0.2892	0.4403	0.1981	0.3134
sentence_{512}	0.432	0.403	0.4262	0.3881	0.3143	0.0821	0.2305	0.5299	0.1625	0.097

Table A.9: Precision and Recall measures for the different CosConcat vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.3058	0.2341	0.2237	0.3817	0.1475	0.2586
tfIdf_{512}	0.2636	0.3378	0.1311	0.3877	0.3333	0.2907
$\text{count}_{512,Un+Bigram}$	0.3348	0.287	0.2083	0.4444	0.1872	0.2923
$\text{tfIdf}_{512,Un+Bigram}$	0.2222	0.3153	0.1361	0.3737	0.2918	0.2678
count_{1024}	0.2857	0.3268	0.2578	0.4928	0.1064	0.2939
tfIdf_{1024}	0.2823	0.2857	0.1485	0.3953	0.2627	0.2749
$\text{count}_{1024,Un+Bigram}$	0.2795	0.2723	0.2947	0.4239	0.2951	0.3131
$\text{tfIdf}_{1024,Un+Bigram}$	0.2252	0.3049	0.1635	0.3775	0.281	0.2704
count_{8192}	0.3571	0.3493	0.2	0.5314	0.1905	0.3257
tfIdf_{8192}	0.3385	0.4435	0.1659	0.3552	0.25	0.3106
$\text{count}_{8192,Un+Bigram}$	0.3404	0.3498	0.1908	0.5286	0.1502	0.312
$\text{tfIdf}_{8192,Un+Bigram}$	0.2573	0.313	0.169	0.3496	0.1765	0.2531
count_{16384}	0.3669	0.3474	0.2051	0.4892	0.1471	0.3111
tfIdf_{16384}	0.3451	0.4065	0.1667	0.3422	0.2727	0.3066
$\text{count}_{16384,Un+Bigram}$	0.2918	0.3036	0.2383	0.4954	0.213	0.3084
$\text{tfIdf}_{16384,Un+Bigram}$	0.2833	0.3784	0.1645	0.3484	0.217	0.2783
ft_{avg}	0.1589	0.1581	0.0947	0.3717	0.2267	0.202
glove_{avg}	0.2769	0.2083	0.0938	0.2609	0.2419	0.2164
ft_{usif}	0.2095	0.1449	0.0617	0.4187	0.3792	0.2428
glove_{usif}	0.2533	0.2468	0.125	0.3491	0.2428	0.2434
sentence_{512}	0.417	0.4062	0.1302	0.3213	0.1215	0.2792

Table A.10: $f1$ measures for the different CosConcat vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.3725	0.4254	0.4155	0.4403	0.1889	0.1269	0.3057	0.4403	0.3913	0.2687
tfIdf_{512}	0.4029	0.4179	0.4658	0.5075	0.1711	0.097	0.3866	0.5597	0.3652	0.3134
$\text{count}_{512,Un+Bigram}$	0.3622	0.3433	0.4898	0.3582	0.2	0.1194	0.2874	0.5448	0.3694	0.306
$\text{tfIdf}_{512,Un+Bigram}$	0.4365	0.4104	0.5294	0.4701	0.1791	0.0896	0.3306	0.597	0.3534	0.306
count_{1024}	0.4	0.3731	0.4062	0.3881	0.15	0.1119	0.3205	0.5597	0.3133	0.194
tfIdf_{1024}	0.4274	0.3731	0.4697	0.4627	0.2621	0.2015	0.3946	0.6567	0.4	0.2836
$\text{count}_{1024,Un+Bigram}$	0.3952	0.3657	0.4215	0.3806	0.1868	0.1269	0.3226	0.597	0.3721	0.2388
$\text{tfIdf}_{1024,Un+Bigram}$	0.4	0.3582	0.439	0.403	0.2041	0.1493	0.371	0.6119	0.3148	0.2537
count_{8192}	0.4257	0.3209	0.5303	0.2612	0.2275	0.2836	0.2996	0.6194	0.3559	0.1567
tfIdf_{8192}	0.4239	0.291	0.5185	0.3134	0.25	0.2985	0.3552	0.6866	0.4231	0.2463
$\text{count}_{8192,Un+Bigram}$	0.4713	0.306	0.6071	0.2537	0.2405	0.2836	0.3253	0.709	0.3247	0.1866
$\text{tfIdf}_{8192,Un+Bigram}$	0.4681	0.3284	0.5	0.306	0.2518	0.2612	0.3495	0.7537	0.4091	0.2015
count_{16384}	0.3846	0.1493	0.4655	0.2015	0.25	0.4254	0.2842	0.5896	0.3704	0.1493
tfIdf_{16384}	0.4177	0.2463	0.5263	0.2985	0.2515	0.306	0.3225	0.6642	0.3816	0.2164
$\text{count}_{16384,Un+Bigram}$	0.3768	0.194	0.5167	0.2313	0.2412	0.306	0.3065	0.709	0.3279	0.1493
$\text{tfIdf}_{16384,Un+Bigram}$	0.4576	0.2015	0.5395	0.306	0.2789	0.306	0.3234	0.7313	0.3294	0.209
ft_{avg}	0.25	0.2164	0.2241	0.194	0.2963	0.0597	0.253	0.4701	0.216	0.2612
glove_{avg}	0.3246	0.2761	0.2803	0.2761	0.0962	0.0373	0.2812	0.3358	0.2075	0.3284
ft_{usif}	0.2574	0.194	0.2604	0.1866	0.2292	0.0821	0.2824	0.5373	0.2588	0.3284
glove_{usif}	0.2903	0.2015	0.2593	0.209	0.1458	0.0522	0.3179	0.4627	0.2345	0.3955
sentence_{512}	0.4505	0.3731	0.44	0.4104	0.2737	0.194	0.5548	0.6418	0.2826	0.3881

