UNIVERSIDAD NACIONAL DE COLOMBIA

An information retrieval strategy for large multimodal data collections involving source code and natural language

Juan Felipe Baquero Vargas

Universidad Nacional de Colombia
Facultad de Ingenieria, Departamento de Ingeniería de sistemas e industrial
Bogotá, Colombia
2019

An information retrieval strategy for large multimodal data collections involving source code and natural language

Juan Felipe Baquero Vargas

Thesis presented as partial requirement to obtain the degree of:
Magíster en Ingeniería de Sistemas y Computación

Director:
Ph.D., Fabio Augusto González Osorio
Co-director:
Ph.D., Felipe Restrepo Calle

Research Line:
Intelligent Systems, Applied Computing, Programming Languages
Research groups:
MindLab - PLaS

Universidad Nacional de Colombia
Faculty of Engineering, Department of Systems and Industrial Engineering
Bogotá, Colombia
2019

Acknowledgments

To my sister who was the main motivation and inspiration to accomplish this work. To my parents for their help and support and to Lina for supporting me and motivating me during these years.

Simplicity is prerequisite for reliability.

Edsger W. Dijkstra

Resumen

Los repositorios de software almacenan datos sobre los productos de software, datos relacionados con la evolución de código fuente, requerimientos de software, reporte de bugs y comunicación entre desarrolladores. Los repositorios de software han crecido rápidamente en los últimos años y con ellos la necesidad the extraer información significativa de ellos. Un repositorio de software intersante es Stack Overflow(SO), este sitio web es uno de los sitios de Question Ansering más grandes y usados por miles de desarrolladores de sofware en su día a día. En SO los desarrollares pueden preguntar cualquier duda relacionada con programación y software que será respondida por otros usuarios. Como SO, existen muchos repositorios de software con código fuente y texto con millones de ejempleares y la posibilidad de combinar ambas fuentes para extraer información de ellos que no es visible a simple vista. En este trabajo de tesis, exploramos cómo representar código fuente y lenguaje natural y cómo combinar estas representaciones. Intentamos resolver la tarea de entender cómo los usuarios de SO hablan sobre un lenguage de programación, que tan similares son los lenguajes de programación basados en cómo los usuarios hablen sobre ellos y, finalmente, proporcionar herramientas para construir una estrategia de information retrival para indentificar post duplicados.

Palabras clave: Stack Overflow, análisis de código fuente, detección de duplicados, predecir el lenguaje de programación.

Abstract

Source code repositories store data from software products. Among this data we can find the evolution of the source code, requirements, bugs and communication between developers[1][14, 26]. Source code repositories have been growing rapidly in the recent years and with them the need of extracting information from them. An interesting source code repository that is growing both in usage and information is Stack Overflow (SO), this website provides one of the biggest Question Answering places used by thousands of developers every day. In SO the developers can ask any question related to a programming issue and it will be answered by other users. We can find a source code repository with both source code and natural language with thousands of samples and the possibility of combining both sources of information to extract useful and not eye-noticeable information from it. In this thesis, we explore how to represent source code and natural language and how to combine these representations. We try to solve the task of understanding how users in SO talk about the programming language, how similar these programming languages are among them based on how users talk about them, and finally, we provide tools on the building of an information retrieval strategy by identifying duplicated post.

[1] http://www.msrconf.org/

Key words: Stack Overflow, source code analysis, duplication detection, predicting programming language.

Content

1 Introduction

1.1. Motivation

Software repositories contain large collections of information about Software such as: source code, discussions about it, reports of bugs and execution. These repositories are, according to [22], "data rich but information poor" because at the beginning they were just seeking for a place to store the data. This huge amount of data is interesting in terms of the information and knowledge that can be obtained through analysis.

That analysis (or knowledge mining) of software elements, especially the ones stored in software repositories, has been motivated by the increasing amount of data and tools to manage and extract information [10, 14]. Also, another interest is to analyze aspects that are present in the process of software development. Some of these aspects are addressed in [6] like clone detection [18], defect localization [24], code optimization, among others. The knowledge mining also opens up new interesting research topics, like identifying latent relationships between developers in a social level [30] and sentiment analysis and their relation with the software development process [13].

Most of the mentioned aspects can be grouped in 2 tasks: code retrieval and code summarization. From the point of view of software evolution these tasks are shown by [14] and [28] and their goal is to understand a software system. During the development of a big software systems a lot of code is involved and usually it is not well commented and the documentation is outdated. This creates, in case new developers join the team, a challenge in understanding the product or reviewing old elements.

Another interesting task is code-change propagation: the process to identify and propagate the changes in the code keeping the consistency and performance[14, 12].

1.1.1. Software Repository

Accordingly to the authors in [14] and [26] a software repository can be described as a data storage for software products. In here the developer not only store source code but how it evolves during the development (different versions of the same product), software definitions and requirements, bugs, issues and messages among the developer team.

Examples and descriptions of different kinds of software repositories are showed by [14]:

- Source control repositories: These repositories record the development history of a project. They track all the changes to the source code along with meta-data about each change, e.g., the name of the developer who performed the change, the time the change was performed and a short message describing the change. Source control repositories are the most commonly available and used repositories in software projects. Concurrent versions systems (CVS)[1], Subversion[2], Perforce[3] and ClearCase[4] are examples of source control repositories which are used in practice.

- Bug tracking systems: These repositories track the resolution history of bug reports or feature requests that are reported by users and developers of large software projects. Bugzilla[5] and Jira[6] are examples of bug repositories.

- Archived communications: These repositories track discussions about various aspects of a software project throughout its lifetime. Mailing lists, emails, internet relay chat (IRC) chats, and instant messages are examples of archived communications about a project.

- Deployment logs: These repositories record information about the execution of a single deployment of a software application or different deployments of the same applications. For example, the deployment logs may record the error messages reported by an application at various deployment sites. The availability of deployment logs continues to increase at a rapid rate due to their use for remote issue resolution and due to recent legal acts.

- Public code repositories: These repositories archive the source code for a large number of projects. Sourceforge.net[7], Github[8] and Google code[9] are examples of large public code repositories.

We can categorize Stack Overflow (SO)[10] as a software repository because the users most of the time ask about issues related to their software products, sometimes also including

[1] http://savannah.nongnu.org/projects/cvs
[2] https://subversion.apache.org/
[3] https://www.perforce.com/
[4] https://www.ibm.com/us-en/marketplace/rational-clearcase
[5] https://www.bugzilla.org/
[6] https://www.atlassian.com/software/jira
[7] https://sourceforge.net/
[8] http://github.com
[9] https://code.google.com/
[10] https://stackoverflow.com

source code. The issues are stored in a question-format, they include a description of the issue and possibles solutions that other developers propose; with this, we can include SO in the categories of bug tracking and archived communications.

1.1.2. Feature representation

One of the most important and challenging activities is the feature representation of multi-modal data, including text and source code. This is important since the representation will affect the performance of any method for data analysis and/or retrieval.

Text representation is a very well explored field [15]. Some strategies to process and represent text are: probabilistic context free grammar (PCFG), n-grams, hashing, latent semantic, based on trees, based on tokens. In contrast, source-code representation is not as studied as text representation. There are two main ways to address it: static analysis and dynamic analysis [17]. The first one uses directly the source code, and the second one uses information produced during the code execution, like traces and logs of execution. Some of the typical strategies in each category are[17]:

- Static analysis:
  - Natural language processing (NLP) over the source code.
  - Clustering.
  - Tokens analysis.
  - Building the abstract syntax tree (AST: A tree representation of the syntactic structure of the source code).
  - Graph dependency.
- Dynamic analysis:
  - Log analysis.
  - Execution patterns.

1.2. Problem Identification

Nowadays, in each step of the software development life cycle a lot of data is generated and stored. This data usually is a mix of source code and text (software documentation, comments in the code, discussions about code, etc.). During the software development process, developers usually need to translate tasks than are in natural language to a source code representation than performs the task. Moreover, another common activity is trying to understand the source code, and perhaps giving a description in natural language. These are

two good examples where source code and text are mixed and it is not possible to think of a solution to these problems without modeling both modalities. Then, given the large amount of data in software repositories, the challenge is how to use it to support a strategy that helps in problems that involve source code and text components like code summarization and code retrieval, where both modalities are needed.
There are some research questions that will be answered during this work:

- What features from text and source code are more useful to perform tasks such as code summarization and code retrieval?
- How to suggest a source-code fragment to solve a problem?
- How to represent efficiently a particular source code?
- How to exploit textual and source-code data in a retrieval task?
- How to use the data of software repositories in a code summarization and code retrieval task?

1.3. General Objective and specific objectives

1.3.1. General Objective

To design, implement and evaluate a strategy for information retrieval on large collections of documents that involve source code and natural language.

1.3.2. Specific Objectives

- To collect data-sets of multimodal (source code and natural language) documents.
- To design, propose or adapt a strategy for representing the content of the multimodal documents.
- To design an information retrieval strategy for searching multimodal document collections.
- To develop a functional prototype of the retrieval system.
- To evaluate the retrieval system on the datasets.

1.4. Structure of the thesis

This thesis is structured as follows: first chapter presents an introduction, problem identification and the objectives we want to accomplish with this work. Second chapter presents the related works. Third chapter presents the first problem we tackled, predicting the programming language of a given piece of source code. Fourth chapter presents the second problem: detecting duplicated questions in a question answering site. Finally, fifth chapter presents the conclusions of the thesis and discusses ideas for future work.

2 Related works

The application of source-code and textual analysis, particularly related to software development has been used in different context and for different applications. For example, in tasks like authorship attribution of source code [7], bug detection [31, 29], plagiarism detection [16], and more related to this work, automatic duplicate code detection, functionalities summarization, software specifications and generation of documentation.

2.1. Textual analysis and representation

Text classification is a classical information retrieval problem [8]. The goal is to extract information from the text to achieve the classification task. For example, some authors [11] proposed ASOBEK to identify twitter paraphrasing. They model the paraphrasing detection as a binary classification task. Other approach is Word2Vec [21], where the authors introduced a very well know technique to create vectors of words from huge datasets. What it is interesting here is that those vectors are meaningful and work for billions of words. In addition, authors in [8] propose a different way to approach to huge datasets: they train a model to classify IRC tech comments using data from Stack Overflow[1] as positive examples and YouTube[2] comments as negative examples.

The work proposed in [11] tries to solve a key problem for question answering: machine translation. In their approach, they extract information using lexical analysis over features from very short text, in this case tweets, but it could also be from titles of posts or comments in a forum. Authors also use simple models for the classification task such as Support Vector Machines (SVM) and they performed much better than other more complex approaches. Regarding how to use Word2Vec and the usage of comments, the authors in [20] propose a model to predict the grade of an student based on the write comments they do during a course.

Other authors like [3] have used textual analysis combined with source code analysis. They propose an automatic classifier of code snippets where they try to classify post written in different programming languages such as *C*, *C++* and *C#*, all the snippets extracted from

[1] https://stackoverflow.com/
[2] https://www.youtube.com/

Stack Overflow. They mention that this classifying task is very difficult because the code snippets are very short (we can compare it with tweets in the textual world) and that is why they try to combine textual and source code analysis to extract the more amount of information possible about the snippets.

2.2. Source-code analysis and representation

The extraction of interesting information from the source code has been used in different tasks, and for each different task, the useful information could be different.

The authorship attribution is a common task. The aim of this task is to find the author of a given source but sometimes the textual information (such as text written by the author, descriptions, etc.) about the author is small, or even non-existent. We should have a lot of source code belonging to the author to perform this task. In [7] the authors extract lexical, syntactical and layout features from a AST representation of the source code. The authors highlight that, to use machine learning to solve this task, the source code representation should clearly express the program style.

Regarding to the plagiarism detection, some tools like MOSS[3], JPlag[4] only use approaches based on text. But regarding the source code, some others tried to solve this problem analyzing the program dependency graphs of the source code. Going deep in the information extracted, in [16] the authors propose a source code representation using five high level features: lexical features, stylistic feature, comments features, programmer's textual features, and structure features. For the source code, n-grams of characters. Punctually, among the ones regarding to source code, the control flow of the graph was took into account, also with analyzing the methods in the code (direct calls, indirect calls, conditional branches, etc.) and counting features like the number of integer constants, static variables and local variables.

