Automatic Extraction of Conceptual Interoperability Constraints from API Documentation

MASTER THESIS

By

Mohammed Ismail Abujayyab

387380

April, 2016

MASTER OF SCIENCE

Department of Computer Science

TECHNISCHEN UNIVERSITÄT KAISERSLAUTERN

Kaiserslautern, Germany

Automatic Extraction of Conceptual Interoperability Constraints from API Documentation

MASTER THESIS

By

Mohammed Ismail Abujayyab

387380

April, 2016

First Supervisor: Prof. Dr. Dr. h.c. H. Dieter Rombach

Second Supervisor: Hadil Abukwaik, MSc.

DECLARATION OF AUTHORSHIP

I, Mohammed Ismail Abujayyab, declare that this thesis titled, 'Automatic Extraction of Conceptual Interoperability Constraints from API Documentation' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- I have acknowledged all main sources of help.

Signature

_____

Place, Date

_____

DEDICATION

This work is dedicated to my family

Mohammed Ismail Abujayyab

ACKNOWLEDGEMENTS

All praises to Almighty Allah Who makes me able to complete this task. My words of special thanks to my thesis supervisors Prof. Dr. Dr. h.c. H. Dieter Rombach and Hadil Abukwaik, MSc. who played a major role in the fulfillment of this thesis, without their valuable directions, this might be an impossible work for me. The door to Hadil Abukwaik, MSc. office was always open whenever I ran into a trouble spot or had a question about my research or writing. She consistently steered me in the right the direction whenever she thought I needed it.

Next, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study, the process of the thesis research and writing.

In the last, I am really thankful to my friends Aied Abujayyab, Mohammed Abufouda, Hafiz Aziza, Ahmed Alaraj, Ramzi Matar and Mahmoud Abujayyab who always spare their precious time for me; remain with me during different phases of life, with their sincere support and encouragement.

This accomplishment would not have been possible without them. Thank you!

Author

Mohammed Ismail Abujayyab

Table of Contents

List of Figures

List of Tables

Abstract

Successfully integrating software systems requires fulfilling their conceptual interoperability constraints that restrict their state or behavior. Typically, the only source for these information that is available for third-party clients is the API documentation. However, manually reading and analyzing the natural language (NL) text within such API documents, which is unstructured textual content, is a tedious and time consuming task and it requires lexical and linguistic analysis skills. Moreover, it might undergo many mistakes and misunderstandings leading to unexpected mismatches and cost consequences to fix them. This encouraged us to provide a means to support software analysts and the architect to help them in increasing their efficiency and effectiveness for identifying the conceptual interoperability constraints automatically rather than manually from the text in API documentations.

To achieve our goals in this research, we followed an empirical-based methodology in incorporating machine learning (ML) technologies together with natural language processing (NLP) ones. The main contributions of this thesis are wrapped within our methodology. First, we started with a manual development for a corpus, which is a collection of relevant sentences we chose from real API documentations then we manually classified them into different classes. This classification is based on the COnceptual Interoperability coNstraints (COIN) model, which has seven classes (i.e. NOT-COIN, Dynamic, Semantic, Structure, Syntax, Context and Quality). Then, we built rules for these classes. Afterwards, we decided to explore the potentials of using the ML classifiers, thus we designed the classification model that defines the frequently used patterns and terms for representing conceptual interoperability constraints in the NL text of API documents. By training the classification model on our developed corpus. We were able to run many text classification algorithms and we have achieved promising results F-measure of **70.0%** for classifying seven-classes and F-measure of **81.9%** for classifying two-classes. Finally, we implemented a plugin tool by utilizing the classifier that we trained, so this tool allows architects to classify any texts into one of these seven classes.

1 INTRODUCTION

1.1 Overview

Conceptual interoperability constraints (COINs) are restrictions on interoperable software units and their related data elements at different conceptual levels (i.e., syntax, semantics, structure, dynamics, context, and quality) [1]. For successful interoperations, such constraints need to be identified and fulfilled. Otherwise, they may cause conceptual mismatches that hinder the interoperation or even produce meaningless results, and consequently lead to expensive resolution at later project stages. Therefore, third-party clients need to effectively analyze the shared documentation of external APIs. However, manual filtering of natural language (NL) text within API documents is a tedious, exhaustive and time consuming task. To cope with these challenges, we elaborate on Abukwaik's [1] ideas of extracting a complementary set of conceptual constraints from text in API documentation using machine learning (ML) and natural language processing (NLP) technologies.
Our goal in this thesis is to support software architects and analysts in performing the conceptual interoperability analysis effectively, while keeping the associated cost of identifying COINs low. In our work, we follow a systematic empirical-based methodology that has two advantages, i.e., tracing and verifying documented results between the research phases, and repeating the defined activities in our protocol by other researchers to address researcher bias threat to validity.

In this thesis, we expand our previous research [2] by extracting more patterns and rules from the API documents and investigating more text classification algorithms, therefore conducting more experiments and then comparing results accuracy and studying their efficiency and effectiveness as well in the usage of the automated COINs classification (which is tedious to do manually). We mainly rely on our manual classification for the COINs as a ground truth, which we created from API documentation to be fed up to our text classification model.

Our previous research [2] shows an acceptable accuracy level in the classification of the COINs automatically and this will benefit designers and architects in finding out COINs from any API documentation, where it is tedious and time consuming to be performed manually.

1.2 Research methodology and contributions

In this, we followed a methodology that included the following main research tasks:

1- Reviewing the State-of-the-Art (SoA): First, a literature review to identify the existing methods and technologies to extract conceptual interoperability constraints from NL documentation.
2- Exploratory multiple-case study: In order to find out the state of current API documentation with regards to the way the conceptual interoperability constraints are documented, we analyzed multiple API documentations (cases). Each case study goes through three main phases:

- Data preparation for collecting evidence where text will be pre-processed into single sentences.

- Data collection starts with labeling each sentence with one of the COINs classes [1]. Then, sentences that agree on the label are grouped together.
- Thematic analysis for the produced groups of sentences will be conducted to find out the frequent terms, patterns and sentence structures that will be encoded into initial themes.

3- Exploring the potentials of ML and NLP in extracting the COINs from API documentation.

Our research contributions are listed as follows:

1- Transforming raw unstructured data (i.e., text in API documents) into structured data with unified format to be used in next research steps.
2- Building the COINs corpus (i.e. ground truth): Manually classifying the conceptual interoperability constraints (COINs) of the collected data with the help of "Constraints of COIN Model". [1]
3- Defining representation patterns of COINs: Manually mining and analyzing of the textual content of the API documents in order to identify the frequently used terms and sentences structures from the collected API documents.
4- Building the text classification model (classifier): Utilizing the obtained corpus, we designed two different classifiers. These classifiers are used for automatically classifying the COINs.
5- Exploratory experiment: Evaluating the efficiency of the created classifiers in terms of accuracy by conducting experiments that utilize different text classification algorithms.
6- Developing a plugin prototype, which is available to be used as web service[1].
7- Parts of our presented work in this thesis has been accepted in The 38th International Conference on Software Engineering (ICSE 2016) Companion, and will be presented during the conference that will held on May 14-22, 2016, Austin, TX, USA [2]

[1] Web service: is a Program Integration across Application and Organization boundaries https://www.w3.org/DesignIssues/WebServices.html

1.3 Outline

The rest chapters of our thesis are organized as the following:

– Chapter 2 presents a background on Conceptual Interoperability Constraints and its different models. It also offers some definitions for the natural language processing, machine learning terms, and the texts classification methods and algorithms, which are used in our proposed solution

– Chapter 3 overviews the related works that deal with our stated problem of identifying software interoperability constraints and highlights their advantages. Afterward, this chapter explains briefly the differences between our approach and the presented related works.

– Chapter 4 presents the research methodology that we followed in solving the problem, and describes the goals of our research methods.

– Chapter 5 poses the first part of the research, which is a multiple-case study. We describe the study design, results, and discussion along with the threats to validity.

– Chapter 6 presents the second research part of our study, which answers the question about the efficiency of the natural language processing and machine learning in solving the problem of text classification. In this section, we offer two different approaches to solve the problem and answering the research questions, with presenting the results, evaluation and clarifying the efficiency of each of the two approaches.

– Chapter 7 introduces a technical solution by developing a tool prototype, in order to provide the software architects and analysts with means to facilitate the COINs classification from any API documentation. This section also explains in details the design, implementation, and performance evaluation of this tool prototype

– Chapter 8 presents the most important challenges and obstacles that we faced during this research work, and how we overcame them.

– Chapter 9 presents the results that have been accomplished through our thesis in meeting the research and answering its related questions.

– Chapter 10 introduces the future vision for extending this research from different aspects. It offers some ideas, which might improve the performance of the automated classification model, and some other suggestions on how to take advantage of this research in other practical area especially in industry.

2 BACKGROUND

In this chapter, we start with introducing the definition of conceptual interoperability and the COnceptual Interoperability coNstraints (COINs). Then, we present a basic introduction to natural language processing, machine learning, and text classification that we have utilized through our research.

2.1 Conceptual Interoperability

In computer science, interoperability is "the ability of two or more systems or components to exchange information and to use the information that has been exchanged." [3] [4].
Interoperability between software systems is one of the most important modern concepts that receive considerable attention recently, because of many considerations such as communication, compatibility and interaction between different systems, which become very important. Besides, the interoperability is facing many challenges and obstacles, such as technical heterogeneity (e.g., different communication protocols, data input and output type and parameters orders) [1].

Due to the importance of the interoperability, multiple classification-models have been proposed for determining, and organizing the interoperability levels in software systems. For example, (1) the Levels of ISs Interoperability (LISI) [5], (2) NC3TA Reference Model for Interoperability (NMI) [6] and (3) the Levels of Conceptual Interoperability Model (LCIM) [7]. The main importance of these models is their ability to identify both the levels of compatibility between systems as well as the effort needed for configuring these systems in order to work interchangeably and integrally [8].

In our thesis, we based our research on the Conceptual Interoperability Constraints (COIN) Model [1], because it focuses on the conceptual constraints that are of our interest and because it can be applied to different software systems (e.g., information systems, embedded systems, mobile systems, etc.).

According to Abukwaik [1] , The COINs are defined as the conceptual characteristics that govern the software system's interoperability with other systems. That is, wrong understanding, misassumption and misuse of these conceptual constraints might defect the desired interoperability causing systems' inconsistency in getting mutually meaningful results and leading to serious consequences accordingly (e.g., cost increase or project failure). Obviously, explicit and clear declaration about the system COINs helps analysts in detecting the conceptual mismatches and thus allows for a more effective and efficient resolving for these mismatches [1].

Table 1 represents the current set of COINs and their classes with examples. We introduce these classes briefly here, but for more details about it, you can see [1].

COIN Classes: Abukwaik et. al. [1], defined the six-classes of the COINs as follow:

1. **Syntax COINs** "*specify the concept-packaging methods (i.e., the conceptual modeling language) and the lexical references used in the system. Examining the syntactic match paves the way towards investigating the semantic one. ".*
2. **Semantic COINs** "*state semantic constraints (e.g., the measurement unit of a calculateDistance service is km not mile), and semantic references (e.g., reference ontologies) that encode the meaning of exchanged*

data and service goals. As no reference ontology has been widely adopted yet, we consider this a theoretical constraint which is left for future advances in the ontology research. ” .

3. **Structure COINs** “*depict system’s elements, their relations, and their arrangements that influence the interoperation results, e.g., interoperating with a software system without being aware of its data distribution may introduce a security threat if network links between remote sites are not encrypted. In this case, the distribution of the system is a structural COIN. “*
4. **Dynamic COINs** “*report information about the behavior of the interoperability elements during interaction. If such details are missed, they can introduce conceptual interaction flaws. For example, interoperating with a software system of regularly changing data may lead to synchronization issues if this property is not declared and addressed properly*. “.
5. **Context COINs** “*pertain to external aspects forming the interoperation settings, i.e., user and usage properties. For example, software systems that are designed to interoperate with software systems on desktop devices may cause display and memory issues on mobile devices.”*
6. **Quality COINs** “*capture required and provided quality characteristics related to exchanged data and services. For example, inaccurate results may occur when interoperating. “*

Table 1. Conceptual Interoperability Constraints [1]

Category	COIN name	Examples of value
Syntax	Lexical references	Dictionary, thesaurus, glossary, etc.
	Modeling lang.	XML, UML, ADL, WSDL, etc.
Semantic	Semantic references	Reference ontologies
	Semantic constraints	Data units and scale ratio
Structure	Data structural constraints	Invariants, inherited constraints, and multiplicity constraints
	Data storage	Relational database, flat files, etc.
	Distribution	Distributed, centralized
	Encapsulation	Encapsulated , not encapsulated
	Concurrency	Single-threaded, multi-threaded
	Layering	Layered, not layered
Dynamic	Data change	Periodic, irregular, continuous, etc.
	Service conditions	Pre, post, and time conditions
	Interaction property	State(ful/less), (a)synchronous, etc.
	Interaction time constraints	Session timeline, acknowledgment timeline, response timeline, etc.
	Communication style	Messaging, procedure call, blackboard, streaming
Context	Usage context	device type, wired/wireless, access rate, time, location, etc.
	Intended users	Human/machine, gender, age, etc.
Quality	Data quality	Security, trust, accuracy, etc.
	Service quality	Safety, availability, efficiency, etc.

2.2 Natural Language Processing (NLP)

Natural Language Processing (NLP) [9] is a field in computer science that combines the usage of both Artificial Intelligence (AI) [10] and Computational Linguistics (CL) [11]. There is a progress in researches that aims at improving the accuracy of finding the grammatical structures of the sentences [12] [13] [14]. Below we will introduce some of the main NLP technologies, which are used in the construction of any linguistic analysis system to identify the grammatical structures.

Parts Of Speech (POS) tagging is known also as a grammar classification of the words in the sentence, this technique is used to identify the part of speech in terms of (noun, verb, pronoun, adjective, etc.) [15] [16] Example: a sentence "*the child watches the match*", here "*the*" is a **determiner**, "*child*" is a **noun**, "*watches*" is a **verb**, "*match*" is a **noun**. For more information, see Fig. 1

Chunking is parsing a sentence into phrases and clauses, in which they are groups of interconnected set of words with logical relation, such as verb phrase and noun phrase [17]. See Fig. 1

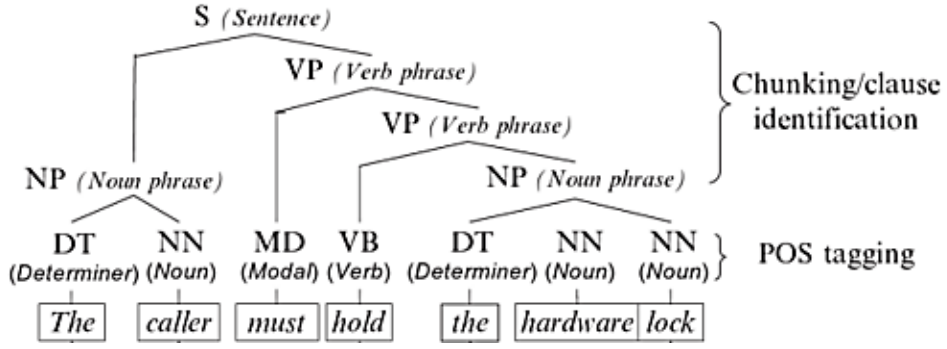


Fig. 1. An example of POS tagging, chunking and clause identification. [17].

Typed Dependencies [12] [14] is a technique to provide a simple description of the grammatical relations, which are oriented in particular toward non-linguistics experts in order to perform tasks related to NLP. It provides a hierarchical structure of the words in order to illustrate the words dependencies in a sentence with a simple description of each dependency. For **example** a sentence "*Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas" is* analyzed into a grammatical relations as shown in Fig. 2 [18].

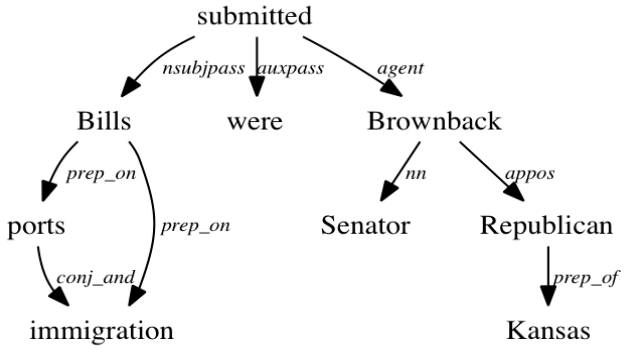


Fig. 2. Standard Stanford dependencies [18]

Named Entity Recognition [19] is also known as entity identification, which is a technique to extract information about words in a sentence by classifying the words based on predefined classes. These classes often have a higher level of abstraction, and depends fundamentally on the semantic meaning of the words. For example "USA, Germany, UK" are transformed into a word "Country". This technique is helpful in facilitating the semantic meaning by finding the main entity that these words belong to.

2.3 Machine Learning (ML)

It is a branch of computer science and a part of Artificial Intelligence (AI). In particular, it refers to training the computer on specific patterns that depend on the problem domain by utilizing some of the machine learning algorithms [20], in order to enable the automatic prediction and detection of these patterns by the machine [21]. In this section, we introduce some of the ML text classification techniques and statistical language modeling that we utilized in our research.

Text Classification (TC)

Text classification (TC) is the process of classifying sentences in documents of text into two or more predefined classes (classes) [22]. In principle, TC is a subjective task, for example, when two experts (human or artificial) decide whether to classify a sentence ***S*** in document ***D*** under class ***C***, they might also disagree, and in fact, this happens with relatively high frequency [23].

There are many traditional text classifier algorithms such as Naive Bayes [24] [25], Support, Vector Machine [26] [27], etc. The performance of any of these classifiers depends mainly on the quality and the quantity of the training dataset, which is manually labeled and carefully selected to be representative as much as possible. The more proper training of labeled data the better accuracy the classifier achieves [28].

Text Classification Workflow

In Text classification, there are two main processes [29]. The first one is the **training** process, in which the classifier is learned on some classified data sample. While, the second is the **prediction** process, in which the classifier assigns the suitable class of a given data. It is important to mention that, before performing the training and prediction processes, two interior procedures need to be performed on the input document, which contains data. These two procedures are the "features extraction" and the "feature selection" that we explain next in details. In Fig. 3, we summarize the text classification workflow.

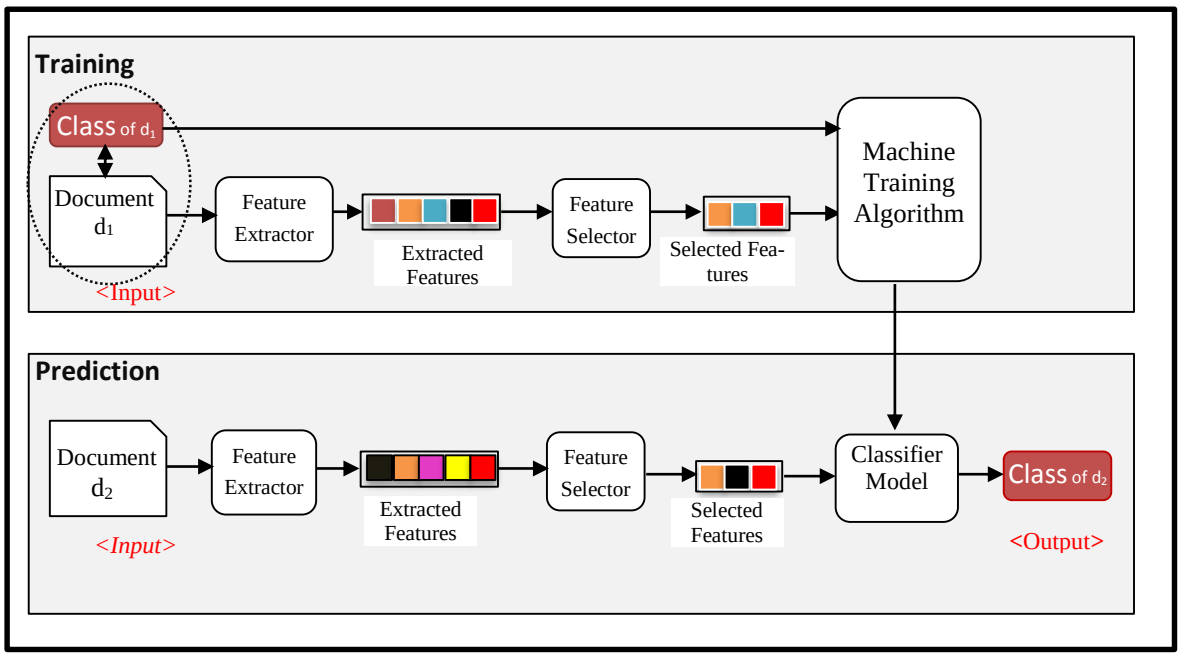


Fig. 3. Text Classification Workflow

Feature Extraction

It is the process of deriving features from the existing raw data [30] . The goal of the feature extraction is to find the most representative characteristics from the original data. These features should be carefully extracted, because they have to represent the important aspects of the sentence structure, semantic, context and all other significant information as well.