Table A.11: Precision and Recall measures for the different UniCosConcat vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.3972	0.4275	0.1518	0.3609	0.3186	0.3312
tfIdf_{512}	0.4103	0.4857	0.1238	0.4573	0.3373	0.3629
$\text{count}_{512,Un+Bigram}$	0.3525	0.4138	0.1495	0.3763	0.3347	0.3254
$\text{tfIdf}_{512,Un+Bigram}$	0.4231	0.498	0.1194	0.4255	0.328	0.3588
count_{1024}	0.3861	0.3969	0.1282	0.4076	0.2396	0.3117
tfIdf_{1024}	0.3984	0.4662	0.2278	0.493	0.3319	0.3835
$\text{count}_{1024,Un+Bigram}$	0.3798	0.4	0.1511	0.4188	0.2909	0.3281
$\text{tfIdf}_{1024,Un+Bigram}$	0.378	0.4202	0.1724	0.462	0.281	0.3427
count_{8192}	0.366	0.35	0.2525	0.4039	0.2176	0.318
tfIdf_{8192}	0.3451	0.3907	0.2721	0.4682	0.3113	0.3575
$\text{count}_{8192,Un+Bigram}$	0.371	0.3579	0.2603	0.446	0.237	0.3344
$\text{tfIdf}_{8192,Un+Bigram}$	0.386	0.3796	0.2564	0.4775	0.27	0.3539
count_{16384}	0.2151	0.2812	0.3149	0.3835	0.2128	0.2815
tfIdf_{16384}	0.3099	0.381	0.2761	0.4341	0.2762	0.3355
$\text{count}_{16384,Un+Bigram}$	0.2562	0.3196	0.2697	0.4279	0.2051	0.2957
$\text{tfIdf}_{16384,Un+Bigram}$	0.2798	0.3905	0.2918	0.4485	0.2557	0.3333
ft_{avg}	0.232	0.208	0.0994	0.329	0.2365	0.221
glove_{avg}	0.2984	0.2782	0.0538	0.3061	0.2543	0.2382
ft_{usif}	0.2213	0.2174	0.1209	0.3702	0.2895	0.2439
glove_{usif}	0.2379	0.2314	0.0769	0.3769	0.2944	0.2435
sentence_{512}	0.4082	0.4247	0.2271	0.5952	0.327	0.3964

Table A.12: $f1$ measures for the different UniCosConcat vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.4262	0.194	0.2879	0.1418	0.2174	0.2612	0.3208	0.5746	0.3028	0.3209
tfIdf_{512}	0.3841	0.3955	0.4621	0.4552	0.2368	0.0672	0.2943	0.5821	0.4124	0.2985
$\text{count}_{512,Un+Bigram}$	0.413	0.4254	0.4068	0.1791	0.2188	0.209	0.2863	0.5448	0.2889	0.194
$\text{tfIdf}_{512,Un+Bigram}$	0.3629	0.3358	0.4545	0.3731	0.2895	0.0821	0.2623	0.597	0.3656	0.2537
count_{1024}	0.3579	0.2537	0.3488	0.2239	0.2328	0.3284	0.4269	0.5448	0.2791	0.2687
tfIdf_{1024}	0.3611	0.3881	0.469	0.5075	0.2273	0.0746	0.29	0.5821	0.3824	0.194
$\text{count}_{1024,Un+Bigram}$	0.3219	0.3507	0.4468	0.1567	0.2586	0.2239	0.3713	0.6567	0.2016	0.1866
$\text{tfIdf}_{1024,Un+Bigram}$	0.3852	0.3507	0.4365	0.4104	0.2174	0.0746	0.2687	0.5896	0.3415	0.209
count_{8192}	0.4175	0.3209	0.3981	0.3209	0.2319	0.2388	0.4404	0.7164	0.2913	0.2239
tfIdf_{8192}	0.4014	0.4254	0.4603	0.4328	0.2698	0.1269	0.2857	0.5522	0.375	0.2239
$\text{count}_{8192,Un+Bigram}$	0.3509	0.2985	0.3711	0.2687	0.1884	0.194	0.3731	0.7239	0.2623	0.1194
$\text{tfIdf}_{8192,Un+Bigram}$	0.3704	0.3731	0.4508	0.4104	0.2581	0.1194	0.25	0.5448	0.2712	0.1194
count_{16384}	0.4396	0.2985	0.3763	0.2612	0.1908	0.2164	0.4424	0.7164	0.2479	0.2164
tfIdf_{16384}	0.4044	0.4104	0.4553	0.4179	0.3016	0.1418	0.2704	0.5448	0.359	0.209
$\text{count}_{16384,Un+Bigram}$	0.4909	0.2015	0.3871	0.3582	0.2047	0.194	0.3785	0.709	0.2212	0.1866
$\text{tfIdf}_{16384,Un+Bigram}$	0.3937	0.3731	0.4912	0.4179	0.2533	0.1418	0.2708	0.5597	0.3766	0.2164
ft_{avg}	0.2188	0.1045	0.2099	0.1269	0.1622	0.0448	0.2657	0.694	0.2464	0.2537
glove_{avg}	0.2906	0.2537	0.2632	0.1493	0.1475	0.0672	0.2289	0.2836	0.204	0.3806
ft_{usif}	0.2297	0.1269	0.2308	0.1119	0.1395	0.0448	0.3028	0.8134	0.3516	0.3358
glove_{usif}	0.34	0.2537	0.3084	0.2463	0.24	0.0896	0.2851	0.4851	0.2054	0.2836
sentence_{512}	0.4203	0.4328	0.4359	0.3806	0.3571	0.0746	0.2445	0.5821	0.1618	0.0821