For clone detection, which differs from the plagiarism detection task in that the clone detection is more oriented to applications, for example, copying-pasting methods, and the plagiarism is more oriented to academic courses or software licensing. An approach to solve this task was presented in [32] where they try to classify the source code fragments in four categories were the first three ones are related to textual similarity and the fourth one is related to functional similitude. Among the techniques described by the authors, we can find the textual, token-based, tree-based and graph-based ones.

The textual ones are only limited to their ability to recognize clones even if the differen-

[3]https://github.com/danainschool/moss-taps
[4]https://github.com/jplag/jplag

ce between them is only renaming identifiers. The token based ones, used in [19, 4], try to operate on a higher level of abstraction than the pure text-based techniques, even though, according to [25], they have a tendency to admit more false positives. The tree-based techniques measure the similarity of sub-trees in syntactic representations, using the fact that these trees are an accurate description of features at a syntactic level of the source code. Regarding the graph-based techniques, these ones are used by doing a static program analysis over the source code to transform code into a program dependence graph (PDG). Some other more general approaches described in [9] are more dependant to the code itself, like to ignore identifiers and operators and instead consider the frequency of the keywords, indentation patterns, length of each line, among others [32].

2.3. Application of interest: duplicate detection

The task of duplicate detection is common in textual elements, like news, book, etc., but in the software development world we can found interesting applications. For instance, in [16] it is used for plagiarism detection in a academic context. Tables **2-1** and **2-2** describe two tasks related to duplicate detection. Table **2-1** describes the task of information retrieval, where, given a post, the idea is to retrieve the most similar post to it; and Table **2-2** describes a work related to the task of, given two post, decide when they are duplicated or not.

For the information retrieval task, a common data set is *msr2013*[5], which is a data set belonging to the *Mining Software Repositories* 2013 that includes all the post from *Stack Overflow* (SO) until 2013.

For the task of identifying when a post is duplicated or not, another SO data set is used, *MSR 2015*[6], which includes posts from SO until 2014. The author in [27] propose two approaches to solve this task, question retrieval and question classification. In the first approach, the aim is to find equivalence among questions. For this case, only using text-based techniques. In the second one, the aim is to classify questions and to understand the knowledge available in SO. In addition, the work proposed in Section 4 describes a way of classify two post as duplicated using semantic similarities between questions and other text-based techniques.

[5] http://2013.msrconf.org/challenge.php
[6] http://2015.msrconf.org/challenge.php

Table 2-1: Related works on the task of information retrieval searching the most similar post

Related work	Year	Dataset	Measure	Results
Multi-Factor Duplicate Question Detection in Stack Overflow[37]	2015	msr 2013	Recall-rate@k	recall5: 0.42 recall20: 0.63
Tag Recommendation in Software Information Sites[33]	2013	msr 2013		
Mining Duplicate Questions of Stack Overflow[1]	2016		Recall-rate@k	recall5: 0.51 recall20: 0.66
Two Improvements to Detect Duplicates in Stack Overflow[23]	2017		Recall-rate@k	recall5: 0.2 recall20: 0.4

Table 2-2: Related works on the task of classifying two post as duplicated

Article	Year	Dataset	Measure	Results
Detecting Duplicate Posts in Programming QA Communities via Latent Semantics and Association Rules[34]	2017	SO 2010-2016	Recall f1 score	0.87 0.9
Duplicate Detection in Programming Question Answering Communities[35]	2017		Recall	0.89
Duplicate Question Detection in Stack Overflow: A Reproducibility Study[27]	2018	msr 2015	Recall	0.84
Feature Analysis for Duplicate Detection in Programming QA Communities[36]	2017		Recall	0.95

3 Predicting the programming language: extracting knowledge from Stack Overflow posts

The goal of this chapter is to explore mechanisms to extract knowledge from the questions in *Stack Overflow*. To accomplish this, we used textual and source code information in every question to find relationships between programming languages used. It is proposed to extract non-evident relationships of the programming languages based in how the developers ask about their problems. Beside, we want to explore the contribution of features extracted from source code and text related to those tags.

This chapter is based on the article *"Predicting the programming language: extracting knowledge from Stack Overflow posts"* [5] published in the Colombian Computing Conference 2017.

This chapter is structured as follows: Section 3.1 shows the proposed method describing step by step the proposal. Section 3.2 presents the conducted experiment: dataset, experimental evaluation, and results. Finally, Section 3.3 concludes the chapter.

3.1. Method

We want to extract information about the relationships from programming languages in Stack Overflow. To perform this task we build a classifier model that uses word embeddings. This classifier allows to classify a post by tag using textual and code information (modalities).

Figure **3-1** shows an overall description of the method, which includes three stages: preprocessing, classification, and evaluation.

3.1.1. Stack Overflow

Given the data of Stack Overflow (SO) we obtain a collection of posts with title, body (text, code and images), tags, score, and information of the tag creation.

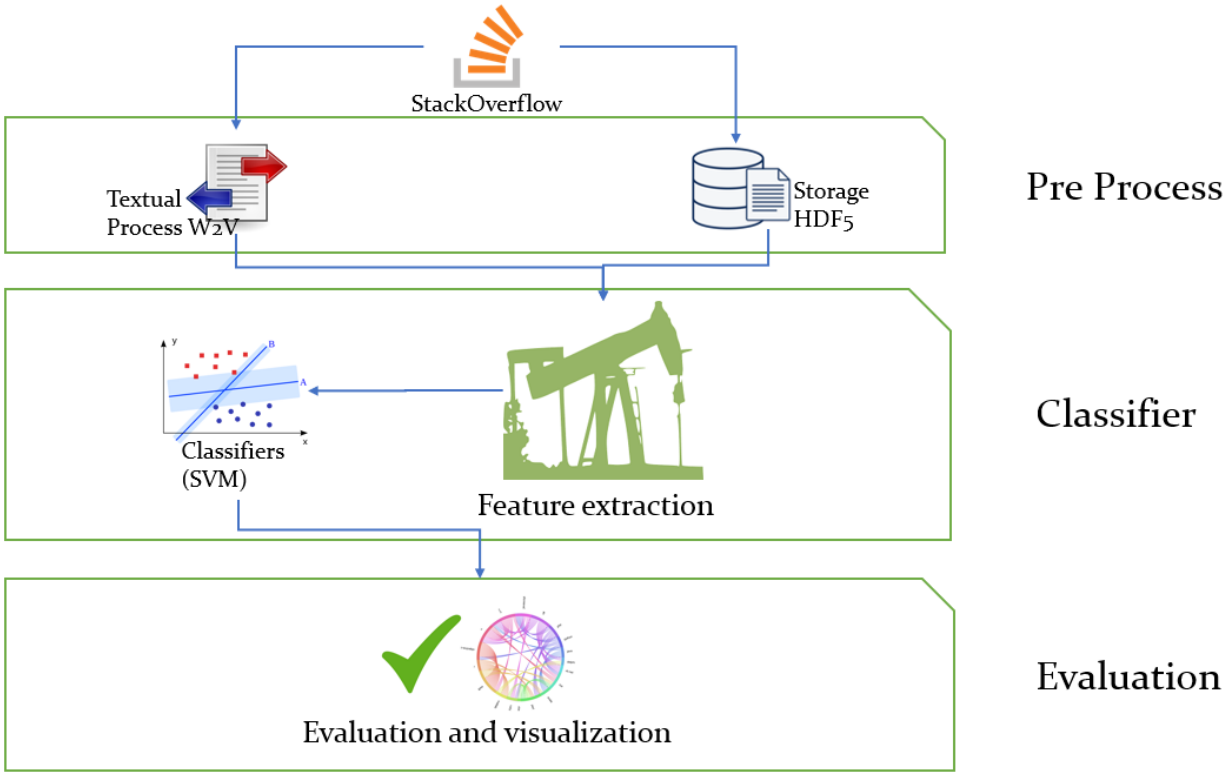

Figure 3-1: Overview of the proposed method divided in three parts: Pre-process, Classifier and Evaluation.

Tags selection

To select the programming languages used in this study, we used the Github repository and extract the list of the most popular languages. We compared that list with the SO list and selected the tags that appear in more than 5000 posts in SO.

3.1.2. Pre-process

First, the data was changed from the original format to the new format HDF5 (Hierarchical Data Format)[1], which is a format developed to work with big and complex collections of data, giving better performance, storage space, and access to data. This format allowed us to have a better mechanism in terms of time and access to information than the original XML file provided by SO.

In the process of changing the format we split the body of each post in two fields: text content and source-code content. We divided the source-code fragments (snippets) and text in columns. This was necessary to obtain a better performance in the classification tasks. The source code was selected only from the fragments in the HTML tag "<code>".

In the pre-processing step we used the text of the posts to train several classification models using different amount of posts in training. We processed the data to generate a set of 3 millions of questions, which was used to train a new *Word2Vec* model to obtain a better performance with SO concepts. Finally, we built a dataset with posts that had the two modalities (text and source code).

Text feature extraction

From the dataset we selected the posts that were identified as *Question post*. Using the text content of each post, we generate a representation of each post using the *Word2Vec* model. The *Word2Vec* model trains a neural network to embedded each word in a new vector representation space **3-2**. Each post was represented as the average of the vectors of the words in the post, generating for each post a vector representation of dimension 300.

Source code feature extraction

To represent the source code we used an approach based on n-grams of characters. We experimented with (2 - 6)-grams. We generated a vector representation of the posts using the TF-IDF and the 300 most frequent n-grams.

[1] https://www.hdfgroup.org/solutions/hdf5/

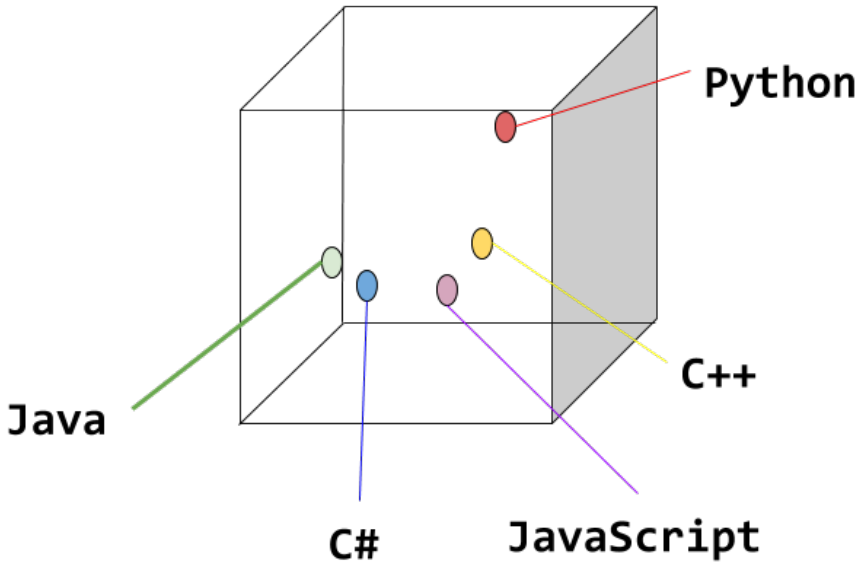

Figure 3-2: Word2Vec space representation for 3 dimensions.

3.1.3. Classifier model

The purpose of this component was to train a classifier that is able to classify a new post in one of the 18 different classes (18 programming languages tags). We selected as a classification method the *Support Vector Machine* (SVM). In this stage we compared the obtained performance of the textual and source code representations.

To represent each programming language as a vector, we used the n-gram representation. We averaged the vectors of the posts that contain source code and have the same programming language tag. It is not possible for two posts having two or more programming languages.

The n-gram representation was selected because we are interested in working with different programming languages and with different syntax. Generally, a post has a small snippet of code that not necessarily is written in the language used to tag the post.

3.1.4. Evaluation

The purpose of this component was to test a *SVM* and visualize the performance apply to this task and dataset. The classifier is evaluated using the metrics of accuracy, precision, recall and f1-score. In addition, for a better understanding of the results, we show the confusion matrix of the experiment and a graphic of the miss-classified tags. Another interesting visua-

Table 3-1: Summary of the SO dataset used in this study

Questions	Answers	Orphaned tag wiki	Tag wiki excerpt	Tag wiki	Moderator nomination
7,990,787	13,684,117	167	30,659	3,058	200

lization is how the programming languages are grouped in a 2D space using the Word2Vec representation.