There are many different feature extraction techniques can be applied for unstructured text. Some of the common ones are called statistical language modeling techniques [31]. In these techniques, the word sequences are assigned a statistical probability [31]. Here are some of these techniques:

1. **Bag of Words** (BOWs) [32] is a simple technique for text classification, in this approach, each word in a sentence is considered as a feature and a document is represented as a matrix of weighted values using some kind of a weighting method such as TF-IDF (Term frequency –Inverse Document Frequency) [33]. However, BOWs model has some limitation like that it does not consider both the grammar and the meaning of the sentence. This is because it ignores the words ordering and losses the semantics of the words, but still it gives a score about the words importance in the whole document [34]. For example: consider a sentence like 'The software has configurations' In BOWs, each sentence is represented as a matrix of features of single separated words like "The",”software”,"has",”configurations”
2. **N-Grams** is a combination representation of all of adjacent words in a sentence [35]. **N** can be any number greater than zero (N > 0). Thus, 1-Gram refers to unigram that is the simplest form of N-Gram model, and in this case, a sentence is represented by a single word. Similarly, 2-Grams stands for bigrams, in which a sentence is represented by two sequence words together. In the same way, 3-Grams are trigrams that represents a sentence by three sequence words together. An example of

each of these aforementioned N-Grams is explained on a sentence “The screen is red” as shown in Table 2.

Table 2. N-Grams example

unigram	‘The’ , ’screen’ , ’is’ , ’red’
bigrams	’The screen’ , ‘ screen is‘ , ‘is red’
trigram	’The screen is’ , ’screen is red’

3. **Skip-gram** is a generalized form of N-Grams with a goal to discover word representations that help in predicting words in the same context in the sentence, which incorporates data sparsity problems [36]. The more data become available for the Skip-gram model, the more the information the model can extract.

   For example the sentences:

   - "I have to return"
   - "I have never had to return"
   - "I finally have to return"
   - "I do not have to return"

   All these sentences are grouped into the skip-gram "I have to return", which means they have similar shape.

Choosing the best techniques for feature extraction depends primarily on the problem domain, for example: sentiment classification[2] might give high performance if the features are extracted by using the Bag of Word technique [32], while news classification using N-Gram technique could achieve better results [35]. Shortly, the high-quality features means better results!

Feature Selection

Feature selection is the process of choosing only the most important features from the extracted features [37]. This is performed by eliminating the redundancy and neglecting the less useful features, while keeping the semantic unchanged [38]. In general, there could be millions of features, especially when working on a huge amount of textual data, for example working on topics modeling [39], in which some texts are given and then identifying what the topic of these texts is, or in another words: what the texts talking about. In Fig. 4, we picture the feature selection concept.

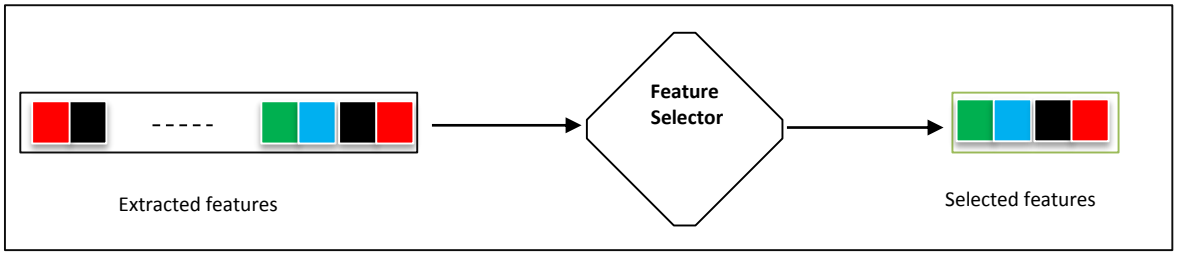


Fig. 4. Feature Selection workflow

[2] Sentiment Classification (SC) is about assigning a positive, negative or neutral label to a piece of text based on its overall opinion. [92]

Applying feature selection has advantages like [40]:

1. Increasing the model prediction accuracy (by avoiding overfitting). Overfitting is a problem of getting inaccurate prediction when testing the classifier. In this problem, the error rate of prediction increases in the testing data set but decreases in the training data set. It happens when the size of the training data set is too small compared with the complexity of classification model [41].
2. Reducing time cost to construct a model and speeding up the model prediction process.
3. Providing a deeper understanding of the process infrastructure that generated the data.

3 RELATED WORK

In this thesis, we present an approach to automatically extract conceptual interoperability constraints, the COINs, from NL text in API documents via NLP and ML technologies. Many recent researches proposed identifying specific types of constraints from API documentation in different ways. Therefore, in this chapter, we describe briefly the similar works to our research.

Wu et al. [42] identified parameters' dependency constraints from multiple library resources, mainly web services and SDK documentation that are expressed in natural language. They proposed an approach called INDICATOR (INference of Dependency ConstrAinTs On parameteRs) that collects information from API documentation about operations' definitions and parameters' descriptions. Their approach has two stages: the first stage is documentation analysis to extract only the constraints candidates, and the second stage is constraints validations, in which the final results are only the validated constraints.

Pandita et al. [43] proposed an approach to automatically infer the formal method specifications from natural text of API documents. They introduced a new technique that assists client-code developers to correctly use methods specifications in terms of method prerequisites, and what is expected after method is executed (i.e., so-called pre-conditions and post-conditions). This approach helps to ensure a legal usage of code contract to avoid inconsistency, misleading and prevent exceptions and bugs during code development cycle. The idea of the approach is based on reading the whole method descriptions from API documents including: summary, argument description, return description, exception description, and remark description. Then, they use a shallow parser to parse the specification in First-Order-Logic FOL [44] expressions, which are extracted using natural language processing NLP Parser. NLP is used as a core-intermediator to analyze and process the code-content and textual content to construct code-contract as a final result.

Zhong et al. [45] proposed an approach called Doc2Spec to recognize and infer resource specifications. In particular, they developed a tool for Doc2Spec that is mainly based on linguistic analysis of the API Documents using natural language processing NLP techniques. The significant importance of their tool is to discover and extraction resources' specifications as a first step and match them with the code-implementation as a second step. The basic functionality of Doc2Spec is to detect both known and unknown bugs in code automatically, which are the consequences of disregard API specifications or misused resources by developers. For example: developer might not close resources properly after the end of their usage. Such tools can play an essential role to avoid errors and refine code quality in implementation phase.

Dekel and Herbsleb [46] introduced an approach for improving API documentation usability by extracting and highlighting the important part of documentation, which includes the sensitive information, instructions, and guidelines to push them into a programming IDE editor. They developed eMoose tool [46], which searches and automatically tracks the content of several major APIs documentation to find the important hidden information to assist developers. These information are called directives that hold method requirements and optimal method invocations. eMoose offers to developers a list of method recommendations in terms of knowledge items, constraints, method invocation dependency, and side effects based on the code context. This allows

developers to work in a safe mode by protecting them against the risks of improper implementation. In addition, the tool increases developer's awareness about the future problems by preventing errors, runtime fails, or encountered code violations, which are potentially hard to predict during code implementation phase. Hence, the tool positively affects software performance and consistency.

Some of the aforementioned approaches like Wu et al. [42] and Pandita et al. [43] used NLP with rule-based identification, while Zhong et al [45] used ML to identify the name of the restricted entities, but not the restriction themselves. In our research, we elaborate on Abukwaik et al [1] idea of extracting different type of conceptual constraints utilizing both NLP and ML technologies together. In addition, we extracted different types of constraints as mentioned before (Not-COIN, Dynamic, Semantic, Syntax, Structure, Context and Quality) constraints. In our research, we followed the empirical methodology [47].

4 RESEARCH METHODOLOGY

In this chapter, we describe our research methodology, starting with the research methods in section "3.1 Research methods" Then; we define our research goals and questions in section "3.2 Goals and Research Questions".

4.1 Research methods

In this thesis, we followed an empirical-based methodology in exploring the potential of automating the extraction of COINs from API documents to support architects and analysts in performing their conceptual interoperability task with the lowest cost possible.
The empirical research provides us with many advantages like allowing us to trace and verify the obtained results between the research tasks and their results. Moreover, it enables other researchers to repeat measure and extrapolate the results independently. Therefore, we performed our research in two parts as follows:

Research Part One (multiple-case study). In the first part of our empirical research, we systematically explored the nature of COINs in many API documentations to explore their current state in terms of their frequently used terms and patterns. Accordingly, we manually built our COINs corpus that holds each investigated sentence in the API documents along with its COIN class.

Research Part Two (ML for automatic COINs Extraction). In the next part of our research, we used the results of the previous research task in directing our investigation about the capabilities of NLP (in representing the observed patterns and rules obtained from analyzing the COINs) and the power of ML (in learning these modeled patterns and rules towards full automation of identifying the COINs in text). Finally, with exploratory experiments we evaluated the accuracy of our produced ML model. This helped us in deciding how useful our automatic extraction idea for software architects and analysts in performing effective and efficient conceptual interoperability analysis.

4.2 Goals and Research Questions

In fact, this work is extending the proposed idea of Abukwaik et al. [5] of automating the extraction of COINs from their API documentation. Hence, we formulated our main goal in terms of GQM-goal template [48], which in turn supports the more comprehensive purposeful goals as the following:

- To: support the conceptual interoperability analysis task
- For the purpose of: improvement
- With respect to: effectiveness and efficiency
- From the viewpoint of: software architects and analysts
- In the context of: analyzing text in API documentation within integration projects

We translate this goal into the following research questions that we try to answer within our research:

- **RQ1:** What are the observed patterns in specifying the conceptual interoperability constraints COINs in the NL text of API documentation?

Rational: This question aims at building an accurate Ground Truth (i.e. COINs corpus) that represents the first building block of our automatic extraction idea. To answer the research question we need to collect adequate data manually (i.e., textual sentences from API documents), then we analyzed it, and identified a set of patterns and extraction rules for the found COINs. The metric we used for this research question are frequent terms and sentence structures.

- **RQ2:** How effective and efficient would it be to use Natural Language Processing (NLP) along with Machine Learning (ML) technologies to automate the extraction of COINs from the text in API documentations?
  Rational: This question aims at tackling the challenges of extracting the COINs from NL text by building a machine model for the COINs to utilize it within already existing ML classification algorithms. The metric we used for this research question are Accuracy, Recall, Precision, and F-Measure.

5 RESEARCH PART ONE: MULTIPLE-CASE STUDY

In this chapter, we present the first part of our research starting with its design in section 5.1, in which we defined a research method. Then we present the execution of our designed multiple-case study in section 5.2. Then we discuss the results.

5.1 Study design (Holistic multiple-case study)

Study goal. The first part of our research has a goal of answering the first research question that we mentioned in chapter 3, which is "**RQ1**: What are the frequently used patterns in specifying the conceptual interoperability constraints COINs in API documentation?"

In order to do so, we need to investigate the current state of COINs (in terms of representation, context and recurring patterns) by exploring real-world API documentation. This investigation facilitated discovering the infrastructure of the building units, which help in finding out the representative terms, structure and patterns, which is very important to be used in ML later on in the next chapter.

Research method. Accordingly, we decided to perform a multi-case study with *literal replication* of cases from different domains. Such study allows us to recognize and perceive variety cases, with important evidence to get with generalizable and more powerful results as drawn independently across replicated cases. Fig. 5 illustrates the Holistic multiple-case study and other cases types [49].

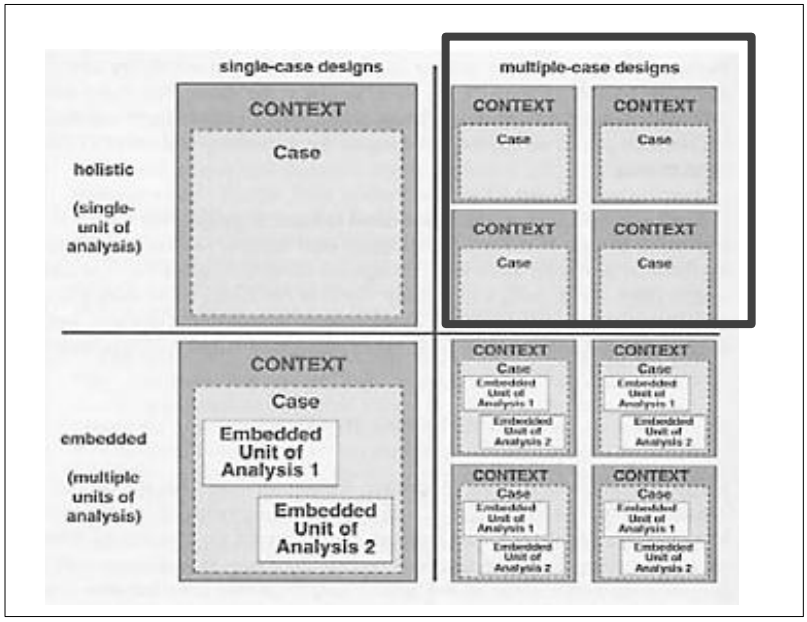


Fig. 5. Holistic multiple-case study [49]

Analysis unit. Our case study has a holistic design, which means that we have a single unit of analysis, which is "the sentences that include COIN instances".

Study protocol. Our multiple-case study protocol includes three main activities, which are case selection, case execution, and cross-case analysis that we detail in the execution section 4.2 "Study Execution".

Design of the data extraction sheet. We designed a "data extraction sheet" that we implemented as an MS Excel sheet (see appendix 7.3). The extraction sheet consists of the following fields:

Sentence ID: it is an auto number; and each sentence has a unique number.

Sentence: the textual value of the sentence that we call the "unit of analysis", which may include a COIN instance.

COIN type: one of the classes of the conceptual interoperability constraints COIN: {Semantic, Dynamic, Syntax, Context, Quality} and {Not-COIN}.

Source API document: to record the original API document name.

5.2 Study Execution

In this section, we present the execution of our designed multiple-case study.

Case selection

We have chosen six API documentations, which are: SoundCloud, GoogleMaps, Skype Instagram, AppleWatch and Eclipse-Plugin Developer Guide. We considered different characteristics and criteria for the choosing like:

- Published statistics[3] on API mashups score, which represents the API popularity in terms of API usage by developers to build web services or even applications. As shown in Table 3.

Table 3. Mashups score of API documentation

API Documentation	Mashups
SoundCloud	34
Google Maps	2580
Skype	30
Instagram	64

- API type: we select different API types in terms of web service and platform. Our selection was as the following (Four API documents from web services and two API documents from platform. Since, web-services covers different services than the platform does.
- API domain: we also consider the diversity of the selected APIs document to cover different domains such as social activates by selecting Instagram API documentation, and from communication services, we selected Skype, and from developing environment; we selected two different APIs, which are AppleWatch and Eclipse-Plugin Developer Guide).

Finally, we summarized our selection cases in Table 4. Which represent all API domains and its API documentations, and for each API documentation we added it API link in order to extract these document from its source, as we will explain in next section Case Execution.

[3] Programmable web: *http://www.programmableweb.com/apis/directory*

Table 4. API documentation's URL

API Domain	Documentation	Links to process
Web-Service	SoundCloud	https://developers.soundcloud.com/docs/api/guide
		https://developers.soundcloud.com/docs/api/reference
	GoogleMaps	https://developers.google.com/maps/web-services/
	Skype	https://msdn.microsoft.com/en-us/library/office/mt124991.aspx
	Instagram	https://instagram.com/developer
Platform API	AppleWatch	https://developer.apple.com/library/prerelease/ios/documentation/General/Conceptual/WatchKitProgrammingGuide/#//apple_ref/doc/uid/TP40014969-CH8-SW1
	Eclipse -Plug-in Developer Guide	http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcore%2Fruntime%2FPlugin.html

Case Execution.

In this stage, we performed for each case the following three steps: data preparation, data collection, and data analysis as shown in Fig. 6

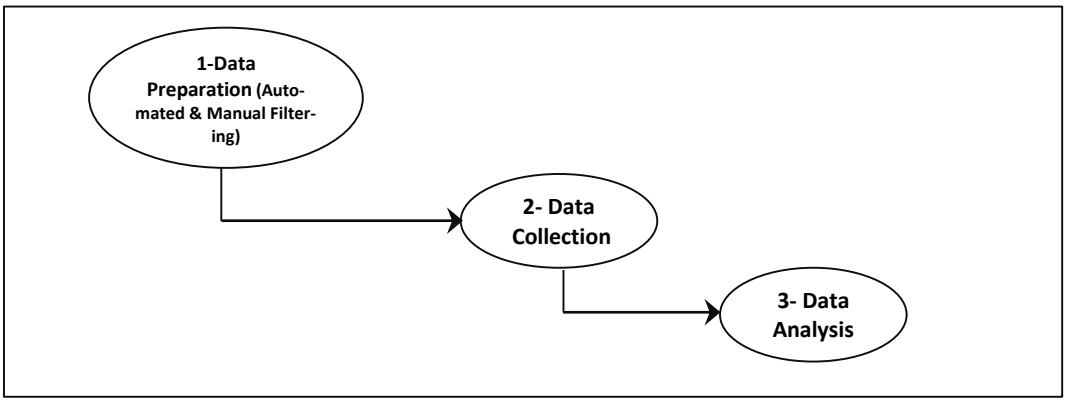


Fig. 6. Case execution process

1. Data Preparation

In this step, we started by fetching the API documents from their online resources in order to process their content, which means that we must a content with pure text data only. In our selected documents, we focused on retrieving the parts or sections that were rich in the textual content about the conceptual interoperability constraints. For example, the Overview, Introduction, Guide, API reference, and Summary webpages of the documentation website. The final output of this preparation is a filtered text. Thus, we performed this preparation as the following into two procedures:

– Automated Filtering

We implemented a simple PHP code using Simple HTML DOM Parser 4 library to filter out the API documentation from noise. (i.e., headers, images, tags, symbols, html and JavaScript code) and to keep only the textual content. In our implementation, we pass an API document link as input for the PHP method in the abovementioned library as the following: file_get_html (link) and we get back the output as a text content. Then, we keep the output as a text file to be manual reprocessed, as we explain in next step (manual Filtering).

- Manual Filtering

The automated filtering described in the first process is limited to detect some usual and known patterns of noise, which do not satisfy other noise cases like text mixed with pure code that occurs frequently in many API documentations. There are also some irrelevant textual sentences that do not match our interest in conceptual constraints and are hard to be filtered automatically (i.e., references like "see also" and "for more information", "copyrights", "related topics", "titles", etc.). Additionally, such sentences could mislead the machine learning in our later research steps. Therefore, we handled these sentences by filtering them out manually for more relevant and accurate data.

2. Data Collection

In this step, we cut the textual input resulted from previous step into single sentences within the data extraction sheet that we created according to the design we mentioned in section 4.1 Study Design. This resulted in a structured and organized sheet (see the excerpt example of the sentence retrieval output in Table 5).

Table 5. Data extraction sheet with example of collected data from 3 cases

Sentence id	Sentence	COINS Type	API Document (case)
1	All images must reside in the Watch app bundle.		AppleWatch
2	A user is encapsulated by a read-only Person object.		Skype
3	All rate limits on the Instagram Platform are applied on a sliding 1-hour window.		Instagram

We filled the data extraction sheet gradually as we execute each selected case. That is, at the very beginning we had only the sentences retrieved from the API documentation of SoundClound case. After analyzing the collected data [see section 2.3], we started the execution of the next case, prepared its data and retrieved its sentences into the data extraction sheet and so on so forth.

Note: We developed a data storage, which is a local repository to store and organize all of the following data

- The original HTML pages of the pre-processed documentation
- The excel sheet for each case (each API documentation)
- Other used artifacts (the links of the API documentation)

[4] Simple HTML DOM: *http://simplehtmldom.sourceforge.net/*

This storage has some advantages like enabling us to access the data sources without reconnecting to their online sources and re-retrieving it from there. This guaranteed data consistency and independency. One remarkable feature for this scenario is that, researchers can easily replicate and re-perform our study on the same data. It is worth mentioning that, when we revisited the API documentation on their online sources, we noticed that, there were some updates and removals in the contents, which means that, in the long-term a lot information may be changed. Therefore, it would be an impossible task for future researchers to perform any kind of replication for our research. Fig. 7 shows our data repository content within the preparation and the data collection processes.