Table A.13: Precision and Recall measures for the different SumConcat vectorisation configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.2667	0.19	0.2373	0.4118	0.3116	0.2835
tfIdf_{512}	0.3897	0.4586	0.1047	0.391	0.3463	0.3381
$\text{count}_{512,Un+Bigram}$	0.4191	0.2487	0.2137	0.3753	0.2321	0.2978
$\text{tfIdf}_{512,Un+Bigram}$	0.3488	0.4098	0.1279	0.3645	0.2996	0.3101
count_{1024}	0.2969	0.2727	0.2724	0.4787	0.2738	0.3189
tfIdf_{1024}	0.3741	0.4875	0.1124	0.3871	0.2574	0.3237
$\text{count}_{1024,Un+Bigram}$	0.3357	0.232	0.24	0.4744	0.1938	0.2952
$\text{tfIdf}_{1024,Un+Bigram}$	0.3672	0.4231	0.1111	0.3692	0.2593	0.306
count_{8192}	0.3629	0.3554	0.2353	0.5455	0.2532	0.3505
tfIdf_{8192}	0.413	0.4462	0.1726	0.3766	0.2804	0.3378
$\text{count}_{8192,Un+Bigram}$	0.3226	0.3117	0.1912	0.4924	0.1641	0.2964
$\text{tfIdf}_{8192,Un+Bigram}$	0.3717	0.4297	0.1633	0.3427	0.1658	0.2946
count_{16384}	0.3556	0.3084	0.2028	0.547	0.2311	0.329
tfIdf_{16384}	0.4074	0.4358	0.1929	0.3614	0.2642	0.3323
$\text{count}_{16384,Un+Bigram}$	0.2857	0.3721	0.1992	0.4935	0.2024	0.3106
$\text{tfIdf}_{16384,Un+Bigram}$	0.3831	0.4516	0.1818	0.365	0.2749	0.3313
ft_{avg}	0.1414	0.1581	0.0702	0.3843	0.25	0.2008
glove_{avg}	0.2709	0.1905	0.0923	0.2533	0.2656	0.2145
ft_{usif}	0.1635	0.1508	0.0678	0.4413	0.3435	0.2334
glove_{usif}	0.2906	0.2739	0.1304	0.3591	0.2382	0.2584
sentence_{512}	0.4265	0.4064	0.1235	0.3444	0.1089	0.2819

Table A.14: $f1$ measures for the different SumConcat vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\not\leftrightsquigarrow}$	$R_{\not\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
count_{512}	0.4444	0.4179	0.4507	0.2388	0.1553	0.1866	0.3891	0.694	0.274	0.1493
tfIdf_{512}	0.3289	0.1866	0.4634	0.2836	0.2364	0.097	0.2281	0.6418	0.275	0.1642
$\text{count}_{512,Un+Bigram}$	0.3585	0.2836	0.4186	0.2687	0.2069	0.2239	0.3506	0.6567	0.2561	0.1567
$\text{tfIdf}_{512,Un+Bigram}$	0.2949	0.1716	0.4	0.2836	0.2653	0.097	0.2207	0.6194	0.2778	0.1493
count_{1024}	0.4045	0.2687	0.3243	0.2687	0.2037	0.2463	0.4096	0.5746	0.2583	0.2313
tfIdf_{1024}	0.2989	0.194	0.3766	0.2164	0.1746	0.0821	0.2193	0.6269	0.2167	0.097
$\text{count}_{1024,Un+Bigram}$	0.3955	0.3955	0.3889	0.209	0.2442	0.1567	0.3308	0.6493	0.2174	0.1866
$\text{tfIdf}_{1024,Un+Bigram}$	0.2857	0.1642	0.4483	0.291	0.1724	0.0746	0.2174	0.597	0.3125	0.1866
count_{8192}	0.3478	0.4179	0.3918	0.2836	0.225	0.2015	0.408	0.6119	0.1978	0.1343
tfIdf_{8192}	0.3611	0.291	0.4667	0.3134	0.1625	0.097	0.2129	0.5672	0.2857	0.0746
$\text{count}_{8192,Un+Bigram}$	0.4286	0.2463	0.3212	0.3955	0.2258	0.209	0.3982	0.6567	0.1807	0.1119
$\text{tfIdf}_{8192,Un+Bigram}$	0.3299	0.2388	0.4524	0.2836	0.25	0.194	0.2266	0.597	0.25	0.0597
count_{16384}	0.3889	0.3134	0.3846	0.3358	0.2469	0.1493	0.3333	0.7761	0.2692	0.1045
tfIdf_{16384}	0.38	0.2836	0.4742	0.3433	0.1842	0.1045	0.2165	0.5672	0.3043	0.1045
$\text{count}_{16384,Un+Bigram}$	0.322	0.2836	0.3737	0.2761	0.2118	0.1343	0.3623	0.7463	0.2065	0.1418
$\text{tfIdf}_{16384,Un+Bigram}$	0.3404	0.2388	0.4634	0.2836	0.1923	0.1493	0.2171	0.5672	0.35	0.1045
ft_{avg}	0.2394	0.1269	0.1918	0.1045	0.2	0.0448	0.252	0.7164	0.2435	0.209
glove_{avg}	0.2945	0.3209	0.2913	0.2761	0.1562	0.0373	0.2553	0.3582	0.1751	0.2313
ft_{usif}	0.2688	0.1866	0.2295	0.1045	0.1714	0.0448	0.2902	0.7537	0.3835	0.3806
glove_{usif}	0.3043	0.2612	0.2842	0.2015	0.1887	0.0746	0.2735	0.4776	0.2023	0.2612
sentence_{512}	0.4412	0.3358	0.4455	0.3358	0.2222	0.0896	0.2203	0.5672	0.1765	0.0896