3.2. Experimental Evaluation

This section presents details of the experimental evaluation conducted to evaluate the proposed method.

3.2.1. DataSet

We started with the SO dataset published in 2014 for the MSR (Mining Software Repositories) challenge 2015[2]. This dataset was divided into XML files, whose size is about 20 GB. We focused only on the text and source-code information of the post and its tags. The file "PostTypeId" was used to identify the relation between the question and answer of the post. A summary of the amount of posts for each type is presented in Table **3-1**.

As mentioned before we worked only with posts of type Question since these posts were originally tagged, and have a good number of related posts to use. The initial set had 38,206 tags with more than 21 millions of posts. Since tags are added manually by users, some of them occur infrequently, or always occur with the same set of tags. This motivated us to do a selection of the most interesting tags.

To do a better selection of elements we analyzed the tags versus its frequency. This was performed the plot presented in Figure **3-3**. We can see that there are some tags that rarely appear. The post that rarely appear can be ignored, so we focused on the most frequent. In addition, a selection of some programming languages was made using the GitHub platform as an external source of information to make a more fair selection of the final tags that were used in this study.

In the pre-processing step we stored the data in the HDF5 format. A new *Word2Vec* model was trained with a set of the SO posts to obtain a set of post/tags to be used in the classification step. The set of tags selected were the 18 most popular programming languages used in GitHub, that is to say, the programming languages that occur in almost 5,000 posts.

[2]http://2015.msrconf.org/

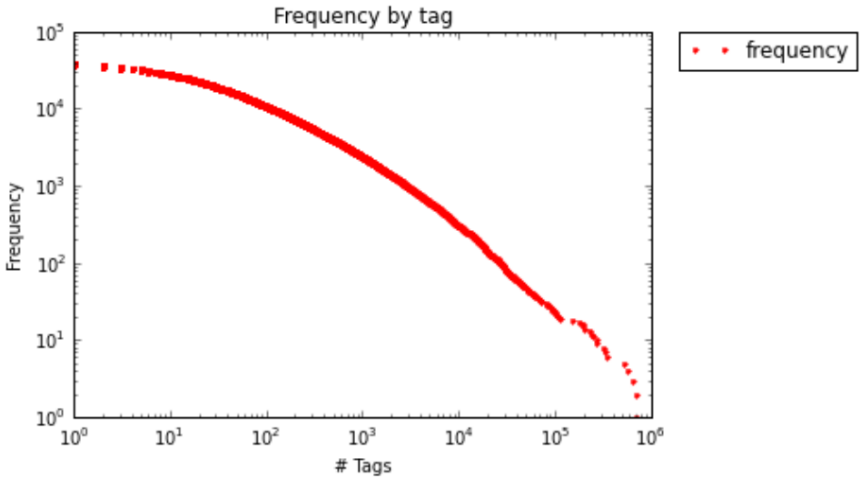

Figure 3-3: Tag frequency in the dataset.

The first tag with more than 5,000 post frequency is LUA, which is only used in 6,867 posts. This number of post gives a lower bound for doing an equitable partition in the train and test sets. We show the frequency of each tag in Figure **3-4**.

3.2.2. Experimental setup

We split the dataset in training and test sets. For training we selected 1,000 posts of each tag to train a SVM, that is to say, we selected 18,000 posts for training. For the test set we also selected 100 posts for each tag. These posts were selected in such way that they do not share a common programming language tag.

To evaluate the result of the classification task we use the *f1*-score, accuracy, precision and recall measures. These measures are commonly used to evaluate classification algorithms. We also used a set of visualizations to qualitatively measure and understand how a programming language is related each other.

We present the results in 2 parts, one part dedicated to show a visualization of the results obtained for the classification task, and the other part dedicated to visualize how programming languages are related in the feature representation space.

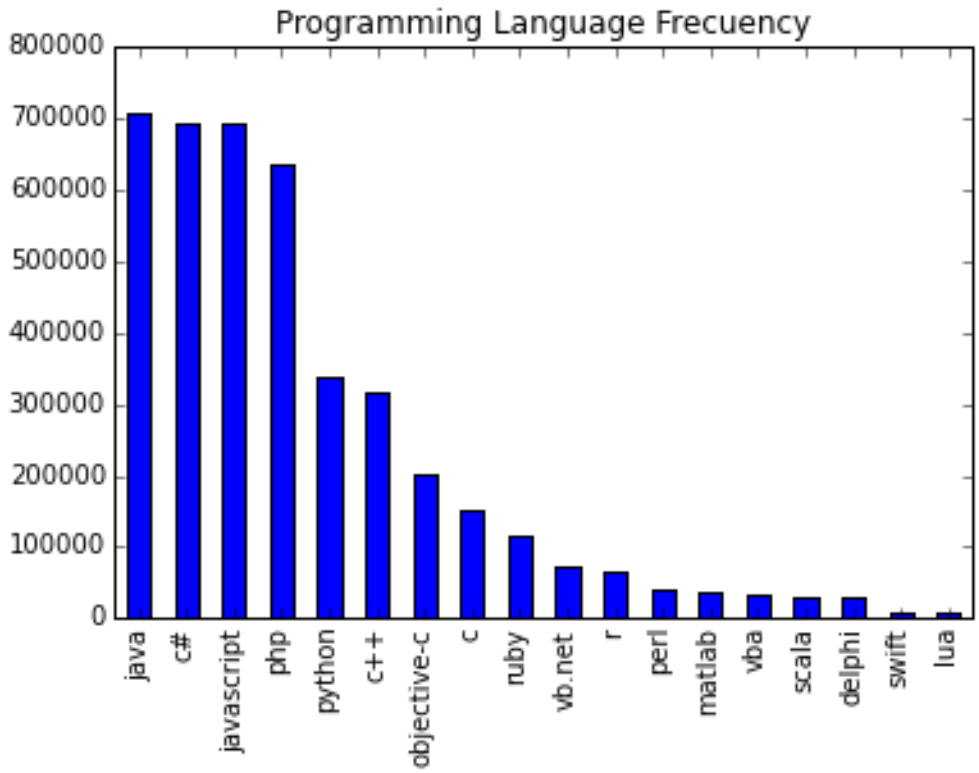

Figure 3-4: Frequency of tags for the 18 selected programming languages in this study.

	java	python	php	c#	javascript	c++	c	objective-c	r	swift	matlab	ruby	vb.net	vba	perl	scala	lua	delphi
java	57	2	5	3	3	5	1	1	2	1	1	2	0	4	0	6	3	4
python	1	57	5	0	1	1	2	1	2	2	4	7	5	1	6	3	1	1
php	2	2	67	0	5	1	0	0	2	2	0	5	4	2	5	1	0	2
c#	4	2	3	27	5	11	6	0	0	3	4	1	16	7	0	3	2	6
javascript	1	3	1	2	82	0	0	3	1	0	0	1	1	0	4	0	0	1
c++	2	1	2	7	1	32	29	1	2	2	3	0	4	1	2	2	6	3
c	3	3	2	5	2	19	37	1	3	5	2	2	0	0	5	2	6	3
objective-c	1	1	1	1	0	2	2	68	1	14	1	2	1	0	0	2	0	3
r	1	2	2	0	1	1	1	1	80	2	7	0	0	1	0	1	0	0
swift	1	2	1	0	0	1	2	20	0	66	1	1	2	0	0	3	0	0
matlab	1	1	2	1	2	0	2	3	10	2	64	2	0	2	6	0	1	1
ruby	1	7	7	2	1	0	1	3	3	1	1	64	2	1	3	3	0	0
vb.net	5	0	3	16	5	2	0	1	1	2	1	1	50	5	1	1	0	6
vba	0	0	0	3	0	1	0	0	4	0	2	0	4	83	1	1	1	0
perl	1	1	5	1	1	2	0	1	2	1	2	2	0	1	74	3	3	0
scala	7	3	2	1	1	0	1	2	0	6	0	0	0	0	2	71	1	3
lua	0	2	7	1	4	1	5	3	2	4	4	2	1	0	2	2	59	1
delphi	7	2	3	2	1	1	4	6	2	4	3	1	1	1	1	1	2	58

Figure 3-5: Confusion matrix of classification using text features.

3.3. Experimental Results

Post label prediction using text features

After running a classifier we are interested in the misclassified posts. This misclassification allows to see the similarity between typical problems in programming languages. Figure **3-5** presents a confusion matrix when the classification model is trained using text features.

Figure **3-5** and **3-6** presents a confusion matrix and the heat map when the SVM is trained using text representation. This matrix shows that most of programming languages are correctly classified, although some of them were not. It is worth noting that some of the misclassified posts belong to a programming language that is considered similar according to the programming paradigm or to the language [2].

Figure **3-7** presents a radial visualization of miss-classified posts for each language with respect to other programming languages. It is interesting to see in this visualization how some programming languages share connection with other programming languages, that is to say, the model missclassifies posts because these programming languages share common problems. For instance, objective-c and swift, C and C++, Matlab and R, etc.

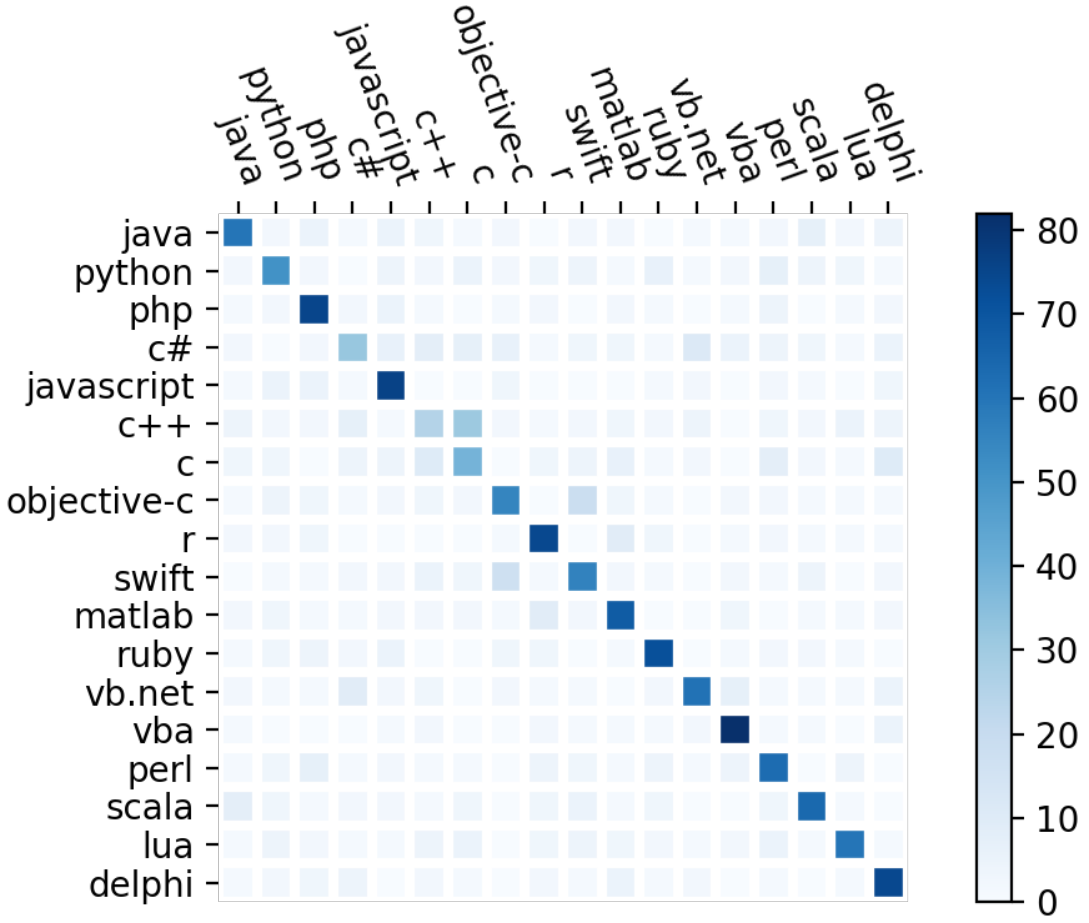

Figure 3-6: heatmap of classification using text features.