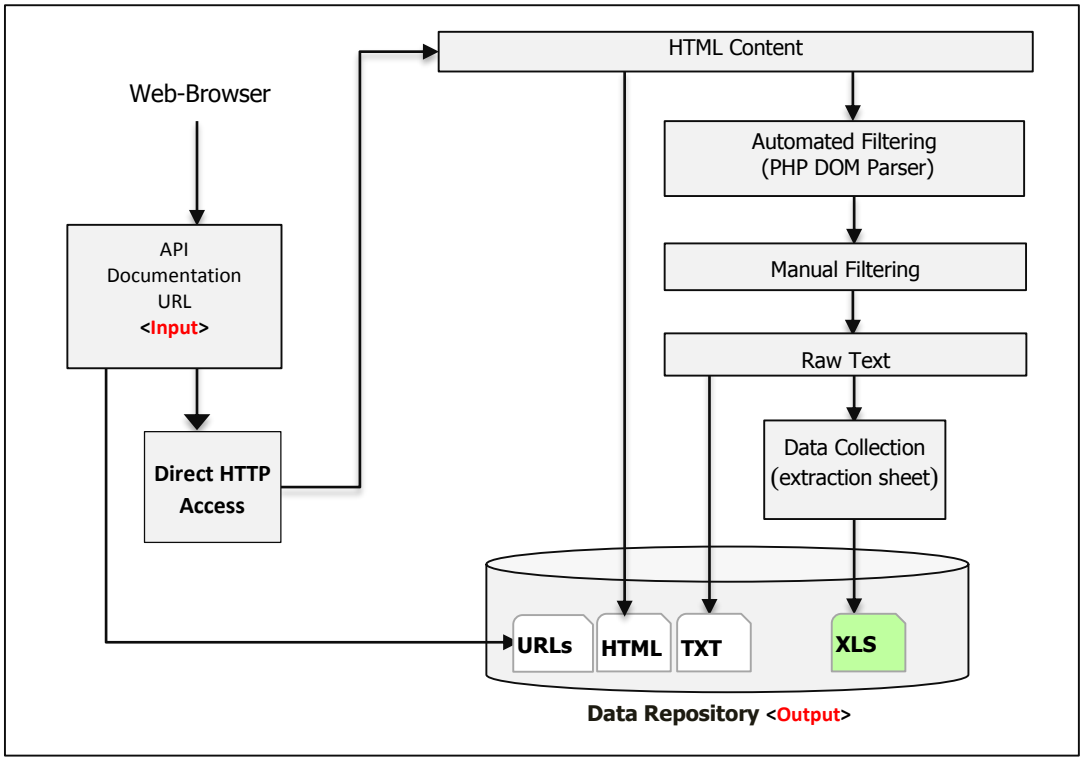


Fig. 7. Data Storage along data preparation and collection

3. Data analysis

In this stage, we performed our content analysis method on the collected data from a case under execution as we depict in Fig. 7 and explain in detail below.

Incremental building of the ground truth.

We manually investigated the meaning of each sentence collected in the data extraction and we checked if it could be mapped to any of the classes covered by our interpretation criteria. This interpretation criteria is the "Constraints of the COIN Model [1], which directs our decision about classifying each sentence as having a COIN instance of a specific class (i.e., syntax, structure, dynamic, context, semantic) or as not having a COIN instance at all (i.e., Not-COIN).

Obviously, this manual analysis took too much mental effort and time to analyze each of the 2283 sentences that we have in our data storage. In fact, this is one of the most challenging phases of our research and it

represents a corner stone in our research. The result of this step in each case was an increment in our ground truth (i.e., COINs corpus), which we will adopt later in the second part of our research. Hence, this process was performed by two researchers, each classified all sentences for each case separately (i.e., each sentence was classified twice in a separated way). In multiple discussion sessions, the two researchers compared their decisions; resolved conflicts based on consensus, and stored the classification decision in the extracted sheet. We summarize our spent effort in manual filtering and classification tasks in terms of time per document as shown in Table 6 and Fig. 8. Total effort in time with respect to the document size.

According to the information, we can easily conclude that there is a relation between the efforts in term of time and the size of the documents being analysed. Hence, we can observe that Eclipse Plugin Dev documentation has the largest size compared with the others. SoundCloud on the other side took much time compared with its document size, because it is the first case study that we analysed, and we spend much time to record the correct COIN type for each sentence.

Table 6. Total effort in time with respect to the document size

API Document	Total number of sentences	Document manual filtering (Minutes)	Sentence Classification (Hours)	Total efforts (Hours)	Total efforts (Minutes)
Sound Cloud	**219**	40	7	7.7	**460**
GoogleMaps	**473**	60	5.5	6.5	**390**
AppleWatch	**360**	60	7	8.0	**480**
Eclipse Plugin Dev	**651**	60	11	12.0	**720**
Skype	**325**	30	4	4.5	**270**
Instagram	**253**	20	4.5	4.8	**290**
Total	**2281**	**270**	**39**	**43.5**	**2610**

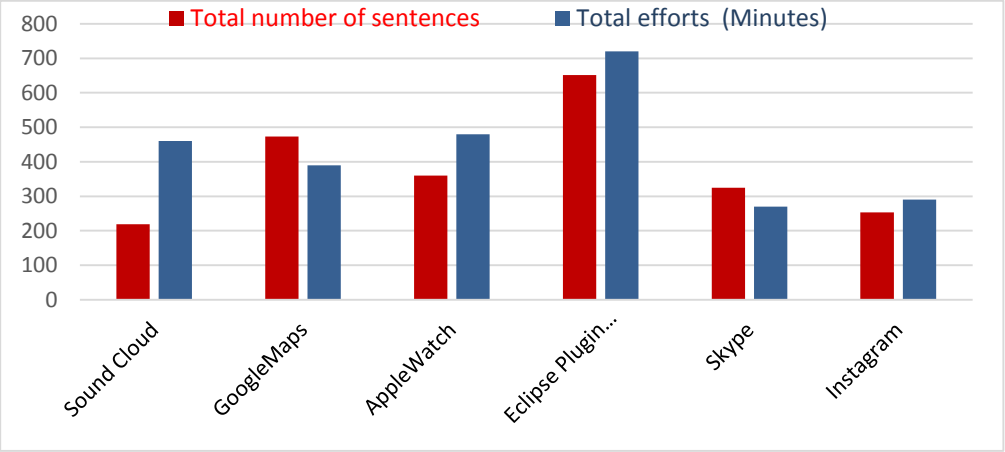


Fig. 8. Total effort in time with respect to the document size

Incremental identification of the COINs' patterns.

The final procedure in the case execution was manually analyzing its sentences that were classified as having COIN instances, in order to identify the patterns and detection rules of each COIN class. In specific, we investigated each sentence of the case and started taking our notes on any observed frequent occurrence of words, sentence structures, or any other noticeable format. We were also looking for any correlation between the sentences' phrases for each COIN class. In addition, we stored these identified patterns into a different data sheets as we will explain in more details in next section. Actually, we incrementally refined these patterns and discovered more patterns as we execute each case. Fig. 9 shows the content analysis method 'process flow' that we followed in our research to identify the patterns and also to create the ground truth (i.e. corpus). While Fig. 10 shows a snapshot of examples of the manual identification of the patterns in GoogleMaps.

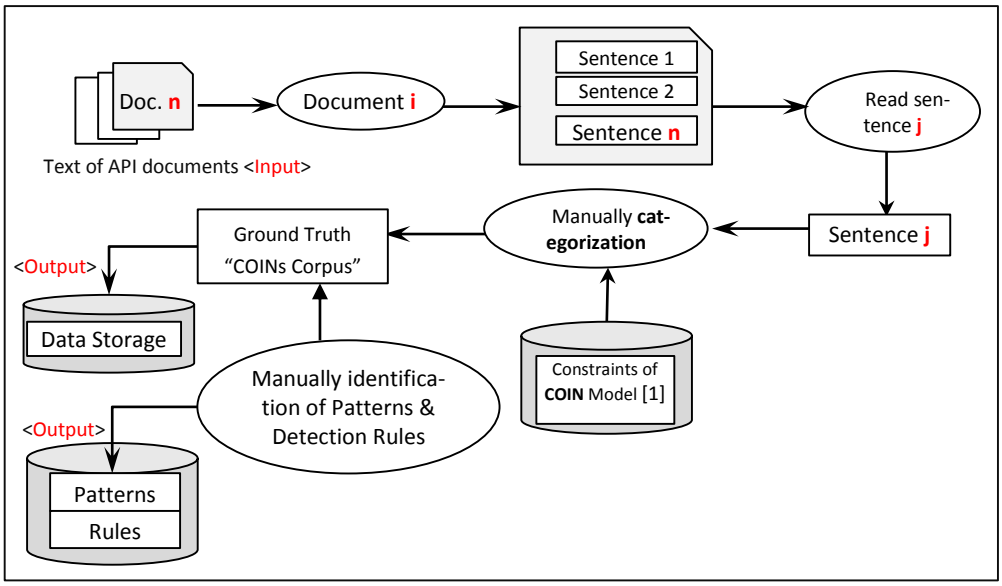


Fig. 9. Content analysis method 'process flow'

sentence	detected Patterns					
	1	2	3	4	5	6
	Input/Output	explantion	Conditional statement	Technical Terms	Structure Terms	method call
These web services use **HTTP requests** to specific URLs, passing URL parameters as arguments to the services.	request			HTTP		
For example,? is used within URLs to indicate the beginning of the query string;		For example			query	
When processing XML responses, you should use an appropriate query language for selecting **nodes** within the **XML document**, rather than assume the **elements** reside at absolute positions within the **XML** markup.			When	XML	nodes, element, document	
By default, **XPath** expressions match all **elements.**				Xpath	elements	
This **object** can then process passed **XML** and **XPath** expressions using the `evaluate()` method.				XML, Xpath	object,	evaluate()

Fig. 10. Snapshot of the manual identification of the patterns

Cross-Case Analysis.

Ground Truth (COINs Corpus)

Two COIN Corpora. After executing all cases, we arranged the incrementally contributed ground truth (i.e., COINs Corpus) into two different versions as the following:

- *Seven-COIN corpus*: in which, each sentence belongs to one of Seven-COIN classes (i.e., not-COIN, dynamic, semantic, syntax, structure, context, or quality).
- *Two-COIN corpus*: in which, each sentence belongs to one of two COINs classes so called: 'Two-COIN' instead of seven classes. That is, each sentence can be either a class of COIN (i.e.; dynamic, semantic, syntax, structure, context, quality) or a class of not-COIN. In fact, the Two-COIN corpus is derived from the Seven-COIN corpus by abstracting six of its classes into one class called 'COIN'. The aim behind deriving this new abstracted corpus is for later training of our ML model on two classes instead of seven, as this would achieve better accuracy results (we explain this issue in details in chapter 5). Fig. 11 shows the algorithm of creating the Two-COIN corpus from Seven-COIN corpus, while Fig. 12 shows the content of each two corpora.

Algorithm 1: Create TwoCOINs Corpus

Data: Seven-COINs corpus
Result: Two-COINs corpus

```
SixClasses [ ] ={'semantic','dynamic','structure','syntax','context','quality'}
Corpus ← ReadCorpus()
TwoCOINsCorpus ← empty
foreach sentence ∈ Corpus do
    temp=sentence.class
    if ( temp ∈ SixClasses ) then
        sentence.class='COIN'
    //update the class of the current COIN
    TwoCOINsCorpus.add(sentence)
return TwoCOINsCorpus
```

Fig. 11. Pseudo code of deriving Two-COIN corpus from Seven-COIN corpus

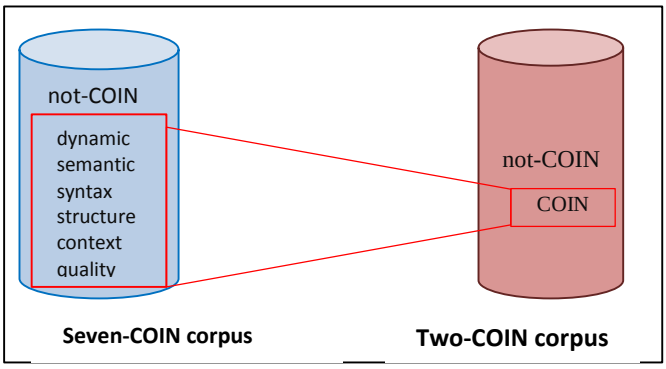


Fig. 12. Seven-COIN corpus and Two-COIN corpus structure

Here, we explain an example of the results of the created Two-COIN corpus from Seven-COIN corpus using the above mentioned algorithm as the following: Table 7 shows an example of the data extraction sheet of the

Seven-COIN as input Corpus, while Table 8 shows an example of the resulting data extraction sheet, which is *the* Two-COIN *Corpus* as output.

Table 7. Example of the data extraction sheet of the Seven-COIN Corpus

Sentence id	Sentence	COIN Type	API Document
1	All rate limits on the Instagram Platform are applied on a sliding 1-hour window.	**not-COIN**	Instagram
2	When it is finished manipulating the object, it releases the lock.	**dynamic**	Eclipse
3	A user is encapsulated by a read-only Person object.	**structure**	Skype
4	indoor indicates that the calculated route should avoid indoor steps for walking and transit directions.	**syntax**	Google-MAP
5	the connection ids can be used to share tracks and playlists to social network.	**semantic**	SoundCloud
6	Directions may be calculated that adhere to certain restrictions.	**context**	Google-MAP
7	your interfaces need to display information quickly and facilitate fast navigation and interactions.	**quality**	AppleWatch

Table 8. The data extraction sheet of Two-COIN

Sentence id	Sentence	COIN Type	API Document
1	All rate limits on the Instagram Platform are applied on a sliding 1-hour window.	**not-COIN**	Instagram
2	When it is finished manipulating the object, it releases the lock.	**COIN**	Eclipse
3	A user is encapsulated by a read-only Person object.	**COIN**	Skype
4	indoor indicates that the calculated route should avoid indoor steps for walking and transit directions.	**COIN**	Google-MAP
5	the connection ids can be used to share tracks and playlists to social network.	**COIN**	SoundCloud
6	Directions may be calculated that adhere to certain restrictions.	**COIN**	Google-MAP
7	your interfaces need to display information quickly and facilitate fast navigation and interactions.	**COIN**	AppleWatch

Case-share of sentences in the Ground Truth (COINs Corpora).

We have summarized this information in **Table 9**, which shows the number of sentences collected from each case.

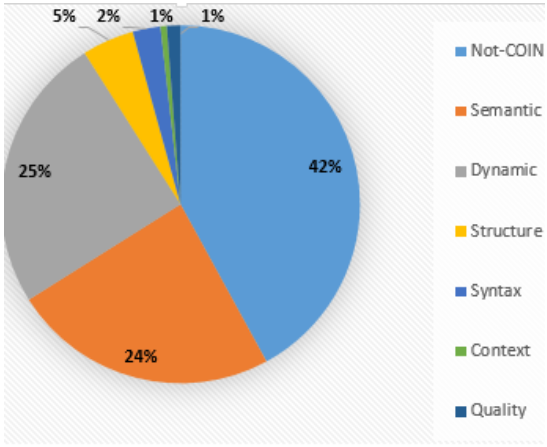
Table 9. The distribution of API Documentation

API Domain	Documentation	# Sentences
Web-Service	SoundCloud	219
	GoogleMaps	473
	Skype	325
	Instagram	255
Platform API	AppleWatch	360
	Eclipse-plugin	651
Total		**2283**

In Table 9, we can see that, the less number of sentences belongs the SoundCloud API documentation, because this documentation has some limitation of the offered services compared with other services like

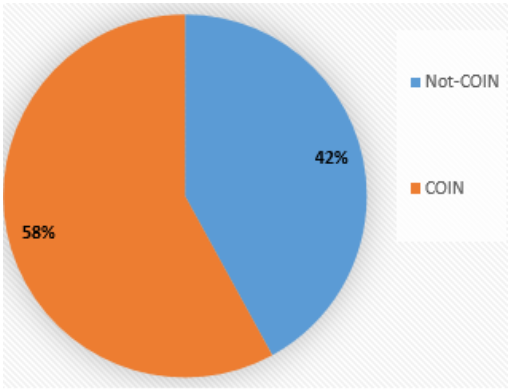
GoogleMaps or even AppleWatch. We also observed that the maximum number of the sentences obtained from Eclipse-plugin, obviously its API documentation is very huge, because there are many methods used for developing plugins inside eclipse platform.

COIN-Class share in the Ground Truth (COIN Corpora). The COIN-class (e.g., Not-COIN, dynamic, semantic, syntax, structure, context, and quality) is distributed over the Ground Truth in the COIN corpora non-equally. There are some classes like not-COIN, dynamic, semantic contribute of the majority of the COINs which is 91% of the total classes. For example, we observed that the Not-COIN class constitutes about 42% of the total classes, while the dynamic class constitutes about 25% and the semantic class constitutes about 24% of the total classes in the Ground Truth (COIN Corpora). On the other hand, we observed there are a few contribution of the other classes like (structure, syntax, quality and context). They comprise together about 9% of the total classes. These statics are illustrated in Fig. 13. While, Fig. 14 demonstrates the distribution of COIN-Classes in the second corpus (Two-COIN corpus).



COIN category	# No. of instances
Not-COIN	960
Semantic	548
Dynamic	570
Structure	107
Syntax	57
Context	13
Quality	28

Fig. 13. Seven-COIN instances distribution



COIN category	# No. of instances
Not-COIN	960
COIN	1323

Fig. 14. Two-COIN instances distribution

COIN-Class share in each case. We have deeply investigated this information and documented the results as shown in Table 10 and Fig. 15. These statistics reveal much information about the structure of each API document. We mean by structure is the contents of each API documents in terms of COINs types (e.g. Not-COIN, dynamic, semantic, structure, syntax, context, quality).

Table 10. COINs classes distribution per each case

COIN Type	Not-COIN	dynamic	semantic	structure	syntax	context	quality
SoundCloud	46.1%	26.9%	18.3%	4.6%	3.2%	0.9%	0.0%
Google Maps	63.0%	11.2%	13.1%	1.7%	**6.6%**	**2.1%**	2.3%
AppleWatch	**40.8%**	26.1%	25.0%	6.1%	1.1%	0.3%	0.6%
Eclipsse-plugin	29.0%	**32.4%**	**30.1%**	**6.5%**	0.9%	0.0%	1.1%
Instagram	**41.6%**	29.8%	25.1%	2.0%	0.0%	0.0%	1.6%
Skype	36.6%	23.7%	29.5%	6.2%	2.8%	0.0%	1.2%
Grand Total	**42.0%**	**25.0%**	**24.0%**	**4.7%**	**2.5%**	**0.6%**	**1.2%**

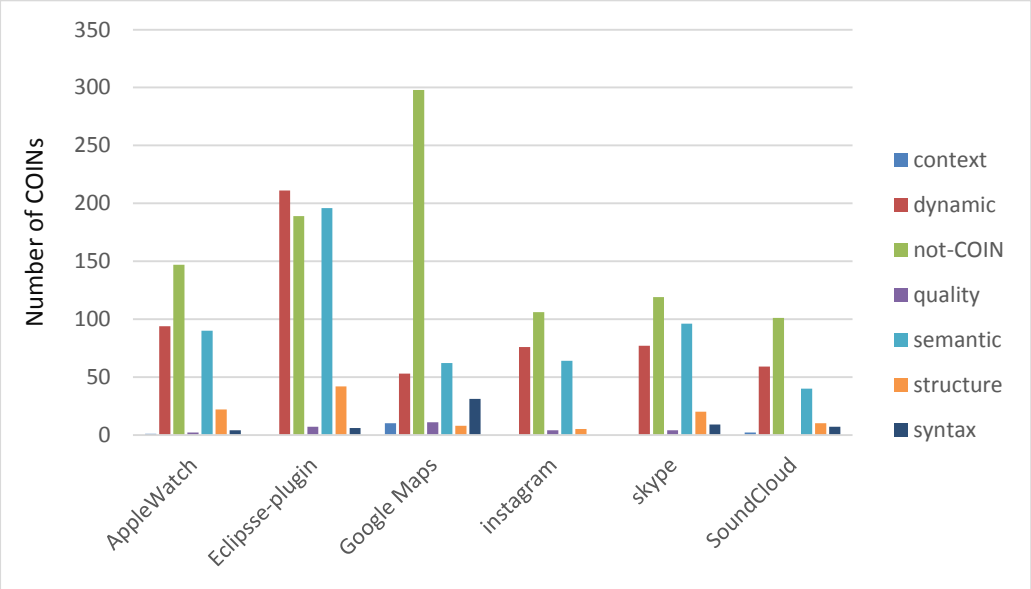


Fig. 15. Cases distribution over COINs

COIN patterns

"RQ1: What are the observed patterns in specifying the conceptual interoperability constraints COINs in the NL text of API documentation?"