Table A.15: Precision and Recall measures for the different Issue-vector vectoriser configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.4308	0.3122	0.1695	0.4987	0.1932	0.3209
tfIdf_{512}	0.2381	0.3519	0.1376	0.3366	0.2056	0.254
$\text{count}_{512,Un+Bigram}$	0.3167	0.3273	0.2151	0.4571	0.1944	0.3021
$\text{tfIdf}_{512,Un+Bigram}$	0.217	0.3319	0.1421	0.3255	0.1942	0.2421
count_{1024}	0.3229	0.2939	0.223	0.4783	0.2441	0.3124
tfIdf_{1024}	0.2353	0.2749	0.1117	0.325	0.134	0.2162
$\text{count}_{1024,Un+Bigram}$	0.3955	0.2718	0.1909	0.4383	0.2008	0.2995
$\text{tfIdf}_{1024,Un+Bigram}$	0.2085	0.3529	0.1042	0.3187	0.2336	0.2436
count_{8192}	0.3797	0.329	0.2126	0.4896	0.16	0.3142
tfIdf_{8192}	0.3223	0.375	0.1215	0.3096	0.1183	0.2493
$\text{count}_{8192,Un+Bigram}$	0.3128	0.3545	0.2171	0.4958	0.1382	0.3037
$\text{tfIdf}_{8192,Un+Bigram}$	0.2771	0.3486	0.2185	0.3285	0.0964	0.2538
count_{16384}	0.3471	0.3586	0.186	0.4664	0.1505	0.3017
tfIdf_{16384}	0.3248	0.3983	0.1333	0.3134	0.1556	0.2651
$\text{count}_{16384,Un+Bigram}$	0.3016	0.3176	0.1644	0.4878	0.1681	0.2879
$\text{tfIdf}_{16384,Un+Bigram}$	0.2807	0.3519	0.1681	0.314	0.1609	0.2551
ft_{avg}	0.1659	0.1353	0.0732	0.3728	0.2249	0.1944
glove_{avg}	0.3071	0.2835	0.0602	0.2981	0.1994	0.2297
ft_{usif}	0.2203	0.1436	0.071	0.4191	0.382	0.2472
glove_{usif}	0.2811	0.2358	0.107	0.3478	0.228	0.2399
sentence_{512}	0.3814	0.383	0.1277	0.3173	0.1188	0.2656

Table A.16: $f1$ measures for the different Issue-vector vectorisation configurations.

Vectoriser	$P_{\rightsquigarrow}$	$R_{\rightsquigarrow}$	$P_{\leftsquigarrow}$	$R_{\leftsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\leftrightsquigarrow}$	$R_{\leftrightsquigarrow}$	$P_{\text{duplicate}}$	$R_{\text{duplicate}}$
$_{512}$	0.3762	0.2836	0.4348	0.3731	0.219	0.1716	0.3134	0.6269	0.2963	0.1791
tfIdf$_{512}$	0.3596	0.2388	0.44	0.3284	0.2262	0.1418	0.2898	0.5299	0.2895	0.3284
count$_{512,Un+Bigram}$	0.3514	0.291	0.3864	0.2537	0.1961	0.2985	0.3676	0.5075	0.2195	0.1343
tfIdf$_{512,Un+Bigram}$	0.3152	0.2164	0.4066	0.2761	0.209	0.1045	0.2552	0.5448	0.2836	0.2836
count$_{1024}$	0.275	0.1642	0.3163	0.2313	0.2153	0.2313	0.3517	0.6194	0.2321	0.194
tfIdf$_{1024}$	0.2793	0.2313	0.3878	0.2836	0.23	0.1716	0.2469	0.4403	0.2295	0.209
count$_{1024,Un+Bigram}$	0.3529	0.3582	0.3582	0.1791	0.2202	0.2761	0.4009	0.6343	0.3218	0.209
tfIdf$_{1024,Un+Bigram}$	0.2929	0.2164	0.3936	0.2761	0.2048	0.1269	0.2764	0.5075	0.2973	0.3284
count$_{8192}$	0.3016	0.1418	0.3545	0.291	0.1931	0.209	0.3636	0.7164	0.3409	0.2239
tfIdf$_{8192}$	0.3559	0.3134	0.4561	0.3881	0.2233	0.1716	0.2546	0.4104	0.2353	0.209
count$_{8192,Un+Bigram}$	0.3855	0.2388	0.3438	0.3284	0.2054	0.1716	0.37	0.7537	0.2973	0.1642
tfIdf$_{8192,Un+Bigram}$	0.3208	0.2537	0.4	0.3134	0.2385	0.194	0.2568	0.4925	0.2151	0.1493
count$_{16384}$	0.3644	0.3209	0.4327	0.3358	0.1958	0.209	0.4097	0.694	0.1923	0.1119
tfIdf$_{16384}$	0.3684	0.3134	0.4444	0.3881	0.2321	0.194	0.2582	0.4104	0.2281	0.194
count$_{16384,Un+Bigram}$	0.3667	0.1642	0.3565	0.306	0.2171	0.209	0.3796	0.7761	0.25	0.1716
tfIdf$_{16384,Un+Bigram}$	0.3333	0.2612	0.5	0.2985	0.1983	0.1791	0.2481	0.4851	0.2353	0.1791
ft$_{avg}$	0.2025	0.1194	0.2192	0.1194	0.2031	0.097	0.296	0.6119	0.226	0.2985
glove$_{avg}$	0.3038	0.1791	0.2353	0.1493	0.1765	0.1343	0.4318	0.1418	0.1917	0.5149
ft$_{usif}$	0.2796	0.194	0.2267	0.1269	0.1852	0.0746	0.3029	0.694	0.3121	0.3284
glove$_{usif}$	0.3304	0.2761	0.2589	0.2164	0.1475	0.0672	0.2868	0.2761	0.207	0.3955
sentence$_{512}$	0.408	0.3806	0.4307	0.4403	0.2653	0.097	0.2477	0.3955	0.1862	0.2015