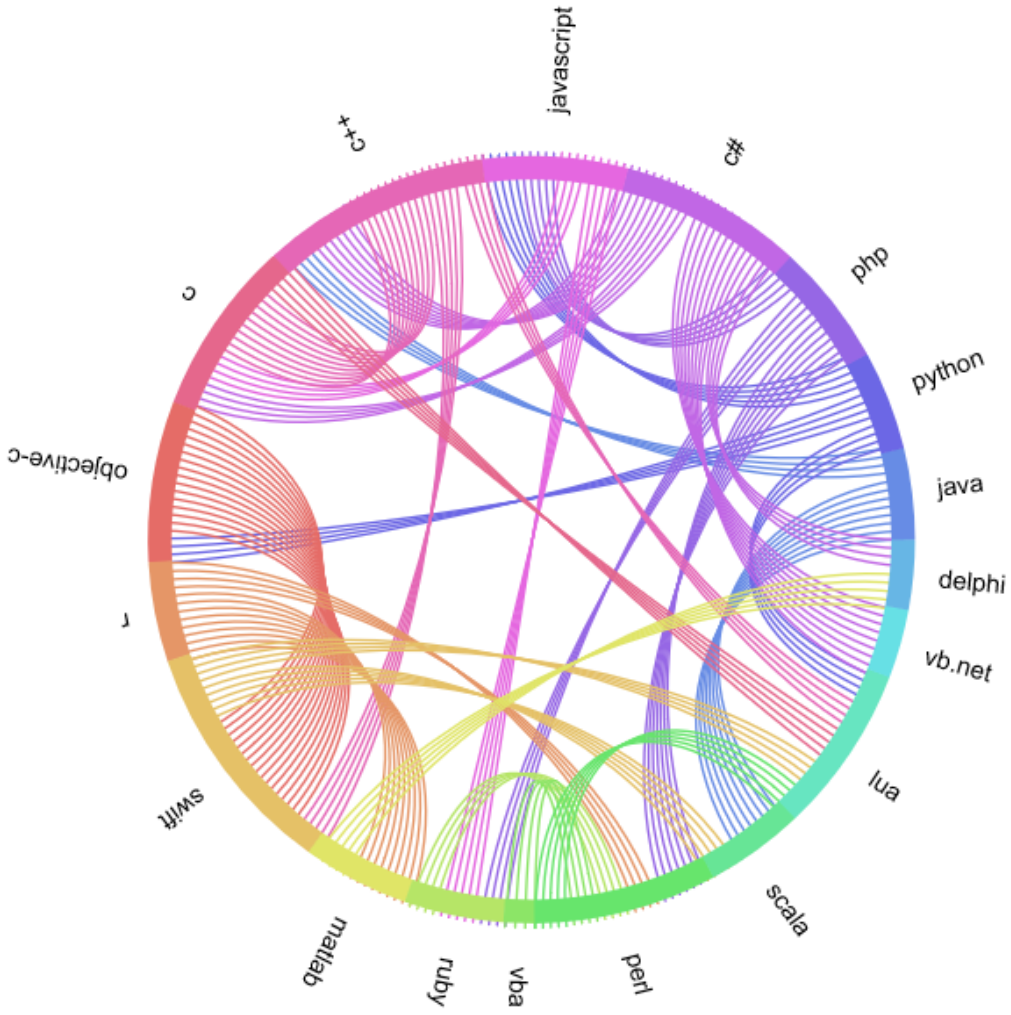

Figure 3-7: Visualization of the miss-classified posts. It is worth noting how the objective-c language is confused with swift.

	java	python	php	c#	javascript	c++	c	objective-c	r	swift	matlab	ruby	vb.net	vba	perl	scala	lua	delphi
java	13	3	3	20	19	4	5	2	2	1	5	3	1	4	1	7	4	3
python	2	44	2	0	6	1	6	6	0	3	9	3	0	5	4	6	2	1
php	6	5	23	5	15	3	5	7	1	1	7	6	1	2	8	0	3	2
c#	5	3	5	26	12	1	6	8	3	4	0	2	2	7	4	3	4	5
javascript	2	1	3	0	71	2	1	6	0	3	2	0	1	1	1	2	1	3
c++	3	4	2	4	6	38	13	6	0	1	6	0	1	3	5	2	4	2
c	2	1	0	2	3	2	61	3	2	3	6	0	0	5	5	0	4	1
objective-c	0	4	3	3	4	2	4	60	1	3	3	1	0	9	0	1	0	2
r	3	4	1	0	8	0	4	2	57	0	7	2	0	3	0	2	5	2
swift	0	5	1	3	3	1	1	10	1	59	5	0	0	2	2	1	2	4
matlab	4	7	2	2	5	4	1	4	5	3	44	4	1	2	2	2	4	4
ruby	0	7	3	3	6	2	3	7	4	1	5	40	3	6	4	2	2	2
vb.net	2	1	1	5	9	1	0	5	1	1	1	4	49	7	0	4	2	7
vba	1	1	1	2	3	0	4	9	3	3	10	2	18	36	2	1	1	3
perl	2	5	10	1	8	2	4	6	1	0	7	6	0	8	33	2	4	1
scala	6	1	0	9	5	5	2	3	1	3	2	3	0	1	3	53	2	1
lua	0	2	5	5	4	10	8	5	2	2	5	2	0	2	6	0	42	0
delphi	2	4	2	4	2	1	0	5	5	5	3	2	1	7	0	2	1	54

Figure 3-8: Confusion matrix of classification using code-source features.

When we evaluate the classifier using only textual information, we obtain a *F1* score = 0.6024, accuracy = 0.6088, precision = 0.6010, and recall = 0.6088.

Post label prediction using source code features

Using only source code information of the question and building a n-gram feature representation, the model presents a lower performance with respect to the model based on text representation. Figure **3-8** and **3-9** shows the obtained confusion matrix and the heatmap for the SVM using code features.

Moreover, Table **3-2** shows the consolidated performance measures for the classifier for the two modalities. For understand more about the misclassification seen in the previous confusion matrices we show in the Figures **3-10** and **3-11** the representation of the programming languages tags in a 2d space, using the textual and code representations, respectively. This is useful to show groups and distances between different tags of programming languages.

Language landscape visualization

We build a 2 dimensional visualization of the programming languages using the feature representation. Figure **3-10** presents a 2D visualization of the feature representation of each

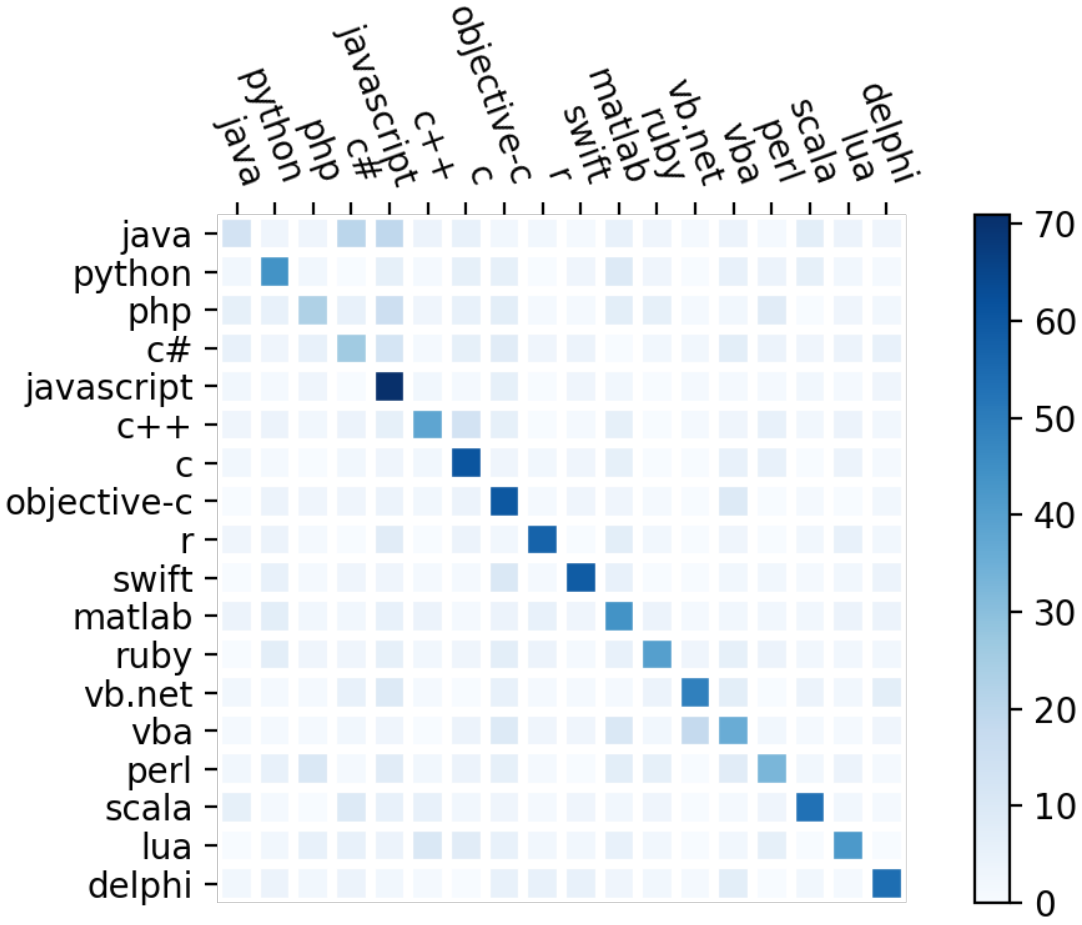

Figure 3-9: Heatmap of classification using code-source features.

Table 3-2: Performance of the classifier using text and source-code representations.

Modality	F1-score	Accuracy	Precision	Recall
Source	0.6024	0.6088	0.6010	0.6088
Code	0.4402	0.4461	0.4509	0.4461

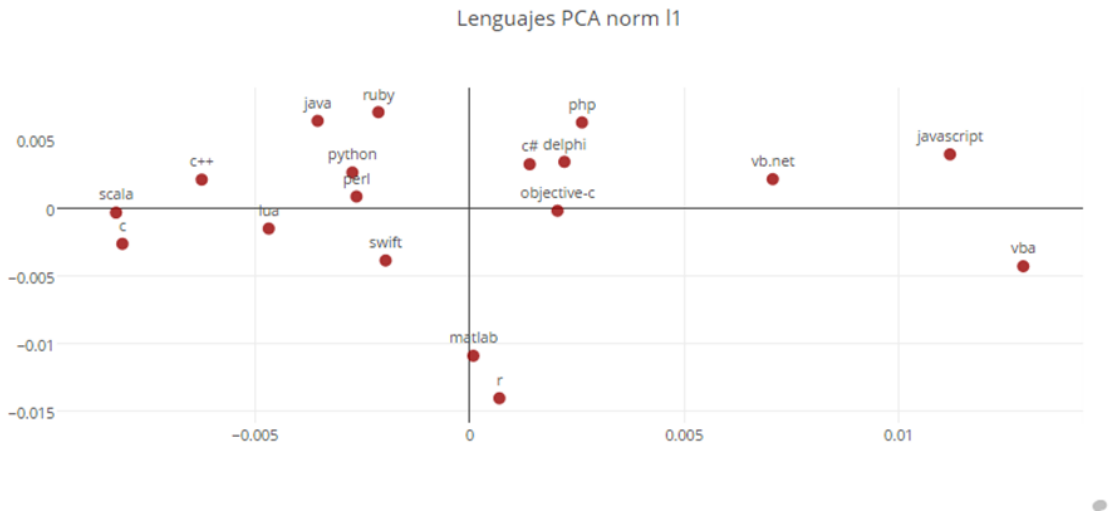

Figure 3-10: Posts in a 2D space using text features.

programming language when using textual features. This representation is an average of all representations of the posts sharing the same programming language. Vectors were normalized using the L1-norm. To plot each programming language vector (embedding space with word2vec), we applied the principal components analysis algorithm (PCA), which generates a set of components from which we selected the two most important to be used as coordinates to plot in a 2d coordinate system. It is expected in this visualization that similar points are projected close each other. Figure **3-10** presents the obtained visualization. This visualization is very interesting because allows to see relations between programming languages. For instance, Matlab is very close to R, which makes sense because they share many things in common, and in this case, they share similar problems in the developers community.