What is patterns?

First, we define patterns as any frequent used of both terms and sentence structure. More specifically, a frequent term is any word is a used repeatedly in some sentences and occurs normally individually (single word per single sentence), for example the terms XML, iOs, XPath, HTTP, etc. are words used many times in different sentences in some COINs classes like Not-COIN class. While, sentence structure is more specific terms than just single term in a sentence. They constitute the formation/construction of a sentence, for example there are sentences begin with the phrase "if" and the clause ", then". Another example some sentences begins with pronouns like 'You' and followed by 'Modal Verb' like you must or it must or we must, etc. these kinds of patterns we define them as sentence structure, because they are not a single term.

After collecting the data and classifying it for each case, we focused on identifying the significant patterns of the text that would put us on the right road towards machine automated identification of COINs in text on behalf of human architects and analysts. As we mentioned earlier in section 4.2 that the analysis process was performed in a gradual manner (i.e., case after case). In each case, we extracted the noticed patterns for each COIN class.

Having this being said, in this cross-case analysis, we studied the textual content of all cases that is gathered within the Seven-COIN Corpus more deeply and carefully. That is, we mined the content of the corpus sentence by sentence and word by word. The more data we studied, the more patterns (terms and sentence structures) we discover, since every case study has different aspects and conceptual constraints.

Such a work is a tedious manual task that took us about 20 days and 8 hours per a day to accomplish. Some sentences required us to read them more than once to comprehend the accurate meaning first, and then extracting the cross-case patterns, proportional relationships, and similarities. This cross-case analysis helped us to refine our identified patterns. In the other words, the more COINs we include in the cross-case analysis, the more accurate and significant patterns we discover.

- Finally, we created a list of the top used terms in each class as shown in Table 11. For a complete list of the most frequently used terms per each class, please see appendix (B. Top frequently terms), which we created during our analysis process.

Table 11. Top 5 terms are frequently used per each class

Dynamic	Semantic	Structure	Context	Syntax	Quality
job	user	interface	direction	calculate	user
user	plug	content	time-share	route	direction
app	app	app	available	indicate	provide
client	provide	contain	bicycle	specify	access
interface	platform	collection	drive	user	api

It is important to mention that, we focused on identifying patterns for three COIN classes (i.e., Not-COIN, dynamic, semantic). This is simply because they have the three biggest shares of sentences in the Seven-COIN corpus. In total, these three classes constitute about 91% of sentences in the corpus as shown in Table 12.

Table 12. Total Ratio of the majority COINs

COIN Type	Ratio
Not-COIN	42.0%
Dynamic	25.0%
Semantic	24.0%
Total	**91.0%**

Next, we show our identified patterns for these three classes with detailed examples. ***Note***: For all three classes we created a table, in which we highlighted the detected pattern in red color within the example.

Patterns of the Not-COIN class. In this class, we observed the following patterns:

- Frequent terms: The predominant part of terms in this class are the Technical Keywords, which are mainly abbreviation of technical terminology and programming keywords. For example (XML, iOS, XPath, JavaScript, ASCII, KB, MB, etc.). In our corpus there are about 30.7% (of the Not-COIN class) has technical keywords. This means that, there are about 295 COINs of 960 COINs have one or more technical term (see Table 13 the first row).

  For further clarification, Table 13 is composed from five columns. First columns is a pattern type (e.g. frequent terms or sentence structure). Second column is a pattern name (e.g. technical keywords, sentence begin with some term, sentence contains some terms, etc.). Third column is example of the pattern term. Forth column is a real example from the corpus, Fifth column is the total number of the occurrence of the pattern in the corpus with respect to the COIN class and last column is the percentage of the occurrence of the pattern in the corpus with respect to the COIN class. Note that in the fifth column the cell values do not add up to 100% as there are minor patterns that take a share of it but we do not cover them in the table.
- Sentence structures: the second part of the patterns are sentence structure, as aforementioned these structures are illustrated as shown in all rows of Table 13 except the first row.

  In this regard, we observed that, there are relatively two significant patterns in this class, which are:
  - Sentences contain variables, symbols and tags. For example a sentence “XML responses consist of zero or more <route> elements.” It is classified as Not-COIN, as you can see contains some tags and technical terms. Such tags and the special characters like ‘/’,’\’,’<’,’>’ constitute 13.9% from the whole not-COIN class.
  - Sentences begin with the terms, like “for example”, ” for information” and sentences contains terms like “see”, “learn”, “let’s” all together constitute 12.8%.

Table 13. Identified patterns of Not-COIN class

Pattern Type	Pattern Name	Example	COIN with pattern example	COINs count	COINs %
Frequent Terms	**Technical keywords**	XML, iOS, XPath, JSON, OSGi, SDK, HTTP, GET, POST, etc.	these resources can be accessed and manipulated using the **HTTP** methods **GET**, **POST**, **PUT** and **DELETE**.	295	**30.7%**
	Begins with	for example	**for example**, a user may enter an address as '5th&Main St.'	123	**12.8%**
		for information	**for information** on how to present new interface controllers, **see** interface Navigation.		
			For more information about notification payloads, see Specifying a Notification Payload for Testing.		
	contains	see	for a full list of properties that can be set on a sound resource, **see** the /tracks endpoint reference.		
			See Transit Details below		
			See also: /me endpoint reference documentation.		
		Learn	**Learn** how to get a key.		
		let's	**Let's** look first at the scopes defined by the platform runtime:		
		the following	Currently the following activity types are supported:	62	**6.5%**
	Begins with	figure shows	**Figure shows** the default JSON file that comes with your project.	10	**1.0%**
	Ends with	: as follows	the help command supports -scope argument, which should be used **as follows:**	85	**8.9%**
	Contains	variables names , path tags: '/' , '<' ,'>'	Neither requires an **access_token** or **client_id**.	133	**13.9%**

Patterns of the dynamic class. In this part, we extracted the most used patterns in terms of terms and sentence structures.

- Frequent terms: during our studying and analysis of the dynamic COINs, we observed that, this class contains many activities, events that depict the data and process flow, and commands to perform direct tasks or activities. Therefore, we came up with a special list of terms and we called it "Action Verbs". Example of these verb terms includes but not limited to (create, use, access, request, etc.). Actually, they represent about 35.8% of the total number of sentences with dynamic COINs. This means that, there are 204 COINs from 570 COINs have at least one or more Action Verb. See Table 14 (first row). We also provide a complete list of the Action Verbs in the please see appendix (B. Top frequently terms).

– Sentence structures: as we explained in the beginning of this section, the sentence structures belong to different terms. This class has specific patterns in terms of conditional statements, method call, and variables. See Table 14.

Table 14. Identified patterns of Dynamic class

Pattern Types	Pattern Name	Example	COIN with pattern example	COINs count	COINs %
Frequent terms	**Action Verbs**	**create**, use, request, access, plug, lock, include, set-up, run, start ,call-up ,redirect. *Please see the Action Verbs list in the appendix for more details.*	instead, **create** a complementary experience to your iOS app.	204	**35.8%**
Sentence structures	**Begin with Conditional statement**	**if** , **when**, once, **while**, as long as ,unless	**if** a command name is specified, the help message for this command is displayed	137	**24.0%**
			note that **as long as** the sound is public, you will only need to provide a client_id when creating a client.		
			once the state is finished it is ready to be embedded or streamed.		
			when building a valid URL, you must ensure that it contains only those characters shown above.		
			while it owns this rule, it is only allowed to modify files within that directory subtree.		
			Unless otherwise stated, the values null and the empty string are equivalent to omitting the property.		
	Contains Conditional statement	if , **when**, once, while, as long as ,unless	the request may succeed **if** you try again.	100	**17.5%**
			this feature is available only **when** connecting through telnet or ssh.		
			you should exercise extreme caution **when** acquiring and releasing scheduling rules using such a coding pattern.		
	Begin with	**You** + **Modal Verb** *(e.g must, have to, should , will, may, can)*	**you can** follow a user using the /me/followings endpoint.	34	**6.0%**

			you must have an existing iOS app to create a Watch app.		
			you should now store the access token in a database		
		imperative(commands verbs)	**click** the Store icon to navigate to the Skype client s entry.	19	**3.3%**
			Build a dynamic view of a user s person list with content from the Groups collection.		
		e.g. (to + verb …… , verb)	**to end** a subscription**, call** the subscription.dispose method.	22	**3.9%**
			to create a new notification interface**, drag** a notification interface controller object to your storyboard file		
	Method call	1. call, invoke, use method/function. 2. 3. function expression e.g. get(),set(a)	you create sets using our API by creating a client and **calling** the post **method** with the /playlists endpoint and information about the set, including a list of track ids.	68	**11.9%**
	Contains	After/Before *as connector*	**after** a job finishes running, its reference to the progress group is lost.	37	**6.5%**
			the setuser call must also be made **before** the job is scheduled.		
	Begin with	note(that)	**note** if you are going to stream from our API you need to attribute properly.	16	**2.8%**
			note that in this case, object_id is the tag to which you would like to subscribe.		
	Contains	via/through	this feature is available only when connecting **through** telnet or ssh.	20	**3.5%**
	Contains	Variables names: '_' , '**/**'	Some API only require the use of a **client_id.**	38	**6.7%**
			you can also optionally include a **transit_mode** and/or a **transit_routing_preference**.		

Patterns of the semantic class. In semantic class, we observed different kinds of terms and sentence structures. Table 15 explains these observed patterns with examples as the following:

- Frequent terms (first three rows in Table 15): In this class, we classified some frequent terms into three different lists as the following: Output/Input verbs, Supporting verbs, and Admission verbs. See Appendix (Table 34. Output/Input verbs, Table 35. Supporting verbs and Table 36. Admission verbs). The total summation of these patterns equals (103+90+74=267), which constitute 48.7% of the overall semantic class.
- Sentence structures (all rows in **Table 15** except the first three columns). In which there are sentences contains some structure like the sentence contains "by" and followed by "Gerund", or there are sentences begin with "for" and followed by "Noun" or "Gerund". As another example, there are sentences contains "so that", "because", "in order to".

Table 15. Identified patterns of Semantic class

Pattern Type	Pattern Name	Example	COIN with pattern example	COINs count	COINs %
Frequent terms	**Output/Input verbs**	return, receive, **display**, response, send, notify retrieve, select, read, recover, access, fetch, upload, download, submit, recall, share, result	Thus the help command will **display** help only for the commands with the specified scope.	**103**	**18.8%**
	Supporting verbs	support, **provide**, Suggest, give, propose.	A dynamic notification interface lets you **provide** a more enriched notification experience for the user.	**90**	**16.4%**
	Admission verbs	**allow**, enable, admit, let, give, grant, permit, facilitate, authorize, prevent, stop, avoid	The console **allows** custom command completers to be provided.	**74**	**13.5%**
Sentence structures	**Contains**	by + {Gerund}	Action buttons save time for the user **by offering** some standard responses for a notification.	**29**	**5.3%**
	Begin with	for + {Noun or Gerund}	**For** remote **notifications**, add the title key to the alert dictionary inside the payload.	**7**	**1.3%**
	Contains	for + Noun/Gerund	Eclipse provides a common user interface (UI) model **for working** with tools. The plug-ins that make up a subsystem define extension points **for adding** behaviour to the platform.	**30**	**5.5%**

			You can search for directions for several modes of transportation, include transit, driving, **walking** or cycling.		
	Begin with	***if you*+…., will + (be) / you can**	**if you** want to load the library separately from the HTML code, **you can** call the oEmbed endpoint with the omitscriptparameter.	**13**	**2.4%**
		in *(that)(this)* **case**	In **that case**, the platform uses some heuristics to determine which one should be selected.	**4**	**0.7%**
		it + modal verb + (be)	**It should be** stable enough so that industrial strength tools can build on top of it.	**5**	**0.9%**
		note(**that**)	**note:** Tapping your app s glance interface always launches the app.	**8**	**1.5%**
		(**to** + **verb)** …… **, verb**	to ***embed*** instagram content you need to first visit the post on the web and get the embed code.	**7**	**1.3%**
		(use)(using) + …. + ***to***	**use** promise chaining **to prevent** application logic from changing the state of an object until the object is initialized and ready	**26**	**4.7%**
		when	***When*** configuring the interface, specify the JSON data file containing the test data you want **delivered** to your interface.	**12**	**2.2%**
		you can, you could	**you can** cancel a presence subscription for a given person at any time.	**29**	**5.3%**
	Contains	**in order to, so that, because**	Many developers use this flow **because** of its convenience.	**16**	**2.9%**
			it should be stable enough **so that** industrial strength tools can build on top of it.		
			we will gloss over a lot of details **in order to** get the plug-in built and running.		
		(user)(you)(we) + modal verb	With a dynamic interface, **you** can display more than just the alert message.	**37**	**6.8%**

5.3 Discussion

After we reviewed the results and statistics from table (Table 13. Identified patterns of Not-COIN class, Table 14. Identified patterns of Dynamic class and Table 15. Identified patterns of Semantic class) which contains the frequent patterns in the sentences and the ratio of each pattern that occurs in each COIN class. (Not-COIN, Dynamic and Semantic) classes. Thus, we observed that, our Seven-COIN Corpus has imbalance amount of sentences from each COIN class as the following:

- The majority of the COINs about 34.4% of the total COINs are classesified as not-COIN class, the reason is because there are many technical description and many technical terms used in the API documentation.
- The minority of the COINs are classesified as (context, structure, syntax, quality), because these kinds of COINs describe the service usage context, terminology definitions or quality attributes of services or systems, which are rarely mentioned in the APIs.
- COINs of type (dynamic and semantic) are distributed equally. The dynamic class has a ratio of 28.3%, while the semantic class is about 27.4% from the whole COINs.

The second observation is that: the statistics in (Table 10. COINs classes distribution per each case) reveals very important information about the COINs distribution over each case study. For example:

- SoundCloud: is easy to read and to find out the conceptual constraints. Most of the sentences in the API documentation are short and direct to understand, which does not confuse the reader to extract any of the COINs. In addition, this documentation has a small size compared with the other API documentation in our corpus.
- The GoogleMaps APIs has the majority of not-COIN class of 298 from the total number of GoogleMaps COINs which are 473, which means there are more than 62% of the COINs in GoogleMaps are only belongs not-COIN class. And this is very huge ratio compared with its COINs size and also with the other cases. This is because the GoogleMaps API documentation has a lot of technical terms. Moreover, there are few concepts, background paragraphs. GoogleMaps APIs seems to be more technical than conceptually.
- Eclipse-Plugin has a balanced COINs distribution, especially for (not-COIN, dynamic, semantic). In addition, it has the highest number of COINs of type structure of 42 COINs. Therefore, we can conclude that, these balances are due to many reasons:
    - Eclipse-Plugin is the largest APIs we extracted and it contains 651 COINs.
    - It has many sections for describing the concepts and the abstract level of knowledge, such as: abstract, introduction, and overview section.
    - It has less technical information.
- AppleWatch: has very well structural paragraphs, even it is long documentation, but it is easy to track and read. In addition, this documentation has three main classes. These are, Not-COIN of 40.8%, Dynamic of 26.1% and Semantic of 25.0%.

5.4 Threats to validity

Generalizability

To avoid having results applicable for one case of API documents and to make our findings generalizable, we decided to include multiple cases in our search for the COIN patterns in textual sentences. That is, we have chosen six API documentations, which are: SoundCloud, GoogleMaps, Skype, Instagram, AppleWatch and Eclipse-Plugin Developer Guide. Moreover, we have collected data from another API documents which is from Amazon Storage Service S3, but unfortunately, we ran out of time to analyze it, although we have processed it. In general, we have covered 2283 sentence from the six cases, which gave us a good impression of the typical amount of COINs as well as their distribution in the API documents.

Completeness

Due to time limitations, we were unable to analyze a large number of API documents despite of its prominent role in finding out more patterns. However, we have selected inclusive parts of the large API documentations (e.g. in the API document of Eclipse, we covered the Plug-in part that has about 651 sentences).

In fact, manual processing of data takes a very long time, and the reason behind that is due to the need firstly to cleaning data from noise (such as images, tables, symbols, etc.) and then organizing paragraphs into short sentences, and after that we analyzed these sentences manually to extract patterns, which we build the rules based on it, then we fed it up to the classification model. The accuracy of the automated classification model affected significantly by the quality and quantity of this data.

Researcher bias

In this thesis, we built our corpus in a way that guarantees results accuracy and impartiality. Accordingly, the manual classification process. The manual classification process is a process of identifying the proper COIN class (i.e. Not-COIN, Dynamic, Semantic, Syntax, Structure, Context and Quality), which is performed through a manual labor by reading the sentence and understand the meaning using of the Constraints of COIN Model [1]. Understanding the sentence correctly plays an important role in determining the right COIN class, and that the

In our research, the manual classification process was performed separately by two researchers from the Software Engineering Research Group (AGSE) of University of Kaiserslautern. Each researcher classified the retrieved sentences from the API documents independently. Our process flow for the document classification is shown in Fig. 16

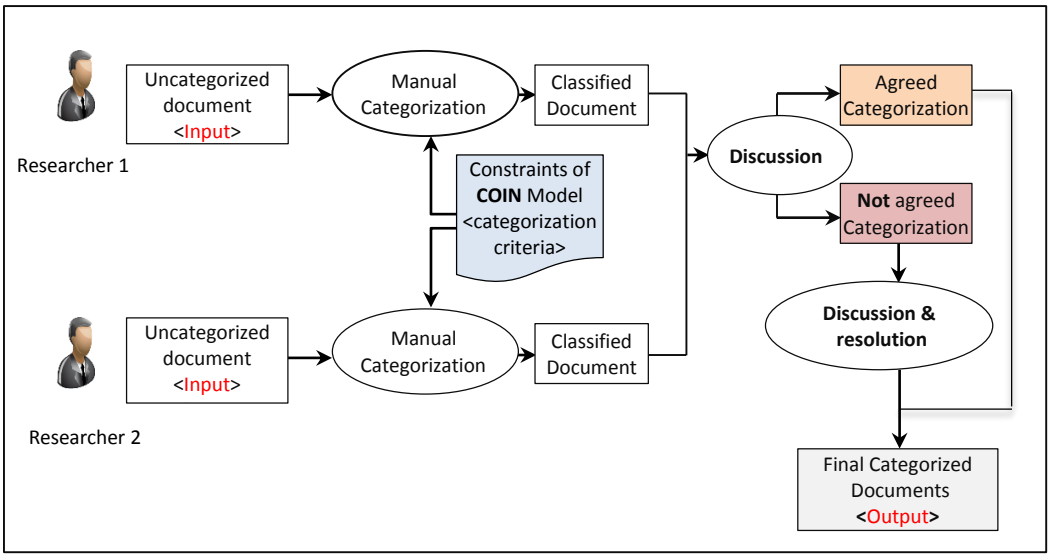


Fig. 16. The classification process performed by two different researchers

The classification processes were performed in four stages as the following:

First: Both researchers started classifying the sentences of each API documents based on the COINs sheet model [1] as shown in Table 1.

Second: After classification and in multiple discussion sessions, the researchers compared their classification decisions. For conflicting classifications, they created a "non-agreed list" to be re-discussed and re-classesified later and continued comparing the rest of the COINs and kept them in an "agreed list". Then, the researchers revisited the "non-agreed list", discussed and resolved based on consensus.

We evaluated the accuracy of this process by using agreement percentage [50], which was almost 75%.

Agreement Percentages is shown in We evaluated the accuracy of this process by using agreement percentage [50], which was almost 75% that we obtained using the following formula:

$$P_A = \frac{N_A}{N_A + N_D} \times 100 \qquad (1)$$

Where P_A refers to the percentage of agreement, N_A the number of agreements, and N_D No the number of disagreements [50].

Third: Collecting the final classified document into one final data sheet, that was the input for all the later analysis activities. It was used to discover the patterns as we described previously in section 5.2 and was used to feed up our ML model as we explain later in the next chapter.

6 RESEARCH PART TWO: AUTOMATIC IDENTIFICATION

In this part of our research, we aim at answering the second research question, which is: "**RQ2:** How effective would it be to use Natural Language Processing (NLP) along with Artificial Intelligence (AI) technologies to automate the extraction of COINs from API documentation?" Please see Fig. 17.