Table A.17: Precision and Recall measures for the different Topic vectoriser configurations.

Vectoriser	$f1_{\rightsquigarrow}$	$f1_{\leftsquigarrow}$	$f1_{\leftrightsquigarrow}$	$f1_{\not\leftrightsquigarrow}$	$f1_{\text{duplicate}}$	macro f1
count_{512}	0.3234	0.4016	0.1925	0.4179	0.2233	0.3117
tfIdf_{512}	0.287	0.3761	0.1743	0.3747	0.3077	0.304
$\text{count}_{512,Un+Bigram}$	0.3184	0.3063	0.2367	0.4263	0.1667	0.2909
$\text{tfIdf}_{512,Un+Bigram}$	0.2566	0.3289	0.1393	0.3476	0.2836	0.2712
count_{1024}	0.2056	0.2672	0.223	0.4486	0.2114	0.2712
tfIdf_{1024}	0.2531	0.3276	0.1966	0.3164	0.2188	0.2625
$\text{count}_{1024,Un+Bigram}$	0.3556	0.2388	0.245	0.4913	0.2534	0.3168
$\text{tfIdf}_{1024,Un+Bigram}$	0.2489	0.3246	0.1567	0.3579	0.3121	0.28
count_{8192}	0.1929	0.3197	0.2007	0.4824	0.2703	0.2932
tfIdf_{8192}	0.3333	0.4194	0.1941	0.3143	0.2213	0.2965
$\text{count}_{8192,Un+Bigram}$	0.2949	0.3359	0.187	0.4963	0.2115	0.3051
$\text{tfIdf}_{8192,Un+Bigram}$	0.2833	0.3515	0.214	0.3376	0.1762	0.2725
count_{16384}	0.3413	0.3782	0.2022	0.5152	0.1415	0.3157
tfIdf_{16384}	0.3387	0.4143	0.2114	0.317	0.2097	0.2982
$\text{count}_{16384,Un+Bigram}$	0.2268	0.3293	0.2129	0.5098	0.2035	0.2965
$\text{tfIdf}_{16384,Un+Bigram}$	0.2929	0.3738	0.1882	0.3283	0.2034	0.2773
ft_{avg}	0.1502	0.1546	0.1313	0.399	0.2572	0.2185
glove_{avg}	0.2254	0.1826	0.1525	0.2135	0.2794	0.2107
ft_{usif}	0.2291	0.1627	0.1064	0.4218	0.32	0.248
glove_{usif}	0.3008	0.2358	0.0923	0.2814	0.2718	0.2364
sentence_{512}	0.3938	0.4354	0.1421	0.3046	0.1935	0.2939

Table A.18: $f1$ measures for the different Topic vectorisation configurations.