Figure **3-11** presents also a 2D visualization but using the code-source embedding representation representation obtained with word2vec. It is worth noting that this visualization allows to see correlations between questions of programming languages but in terms of code snippets. It is important to note that the obtained relations depicted in this visualization are different to the text visualization.

In this experiment we explored over extracting information from questions in SO. To represent the questions, we used the textual and the source code components. For the specific case of SO, we saw that the source code itself was not providing enough information about the programming language but the textual component was more meaningful in this task. We visualized 18 programming languages and how they relate to each other. We saw a cluster of some programming languages used in similar topics and context. For instance, Matlab and R are close to each other and far away from another programming languages in our visualiza-

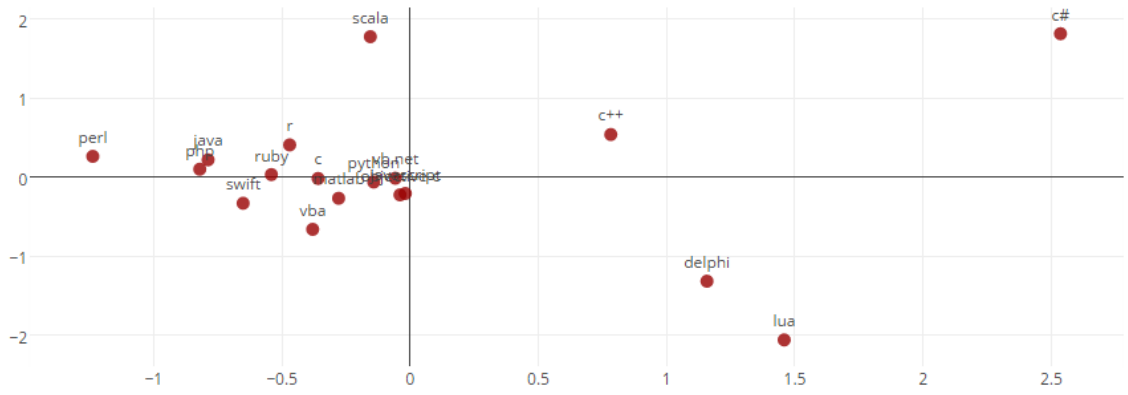

Figure 3-11: Posts in a 2D space using code features.

tions. We can also see the relationship between programming languages that are close to each other in the visualization and in fact they are similar, for example PHP and Objective-c. Results show a relationship between programming languages based on how people ask about related problems.

4 Identifying duplicated posts in Stack Overflow

The experiments described in this Chapter are related to extract specific information from the questions in Stack Overflow unlike the experiment described in the Chapter 3 that was aimed to extract more general relationships between the posts. The goal in this experiment is to identify when a post is duplicated or not using a multimodal approach. We extracted features from the text in the post (tittle and description of the questions itself) and from the snippets of source code in the post. Another goal is to analyze the impact of using the code snippets in the task of finding duplicated post in SO.

Our approach was modeled as a classification problem. We tried to classify a set of posts as duplicated or not given a post considered duplicated. This is basically a two-classes classification problem, being *duplicated* and *not duplicated* the two classes. The possible application of this experiment is being able to identify from a set of several post duplicated ones and then provide a less noisy result in the case of someone consulting for a specific question.

We wanted to show the relevance of the source code to identify duplicates. In order to create a more robust representation we use both features coming from the question itself (text), and features coming from the source code in the question to try to obtain more information from the post.

This chapter is structured as follows: Section 4.1 describes the method used. In this Section we present details describing the pre-process of the data, the feature extraction process, and the general classification stage. Next, Section 4.2 describes the experimental evaluation, including the dataset, the classification models used, and the results to conclude the experiment.

4.1. Method

In this Section we give a context of the proposal method, and a description of it's modules. Figure **4-2**, presents the big picture of the method. It is subdivided in 3 stages: Pre-process, Classification, and Evaluation.

The first stage, **Pre-process**, described in the Subsection 4.1.1, presents the processing of the data found in Stack Overflow, and the explanation of how to create a new structure that helps in the next steps. For instance, creating the AST of the snippets of source code and the process of a more specific Word2Vec (trained with posts from Stack Overflow) for the textual elements that can be found in a common post in Stack Overflow.

The second stage, **Classification**, described in the Subsection 4.1.2, describes 3 things: the feature extraction from source code and text components, common approaches which features to extract and how to use both type of features to find how relevant are them to solve the task.

The third and last stage, **Evaluation**, describes the result of using the classification models with the data sets.

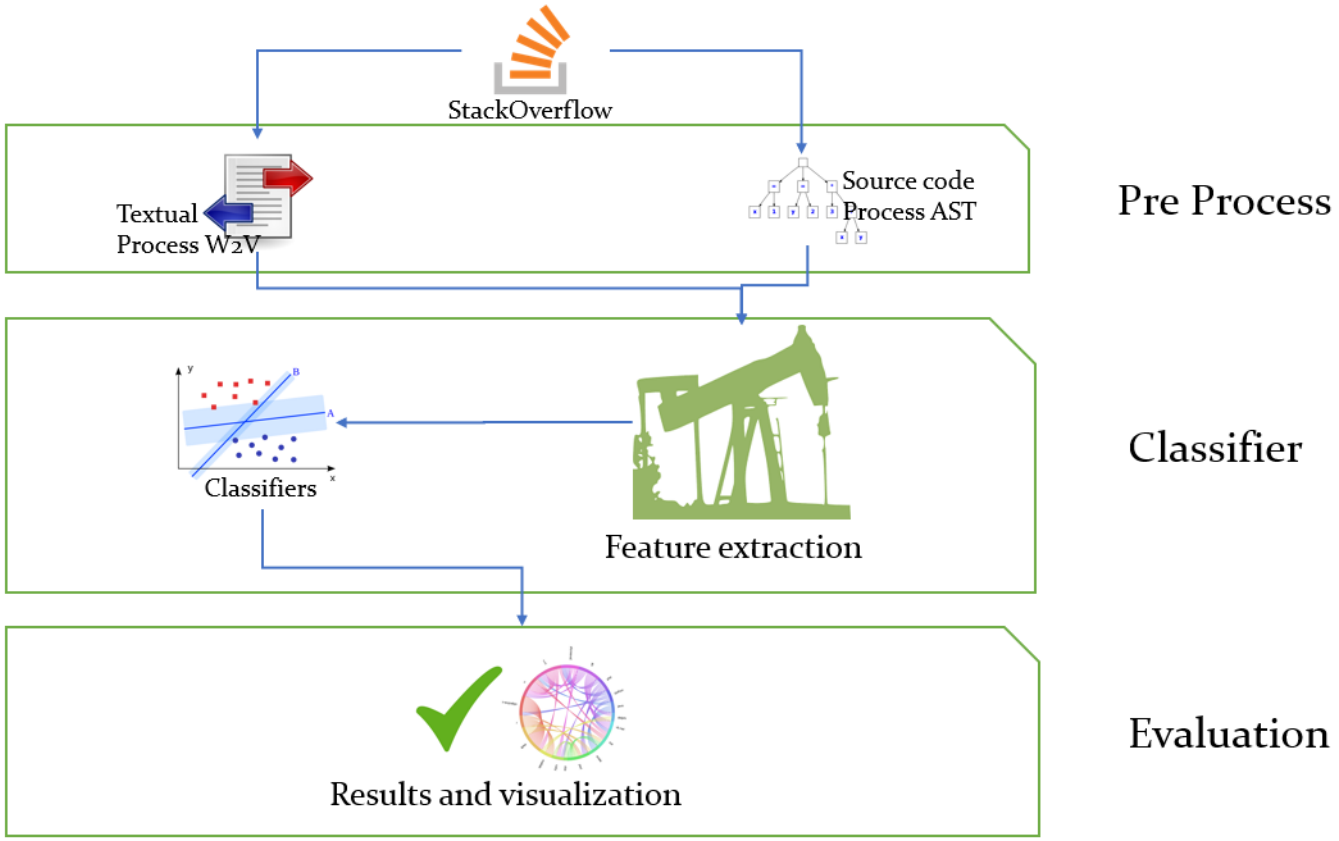

Figure 4-1: Method for the duplicate posts detection in Stack Overflow

4.1.1. Pre-processing of the data

We extracted posts from Stack Overflow including information such as the title, body (text and source code), tags, score, and duplicated status. After, a new Word2Vec model was trained with the text from the posts to, as mentioned in the Section 3.1.1, to have a better idea of the concepts in SO.

Beside creating a new Word2Vec model, an Abstract Syntax Tree (AST) representation was build for the source code (for every snippet of source code in each post) using ANTLR[1], which is a well-known parser generator tool. ANTLR provides a Java grammar[2] to create ASTs from Java source code. However, since the snippets were, sometimes, only a small fraction of code, we needed to adapt the Java grammar to supports, compile and create ASTs from the small excerpts of source code. This grammar supports *no usual* Java source code such as not starting with complete Java classes. Its purpose was to support the biggest amount of snippets and to parse the largest amount of source code. Nonetheless, the proposed grammar is ambiguous. An example of one of the snippets supported by this new Java grammar for snippets, and not by the well known Java grammar included in ANTLR is shown in Figure **4-2**. This is an example of an usual post of Stack Overflow: a short fragment of code that maybe will not compile.

Figure 4-2: Example of snippet of a Java code in SO

[1]https://www.antlr.org/
[2]https://github.com/antlr/grammars-v4

Table 4-1: ASOBEK features

Name	Description
c1w2	Characters Unigram - Word Bigram
c1c2	Character Unigram - Character Bigram
w1w2	Word Unigram - Word Bigram
w1c2	Word Unigram - Character Bigram

4.1.2. Classification

Text feature extraction

We generated a representation of the text in each post using several representations:

- Word2Vec: Each post was represented as the sum, concatenation, subtraction and multiplication of the vectors.
- Asobek: Each post was represented as different combinations of unigrams of characters, unigramas of words, bigrams of character and bigrams of words **4-3**. The features are described in the Table **4-1**.
- n-grams: Each post was represented with several n-gram with several sizes for n from 1 to 6 n-grams. The description of the features using n-grams are described in the Table **4-2**.

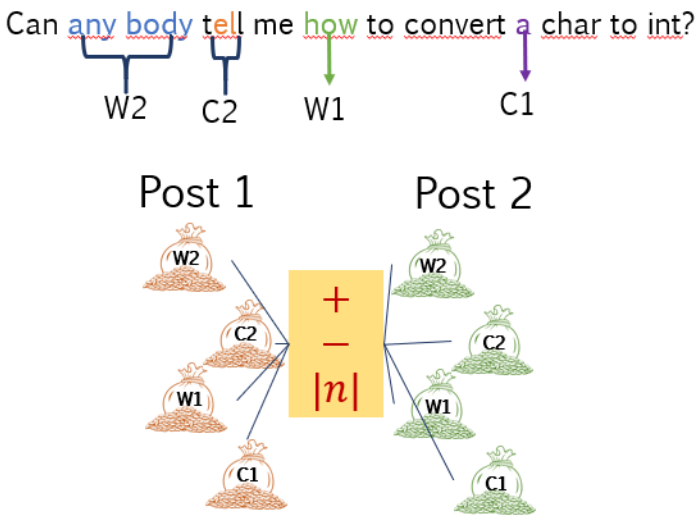

Figure 4-3: ASOBEK representations

Table 4-2: N-grams features

Name	Description
Sum	Sum of the n-grams vector
Dot	Dot product of the n-grams vector
Conc	Concatenation of the n-grams vector
Rest	Subtraction of the n-grams vector
Mult	Multiplication of the n-grams vector

Table 4-3: Source code features extracted from the AST

Definition	Count	Type
Number of if, if-else, do, while, for, switch	6	Integer
Number of literals	1	Integer
Depth of the AST	1	Integer
Average of children nodes in the AST	1	Decimal
Number of nodes	1	Integer
Number if + Number of cycles + 1 (Cyclomatic complexity)	1	Integer

Source-code feature extraction

For the source code, as mentioned in the pre-processing of the data, an AST was extracted from each snippet of code. From the AST, 11 features were extracted. They are shown in the Table **4-3**. Each post was represented as a 11 dimension vector.