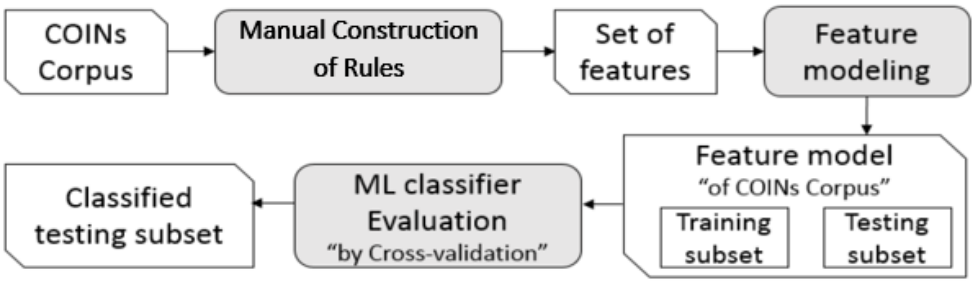


Fig. 17. 'Process Flow' of the first machine learning classification approach

To achieve this, we used two different approaches to investigate the potentials of using technologies from machine learning and natural language processing. The first approach is Rule-based classification system using NLP with ML. While, the second approach is ML classifiers using Bag-of-Words model.

In this chapter, we explain each approach in details showing the exploratory experiment configuration, execution, and performance results in terms of accuracy.

6.1 First Approach: Rule-based Machine Learning Classification

Within this approach, we aimed at investigating the benefits of utilizing our manually identified patterns through NLP technologies in extracting the representative features of the textual sentences in the COIN Corpus as a matrix of attributes. For this goal, we adapted and extended our discovered patterns (which we observed in our multi-case study) into rules that we could use for training a machine learning model.

Rule construction using NLP

As we mentioned earlier, we identified the frequently used terms and sentence structures for the main three COIN classes (Not-COIN, Dynamic, and Semantic).

In this class, we observed the following patterns:

Frequent terms: The predominant part of terms in this class are the Technical Keywords, which are mainly abbreviation of technical terminology and programming keywords. For example (XML, iOS, XPath, JavaS-

cript, ASCII, KB, MB, etc.). In our corpus there are about 30.7% (of the Not-COIN class) has technical keywords. This means that, there are about 295 COINs of 960 COINs have one or more technical term (see Table 13 the first row).

For further clarification, Table 13 is composed from five columns. First columns is a pattern type (e.g. frequent terms or sentence structure). Second column is a pattern name (e.g. technical keywords, sentence begin with some term, sentence contains some terms, etc.). Third column is example of the pattern term. Forth column is a real example from the corpus, Fifth column is the total number of the occurrence of the pattern in the corpus with respect to the COIN class and last column is the percentage of the occurrence of the pattern in the corpus with respect to the COIN class. Note that in the fifth column the cell values do not add up to 100% as there are minor patterns that take a share of it but we do not cover them in the table.

Sentence structures: the second part of the patterns are sentence structure, as aforementioned these structures are illustrated as shown in all rows of Table 13 except the first row.

In this regard, we observed that, there are relatively two significant patterns in this class, which are:

Sentences contain variables, symbols and tags. For example a sentence “XML responses consist of zero or more <route> elements.” It is classified as Not-COIN, as you can see contains some tags and technical terms. Such tags and the special characters like ‘/’,’\’,’<’,’>’ constitute 13.9% from the whole not-COIN class.

Sentences begin with the terms, like “for example”, ” for information” and sentences contains terms like “see”, “learn”, “let’s” all together constitute 12.8%.

We improved and reformulated these observed patterns. That is, we involved a wider range of terms and sentence structures by utilizing both the observed (terms, patterns and rules in the multi case study) and the “Constraints of COIN Model”. Based on this, we constructed the rules needed in the Rule-based classification approach. This rule construction was performed by utilizing NLP techniques (i.e., sentence tokenizing, stemming, stopwords removal, part of speech recognition, N-Grams). In our research, we used NLTK (i.e., a leading platform for building Python programs to work with human language data). Table 16 summarizes our rules with examples for more than 13 rules, but we described the most 13 significant ones, and these rules are (Definition, Goal, Conditional, Explanation/Example, Method Call, Modal Verb, Resource, Structure, Technical Term, Variable, Warning, Output/Input and Action Verb)

Table 16. Rules Names with examples

#	Rule name	Example of sentences satisfying the rule
1	Definition	oEmbed **is an** open standard to easily embed content from oEmbed providers into your site
2	Goal	Background actions launch the containing iOS app in the background so that it can process the action
3	Conditional	**if** a command name is specified, the help message for this command is displayed.
4	Explanation/Example	**for example**, you can use this to protect against CSRF issues.
5	Method Call	in order to embed a player widget using JavaScript SDK, you can **call SC.oEmbed()** function
6	Modal Verb	**you can** also get a list of comments for a specified sound
7	Resource	artifacts for each tool, such as **files data**, are coordinated by a common platform **resource** model.
8	Structure	Fundamentally, a bundle is just a **collection** of **files** (resources code) installed in platform
9	Technical Term	instead, create a complementary experience to **iOS** app.
10	Variable	Some API only require use of a **client_id.**
11	Warning	**do not assume** access_token is valid forever.
12	Output/Input	on success, function **returns** true.
13	Action Verb	to perform a task, a plug-in creates a job then **schedules** it. // see Appendix

1- **Definition**: This rule is implemented to check the sentence grammar or structure looking for definitions. This rule extends the linguistic rules that are stated in [51], to cover more cases that what we observed in our multiple-case study. For instance, we included additional patterns for sentences including definitions like "is called as", "is known as" and "is declared as". This rule is mapped to Syntax COIN.

2- **Goal**: We established some rules to discover if a sentence is stating a goal. For this purpose, we implemented a method that utilized NLTK[5] & Python[6]. For example, a sentence that contains terms like: (so that, in order to, to +verb + any word(s) +',' + any verb, etc.). This rule is mapped to Semantic COIN.

   Examples: a sentence: "a client must have a user_name and a password in **order to** log in to the server".

3- **Conditional Statement:** This rule detects the preconditions by checking if the sentence begins with a conditional clause that starts with like (if, when, once, while, until) and its other clause begin with ',' + then. This rule is mapped to Dynamic COIN.

4- **Explanation/Example:** This rule detects if the sentence contains some kind of further explanation or examples. There are special words that we observed them to be used in such statements like: for example, as an example, for instance, etc. This rule is mapped to Not-COIN.

[5] NLTK is a leading platform for building Python programs to work with human language data. http://www.nltk.org/

[6] Python is a programming language developed under an OSI-approved open source license, making it freely usable and distributable. https://www.python.org/about/

5- **Method Call**: This rule aims at discovering the statements that has function calls. We developed a regular expression rule to detect if the sentence contains a function signature (e.g., Class.setText()) or some keywords (e.g., call, invoke, function, method, etc.). This rule is mapped to Dynamic COIN.
6- **Modal Verb**: This rule aims at detecting modal verbs from the sentence. This rule is mapped to Dynamic COIN.
7- **Resource**: This rule aim at detecting statements about required resources to use a function. We defined a list of keywords, which are used to indicate resources words. Some instances of these keywords in our list are (access, client, file, network, disk, and more.) This rule is mapped to Dynamic COIN.
8- **Structure**: This rule aims at finding any structure design decisions declared in the textual sentences. Hence, we created the list of top 500 keywords for this rule based on the terms of the "Data Structures and Algorithms in Java, 6th Edition" [52]. These keywords list include (e.g., database, inherit, override, implement, extend, etc.). This rule is mapped to Structure COIN.
9- **Technical term**: This rule aims at detecting technical terms like any keywords looks like abbreviation such as (XML, XPath, SQL, SSL, etc.). For this purpose, we use a regular expression (e.g., word with all capital letters or/and words with short characters and capitalized like ABC, SSL, etc.). This rule is mapped to Dynamic COIN.
10- **Variable**: This rule aims at finding any variable in the sentence. This rule is developed by using regular expression, in which a given sentence is checked whether it contains any word represents a variable (i.e. client_id, _parameter, user_name, etc.). This rule is mapped to Dynamic COIN and Not-COIN.
11- **Warning**: This rule aims at detecting sentence that contains warning statements (i.e. take care, pay attention, be aware, be careful, etc.). In this case, we defined a list of terms that contains similar meaning of warning. This rule is mapped to Semantic COIN.
12- **Output/Input**: The rule aims at detecting the activities of type input or output. We developed a method to check our predefined list, which contains keywords like (return, output, display, throw, etc.). This rule is mapped to Semantic COIN.
13- **Action verb**: This rule is used to detect action verbs that describe activities (which we noticed to appear frequently in Dynamic COINs. For this purpose, we defined a list for these verbs (i.e. run, complete, open, process, start, etc.). For complete information about the Action Verbs, please see Appendix (Table 33). This rule is mapped to Dynamic COIN.

Exploratory Experiment

After having our rules ready, we conducted the exploratory experiment to see the potential that Rule-based classification can bring to our goal to automate extracting the COINs from API documents. We performed this experiment in two phases as the following:

Phase 1: Preparing the training data set. We aim by this phase at generating a data set to be fed back as a training set to the classification model, which in turn, train on it and later on predicts the right COIN Type. Bellow, we describe the process of this phase as the following:

Input: The Seven-COIN corpus

Process:

- For each sentence in a corpus

1- If the rule is satisfied by a sentence, then the method for that rule return 1,

2- Otherwise return 0.

However, in some cases, methods might return integer value greater than 1, which is a summation of how many times the term in a sentence occurs. For example: in case of the rule "Technical Term", the method is developed to scan the sentence and return the total number of technical terms (i.e. SQL, XML, iOs, etc.), which can be any number greater or equal to 0.

Output: The result of this phase is a matrix of rules and sentences.

This means, for each sentence it gets a score for each of the rules as seen in Fig. 18, which shows the snapshot of only the first 8 rules of the matrix for seven sentences.

Sentence	Definition	Goal	Conditional Statement	Explanation/ Example	Method Call	Modal Verb	Resource	Structure
a user's availability can be free and their activity can	0	1	0	0	0	0	0	0
Because this authorization flow depends on passing a set of user credentials, it	0	1	0	0	1	0	2	0
Depending on your needs, you can embed a player widget, use the JavaScript	0	1	0	0	0	0	2	2
for a full list of properties that can be set on a sound	0	0	0	1	0	0	3	1
Given a sound or set URL, you can retrieve all of the	0	1	0	0	0	1	3	1
if you are writing an application that runs in the browser, you should	0	0	1	0	2	1	1	1
if you donot want to use the SoundCloud widget, our API gives you the	0	0	1	0	0	0	5	0

Fig. 18. Snapshot of an excerpt of the rule matrix

We also show in Fig. 19 the distribution of our rules over the Seven-COIN classes.

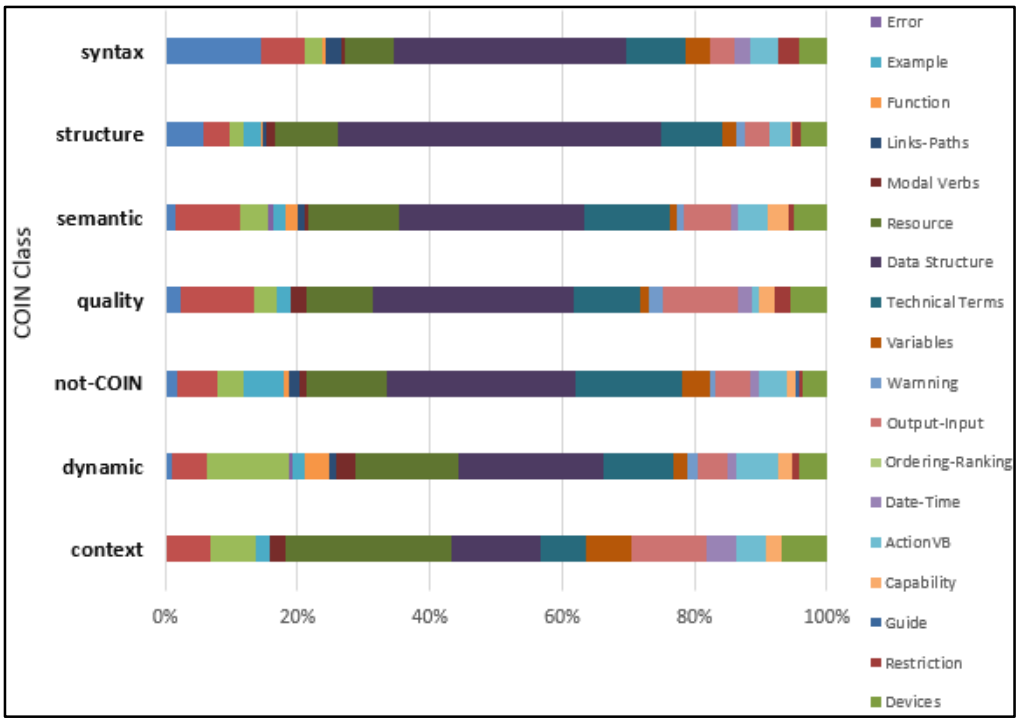


Fig. 19. Rules distribution over the COINs Classes

It can be noted from Fig. 18 some rules apply to all COIN Classes, but the decision is not based on one rule satisfaction, but on all rules together.

Generally, we can observe that different COIN classes satisfy more than one rule. For example, all COIN classes share rules on structure keywords and technical terms. On the other hand, each COIN class has some special rules that is satisfies unlike the other classes. For example:

- Syntax COINs: "Definition" rule is one of the predominant rules of this COIN class compared to the other classes.
- Structure COINs: “Structure” rule is the mainly satisfied rule.
- Semantic COINs: This class has a balanced distribution between three main rules (Goal, Conditional, and Explanation/Example).
- Quality COINs: As we have a few instances of this class (i.e., 28 sentences only), we could not determine the exact features or rules for this COIN class.
- Not-COIN: This class mostly satisfies the “Technical term” and the “Conditional” rules. It also satisfies the “Goal” rule to some extent.
- Dynamic COINs: There are many instances of this class that satisfies the “Conditional” sentence rule, “Resource” terms, and “Action verb” rules.
- Context COINs: This COIN class is similar to the Quality COIN class in terms of the few number of instances. Hence, we do not have enough data (i.e., 13 sentences only) to decide on the rules it satisfies more frequently. For this limited instances we noticed they satisfy the “Conditional” rule.

Phase 2: Selecting ML classification algorithms. Based on our literature review and deep investigating for the different existing ML algorithms that are specifically used for text classification, we found that Naïve Bayes (NB) [25], and Support Vector Machine (SVM) [26] are the most effective and recommended ones

[53] [54] [55]. Nevertheless, our curiosity made us decide to try the different versions of Naïve Bayes (i.e., Complement Naïve Bayes [56], Naïve Bayes Multinomial updatable [57], Naïve Bayes Multinomial [58], Naïve Bayes Updatable [59]). In addition, we included other algorithms like Decision Tree (J48) [60], Random Forest Tree [61], Simple Logistic [62], Logistic Regression [63], and K-Nearest Neighbor (KNN) [64].

Phase 3: Configuring and running tests for the ML classification algorithms. Having the classification algorithms selected, we trained the ML classification model using the matrix that we produced in phase 1. The test was run on the two versions of our COIN Corpus (i.e., the Seven-COIN Corpus and Two-COIN Corpus). For the activities in this phase, we used Weka 3.7.13 [7] to train and test the classification models.

1- Seven-COIN classification experiment run.
   In this experiment run, we trained the classification model using the rules' matrix that resulted from phase 1. Then we ran the selected above mentioned text classification algorithms.
2- Two-COIN classification experiment run.
   Similarly, in this experiment run, we trained the classification model using the rules' matrix that resulted from phase 1. Then we ran the selected above mentioned text classification algorithms.

Generally, for each test run, the corpus was divided into a training and testing sets. The training set we used for teaching the classification model about the rules, while we used the testing set for determining the model's classification accuracy. In specific, we used the k-fold cross-validation [65] [66], in which the data set in our corpus is divided into k subsets. Then, (k-1) subsets of the data set are used for training and one subsets used for testing. As we used k = 10 for 10 rounds, then we got in each round 9 subsets are used for training the classification model and only one subset is used for testing. Finally, we computed the average of the 10 runs.

Phase 4: Evaluating the experimental results. Next, we briefly introduce the metrics we used in evaluating our experimental results to evaluate the results. Then, we use them in interpreting the results in details.

Evaluation Metrics

Recall, Precision and F-Measure are the most commonly used metrics for evaluating the accuracy of text classification models [67]. Hence, we used these metrics, which we explain below, for evaluating the results of all our tested ML classification algorithms.

Recall (R) is the ratio of the records that are correctly predicted to the total number of relevant records in the data set. Recall is a fractional number between 0 and 1 and usually expressed as a percentage [68] and is calculated using the following formula:

$$Recall\ (R) = \frac{\#\ of\ correct\ prediction}{\#\ of\ relevant\ records} = \frac{\#TP}{\#TP + \#\ FN} \qquad (2)$$

Precision (P) is a ratio of the documents that are correctly predicted to the total number of relative and irrelative records that are retrieved from the data set [68]. Precision is a fractional number between 0 and 1 and expressed as a percentage and is calculated using the following formula [68]:

[7] Weka is a collection of machine learning algorithms for data mining tasks. http://www.cs.waikato.ac.nz/ml/weka/

$$Precision\ (P) = \frac{\#\ of\ correct\ prediction}{\#\ of\ (relevant + irrelative)\ records} = \frac{\#TP}{\#TP + \#FP} \qquad (3)$$

F-Measure (F) is a combination of recall and precision. F-Measure is a popular evaluation metric for imbalance problem, in which the data set are not classified equally (e.g., some of the classes are more than the others) [69] [70]. F-Measure is calculated using the following formula:

$$F = \frac{2 \times precision\ \times recall}{precision + recall} \qquad (4)$$

Results and Evaluation

As mentioned earlier, we performed the experiment on two versions of the corpus, first one on the Seven-COIN Corpus, and the next on the Two-COIN Corpus.

With regards to the Seven-COIN classification results, we found that the best F-Measure was achieved by Logistic Regression, Recall of 47.0%, Precision of 57.7% and F-Measure of 47.6% (See Table 17).

Table 17. Model performance for classifying Seven-COIN

Classification Algorithm	Recall	Precision	F-Measure
Logistic Regression	47.0%	51.7%	**47.6%**
Naïve Bayes	50.2%	45.8%	46.5%
J48	49.8%	46.1%	46.5%
Complement Naive Bayes	45.6%	49.2%	46.4%
Neural Network	49.2%	45.8%	46.2%
Random Forest Tree	47.1%	44.4%	45.0%
KNN, k=18	49.6%	46.7%	43.7%
Support Vector Machine	49.6%	43.9%	43.7%

Similarly, we found that the best results of the Two-COIN classification was achieved by Logistic Regression, Recall of 66.5%, Precision of 66.1% and F-Measure of 65.7% (See Table 18).

Table 18. Model performance for classifying Two-COIN

Classification Algorithm	Recall	Precision	F-Measure
Logistic Regression	66.5%	66.1%	**65.7%**
Naïve Bayes	66.0%	65.5%	**65.3%**
Complement Naive Bayes	64.3%	64.8%	64.5%
J48	64.5%	64.0%	63.9%
Neural Network	63.4%	62.7%	62.4%
KNN, k=18	62.3%	61.9%	62.0%
Random Forest Tree	62.2%	61.9%	62.0%
Support Vector Machine	64.0%	65.6%	59.1%

Conclusion on rule-based classification

According to the results we obtained from these experiments, we conclude that:

- The accuracy of the first model (i.e., Seven-COIN classification) gave a maximum accuracy F-measure of 47.6%, which is obtained by applying the logistic regression algorithm.
- On the other hand, the second model (i.e., Two-COIN classification) gave a little bit improved accuracy with F-measure of 65.3%, which is achieved using a Naïve Bayes algorithm.

To the best of our knowledge, these results can be improved if we have a larger data set (i.e., more manually classified sentences in the corpus). That is, our contributed corpus has a small size (less than 3K of COIN sentences). Still, we believe that these achieved results are promising and this encourages us to investigate different strategies to optimize the results by using other possible text classification algorithms.

6.2 Second Approach: Bag-of-Words-based Machine Learning Classification

In this section, we explore another approach for automating the extraction of COINs from textual content of API documentation by using ML classifiers along with the Bag-of-Words (BOWs) model [32].

As we saw in the previous section, Rule-based ML classification using our manually identified rules did not provide high accuracy results. Hence, we expanded and intensified our efforts toward exploring other text classification strategies. By reviewing further research papers in machine learning and text classification methods, we found that there is a representation model for the data that could show better effectiveness and efficiency in classifying natural language text called Bag-of-Words. In BOWs model, each sentence is represented as a collection of words after tokenizing it using natural language processing techniques. For example, a sentence like "This is a model" is represented as {'This', 'is', 'a', 'model'}. Thus, each word represents an independent feature. The co-occurrence of words is weighted using a model called TF-IDF (i.e., Term frequency –Inverse Document Frequency) [33] that we will explain in more details later in this chapter.