12699freeCodeCamp/freeCodeCamp	2muchcoffeecom/ngx-restangular	AuthorizeNet/sdk-node	Automattic/mongoose	Azure/AKS	BishopFox/h2csmuggler
CaliStyle/ng-intercom	Capgemini/Apollo	CaptainFact/captain-fact-frontend	CellProfiler/CellProfiler	CodeChain-io/codechain-explorer	Concorda/concorda-dashboard
ContinuumIO/anaconda-issues	DHTMLX/angular2-gantt-demo	DamonOehlman/travis-multirunner	DarkaOnLine/SwaggerLume	Dashticz/dashticz	DataDog/docker-dd-agent
DefinitelyTyped/DefinitelyTyped	EugenMayer/docker-sync	ExpDev07/coronavirus-tracker-api	FountainJS/generator-fountain-webapp	FreeCodeCamp/FreeCodeCamp	Gbuomprisco/ngx-chips
GustavoCostaW/ngc-float-button	HabitRPG/habitica	Homebrew/legacy-homebrew	IBM-Cloud/ibm-cloud-developer-tools	InterDigitalInc/AdvantEDGE	Invis1ble/assistant-client
JamesMGreene/node-flex-sdk	JedWatson/classnames	JeffreyWay/laravel-mix	Julian/jsonschema	Ks89/angular-modal-gallery	MartinoMensio/spacy-universal-sentence-encoder
MichaIng/DietPi	Miserlou/Zappa	NationalBankBelgium/stark	NativeScript/nativescript-plugin-seed	NetApp/trident	Norkart/L.Control.NorkartSearch
PIVX-Project/PIVX	Painted-Fox/docker-mariadb	Painted-Fox/docker-postgresql	PatrickJS/angular-starter	PyWavelets/pywt	RaRe-Technologies/gensim
RasaHQ/rasa	ReactTraining/react-router	ReactiveX/rxjs	Redocly/redoc	SamSaffron/graphite$_docker$	SassDoc/gulp-sassdoc
Semantic-Org/Semantic-UI-React	Shippable/support	SoftwareCarpentryLessonManager/lesson-manager	Stannieman/audacity-with-asio-builder	SteveLTN/https-portal	TrilonIO/aspnetcore-angular-universal
TypeStrong/atom-typescript	TypeStrong/ts-loader	Unidata/thredds-docker	Unitech/pm2	Varying-Vagrant-Vagrants/VVV	WoltersKluwerPL/ng-spin-kit
WordPress/gutenberg	XervoIO/demeteorizer	Yelp/dumb-init	Zulko/moviepy	aberezkin/ng2-image-upload	abiosoft/caddy-docker
adopted-ember-addons/ember-electron	agdsn/pycroft	ajv-validator/ajv	andrewrk/node-s3-cli	angular/angular	angular/angular-cli
angular/devkit	angular/protractor	angular/quickstart	ansible-collections/community.kubernetes	ansible/ansible-modules-core	ansible/awx
ant-design/ant-design	apache/openwhisk	archfz/drup	arkon/ng-sidebar	aspnet/JavaScriptServices	asreview/asreview
atom/atom	aurelia/webpack-plugin	auth0/angular2-jwt	aws-samples/amazon-rekognition-video-analyzer	aws/amazon-ecs-agent	aws/amazon-sagemaker-examples
aws/amazon-vpc-cni-k8s	aws/aws-cli	aws/aws-sdk-js	aws/aws-sdk-ruby	aws/containers-roadmap	awslabs/amazon-eks-ami
bcgov/ckanext-openapiviewer	beeworking/voyant	benpolinsky/artvsart$_react$	big-data-europe/docker-hadoop-spark-workbench	bitcoin/bitcoin	bitcraze/crazyflie-clients-python
bitnami/bitnami-docker-wordpress-nginx	bitnami/charts	boot2docker/boot2docker	boto/botocore	bower/bower	brave/browser-laptop
brikis98/docker-osx-dev	browserify/module-deps	brunch/brunch	bsidelinger912/react-tooltip-lite	callmehiphop/backend	canjs/canjs
captivationsoftware/react-sticky	carlosedp/cluster-monitoring	ceph/ceph-csi	channl/dynamodb-lambda-autoscale	che-incubator/chectl	chenkie/angular-cli-heroku
chili-epfl/FROG	cilium/cilium	clarity-h2020/simple-table-component	clearcontainers/runtime	cleverbeagle/pup	cli/cli
cloudfleet/blimp-engineroom	cloudinary/cloudinary$_angular$	cloudnativelabs/kube-router	clux/symlink	codekitchen/dinghy	codemirror/CodeMirror
commitizen/cz-cli	concourse/concourse	conda/conda	connext/indra	coredns/deployment	coreos/bugs
coreos/flannel	crisbeto/angular-svg-round-progressbar	cypress-io/cypress-test-node-versions	d-akara/vscode-extension-fold	danielhusar/gulp-remove-empty-lines	datacats/datacats
datawire/ambassador	davidkpiano/react-redux-form	dbashford/mimosa	dbfannin/ngx-logger	dduportal-dockerfiles/docker-compose	deis/deis
deis/postgres	deis/workflow	derhuerst/vbb-stations-cli	deviantony/docker-elk	devilbox/docker-php-fpm	dhaus97/pi-gen-k8s
digitalascetic/ngx-pica	digitalocean/digitalocean-cloud-controller-manager	dimpu/ngx-md	dminca/docker-phabricator	docker-archive/for-azure	docker-flow/docker-flow-monitor
docker-java/docker-java	docker-library/mongo	docker-library/openjdk	docker-library/redis	docker-library/tomcat	docker-library/wordpress
docker-mailserver/docker-mailserver	docker-php/docker-php	docker/classicswarm	docker/cli	docker/compose	docker/docker-py
docker/for-linux	docker/for-mac	docker/for-win	docker/machine	docker/toolbox	dockerstuff/docker-dachs
dokku/dokku	dokku/dokku-rabbitmq	dotnet-architecture/eShopOnContainers	dotnet/core	dotnet/dotnet-docker	dpkp/kafka-python
drone-plugins/drone-s3	drud/ddev	drud/vault-consul-on-kube	dschnelldavis/angular2-json-schema-form	dunovank/jupyter-themes	dustinspecker/eslint-plugin-no-use-extend-native