Classifiers

The following classifiers were used for the classification problem:

- Nearest Neighbors (NN)
- Linear SVM (SVM L)
- RBF SVM (SVM R)
- Decision tree (DT)
- Random forest (RF)
- Multilayer Perceptron (MLPC)
- AdaBoost (ABoost)
- Naive Bayes (NB)

- Quadratic Discriminant Analysis (QDA)

4.2. Experimental Evaluation

4.2.1. Dataset

For this experiment, two datasets were created from the main dataset: Big dataset and Small dataset. Both datasets have two main components that could be defined as positive and negative duplicate posts: posts that were marked as duplicated, and posts not marked as duplicated.

The first set of posts, which are the posts marked as duplicated, were identified easily from the published dataset: Each post has a label indicating if it was marked as duplicated or not in SO. Nevertheless, the second part was not as easy, so it was created artificially by groups:

- The ones that shared the same tag.
- The ones that do not shared the same tag.
- The ones related to each other by links, this means that a question is related to another, but they were different questions.

The reason to create two dataset is that the first one was generic and exploratory, for this one we included post with several programming languages; while the second one is a subset of the first one, only using posts with Java as programming language.

Big dataset

The main characteristic of this dataset is that it includes several programming languages such as C++, Java, PHP, JavaScript, among others. The Table **4-4** shows the amount of posts by programming language in the dataset.

In this dataset, there are 5478 pairs of posts. Half of them are tagged as duplicated (positive class), and the other half are not duplicated (negative class). 60 % of the dataset was selected for training, 20 % for validation, and the remaining 20 % for testing.

The frequency of characters in the text per programming language is shown in the Table **4-5**, whereas Table **4-6** shows the frequency of characters in the source code part for programming language.

Table 4-4: Amount of post by programming languages in the dataset

Programming language	**Frequency**
Javascript	11674
PHP	10924
Java	10153
C#	7467
C++	6845
Python	6019
Jquery	5754
C	4201
HTML	3861
MySQL	3742

Table 4-5: Frequency of characters per post in the text per programming language

Programming language	**Frequency**	**Average characters**
Javascript	3626687	310
PHP	3545937	324
Java	3191030	314
C#	2497215	334
C++	2283501	333
Python	1934156	321
Jquery	1654341	322
C	1224923	291
HTML	1249732	323
MySQL	1242993	332

Table 4-6: Frequency of characters in the source code per programming language

Programming language	Frequency	Average characters
Javascript	2977354	255
PHP	3069451	280
Java	2858716	281
C#	2057990	276
C++	1730543	252
Python	1444160	239
Jquery	1693213	294
C	923656	219
HTML	1053740	275
MySQL	1118859	299

Small dataset

We decided to narrow to only use one programming language to make the extraction of features from the source faster, easier and more specific code. Java was selected because of the large amount of posts about this programming language.

This dataset has 3250 pairs of duplicated post: 600 pairs were used as validation and 650 as test. The rest of them were used for training.

4.2.2. Results

The experiments were executed with several representations over both datasets as mentioned in the Section 4.1.2. Figure **4-4** presents a summary of the proposed experiment. For the big dataset, only the two-classes classification was implemented. For the small dataset, we implemented the classification and retrieval experiments. The experiments were evaluated using *f1*-score where the *f1* is defined in the Equation 4-1.

$$f1 - score = 2 * \frac{precision * recall}{precision + recall} \tag{4-1}$$

Big dataset - Classifying duplicated post

In this experiment, we were interested on how to classify a post as duplicated. This allows us to see the similarity between two posts and then provide a less noisy environment when asking a question in SO. The experiments were executed in several groups depending on the text used (text in the snippet and tittle), source code in the snippet, and different combinations between them and different features representations.

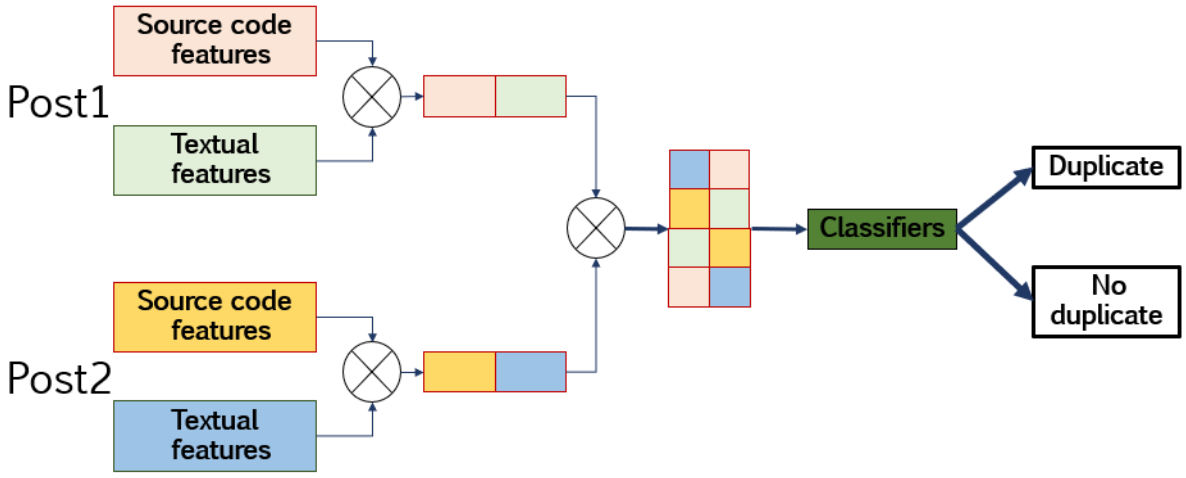

Figure 4-4: Summary of the experimental process

Table 4-7: Big dataset. *f1* score using ASOBEK with titles

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
c1w2	.63	.68	.65	.70	.68	**.71**	.68	.66	.70
w1w2	.71	.74	.73	.73	.74	**.74**	.74	.70	.59
c1c2	.75	.80	.78	.78	.77	**.80**	.78	.72	.77
w2c2	.73	.80	.77	.78	.76	**.80**	.77	.70	.60

The first group of experiments were held only using the text from the question itself and the source code in the snippet. Table **4-7** shows the f1-score for the nine classifiers mentioned using the Asobek representations extracted from the titles in the posts. On the other hand, Table **4-8** shows the f1 results for the classifiers also using Asobek, but this time with the text from the posts. In both cases the Multilayer Perceptron got the best results on the *f1*-score for all the representations, except for unigrams of characters and bigrams of characters in the Table **4-8**, where the SVM L got the best f1 score, but only for few centimes points.

Regarding to the titles, the best *f1* score was obtained using the representation of bigrams of words and the bigrams of characters, in opposite to the Table **4-8** where the best results were obtained using the unigrams of characters and bigrams of characters.

For Asobek using the source code as text, the Table **4-9** shows that the best *f1* scores were also obtained with the MLPC. The only instance where MLPC didn't have the hightest *f1* score was using unigrams of words and bigrams of words. In this case, the SVM L was also the second classifier with best performance.

Table 4-8: Big Dataset. *f1* score using ASOBEK with Text

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
c1w2	.57	.65	.61	.63	.65	**.66**	.65	.62	.61
w1w2	.65	.68	.70	.65	.66	**.70**	.69	.63	.63
c1c2	.57	**.72**	.63	.61	.61	.72	.67	.56	.70
w2c2	.63	.70	.68	.65	.65	**.71**	.68	.62	.61

Table 4-9: Big Dataset. *f1* score using ASOBEK with source code.

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
c1w2	.64	.67	.68	.69	.68	**.70**	.66	.65	.66
w1w2	.65	.66	**.70**	.67	.68	.69	.69	.63	.64
c1c2	.64	.70	.69	.64	.65	**.73**	.69	.64	.72
w2c2	.67	.70	.70	.67	.69	**.72**	.70	.64	.66

Another group of experiments were held using features extracted from the vectors of Word2Vec. Several combinations of those vectors were used such as:

- Sum: Sum of Word2Vec vectors.
- Subs: Substraction of Word2Vec vectors.
- Dot: Dot product of Word2Vec vectors.
- Mult: Multiplication one by one of the elements belonging to the Word2Vec vectors.
- Conc: Concatenation of Word2Vec vectors.

In general the MLPC classifier had the higher *f1* scores. Regarding the source of the features, Tables **4-11** and **4-12** reflect that the worse *f1* scores were obtained from plain text such as the Tittle of the post and the Text of the posts. Being the Text the source with the lowest-highest *f1* score: 0.68 with a SVML and a multiplication of vectors.

Table 4-10: Best *f1* scores in the Big dataset per representation and models using ASOBEK

ASOBEK		
Representation	**Model**	**f1 Score**
Titles	**W2C2 MLPC**	**.80**
Text	C1C2 SVML	.72
Code	C1C2 MLPC	.73

Table 4-11: Big Dataset. *f1* score using Word2Vec with Titles

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.60	.52	.50	.54	.57	.60	**.66**	.59	.50
Subs	.51	.57	.51	.59	.62	.65	**.66**	.64	.63
Dot	.55	.57	.54	**.59**	.54	.58	.56	.51	.50
Multi	.60	**.77**	.50	.60	.62	.75	.69	.68	.67
Conc	.58	.56	.50	.60	.52	**.66**	.58	.57	.53

Table 4-12: Big Dataset. *f1* score using Word2Vec with Text

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.56	.54	.50	.52	.54	**.57**	.56	.57	.55
Subs	.53	.57	.51	.56	.60	.62	.61	.61	**.62**
Dot	.56	.60	.57	.57	.54	**.62**	.56	.56	.62
Multi	.56	**.68**	.51	.55	.55	.67	.60	.57	.56
Conc	.55	.60	.50	.52	.53	**.64**	.57	.54	.63

When we tried to combine both sources, title and text, as presented in the Table **4-13**, we saw a substantial increase in the *f1* score in the highest score with the multiplication representation, but also with the others representations. In the Tables **4-11**, **4-12** and **4-13** we can also see that the best representation for text in our data-set was the multiplication one by one of the elements of the vector combined along with a MLPC classifier. Other classifiers such as SVML and Ada Boost got also one of the highest *f1* scores combined with other representations such a sum of the vectors and dot product even though none of those combinations were better than Multiplication of vectors along with MLPC.

Regarding the source-code component of the post, we made several experiments combining text with source-code. Table **4-14** presents the results of the classification combining the title and the source-code snippet. Table **4-15** presents the classification results combining

Table 4-13: Big Dataset. *f1* score using Word2Vec with Titles and Text

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.60	.56	.50	.55	.56	.59	**.65**	.62	.53
Subs	.50	.59	.51	.56	.59	.66	.68	**.69**	.67
Dot	.57	**.62**	.54	.60	.54	.60	.59	.57	.58
Multi	.61	.77	.50	.60	.61	**.77**	.69	.70	.68
Conc	.58	.57	.50	.60	.53	**.65**	.58	.58	.51

Table 4-14: Big Dataset. *f1* score using Word2Vec with Titles and Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.60	.56	.50	.60	.61	.61	**.65**	.62	.51
Subs	.52	.57	.51	.60	.63	.66	**.69**	.68	.68
Dot	.58	.60	.54	.59	.58	**.63**	.57	.58	.60
Multi	.61	.78	.50	.62	.62	**.79**	.70	.67	.67
Conc	.60	.59	.50	.61	**.69**	.60	.59	.58	.50

Table 4-15: Big Dataset. *f1* score using Word2Vec with Text and Source-Code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.59	.55	.50	.57	.60	.59	**.63**	.59	.58
Subs	.54	.60	.51	.60	.62	**.67**	.66	.65	.60
Dot	.59	.57	.55	.58	.57	**.61**	.57	.50	.62
Multi	.60	.72	.50	.56	.54	**.74**	.63	.58	.58
Conc	.62	.59	.50	.60	.58	**.66**	.58	.60	.51

the text in the post and the source-code snippet and the Table **4-16** presents the results of combining both the text from the tittle and post and the source-code snippet. In all of them, the best results were obtained with the multiplication of vectors along with the MLPC classifier.