Accordingly, we decided to adopt the BOWs modeling in our research to see its potentials in classifying the sentences of API documents into the COIN classes. This required us to follow the process as shown in Fig. 20, which we have published in a paper related to the thesis work [2]. In next sections, we describe the details for each step of this process.

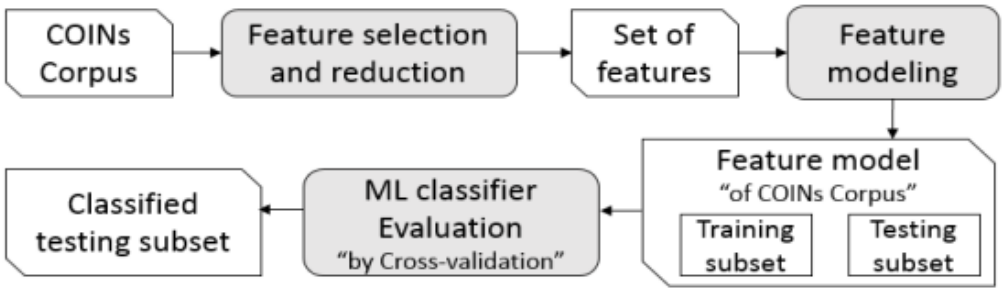


Fig. 20. 'Process Flow' of our model [2]

Data preparation

In this stage, we prepare the data set (i.e., the sentences in the Seven-COIN Corpus) by transforming the format and cleaning the content that we describe next.

Format transformation. We transformed the format of the sentences in the corpus from CSV (i.e. Comma Separated Values) to ARFF (i.e. Attribute-Relation File Format[8]).

Content cleaning. For performance consideration of the classifier, some sentences needed cleaning to remove the technical noise that existed in the non-natural language text (e.g., http links, resources and path, service location, variable definitions, or functions call). Such technical noise exists frequently in API documents text to explain the technical usage of the offered APIs. Therefore, we developed some text manipulation techniques to reduce the technical noise as the following:

[8] ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. URL:https://weka.wikispaces.com/ARFF+(stable+version)

- ***Hyperlinks***: basically, we defined a regular expression to replace all hyperlinks that might exist in the text with a constant term (i.e., 'Hyperlink'). That is, the whole textual content is checked to find if it has any hyperlink to replace it with this constant using the following regular expression that we defined:

```
(https?:\/\/)?(www\.)?(ftp\:\/\/)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,256}\b([-a-zA-Z0-9@:%_\+.~#?&//=]*)+[^\( | ^\)]
```

The above regular expression is used to detect hyperlinks like:

https://www.facebook.com, http://yahoo.com, www.google.com, http://speedtest.tele2.net .

- ***Resources' path:*** API documents might contain some paths that point to particular resources, locations, online data, or further information regarding the usage of the API. Such technical noise is very similar to the hyperlink noise. In order to manipulate this noise, we also developed another simple regular expression as the following:

```
\w*(((\/)|(\\))+(.*))+
```

The above regular expression is used to detect hyperlinks like:

file/document/ , /location/windows/abc , \server\pc\ , \\file\\system.

- ***Variables:*** Any expression of words in the form of X_Y or _X can be considered as variable and here we replace it with a constant term "**VARIABLE**". For that, we developed the following regular expression:

```
\w+_\w+  and \s\w+\/\w+\s
```

As an example: "if neither time specified, departure_time defaults now (that is, departure time defaults current time)." After replacement by our regular expression, it looks like " if neither time specified, **VARIABLE** defaults now (that is, departure time defaults current time)."

- ***Function Call:*** In some sentences, there are a piece of code used as examples to explain how the function or method works, this code contain a function call. Hence, we developed a simple regular expression to capture this pattern as the following:

```
\w*\.*\w+\(\w*\)
```

Example: set( ), get( ), add(a,b), print(x).

Perquisites input for our ML classification model.

In our study, we observed that the conceptual constraints are the non-technical information that their main concern is not "how" to implement, configure, or deploy the service or system.

According to these observations in our experiments, we ignored some technical information from the analysis process. For example, we did not include pure code, and partially technical terms explanation in some section. Especially in development pages, examples and technical help which in general, they have non-representative information about concepts, same as the (mixed-code with technical terms), or non-meaningless sentences (basically: our model is not customized for grammar/spelling checking). Thus, we assume that the grammar of the document's content should be well formed with the right spelling. Therefore, textual input to our model should be correct, complete, and meaningful sentence(s).

In addition, we also exclude headers, paragraph titles, footers, image/figure description, and table description, which mostly do not help to detect any useful information about the system/service concept. In addition, we did not include "text-as-link", and some sentences that contain one or more function(s)/variable(s)/parameter(s)/link(s) or/and sentence that contains many non-natural language (NL) terms. As an *exception*, in some cases we included some sentences, which contain non-NL, like functions/parameters/etc., sentences that have technical keywords, etc., only if the context is about a concept or non-technical constraints.

On the other hand, we excluded sentences from the SDK documentation part of the API documentation. This due to our awareness of the technical dominance in this part of the API documents, which we observed and concluded through the manual analysis, processing, and classification of sentences in out multiple-case study. Beside, our prior knowledge, excluding this technical part allows better learning for the ML classifiers and consequently better classification results later.

Exploratory Experiment

In this section, we conduct two different experiments, one for Seven-COIN classes and the second for the Two-COIN classes. These experiments are performed to measure the performance of the text classification algorithms in terms of accuracy.

Phase1: Applying NLP Pipelines.

The purpose of NLP Pipelines (processes/tasks) is to select the most presentative features (keywords) by first cleaning corpus from noise (insignificant words like Stopwords, punctuation, etc.) and then grouping similar words into one form using stemming, in which the word will reduced to its root (e.g. Recording → record, operation → operate, playing →play, etc.). For more details, our Seven-COIN corpus contains 2283 COINs (sentences), these COINs consist of 41,287 words, which in average there are 18 words per one COIN (sentence). Hence, we aim at representing each COIN with the most informative words and filtering out the less important words, thus NLP pipelines helps to perform such a task if we applied the previous tasks, then we will get only small number of features (keywords) compared with if select all words in the corpus. Then, by using TF-IDF (which is the last process performed on the resulting features from the previous processes above-mentioned), then these features will be weighted according to its importance in the corpus.

The input is: the sentences in the corpus.
This is to transform the textual data set into a mathematical representation that is the required form to be fed up to the classification model. More specifically, the input to this phase is the whole manipulated Seven-COIN corpus resulting from the previous data preparation.
The output is: a weighted matrix of the weighted features.

Note that our NLP pipe lines are performed completely using Weka v 3.7.11 1[9] as the following:

- ***Word tokenizing***: Once we obtained the sentence from the previous procedure (i.e., Sentence Tokenizing), we splitted each sentence into a subset of individual words. For example, the following sentence S1="All images must reside in the Watch app bundle" will be represented as a subset of keywords such as S1' = { 'All', 'images', 'must', 'reside', 'in', 'the', 'Watch' ,'app' , 'bundle'}.

- ***Lowering cases***: A word can be written in two different forms, but still the same. For example, at the beginning of the sentence, the first letter of the word is always in uppercase, while, in the middle of the sentence the same word would be written in lowercase. However, in machine learning technology, text classification algorithms do not consider such cases as the same word. This affects the performance results of classifiers in a negative way. Thus, we normalize all words to be in one form (i.e., lowercase). As an example, the word 'All' is converted to 'all' and the word 'The' is converted also to 'the'. Therefore, in this phase, we converted the all the words in Seven-COIN Corpus into the lowercase form.

- ***Stopwords Eliminating:*** Stopwords refer to the commonly used words that are considered as a conjunctive words, prepositions, adverbs, or pronouns. In our work, we adapted the default English Stopwords list, which is a list of English words are used in Weka. Hence, we adapted this list after we performed some experiments. The experiments were conducted many times by training a classification model on the data set of the Seven-COIN corpus, each time we used different Stopwords, until we got a best accuracy. For example, we found that some words like 'if', 'then', 'while', 'when' and modal verbs: 'could', 'can', 'would', 'will', 'shall', 'should', 'may', and 'might' should not be considered as Stopwords because it can change the accuracy of the classification model based on our observation during the experiments we conducted. The reason behind this is that, these words are used so frequently in the sentences within our Seven-COIN corpus especially for the following COIN classes: dynamic, semantic, and Not-COIN. Hence, if we exclude these words, then it will decrease the accuracy of the classification model. Thus, we defined a special Stopwords list for our model that does not include the modal verbs and some temporal conjunction, see Appendix (Table 37. Defined Stopwords)

- ***Words stemming:*** One useful and important NLP technique is to stem a word into its root that is considered to be the primitive lexical unit of any similar words [71]. Hence, we applied the stemming to aggregate all similar forms of the words into one unified form. This process reduces the number of keywords that share a similar root. After reviewing the comparative study of Stemming Algorithms [72], we had tested the performance of many stemming algorithms like Porter (Snowball) stemmer [73], Lovins stemmer [74]. Then, we decided to choose Snowball stemmer due its performance in terms of F-measure when used in classifying the COINs.

[9] Weka: http://www.cs.waikato.ac.nz/ml/weka

It is worth to mention, that we conducted an experiment to compare between the effects of the stemming and the lemmatization processes [71] on that accuracy of the classification algorithms (i.e., NB [25], SVM [26] and Complement NB [56]). The experiment results showed that the stemming process was better than lemmatization in terms of the classification models accuracy in terms of f-measure as shown in Table 19.

Table 19. Comparision between Stemming and Lemmatization in terms of F-Measure

Process	F-Measure		
	NB	SVM	Complement NB
Snowball Stemming	**62.8**%	**59.0**%	**70.0**%
Lemmatization	60.7%	57.6%	66.5%

- *Feature extraction using N-Gram combination:* At this phase, we aimed at extracting the features from the sentences in the Seven-COIN Corpus. At the beginning, we considered each single word in the sentence as a feature (Uni-Gram), but after performing a number of experiments, as we will show later, we decided to use N-Gram [35] where N is between 1 and 3. That is, we considered the features as each single word, each combination of two consecutive words, and each combination of three consecutive words. Such technique enables us to preserve the words' order and to keep the context of the sentence as well.

Terms weighting: In this stage, the whole COINS corpus is transformed into a mathematical model that is a matrix. In this matrix, the header contains all the extracted features from the previous phase, while each row represents a sentence in the corpus. Thus, each cell [row, column] holds the weight of a feature in the sentence. For achieving the best weighting, we used Term Frequency-Inverse Document Frequency (TF-IDF) [33].

Evaluation Metrics

We used the same metrics as the ones we used for the Rule-based classification experiment. That is we used in the first Approach: Rule-based Machine Learning Classification in the evaluation section as the following:

We used k-fold cross-validation, in which the data set in our corpus is divided into k subsets. Then, (k-1) subsets of the data set are used for training and one subsets used for testing. As we used k = 10 for 10 rounds, then we got in each round 9 subsets are used for training the classification model and only one subset is used for testing. Finally, we computed the average of the 10 runs.

Results and Evaluation.

In this section, we present the results that we have obtained after we have conducted the experiment on two different types of corpus, Seven-COIN corpus and Two-COIN corpus by applying different classification algorithms. And finally, we perform a statistical comparison between these results.

Classification accuracy achieved by the different ML Classifiers **(Seven-COIN Corpus case):** The results showed different values for different text classification algorithms. For classifying seven classes, we have achieved the best accuracy using 1,2,3-Gram with recall of 70.2%, precision of 72.4% and f-measure of 70% by using ComplementNaïveBayes algorithm (see Table 20). While, in the second place comes NaïveBayesMutinomialupdatable with accuracy recall of 65.1%, precision of 66% and f-measure of 65.4%. The rest of the results show accuracy f-measure between 62.8% and 52.3%. The worst results were obtained by Decision Tree J48 and KNN where (K=1, 2) algorithms. These results are better than our results which are obtained in our recent published paper [2] which reported f-measure of 62.2% using 1,2,3 Gram with NB algorithm.

Table 20. Accuracy comparison between different classification algorithms

Classification Algorithm	1,2,3 Gram		
	Precision	Recall	F-Measure
ComplementNaïveBayes	**70.4%**	**70.2%**	**70.0%**
NaïveBayesMutinomialupdatable	**66.0%**	**65.1%**	**65.4%**
NaiveBayes	64.3%	62.4%	62.8%
NaivebayesMultinomial	66.3%	59.5%	61.9%
Support Vector Machine SVM	59.3%	60.0%	59.0%
NaïveBayesUpdatable	55.3%	51.7%	52.5%
Simple Logistic	52.5%	54.4%	52.4%
Random Forest Tree	60.4%	56.3%	52.3%
Decision Tree J48	48.5%	49.6%	48.3%
KNN K=1	54.8%	45.5%	40.8%
KNN K=2	49.8%	36.1%	30.1%

In Fig. 21, we can see clearly that, the accuracy of 1,2 Gram is very similar to 1,2,3 Gram with very small difference (i.e., F-measure improved from 68.4% to 70.0%).

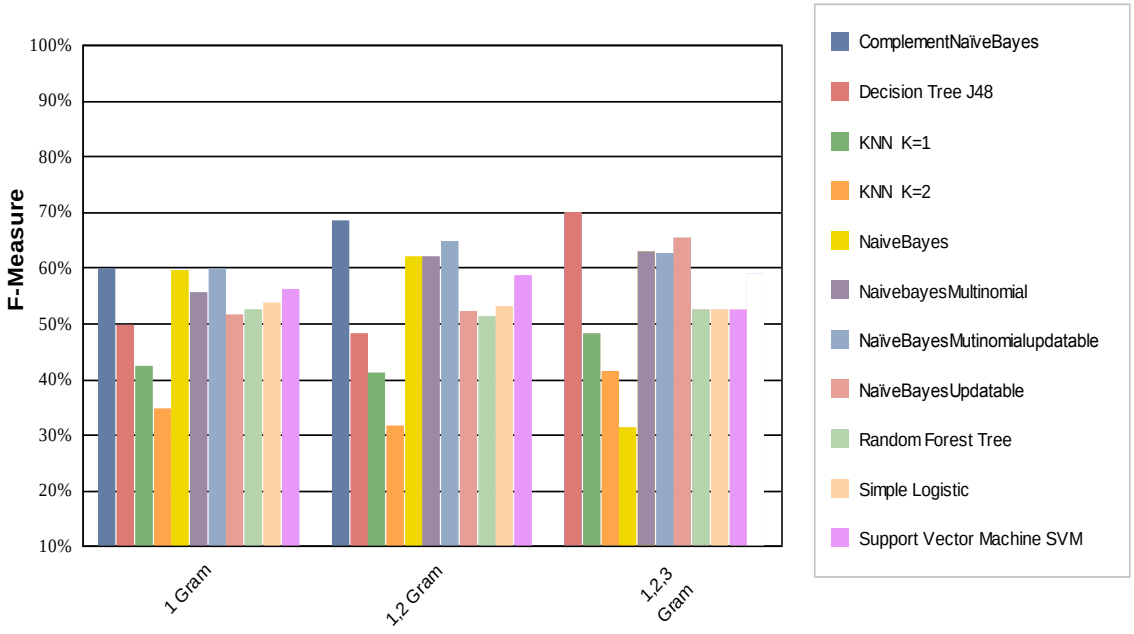


Fig. 21. Text algorithms performance via N-Grams

Table 21 shows the difference between these two variations in terms of F-measure.

Table 21. Accuracy comparison by using all words and top 1500 words

Algorithm	1,2,3 Gram					
	All words			1500 words		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
ComplementNaïveBayes	**70.4%**	**70.2%**	**70.0%**	**67.8%**	**67.9%**	**67.7%**
NaïveBayesMutinomialupdatable	**66.0%**	**65.1%**	**65.4%**	**65.3%**	**65.2%**	**65.2%**
NaiveBayes	64.3%	62.4%	62.8%	63.3%	62.0%	61.1%
NaivebayesMultinomial	66.3%	59.5%	61.9%	65.1%	61.1%	62.7%
Support Vector Machine SVM	59.3%	60.0%	59.0%	58.4%	59.0%	57.8%
NaïveBayesUpdatable	55.3%	51.7%	52.5%	55.1%	51.3%	52.2%
Simple Logistic	52.5%	54.4%	52.4%	53.1%	54.5%	52.3%
Random Forest Tree	60.4%	56.3%	52.3%	61.0%	53.5%	47.3%
Decision Tree J48	48.5%	49.6%	48.3%	48.5%	49.6%	48.3%
KNN K=1	54.8%	45.5%	40.8%	55.0%	47.1%	41.5%
KNN K=2	49.8%	36.1%	30.1%	51.6%	44.6%	31.3%

It is clear, that the usage of all corpus words gives more accuracy than just using only top 1500 words. This is also shown in Fig. 22. In addition, using 1,2,3 Gram with all corpus words is not only better than the using the top 1500 words, but it is also better than using bi-gram or uni-gram.

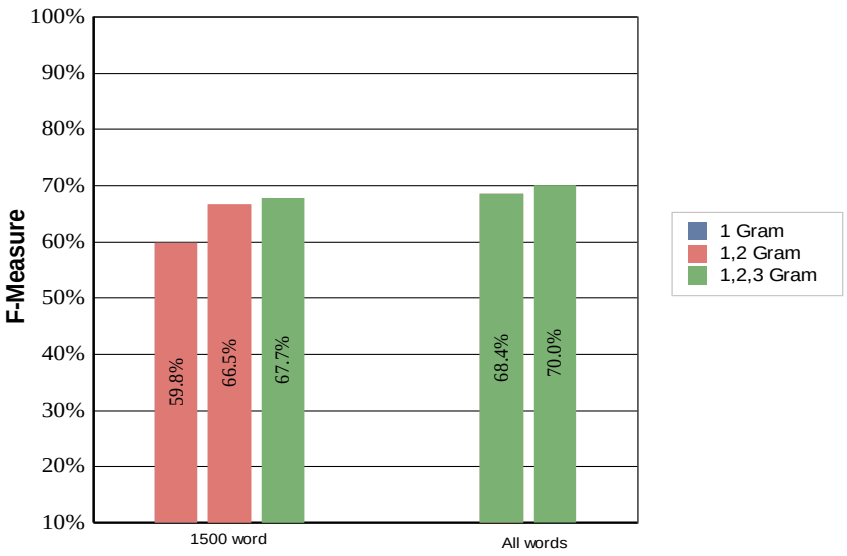


Fig. 22. Comparison between using corpus size corresponding to different N-Grams

For more precision, we conducted the experiment over the text classification algorithms and we observed that ComplementNaïveBayes achieves the best performance over all N-Gram combinations. See Fig. 23.

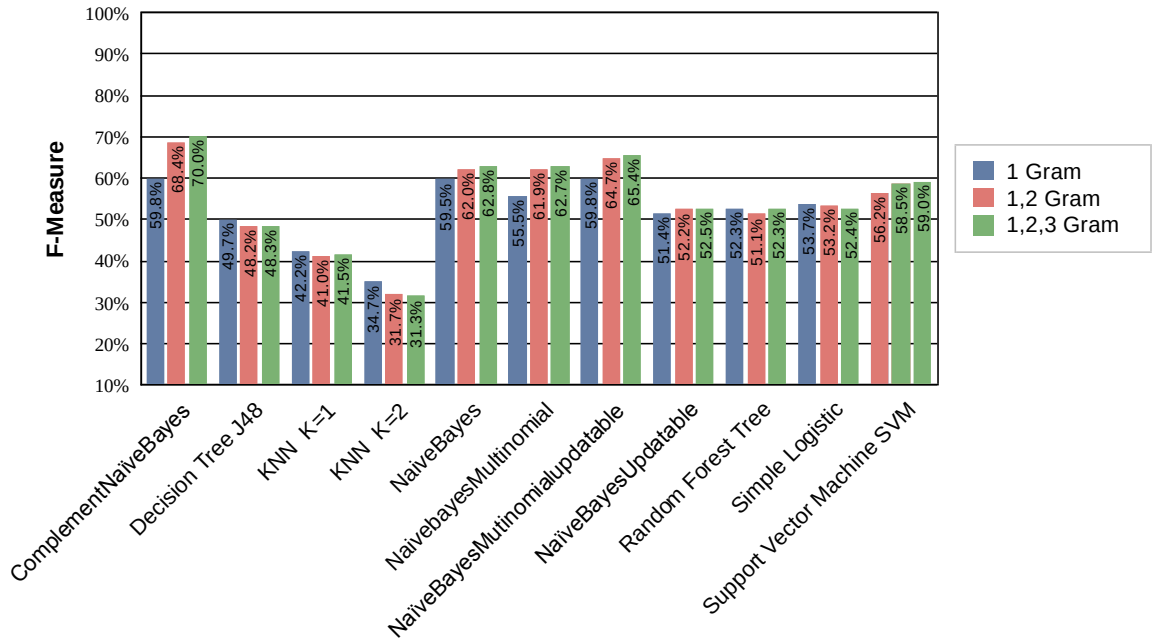


Fig. 23. Performance of text classification algorithms via different N-Gram combinations

Results improvement using linguistic knowledge

In an attempt to enhance the accuracy results, we incorporated linguistic knowledge by using WordNet [75] as stated to have a positive effect [76] [77]. In this regard, we decided to use hypernym[10], which employs the semantic relationship between similar words. For example, a set of words {'Blue','Red','Green'} has a common hypernym called '**Colour'**; as explained in Fig. 24. Explanation of Hypernym

[10] Hypernym is the name of a broader category of things [91]. For example, "colour" is hypernym of "red".