ebi-ait/hca-ebi-dev-team	elabftw/elabctl	elabftw/elabftw	elastic/kibana	electron-userland/electron-forge	emacs-pe/docker-tramp.el
ember-cli/ember-page-title	encode/starlette	enricomarino/is	enthought/traits	ericgio/react-bootstrap-typeahead	etcd-io/etcd
ethereum/go-ethereum	evertramos/docker-compose-letsencrypt-nginx-proxy-companion	explosion/spaCy	ezequiel/react-typeahead-component	facebook/create-react-app	facebook/flow
facebook/jest	facebook/react	facebook/react-native	fastify/fastify-swagger	fecgov/fec-cms	fecgov/openFEC
filipesilva/angular-quickstart-lib	final-form/react-final-form	firebase/firebase-js-sdk	firoorg/firo	fission/fission	flannel-io/flannel
flapjack/omnibus-flapjack	flowtype/flow-bin	flynn/flynn	freeCodeCamp/freeCodeCamp	fulcrologic/fulcro	ga4gh-beacon/specification
gamechanger/dusty	gardener/dashboard	gardener/gardener	gardenlinux/gardenlinux	gdlg/panoramic-depth-estimation	geerlingguy/ansible-role-docker
geerlingguy/ansible-role-docker$_arm$	geerlingguy/raspberry-pi-dramble	getsentry/onpremise	gliderlabs/docker-alpine	gliderlabs/herokuish	gmacario/easy-jenkins
go-delve/delve	goharbor/harbor	golang/go	google-research/motion$_imitation$	google/cadvisor	google/material-design-icons
googleapis/gcs-resumable-upload	googleapis/google-cloud-node	googleforgames/agones	grafana/grafana	grpc-ecosystem/grpc-gateway	grpc/grpc
gruntjs/grunt	handsontable/angular-handsontable	hapijs/hapi	hardkernel/linux	hardware/mailserver	hashicorp/consul
hashicorp/consul-helm	hashicorp/nomad	hashicorp/packer	hashicorp/terraform	hashicorp/terraform-provider-aws	hashicorp/vault
hasura/graphql-engine	helm/charts	helm/helm	hemanth/node-nightly	heroku/heroku-buildpack-ruby	hetznercloud/csi-driver
hetznercloud/hcloud-cloud-controller-manager	hexops/vecty	hilbert/hilbert-cli	hirak/prestissimo	home-assistant/core	hpe-storage/csi-driver
hpe-storage/python-hpedockerplugin	hydraslay/ng2-polymer-static-gen	hyfen-nl/PIVT	hyperledger/besu	igraph/python-igraph	imagemin/imagemin-guetzli
indiana-university/itpeople-functions	intelsdi-x/snap-plugin-collector-docker	ionic-team/ionic-app-scripts	ionic-team/ionic-framework	ios-control/ios-deploy	ipfs/js-ipfs
iproduct/course-angular	ipython/ipython	istio/istio	istreamlabs/pebble	jadjoubran/webdash-package-json	javivelasco/react-css-themr
jaymoulin/google-musicmanager-dedup-api	jcmoraisjr/haproxy-ingress	jdleesmiller/docker-chat-demo	jediproject/generator-jedi	jesseduffield/lazydocker	jetstack/cert-manager
jfrog/terraform-provider-artifactory	jhipster/generator-jhipster	johnagan/clean-webpack-plugin	jonathantneal/precss	joshswan/react-native-globalize	jquery/jquery
jschneier/django-storages	jshimko/meteor-launchpad	jupyter/docker-stacks	jupyterlab/jupyterlab	just-containers/s6-overlay	jvandemo/generator-angular2-library
k3s-io/k3s	kaliber5/ember-fastboot-addon-tests	kashjs/angular-workshop	kata-containers/kata-containers	kbst/terraform-provider-kustomization	keystonejs/keystone-classic
knative/eventing	kolide/launcher	krispo/ng2-nvd3	kubeflow/pipelines	kubeflow/tf-operator	kubermatic/machine-controller
kubernetes-client/csharp	kubernetes-client/java	kubernetes-retired/heapster	kubernetes-retired/kube-aws	kubernetes-sigs/kind	kubernetes-sigs/kubespray
kubernetes-sigs/sig-windows-tools	kubernetes-sigs/vsphere-csi-driver	kubernetes/client-go	kubernetes/dns	kubernetes/ingress-nginx	kubernetes/kops
kubernetes/kube-state-metrics	kubernetes/kubeadm	kubernetes/kubernetes	kubernetes/minikube	kubernetes/node-problem-detector	kubernetes/release
kubernetes/website	kubevirt/kubevirt	kuzzmi/ember-cli-webfontloader	kvaps/kube-linstor	l-lin/angular-datatables	laantorchaweb/ember-cli-slick
laboratoriobridge/bold	lando/lando	lerna/lerna	less/less.js	lfarran/ngx-soundmanager2	liberodark/ODrive
linkerd/linkerd2	linnovate/mean	llSourcell/tensorflow$_chatbot$	lodash/lodash	longhorn/longhorn	lovell/sharp
lowerquality/gentle	luzifer-docker/mumble	lxc/lxc	mafintosh/tar-fs	mailcow/mailcow-dockerized	manekinekko/angular-web-bluetooth
mapbox/npm-internal	mapbox/rasterio	marcorinck/ngStart	mariocasciaro/npm-workspace	marko-js/marko	matiboy/angular2-prettyjson
matplotlib/matplotlib	mattermost/desktop	maximegris/angular-electron	maximelafarie/ngx-smart-modal	mdn/infra	mesosphere/marathon
metallb/metallb	meteor/meteor	meumobi/ion-ams-report	mgechev/angular-seed	microsoft/OMS-Agent-for-Linux	microsoft/TypeScript
microsoft/TypeScript-React-Starter	microsoft/WSL	microsoft/dotnet-computevirtualization	microsoft/fluentui	microsoft/go-winio	microsoft/hcsshim