On average, as can be seen in the Tables **4-14** and **4-15** the *f1* score was higher using the source-code along with the title of the post instead of the text of the post. The difference in the highest score is noticeable, from 0.74 with the text to 0.79 with the tittle. Even though, the best results were obtained using the title, the text of the post and the source-code snippet, in this case the *f1* score was 0.80. This is the best *f1* score of all this set of experiments.

As seen in the Table **4-10** the text provides good information in the task of finding duplicates, this could be because the title can be considered as a short description of the whole content

Table 4-16: Big Dataset. *f1* score using Word2Vec with Text, Titles and Source-Code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.62	.55	.50	.56	.61	.61	**.66**	.64	.57
Subs	.50	.58	.51	.61	.63	.66	.72	**.68**	.59
Dot	.58	.61	.54	.58	.58	**.63**	.58	.57	.62
Multi	.60	.79	.50	.62	.58	**.80**	.69	.67	.62
Conc	.62	.60	.50	.62	.56	**.69**	.59	.60	.50

Table 4-17: Best *f1* scores for the big dataset using Word2Vec

Word2Vec		
Representation	**Model**	**f1 Score**
Titles	MULTI SVML	.77
Text	CONCA SVML	.68
Titles + Text	MULTI MLPC	.77
Titles + Code	MULTI MLPC	.79
Text + Code	SUBS MLPC	.74
Text+ Titles+ Code	**MULTI MLPC**	**.80**

of the post meanwhile the text in the post is a description of a particular case, so the text in the post may contain more noise. We can also see the effects of including the source code reflected in higher *f1* score, in the table **4-17**, we can see that the worst *f1* score is obtained by only using the text (0.68), but adding the source code increases the *f1* score to 0.74, an increase of 0.06 points. We can also see that using the titles combined with the text does not make a great difference than only using the titles, both had a 0.77 *f1* score. Combining the three sources (title, text and source code) we had the best results, a *f1* score of 0.80 which is an 0.01 increase on the score than only using the titles and the code. This may imply that even though the text of the post is not providing much information, it helps to solve the classification task but is the tittle and the source code the ones providing more information.

Small dataset - Classifying duplicated post

Using this data set we solved the task of classifying posts as duplicated or not; In this case we included features not only from text but also from source code as mentioned in the Section 4.1.1.

In this dataset, several configuration of the classification were implemented also with Asobek and Word2Vec. Regarding Asobek, Table **4-18** describes the classification only using the text from the source-code snippet; Table **4-19** describes the results of using the title of the posts and Table **4-20** describes the results using the text of the posts. In all the cases the highest *f1* score was obtained with a unigram of character and bigrams of character along with a SVM R.

In this case, in the Table **4-21** we can see that the best *f1* was obtained only using the title of the post and the difference of classification is considerable, being 0.87 for the title 0.81 for the source-code and 0.80 for the text in the post.
Regarding the vector representation using n-grams, we tried the classification task with several combinations:

Table 4-18: Java Dataset. *f1* score using ASOBEK with Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
c1w2	.70	.70	**.80**	.70	.76	.73	.72	.68	.70
w1w2	.68	.68	**.75**	.70	.68	.71	.72	.67	.65
c1c2	.73	.71	**.81**	.70	.72	.74	.73	.67	.78
w2c2	.70	.70	**.79**	.72	.73	.75	.76	.67	.68

Table 4-19: Java Dataset. *f1* score using ASOBEK with Titles

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
c1w2	.66	.70	**.77**	.69	.70	.70	.68	.68	.70
w1w2	.72	.73	**.76**	.74	.74	.73	.74	.70	.71
c1c2	.75	.81	**.87**	.79	.76	.82	.79	.74	.82
w2c2	.75	.80	**.84**	.79	.79	.79	.77	.71	.77

Table 4-20: Java Dataset. *f1* score using ASOBEK with Text

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
c1w2	.63	.65	**.78**	.68	.67	.65	.69	.62	.62
w1w2	.67	.65	**.73**	.66	.69	.67	.71	.62	.63
c1c2	.62	.70	**.80**	.68	.68	.70	.72	.55	.69
w2c2	.69	.67	**.76**	.66	.71	.70	.72	.62	.62

Table 4-21: Best *f1* scores in the Java dataset per representation and models using ASOBEK

ASOBEK		
Representation	**Model**	**f1 Score**
Titles	**W2C2 SVMR**	**.87**
Text	C1C2 SVMR	.80
Code	C1C2 SVMR	.81

Table 4-22: Java Dataset.*f1* score using N-grams with Titles

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.64	.60	.78	.62	.66	**.80**	.69	.63	.78
Subs	.52	.60	.78	.61	.68	**.83**	.72	67	.82
Dot	.65	.64	.79	.59	.58	**.83**	.62	.62	.77
Multi	.64	.81	.78	.63	.70	**.88**	.78	.72	.80
Conc	.67	.70	.78	.61	.59	**.82**	.61	.62	.77

- Only using the titles of the posts. Table **4-22**.
- Only using the Text of the posts. Table **4-23**.
- Combining the Title and the Text of the posts. Table **4-24**.
- Combining the Title and the source-code of the post. Table **4-25**.
- Combining the Text and the source-code of the post. Table **4-26**.
- Combining the Title, the Text and the source-code of the post. Table **4-27**.

The best *f1* scores in every representation were obtained using the MLPC classifier and in most of the cases using the multiplication of elements of the vector. Comparing the representations that only contain information from text, such as the title and the content of the post, we can see in Tables **4-22** and **4-23** that the tittle provides more information for the task of classifying. The difference between the best classifier in each representations is 0.03. In addition, as can be seen in the Table **4-24**, where the highest *f1* score is 0.86, using the text from the title and the post does not help in the classification task, the *f1* score is still better when classifying the post by titles.

As can be seen in the Tables **4-25**, **4-26** and **4-27**, adding the source-code boosts the *f1* score. All the high scores per representation were higher than 0.80. In this case the combination of title and source-code has the highest *f1* score (0.89) and using the 3 components (text, title and source-code in the Table **4-27**) has an *f1* score of 0.90. Table **4-28** presents the best results per experiments regarding this set of experiments.

On the other side, we also held the classification experiments using a n-grams representation for the text and tokens representations for the source-code. In this case we tried the classification task using the following combinations:

- Only using the snippet of source-code. Table **4-29**.
- Combining the Title and the source-code **4-30**

Table 4-23: Java Dataset. *f1* score using N-grams with Text

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.62	.59	.78	.60	.61	**.80**	.62	.63	.79
Subs	.56	.66	.78	.58	.62	**.83**	.68	.63	.78
Dot	.60	.65	**.79**	.62	.62	.77	.73	.56	.77
Multi	.62	.70	.77	.58	.62	**.80**	.68	.58	.78
Conc	.58	.71	.76	.65	.61	**.85**	.66	.60	.82

Table 4-24: Java Dataset. *f1* score using N-grams with Titles and Text

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.65	.68	.77	.61	.61	**.80**	.70	.65	.78
Subs	.50	.71	.78	.61	.65	**.83**	.74	.71	.78
Dot	.60	.73	.79	.60	.60	**.83**	.68	.53	.80
Multi	.63	.82	.77	.65	.62	**.86**	.72	.72	.84
Conc	.65	.82	.77	.65	.57	**.83**	.66	.65	.82

Table 4-25: Java Dataset. *f1* score using N-grams with Titles and Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.61	.67	.78	.65	.67	**.80**	.70	.65	.77
Subs	.54	.69	.78	.68	.72	**.83**	.76	.71	.81
Dot	.65	.73	.79	.65	.62	**.86**	.68	.59	.80
Multi	.65	.85	.76	.65	.66	**.89**	.78	.70	.80
Conc	.68	.81	.77	.66	.64	**.85**	.66	.65	.78

Table 4-26: Java Dataset. *f1* score using N-grams with Text and Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.62	.66	.78	.69	.68	**.81**	.68	.63	.80
Subs	.58	.72	.78	.66	.70	**.87**	.73	.69	.78
Dot	.62	.72	.79	.67	.68	.76	.70	.58	**.80**
Multi	.61	.80	.78	.63	.67	**.85**	.71	.61	.82
Conc	.66	.82	.78	.65	.64	**.84**	.71	.62	.80

Table 4-27: Java Dataset. *f1* score using N-grams with Titles, Text and Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.66	.73	.78	.67	.67	**.81**	.72	.67	.79
Subs	.52	.80	.78	.66	.66	**.85**	.75	.72	.77
Dot	.62	.76	.78	.63	.62	**.83**	.72	.55	.80
Multi	.60	.87	.77	.65	.64	**.90**	.77	.70	.83
Conc	.67	.82	.77	.68	.64	**.84**	.67	.65	.80

Table 4-28: Best *f1* scores for the Java dataset using n-grams

N-GRAMS		
Representation	**Model**	**f1 Score**
Titles	MULTI MLPC	.88
Text	CONCA MLPC	.85
Titles + Text	MULTI MLPC	.86
Titles + Code	MULTI MLPC	.89
Text + Code	Subs MLPC	.87
Text+ Titles+ Code	**MULTI MLPC**	**.90**

- Combining the Text and the source-code. Table **4-31**.

Once again, the MLPC was the best classifier. With most of the representations it had the highest *f1* score. In addition, as can be seen in the Table **4-31**, using the text of the post and source-code from the snippet, other classifiers such as SVML and QDA also performed well with 0.72 and 0.82 in the *f1* score, respectively. Using the text of the post and the source-code increases the *f1* score, in this case it went from 0.83 (Table **4-29**) only using source-code to 0.84 using both the text and the source-code of the post (Table **4-31**).

As it can be seen in the Table **4-29**, the source-code itself was useful and performed well in the classification task. In this case, the best representation was the concatenation of the vectors. Nevertheless, when we used the title of the post along with the source-code we had the best results. We can see the results of this classification in the Table **4-30** where all the highest *f1* score were above 0.82 being 0.86 the highest *f1* score. The Table **4-32** presents the highest score per experiment with the representation and the mode used.

The last classification experiment was using the features described in the Table **4-33** extracted from an syntactical and lexical analysis of the source-code. In this case the MLPC classifier along with multiplication of vectors element by element had the highest *f1* score: 0.77. We thought that only using the source code by itself, taking into account the syntax and lexical characteristics was going to give better results, but the source code itself was not

Table 4-29: Java Dataset. *f1* score using N-grams and tokens with Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.62	.64	.78	.64	.68	**.80**	.61	.52	.69
Subs	.64	.65	.78	.64	.69	**.83**	.68	.62	.79
Dot									
Multi	.63	.68	.71	.61	.58	**.73**	.67	.60	.72
Conc	.67	.71	.78	.66	.68	**.83**	.64	.61	.79

Table 4-30: Java Dataset. *f1* score using N-grams and tokens with Titles and Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.58	.67	.78	.68	.65	**.82**	.71	.61	.77
Subs	.52	.68	.78	.69	.69	**.83**	.76	.68	.80
Dot	.61	.72	.79	.66	.63	**.83**	.67	.59	.81
Multi	.64	.84	.78	.66	.64	**.86**	.73	.69	.64
Conc	.66	.78	.78	.67	.60	**.84**	.66	.67	.79

Table 4-31: Java Dataset. *f1* score using N-grams and tokens with Text and Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.61	.68	.78	.65	.66	**.78**	.66	.55	.78
Subs	.57	.70	.78	.66	.62	**.83**	.75	.65	.80
Dot	.61	.65	**.72**	.60	.58	.59	.52	.54	.68
Multi	.62	.79	.78	.64	.59	.81	.67	.62	**.82**
Conc	.64	.79	.78	.68	.62	**.84**	.65	.64	.82

Table 4-32: Best results in the Java dataset using N-grams and Tokens

N-GRAMS TOKENS		
Representation	**Model**	**f1 Score**
Code	CONCA MLPC	.83
Title + Code	**MULTI MLPC**	**.86**
Text + Code	CONCA MLPC	.84

Table 4-33: Java dataset. *f1* score using Lexical and syntactical features over Source-code

	NN	SVM L	SVM R	DT	RF	MLPC	ABoost	NB	QDA
Sum	.60	.68	.65	.70	.68	.71	.68	.66	.70
Subs	.60	.70	.73	.72	.60	.71	.71	.69	.61
Dot	.67	.66	.68	.72	.66	.74	.65	.66	.56
Multi	.66	.64	.56	.71	.65	**.77**	.65	.64	.55
Conc	.63	.70	.69	.72	.60	.76	.67	.69	.59

enough.