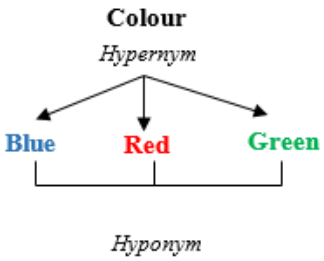


Fig. 24. Explanation of Hypernym [11]

Hence, we developed a python method to extract the hypernym of all words of type verb or noun from the WordNet as shown in Fig. 25. Then we replaced the word in our corpus with the corresponding hypernym word. In case there is no hypernym, then we return the same word.

```
def getHypernym(word,pos_type):
      if(pos_type=='noun'  and len(word)>=2
                           and len( wordnet.synsets(word, pos=NOUN))>0):
           if(wordnet.synsets(word, pos=NOUN)[0].hypernym !=None) :
               return (wordnet.synsets(word, pos=NOUN)[0].hypernym[0])

      elif(pos_type=='verb' and len(word)>=2
                            and len( wordnet.synsets(word, pos=VERB))>0):
           if(wordnet.synsets(word, pos=VERB)[0].hypernym !=None) :
               return   (wordnet.synsets(word, pos=VERB)[0].hypernym[0])

          return word

     else:
        return word
```

Fig. 25. An excerpt of the developed Python code to extract Hypernym using the WordNet

Then we performed the experiment on the corpus using the same setting with the linguistic knowledge (i.e. WordNet) and we achieved a less accuracy f-measure of 63.8% compared with 70.0% using ComplementNaïveBayes algorithm as shown in Table 22. F-Measure of using the WordNet with respect to non-using of WordNet

It is worth mentioning that, we used hypernym method, to the best of our knowledge this method is one of the proposed methods besides other widely used methods like (synonyms, antonym) . In our case, the experiments took 11 hours, from 10:00 PM to 09:00 AM to extract all hypernym of words in our corpus, which is too much consuming time. Note that, the processor we ran the experiment on is Intel core i5 460 M with 2.5 GHZ.

[11] Adapted from http://ohmyluna.blogspot.de/2011/01/hypernym-and-hyponym.html

Table 22. F-Measure of using the WordNet with respect to non-using of WordNet

	F-measure		
	NB	SVM	Complement NB
Without WordNet	**62.80%**	**59.0%**	**70.0%**
Using WordNet	57.90%	55.1%	63.8%

***Classification accuracy achieved by the different ML Classifiers* (Two-COIN Corpus case):** As we mentioned earlier, the Two-COIN Corpus is derived from the Seven-COIN Corpus by abstracting into 'COIN' and 'Not-COIN. Therefore, we ran a second round of the experiment where we repeated the same steps as explained in the first round with the Seven-COIN Corpus.

We tested the performance of our model using ten classification algorithms as shown Table 23 and Fig. 26.

Table 23. Accuracy comparison between different classification algorithms

Classification Algorithm	1,2,3 Gram all words		
	Precision	Recall	F-Measure
ComplementNaïveBayes	81.9%	82.0%	**81.9%**
NaïveBayesMutinomialupdatable	81.9%	82.0%	**81.8%**
NaïveBayesUpdatable	70.5%	70.8%	70.5%
NaiveBayes	76.7%	74.5%	74.6%
NaivebayesMultinomial	81.8%	81.9%	**81.8%**
Support Vector Machine (SVM)	75.7%	75.7%	75.7%
Decision Tree J48	65.0%	65.2%	65.1%
Random Forest Tree	73.7%	73.9%	73.7%
KNN K=1	64.2%	52.3%	47.8%
KNN K=2	64.4%	48.7%	40.6%
Simple Logistic	68.2%	68.4%	67.2%
Logistic	67.1%	67.5%	66.5%

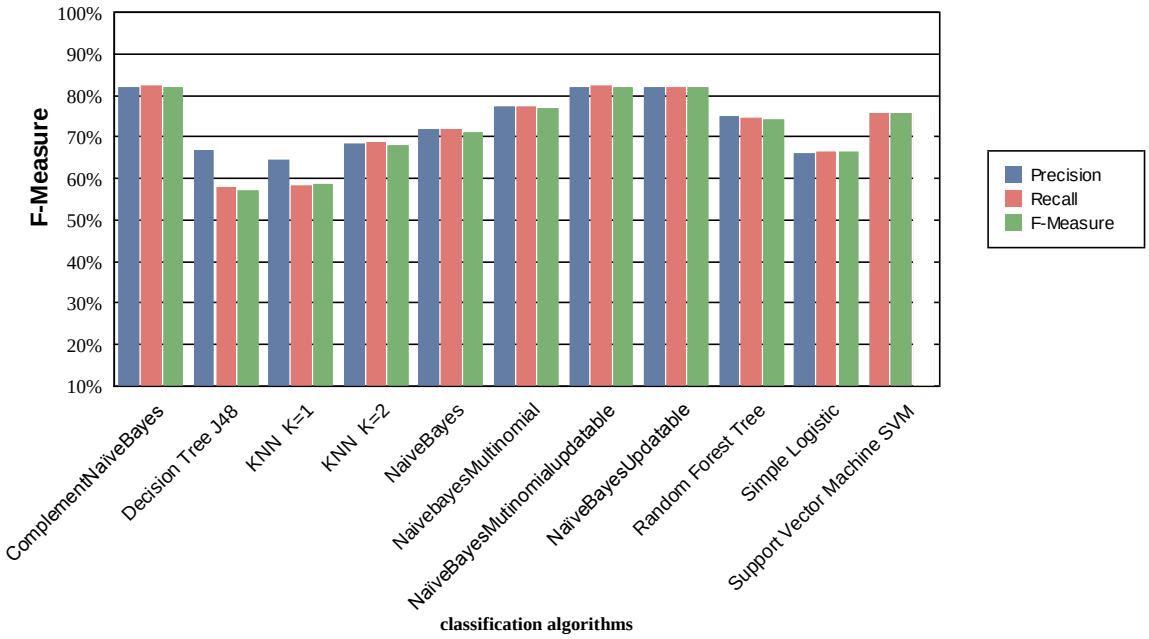


Fig. 26. Accuracy comparison between different classification algorithms

Next, we performed the experiment by applying the classification algorithms on different combination of N-Gram. As expected, the result was similar to the Seven-COIN round. The results revealed an improvement in the accuracy compared to the Seven-COIN classification. That is, we have achieved the best accuracy using 1,2,3-Gram with recall of 82.0%, precision of 81.9% and f-measure of 81.9% by using ComplementNaïveBayes algorithm. In the second place came the NaïveBayesMutinomialupdatable with accuracy recall of 82.0%, precision of 81.9 % and f-measure of 81.8%. These results are much better than what we reported in our published paper [2], in which we got accuracy of f-measure 76.0% using 1,2,3 Gram with NB algorithm. Besides, we compared the performance of the learning algorithms of our model with respect to different combination of N-Gram. The results are shown in Fig. 27 and Fig. 28 respectively.

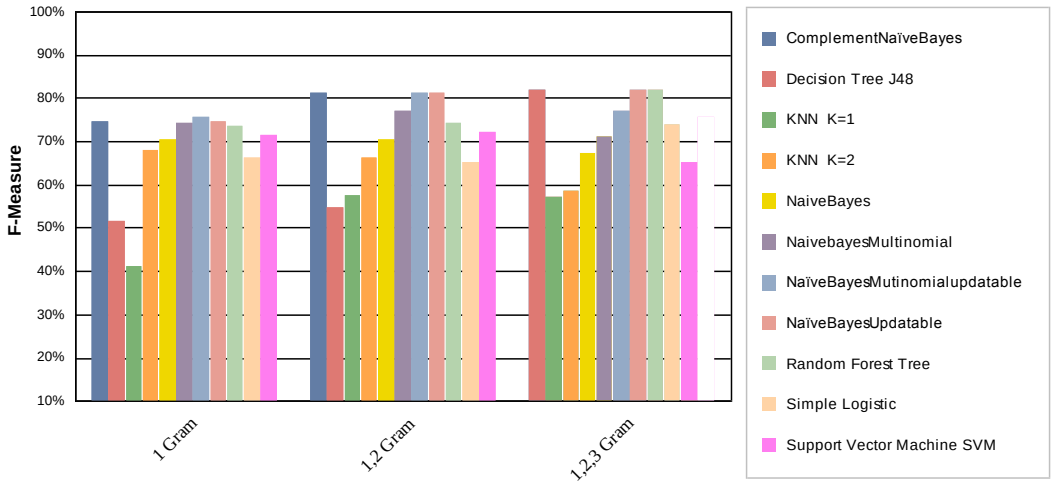


Fig. 27. Algorithms performance via N-Gram

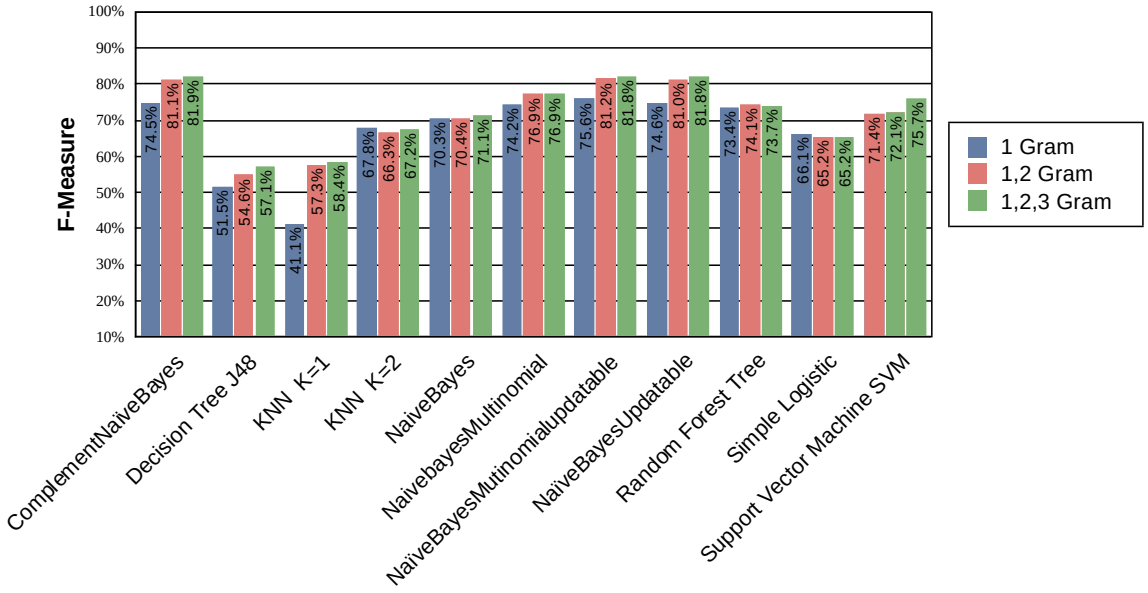


Fig. 28. Classification performance via different N-Gram combinations in Two-COIN corpus

Finally, we observed that, if we limited the classification classes to COIN and Not-COIN, then the f-measure score increases to reach 81.8% instead of 70.0%. Our results are recorded in Table 24 and illustrated in Fig. 29.

Table 24. Accuracy comparision between two different corpora

COINs Type	NaïveBayesMutinomialupdatable (1)	ComplementNaïveBayes (2)
	F-Measure	F-Measure
Seven-Classes	65.4%	**70.0%**
Two-Classes	81.8%	**81.9%**

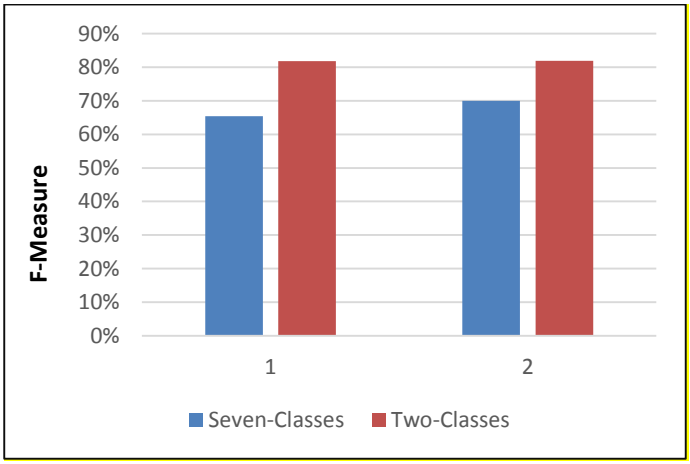


Fig. 29. Accuracy comparision between two different corpora

7 TECHNICAL SUPPORT (A TOOL PROTOTYPE)

To bring our ideas into industry and make it practical, we designed a simple plugin tool so software architects or analysts can benefit from our ML classification ideas and achievements effectively. The tool aim sat making it easy for architects to shape a general perception, which helps them in extracting the conceptual interoperability constraint from API documents automatically. We built a prototype for the tool using web service technologies. Through the tool, the architect selects any piece of text, and requests the COIN type that the sentence could have just in one simple step. Such a service offered by the tool is very easy to use by an architect in terms of usability, and accessibility. It also reduces the needed time and effort to analyze large textual content searching for the COINs.

We implemented our tool prototype as a simple plugin for the Chrome web browser and we call it the *CEP-COIN* (an abbreviation for Classifier Ensemble Plugin–COIN). We use the Java and JavaScript languages for the implementation and we designed the tool with both front-end side and a back-end. Shortly, the front-end side is the user interface (UI), while the back-end one is the core unit of our CEP-COIN.

Principle of work

A client uses CEP-COIN plugin from a web browser to send a piece of text from an API document through an http request to the web server, which hosts and runs the web service. We call this service 'COIN Classifying Service'. Given a sentence, the web service predicts the COIN type by using the machine learning classification model that we introduced in section 6.2. Then, this web service responds to the client request by sending back the result (i.e., the COIN class that the sentence has). A simple overview of the described process flow is shown in Fig. 30

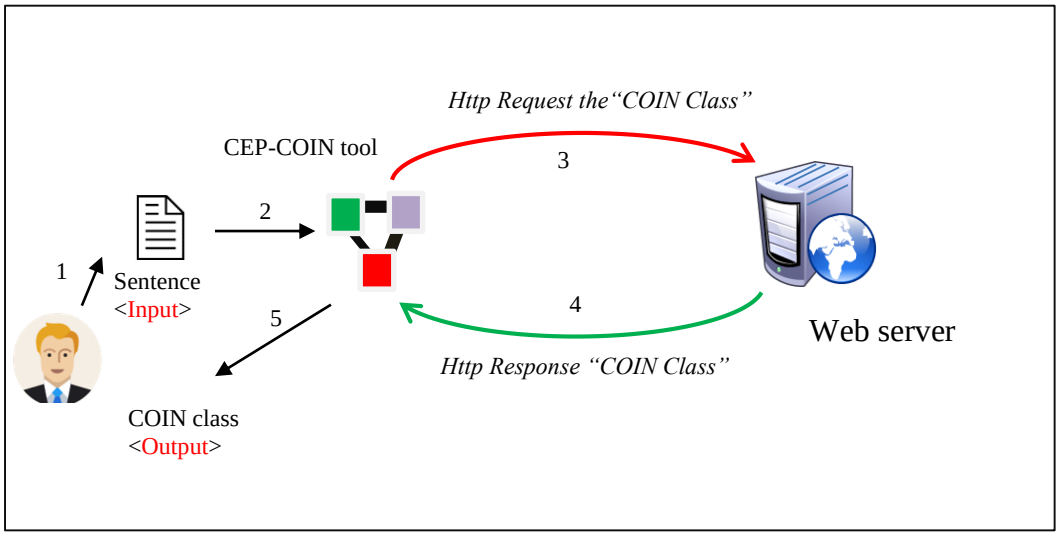


Fig. 30. Process Flow of the CEP-COIN

Using the CEP-COIN Tool

In this section, we explain how software architect can install the tool prototype and how to use its functionalities that are easy to follow.

CEP-COIN Installation. CEP-COIN prototype tool is very easy to install as Add-In for Chrome (we have tested it Chrome version 49"). The software architect can import this tool from Chrome settings through the extension menu. From this menu the architect would load the unpacked extension then select the folder of the CEP-COIN as shown in Fig. 31

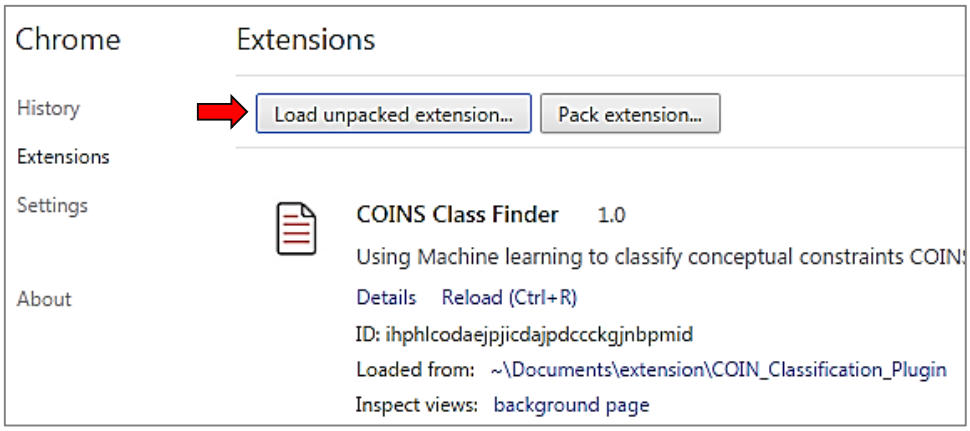


Fig. 31. Installing CEP-COIN Tool

Using CEP-COIN functionalities. The CEP-COIN service is offered in three different forms:

1. *Using CEP-COIN Context Menu.*
2. *Using CEP-COIN Plugin GUI.*
3. *Using CEP-COIN Web Page.*

1. *Using CEP-COIN Context Menu*: Once the CEP-COIN plugin is installed, you can select any text on the webpage of an API document, then right click on the mouse and select COIN Classification. Immediately, a popup window will appear with a COIN-Type as shown in Fig. 32. This context menu is very friendly if the user want to classify any text by selection, but if he want to write his own text to classify, then he should use another functionality of CEP-COIN plugin as it is explained next.

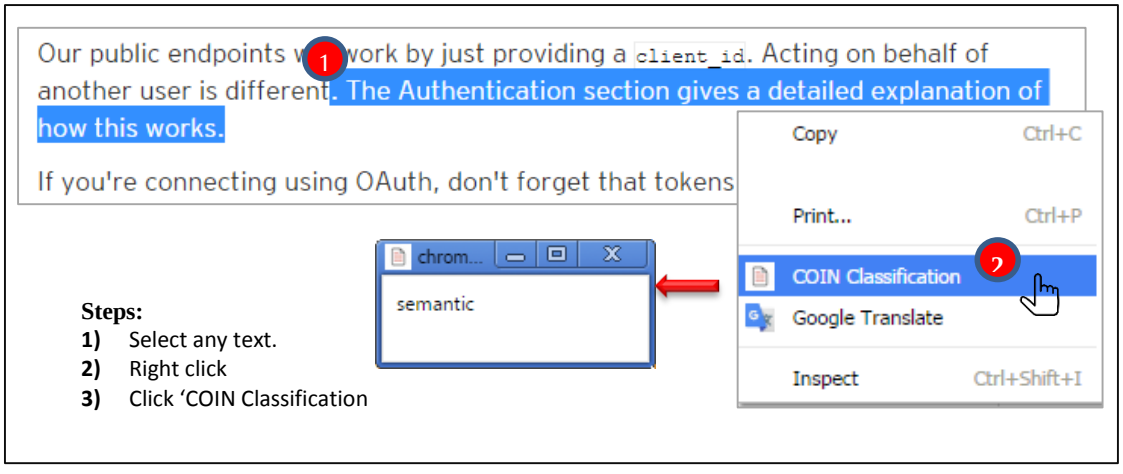


Fig. 32. Using CEP-COIN Tool from context menu

2. *Using CEP-COIN Plugin GUI*: In this case, the software architect can enter any sentence without leaving the webpage of the API document. The GUI of this form has a very simple interface. With this GUI, the software architect can write any text and by pressing the button "Get COIN Class" then he can get the corresponding COINs class in the result field as shown in Fig. 33.