microsoft/navcontainerhelper	microsoft/pxt	microsoft/types-publisher	microsoft/vscode	microsoft/vscode-vsce	miguelcobain/ember-cli-selectize
mininet/mininet	minio/mc	minishift/minishift	mitodl/micromasters	mkuchin/docker-registry-web	mne-tools/mne-python
mobxjs/mobx-react	moby/libnetwork	moby/moby	moff/angular2-flash-messages	mohsinulhaq/react-popper-tooltip	mono/docker
moribvndvs/ng2-idle	moroshko/react-autosuggest	mozilla-mobile/firefox-ios	mozilla/fxa-auth-server	mozilla/vinz-clortho	mpalourdio/ng-http-loader
mperrin/poppy	mquan/cortex	mui-org/material-ui	nats-io/k8s	ng-alain/ng-alain	ng-hal/ng-hal
ng2-ui/map	nginx-proxy/docker-letsencrypt-nginx-proxy-companion	nginx-proxy/nginx-proxy	nginxinc/docker-nginx	ngrx/store	ngx-formly/ngx-formly
ngx-translate/core	nodeca/pako	nodejs/docker-node	nodejs/nan	nodejs/node	nodejs/node-v0.x-archive
nodeschool/discussions	nosir/cleave.js	npm/npm	npm/npme-installer	numpy/numpy	ohmyzsh/ohmyzsh
olivere/elastic	onehungrymind/ng2-reactive-app	onivim/oni	open-api-spex/open$_api_spex$	opencontainers/runc	openebs/openebs
openfaas/faas	openhab/openhab-docker	openshift/openshift-ansible	openshift/origin	oracle/oci-volume-provisioner	orchardup/docker-mysql
ossec/ossec-hids	outlyerapp/dataloop-docker	ova2/angular-development-with-primeng	overleaf/overleaf	pandas-dev/pandas	paulmillr/chokidar
pgilad/gulp-angular-htmlify	phac-nml/staramr	phusion/passenger-docker	pinellolab/STREAM	pingcap/tidb	pingcap/tidb-operator
pnpm/pnpm	polyaxon/polyaxon	portainer/portainer	pravega/pravega-operator	prawnsalad/KiwiIRC	pre-commit/mirrors-eslint
pre-commit/pre-commit	preactjs/preact-cli	prettydiff/prettydiff	projectcalico/calico	prometheus-operator/prometheus-operator	prometheus/prometheus
psfinaki/CheckYourCzech	puckel/docker-airflow	puntonim/ansible-biostar	puntonim/docker-postgresql93	pupil-labs/pupil-detectors	pypa/pip
quasarframework/quasar-cli	r0man/sablono	rackerlabs/docs-dedicated-networking	rancher/os	rancher/rancher	rancher/rke
raspberrypi/linux	rd-dev-ukraine/angular-io-datepicker	react-bootstrap/react-router-bootstrap	react-toolbox/react-toolbox	reactioncommerce/reaction	reactjs/react-chartjs
reactjs/react-modal	redux-form/redux-form	reduxjs/react-redux	reduxjs/redux	renato-bohler/redux-form-input-masks	request/request-promise-native
requireio/aperture	rescript-lang/rescript-compiler	ritzyed/ritzy	rlidwka/sinopia	rmasters/laravel-vm	robcowart/elastiflow
rodolfocop/ng2-mask-money	rook/rook	rroemhild/docker-ejabberd	runem/lit-analyzer	s-panferov/awesome-typescript-loader	salesforce/design-system-react
sameersbn/docker-gitlab	sanniassin/react-input-mask	sbt/sbt	scaleway/kernel-tools	scality/metalk8s	scipy/scipy
sclorg/postgresql-container	scrapinghub/splash	sdelements/lets-chat	seishun/node-steam	select2/select2	sequenceiq/docker-spark
shama/webpack-stream	signavio/react-mentions	silveridea/ngx-push-notifications	sindresorhus/downgrade-root	sindresorhus/open	sitespeedio/sitespeed.io
slara/generator-reveal	smallstep/cli	smoketurner/dropwizard-swagger	sollenne/angular-fittext	sous-chefs/postgresql	spotify/helios
statianzo/webpack-livereload-plugin	statsmodels/statsmodels	steam-forward/node-steam-forum	stefan-jansen/machine-learning-for-trading	stellar/docker-stellar-core-horizon	stoplightio/prism
storj/storj	strapi/strapi-docker	strimzi/strimzi-kafka-operator	strongloop/loopback-next	stylelint/stylelint	substack/html-select
swagger-api/swagger-editor	swagger-api/swagger-js	swagger-api/swagger-ui	syncfusion/ej2-angular-ui-components	syncthing/syncthing	technomancy/leiningen
telerik/kendo-angular	tensorflow/text	tessel/node-usb	therecipe/qt	tiagob/create-full-stack	tiangolo/fastapi
tianon/dockerfiles	tinesoft/generator-ngx-library	tmspzz/Rome	tomdale/ember-cli-addon-tests	tracer0tong/android-emulator	traefik/traefik
trailsjs/trails	travelist/ng2-file-tree	travis-ci/travis-ci	typings/typings	uber/peloton	unstubbable/custom-tslint-formatters
vauxite-org/typescript-logging	vercel/next.js	vimalavinisha/angular2-google-chart	vlad-ignatov/react-numeric-input	vmware-archive/vsphere-storage-for-docker	vpicavet/docker-pggis
weaveworks-experiments/weave-kube	weaveworks/eksctl	weaveworks/weave	webcompat/web-bugs	webfactorymk/ng2-canvas-whiteboard	webpack/webpack
websockets/ws	wechaty/wechaty	why520crazy/ngx-validator	wilmoore/selectn.js	wordpress-clients/wp-api-angular	worstcase/blockade
wurstmeister/kafka-docker	xianyi/OpenBLAS	xolvio/chimp	yakyak/yakyak	yarnpkg/yarn	ypinskiy/GBF-Raiders
yuyang041060120/ng2-animate	zammad/zammad-docker	zefoy/ngx-color-picker	zooniverse/Panoptes-Front-End	zxing-js/ngx-scanner	

Table A.19: A list of the repositories we scraped issues from.

Declaration