In general, we had higher *f1* scores while using the small dataset than the big data set. This is because the dataset was smaller and had only source code from one programming language, so we were able to focus more on the features related to that programming language.

Table **4-21** shows the best results using ASOBEK. In this case, the title was the best source of information, it had an *f1* score of 0.87, which is 0.06 points higher than the source code, and 0.07 points higher than the text from the posts. This could be because, as said, the title have concrete and punctual information about the post.

In this case, as can be seen in the Table **4-32**, the combination of title and source code had the best *f1* score using the n-grams of tokens. This was not an increasing of the performance in comparison of the ASOBEK representation, but here we can reaffirm that the title is a better source of information than the text. We had the best *f1* score (0.90) in the whole experiment using n-grams over the text, the title and the source code (Table **4-28**). The reason behind this is that the n-grams truly extracts information from the source code itself, and because we are only handling one programming language, the noise is less and the posts became easier to identify.

In this experiment we implemented several representations and we applied 8 different classifiers for 2 data sets. We explored different combinations between the representations and the classifiers looking for the source of information that provided the greater added value during the duplicate detection task.

Regarding to the classifiers, it is worth to say that the MLPC got the highest *F1* score in most of the experiments. Given this, we can thing about using more complex networks for future explorations on solving this task. Concerning to the representations, we can highlight that separating the title from the rest of the text was a good strategy. The title by itself can be considered as a summary and, because sometimes developers ask as question in SO before searching other similar questions, several duplicated questions can be identified from

the title. In addition, the usage of the source code was making the *F1* scores higher, this can suggest a wide open field of using the source code to solve this task, even though the big disadvantage could be the several amount of programming languages that need to be analysed.

5 Conclusions and future work

5.1. Conclusions

In general terms, we explored different ways to represent source code and text to solve the task of finding duplicates and identifying similar programming languages, also we explored how to visualize the source code as textual information. In the first place, this exploration held to a better understanding of the source code and its relationship with how the people talk about it in Question answering web pages such as Stack Overflow. We found a relationship between some programming languages and how people ask questions about them. This was possible because of the nature of Stack Overflow; It was very handy to have the amount of data SO provided and most of them with related source code. This confirms data repositories are, indeed, a great source of information that could be exploited for several others retrieval tasks.

Given the results in both experiments we can say that the source code in the task of finding duplicates is very useful, but the best representation of the post was using a multimodal approach by combining both the source code, the title and the text of the post. Even though, seems that the title was providing more information about the post, this could be because the title is a summary of the post, and Stack Overflows encourage the practice of describing the content of the post as accurate as possible using the title meanwhile in the text people have more freedom so they can talk about the particular case they are asking about, which could be very noisy. To wrap the conclusions regarding to the features, features that come from text, in this case, the title, where the ones that provided more information to solve the task nevertheless adding the source code also provided information being the source code only snippets (very short code). The source code being very useful and at the same time very short was very remarkable; most of the times we just had a few lines of source code with incomplete functionalities even though we were capable of use them to improve the results.

It was very interesting seeing how programming languages are similar based on how people talk about them. We provided a good research on this showing that there might be hidden information about source code not only in the source code but also in the natural language that describes it.

5.2. Future work

An immediate future work is extending the representation of both the source code and the text. For example, by exploring more n-grams and building more combinations over them. Also other syntax and lexical features can be calculated over the source code. We can also think of creating different datasets per programming languages, such as we did with Java, and make more features engineering focused on the specific programming languages.

It would be helpful to create a tool to, given a short description in natural language, find the source code that solves the problem in Stack Overflow. Taking into account that the title by itself was a good representation, we can think about a way of finding different manners to solve programming problems, by filtering and grouping the post by its title and then checking upon the source code used. The other way around could be also implemented using the work presented here: given a source code, find different natural language descriptions that could describe the given source code.

Beside exploring the representations, other models could also be explored to solve the task. In this case, as we had the best results using MLPC, i.e. a neural network, we think it is worth to explore other neural networks approaches to solve the task.

Bibliography

[1] Ahasanuzzaman, Muhammad ; Asaduzzaman, Muhammad ; Roy, Chanchal K. ; Schneider, Kevin A.: Mining duplicate questions in stack overflow. En: *Proceedings of the 13th International Conference on Mining Software Repositories* ACM, 2016, p. 402–412

[2] Allamanis, Miltiadis ; Sutton, Charles: Mining idioms from source code. En: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* ACM, 2014, p. 472–483

[3] Alreshedy, Kamel ; Dharmaretnam, Dhanush ; German, Daniel M. ; Srinivasan, Venkatesh ; Gulliver, T A.: [Engineering Paper] SCC: Automatic Classification of Code Snippets. En: *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)* IEEE, 2018, p. 203–208

[4] Baker, Brenda S.: On finding duplication and near-duplication in large software systems. En: *Proceedings of 2nd Working Conference on Reverse Engineering* IEEE, 1995, p. 86–95

[5] Baquero, Juan F. ; Camargo, Jorge E. ; Restrepo-Calle, Felipe ; Aponte, Jairo H. ; González, Fabio A.: Predicting the programming language: Extracting knowledge from stack overflow posts. En: *Colombian Conference on Computing* Springer, 2017, p. 199–210

[6] Binkley, David ; Feild, Henry ; Lawrie, Dawn ; Pighin, Maurizio: Software fault prediction using language processing. En: *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)* IEEE, 2007, p. 99–110

[7] Caliskan-Islam, Aylin ; Harang, Richard ; Liu, Andrew ; Narayanan, Arvind ; Voss, Clare ; Yamaguchi, Fabian ; Greenstadt, Rachel: De-anonymizing programmers via code stylometry. En: *24th USENIX Security Symposium (USENIX Security), Washington, DC*, 2015

[8] Chowdhury, Shaiful A. ; Hindle, Abram: Mining StackOverflow to filter out off-topic IRC discussion. En: *Proceedings of the 12th Working Conference on Mining Software Repositories* IEEE Press, 2015, p. 422–425

[9] Davey, Neil ; Barson, Paul ; Field, Simon ; Frank, Ray ; Tansley, D: The development of a software clone detector. En: *International Journal of Applied Software Technology* (1995)

[10] Di Penta, Massimiliano ; Xie, Tao. *Guest editorial: special section on mining software repositories.* 2015

[11] Eyecioglu, Asli ; Keller, Bill: ASOBEK: Twitter Paraphrase Identification with Simple Overlap Features and SVMs In Proceedings of 9th International Workshop on Semantic Evaluation (SemEval). (2015)

[12] Goknil, Arda ; Kurtev, Ivan ; Berg, Klaas van d.: A rule-based change impact analysis approach in software architecture for requirements changes. En: *arXiv preprint arXiv:1608.02757* (2016)

[13] Guzman, Emitza ; Azócar, David ; Li, Yang: Sentiment analysis of commit comments in GitHub: an empirical study. En: *Proceedings of the 11th Working Conference on Mining Software Repositories* ACM, 2014, p. 352–355

[14] Hassan, Ahmed E.: The road ahead for mining software repositories. En: *2008 Frontiers of Software Maintenance* IEEE, 2008, p. 48–57

[15] Hotho, Andreas ; Nürnberger, Andreas ; Paass, Gerhard: A brief survey of text mining. En: *Ldv Forum* Vol. 20 Citeseer, 2005, p. 19–62

[16] Katta, Jitendra Yasaswi B.: *Machine Learning for Source-code Plagiarism Detection*, International Institute of Information Technology Hyderabad, Tesis de Grado, 2018

[17] Khatoon, Shaheen ; Li, Guohui ; Mahmood, Azhar: Comparison and evaluation of source code mining tools and techniques: A qualitative approach. En: *Intelligent Data Analysis* 17 (2013), Nr. 3, p. 459–484

[18] Kim, Miryung ; Notkin, David: Using a clone genealogy extractor for understanding and supporting evolution of code clones. En: *ACM SIGSOFT Software Engineering Notes* Vol. 30 ACM, 2005, p. 1–5

[19] Li, Zhenmin ; Lu, Shan ; Myagmar, Suvda ; Zhou, Yuanyuan: CP-Miner: Finding copy-paste and related bugs in large-scale software code. En: *IEEE Transactions on software Engineering* 32 (2006), Nr. 3, p. 176–192

[20] Luo, Jingyi ; Sorour, Shaymaa E. ; Goda, Kazumasa ; Mine, Tsunenori: Predicting Student Grade Based on Free-Style Comments Using Word2Vec and ANN by Considering Prediction Results Obtained in Consecutive Lessons. En: *International Educational Data Mining Society* (2015)

[21] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Efficient estimation of word representations in vector space. En: *arXiv preprint arXiv:1301.3781* (2013)

[22] Mishra, P ; Padhy, N ; Panigrahi, R: The survey of data mining applications and feature scope. En: *Asian Journal of Computer Science and Information Technology* 2 (2012), Nr. 4, p. 68–77

[23] Mizobuchi, Yuji ; Takayama, Kuniharu: Two improvements to detect duplicates in Stack Overflow. En: *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on* IEEE, 2017, p. 563–564

[24] Rahman, Mohammad M. ; Yeasmin, Shamima ; Roy, Chanchal K.: Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions. En: *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)* IEEE, 2014, p. 194–203

[25] Roy, Chanchal K. ; Cordy, James R. ; Koschke, Rainer: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. En: *Science of computer programming* 74 (2009), Nr. 7, p. 470–495

[26] Scheidgen, Markus ; Smidt, Martin ; Fischer, Joachim: Creating and Analyzing Source Code Repository Models. (2017)

[27] Silva, Rodrigo F. ; Paixão, Klérisson ; de Almeida Maia, Marcelo: Duplicate question detection in stack overflow: A reproducibility study. En: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* IEEE, 2018, p. 572–581

[28] Śliwerski, Jacek ; Zimmermann, Thomas ; Zeller, Andreas: When do changes induce fixes? En: *ACM sigsoft software engineering notes* Vol. 30 ACM, 2005, p. 1–5

[29] Sun, Chengnian ; Lo, David ; Wang, Xiaoyin ; Jiang, Jing ; Khoo, Siau-Cheng: A discriminative model approach for accurate duplicate bug report retrieval. En: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* ACM, 2010, p. 45–54

[30] Tang, Lei ; Liu, Huan: Relational learning via latent social dimensions. En: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* ACM, 2009, p. 817–826

[31] Wang, Xiaoyin ; Zhang, Lu ; Xie, Tao ; Anvik, John ; Sun, Jiasu: An approach to detecting duplicate bug reports using natural language and execution information. En: *Proceedings of the 30th international conference on Software engineering* ACM, 2008, p. 461–470

[32] White, Martin ; Tufano, Michele ; Vendome, Christopher ; Poshyvanyk, Denys: Deep learning code fragments for code clone detection. En: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* ACM, 2016, p. 87–98

[33] Xia, Xin ; Lo, David ; Wang, Xinyu ; Zhou, Bo: Tag recommendation in software information sites. En: *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on* IEEE, 2013, p. 287–296

[34] Zhang, Wei E. ; Sheng, Quan Z. ; Lau, Jey H. ; Abebe, Ermyas: Detecting duplicate posts in programming QA communities via latent semantics and association rules. En: *Proceedings of the 26th International Conference on World Wide Web* International World Wide Web Conferences Steering Committee, 2017, p. 1221–1229

[35] Zhang, Wei E. ; Sheng, Quan Z. ; Lau, Jey H. ; Abebe, Ermyas ; Ruan, Wenjie: Duplicate Detection in Programming Question Answering Communities. En: *ACM Transactions on Internet Technology (TOIT)* 18 (2018), Nr. 3, p. 37

[36] Zhang, Wei E. ; Sheng, Quan Z. ; Shu, Yanjun ; Nguyen, Vanh K.: Feature analysis for duplicate detection in programming QA communities. En: *International Conference on Advanced Data Mining and Applications* Springer, 2017, p. 623–638

[37] Zhang, Yun ; Lo, David ; Xia, Xin ; Sun, Jian-Ling: Multi-factor duplicate question detection in stack overflow. En: *Journal of Computer Science and Technology* 30 (2015), Nr. 5, p. 981–997