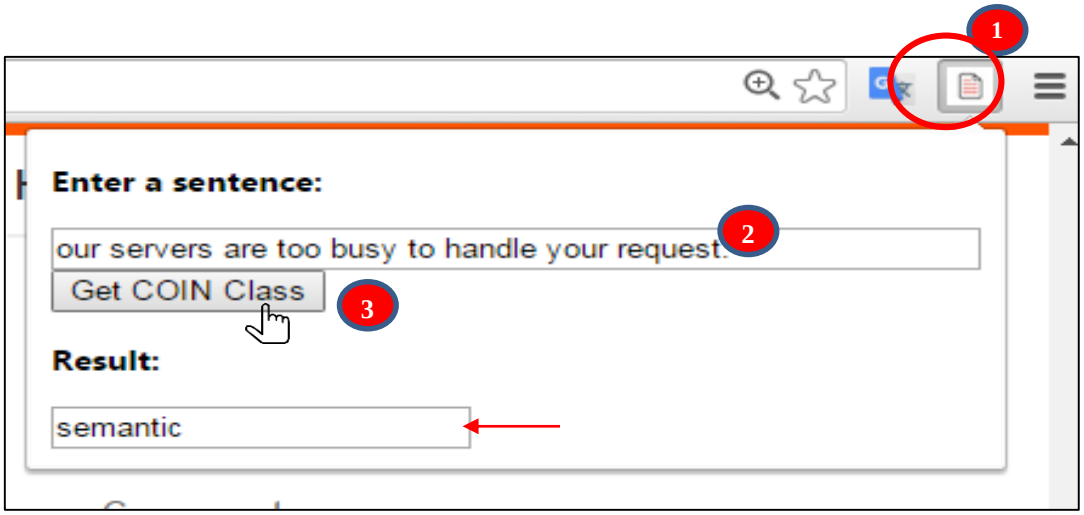


Fig. 33. Using CEP-COIN from Plugin GUI

3. *Using CEP-COIN Web Page*: We developed a simple JSP page, which takes a textual sentence as an input and returns the COIN type as an output. This service differs from the previous menu services in terms of enabling the software architect to write any sentence without need to install the CEP-COIN. In addition, he can use this web page from any web browser. See Fig. 34.

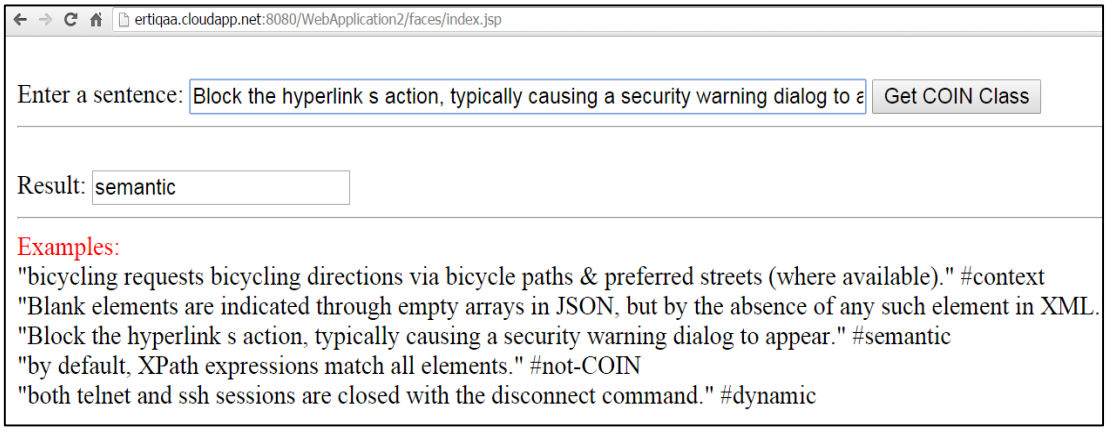


Fig. 34. Using CEP-COIN service from the JSP page

Technical Support (A Tool Prototype)

CEP-COIN Architecture

The Architecture of our CEP-COIN tool consists of two separated components as illustrated in Fig. 35

1- Front-End component: developed using JavaScript.

This component consist of one layer, which is a User Interface (UI) layer to provide a graphical user interaction (i.e., GUI). The software architect sends an http request from the browser via CEP-COIN tool. The tool communicates with the server side directly using Ajax. Then, CEP-COIN passes the result back to the browser using JavaScript & JQuery.

2- Back-End component: developed using Java.

There are three different layers. First layer is a business layer, which is responsible for finding the service location. Then, it requests the service from a web services, and passing the sentence to the classification service, which in turn sends a Simple Object Access Protocol (SOAP) [78], Description Language (WSDL) [79] [80] file that contains:

1- The abstract service interface definition.
2- How to interact with service.
3- The location of the service.

The second layer is the data access layer, which is responsible for creating an instance from the classification model and for applying the text classification algorithm to find out the corresponding COIN class for the sent sentence.

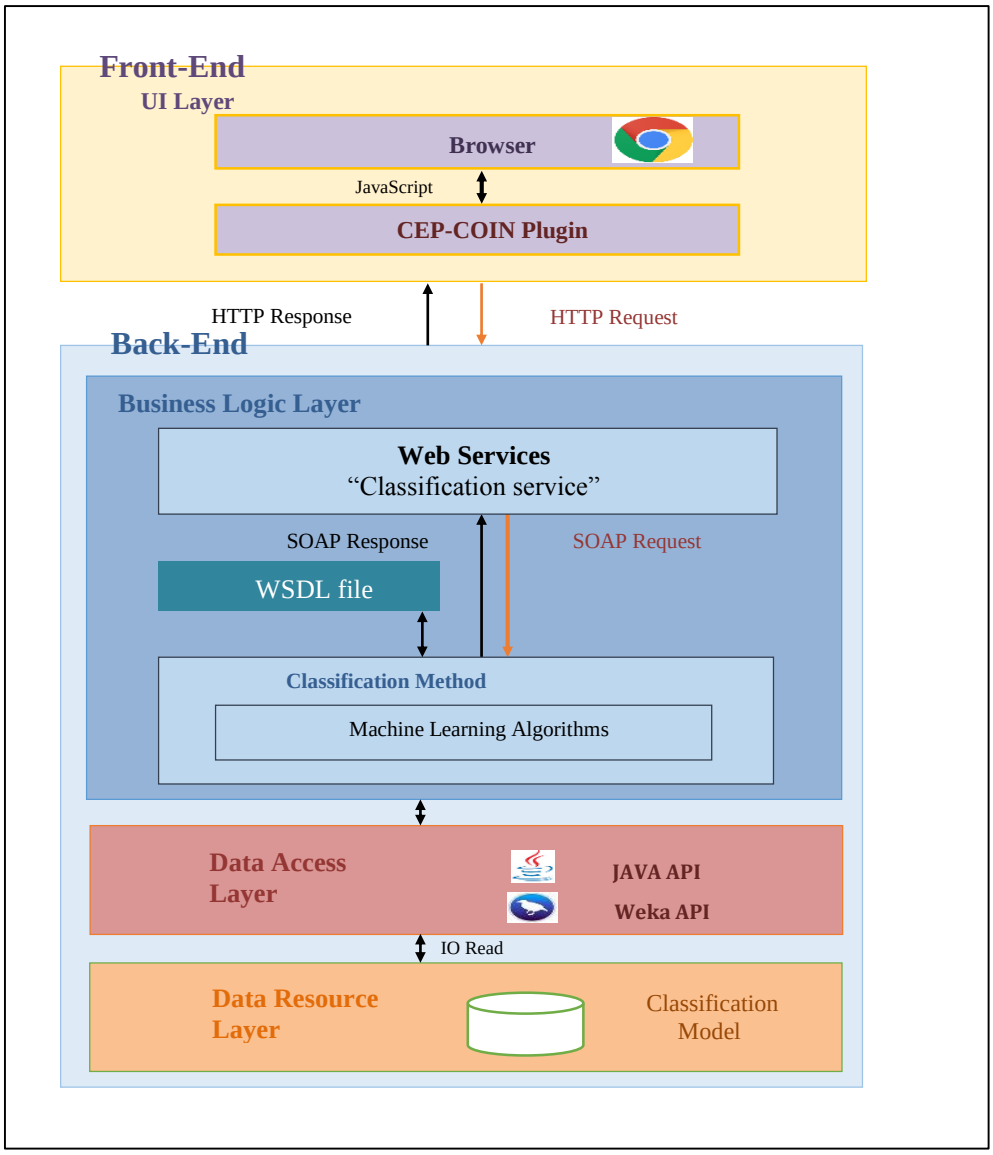


Fig. 35. Architecture of the CEP-COIN

CEP-COIN Implementation

We created two independent implementation parts: one for the client component (i.e., the representation layer) using Ajax code to request our web service, and the second for the server component (which responds to the client request and retrieves the COINs class using Java language. Next, we explain this implementation part in more details.

Client Component (Front-end) implementation. In Fig. 36, we show an excerpt of the JavaScript[12] and JQuery[13] code for requesting the COIN classification from server side. And this implementation is used in all of the three forms of the client services (Context Menu, Plugin-GUI and Web form) as stated above.

[12] https://www.w3.org/community/webed/wiki/What_can_you_do_with_JavaScript
[13] https://jquery.com/

```
function getTheCOINType [2]()
{
 var sentence=getParameterByName('textCOIN');
 var posting =
   $.post("localhost:8080/WebApplication2/faces/action.jsp",
          {textCOIN: sentence});
           posting.done(function (data) {
                 document.write(data.trim()); }
                   });
}
```

Fig. 36. JQuery for requesting the Classification Service

The implementation defines a simple http request using post command. Hence, the command requires the web service URL and retrieves the COIN type. Then, it writes back the result immediately on the webpage od the browser document.

Server Side (Back-end) implementation. Here we depict the three main processes that are running on the server side of our tool. Note that we used GlassFish Server 4.1.1[14] for deploying and running the CEP-COIN web service.

The core functionalities of our server side are performed in the business logic layer. There are three essential processes to retrieve the COIN type. These three processes are (load ML model, classify, and response to client request) these processes are shown in Fig. 37

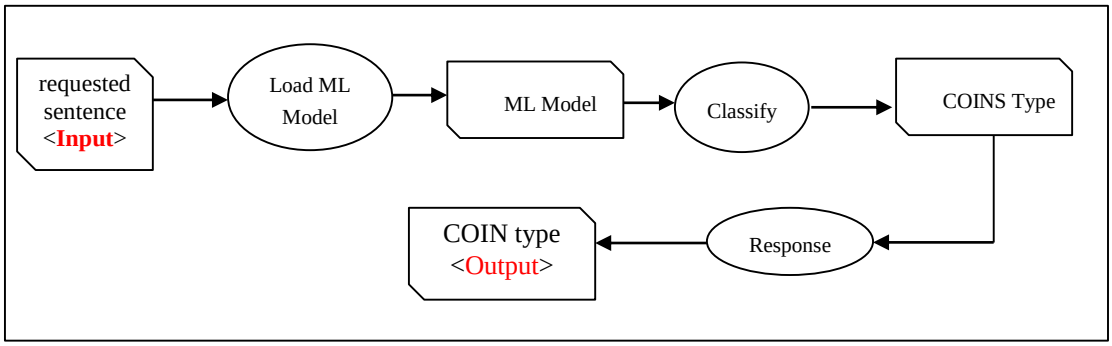


Fig. 37. Server Side Processes Flow

These processes are implemented completely in pure Java. We used NetBeans 8.1 as an IDE and Java EE [15] with JDK 1.8 [16] for developing the server side code. For loading ML libraries and algorithms, we used Weka API [81]. The reason to use Java API and Weka API. Therefore, all Weka resources and packages are totally Java callable, reliable, stable and compatible.

[14] https://glassfish.java.net/
[15] http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html
[16] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

We implemented a method called loadModel to load the ML model from the server drive into the server memory using Java API. In **Fig. 38** we show the complete implementation of this method.

```
public void loadModel(String ModelfileName) {
 try {
        FileInputStream FS=new FileInputStream(ModelfileName);
        ObjectInputStream inModel = new ObjectInputStream(FS);
          Object ClassifierObject = inModel.readObject();
        classifier = (FilteredClassifier) ClassifierObject;
          inModel.close();
        System.out.println("Loaded model: " + ModelfileName );
         }
 catch (Exception ex)
       {
          System.out.println("Problem:" + ModelfileName );
       }
 }
```

Fig. 38. loadModel method implementation

The Object *ObjectInputStream* holds the model into the memory. Then, we defined a Weka API object classifier of type *FilteredClassifier*. This object is responsible for the classification process, which holds the *ML algorithm* that s used in the training phase towards finding out the appropriate COIN type.

Tool Performance

We evaluated the tool performance in terms of time spent to classify textual sentences. This evaluating was performed by classifying 10 sentences with average of 20 words per a sentence. The average time required to classify each sentence is 1.0 second. Remember that the tool effectiveness in classifying the sentences in identical to our described achievements in section 6.2 (i.e., recall =70.2%, precision =70.4% and f-measure =70.0%). We have tested our tool prototype on the platform configuration shown in Table 25

Table 25. Platform Cofiguration

	CPU		RAM	Bus Speed
	Brand	Speed		
Server configuration	Intel Xeon E5	2.2 GHZ	EDO 3.5 GB	200 MHZ
Client configuration	Intel core i5 460 M	2.5 GHZ	DDR3 4GB	133 MHZ

Future work and development

Since our tool is currently implemented as a prototype, it is designed only to operationalize our contributed ML classification model to show its applicability. That is, we just pointed out the potential practical advantages that our automatic classification ideas would bring to software architects and analysts in retrieve the required COINs from verbose of text in API documentation.

For research time restrictions, we limited our plugin implementation to work only on one example of web browsers (we chose Chrome because of its popularity). In future, we are planning to develop CEP-COIN tool to work on other web browsers like Internet Explorer (IE), Firefox. On the other hand, we aim at extending the capabilities of our CEP-COIN tool to classify multiple sentences or even a complete document at once instead of classifying just a single sentence at once. In this case, CEP-COIN returns a list of COINs instead of single COINs as in our current version.

Additionally, We also intend to benefit from users feedback on the automatic classification results of the new sentences they send to the tool (e.g., to offer an optional report for their agreement on the provided classification or their disagreement and suggested classification). Keeping record of such data could be used to enrich our corpus content, and consequently we could use them in improving our classification model accuracy through continuous learnning.

Moreover, we will support CEP-COIN tool with extra features like reusability. The idea of that is by tracking the users' recorded data in our DB system to provide them with reports of the classified documents. These reports can be formalized in different formats like (excel, xml, text, doc, etc.) to be incorporated in different analysis systems in order to save time and efforts by reusing it.

Furthermore, we are planning to support the architects and software engineers with different types of statistics and recommendations about the conceptual interoperability constraints that their analyzed document

have. For example, the CEP-COIN can return statatistics about sentences and their COINs type distribution for a complete document.

8 RESEARCH CHALLENGES

Despite many of the challenges that we encountered, we learned something new and gained more skills and experiences. Due to the clear goal and research plan, we continued our work and confront these difficulties in various ways until we overcame many of them.

Along our researching which we were exploring the capabilities of using the NLP and ML technologies to automate the extraction of the COINs from API documents we faced many challenges that we describe bellow along with our solutions as well.

8.1 Lack of labeled data

For using ML classification techniques, we needed a ground truth, which must include as much as possible of already classified or labeled textual sentences from API documentations according to the COIN model. As such, requirement did not exist; we had to build it by ourselves at the beginning of our research. This manual task was very tedious and time consuming as we analyzed and classified each sentence from the API documents we selected (see chapter 4). In this task, we encountered the following three sub-challenges:

1. Selecting representative API documentation cases.

The first obstacle we faced in our research was selecting the API documentation that meet our research requirements (i.e., API documents that contain conceptual information and not only technical ones, and that are diversified and widely used). This required us to define appropriate selection criteria that directs our search and nomination for included documents in our research (see the research methodology chapter for details).

2. Extracting relevant content from the selected API documentations

After we finished reading the selected API documentations, we faced an obstacle of content representation, in which we found the conceptual information was not pure, but rather it contained technical noise (e.g., sentences contain natural language text along with code, symbols, tags, etc.). This noise was unwanted as it was irrelevant to our interest in the conceptual information only. Not only such a mix confuses the machine, but also worse, it gives the human reader hard times to interpret the text. For our work, it was necessary to clean and organize the textual content to facilitate the pattern recognition and keyword extraction for both approaches we described at section (6.1 and 6.2). In order to perform that, we first implemented a simple PHP code using Simple HTML DOM Parser library to filter out the API documentation from noise. (i.e., headers, images, etc.). However, we found that, these tools were not sufficient and they were poor to meet the required purpose. For example, the available tools are not able to filter out the code-content, which is so frequent in many parts of the API documentation. These tools also do not support sentence tokenization into separated lines. Therefore, we decided to extend our manual efforts along with these tools to clean the content. Although this manual cleaning resulted in absolutely better content, it consumed too much time and mental effort to check sentence by sentence and word by word.

3. Sentences with poor description.

Classifying the sentences of the textual content in the API documents required linguistic skill in to understand the sentence meaning and classify according to the seven COIN classes. We needed to identify the message delivered by each sentence, which sometimes was not trivial due to multiple messages in one sentence. That is, some sentences had more than one COIN class within it. In addition, the document had some grammar mistakes and ambiguous words that made it harder to interpret sentences. Actually, it was a challenging task especially at the beginning of the research.

However, we got over this obstacle by training and practicing on the included cases and by the reviewing and discussing session that we had as described in chapter 5 section 4.

4. Aggregating the collected data from across the cases.

Another challenge that we faced was to aggregate and organize the collected data from the distinctive cases each with its own style. It was not trivial to put such incoherent data with different formats and structures. This task required additional time and many attempts to figure out a suitable procedure to gather them into one data container. The result of our effort produced one consistent database accommodating clean and organized sentences for later usage within automatic ML classifiers.

8.2 Identifying cross-case COINs identification rules.

One of the most challenging phases of our research was to come up with a set of effective extraction rules and features to be fed up to the machine learning algorithms in our first classification approach (i.e., rule-based classification). It was a tedious manual investigation that consumed a lot of our research time that took approximately 35% of our effort. In particular, it was not a trivial task to identify the representative features for each COIN class from the sentences, which contained instances of the class manually. That is, for each phrase independently, we looked after the patterns no matter how diverse they are, then, we selected only the appropriate ones and neglected the insignificant ones in terms of number of occurrences. We tried different NLP techniques (i.e., word tokenizing, stemming, Stopwords elimination, N-Gram) as aids for us to overcome these obstacles and get better results.

8.3 Understanding the semantics and contexts.

This is considered to be the most troublesome among the issues that faces the researchers in the artificial intelligence area. Thus, we spent part of our research time reading related papers and scientific article regarding this issue, and we tried different approaches to maintain this problem. Finally, we settled on using N-Gram techniques and WordNet [75] to cover meaning and context issue. For example if we consider two consecutive words together by using N-Gram with N=2, then this helps us to preserve the word in a context, for example: two words like “perform jobs”, “user name” allows the classification model to recognize the context of these two words together instead of using a single word as a feature. This also was shown in our experiment, where the prediction accuracy is better when we use N-Gram with N between 1 and 3.

8.4 Limitation of resources.

Working with text processing technologies has a very high consumption of resources (i.e., memory and CPU speed). In our experiments, there are two main phases require high-performance and hardware resources. First phase (Rule-based classification) and the second phase (BOWs-based classification, in which we extracted the features by applying NLP-Pipeline. As mentioned previously, our corpus size is 2283 COINs, hence, the process of extracting features requires high hardware resources. In particular, this required a big size of memory to accommodate the representation of the mathematical model resulting from this process. In fact, it was a very difficult situation, almost ending with memory overflow after performing many experiments. Then we restart again Weka many times to perform the experiment from scratch. To overcome this problem, we recorded the results of the experiment after each classification process manually to assure that at least we have the measures of the last experiments. And we faced the same problem in the second phase (BOWs). Finally, we have solved this problem partially by configuring RunWeka.ini and then set parameter (*maxheap* = 10024M instead of 1024M), then it works longer time than before but then again it shut down after some period of time.

In fact, the consumption of resources is a common problem in machine learning and is expected to occur frequently. Although, it was not easy to run the experiments with such resource limitations, we did not have other alternatives.

9 OVERALL DISCUSSION AND CONCLUSION

The main goal in this thesis is to support software analyst and architect in identifying the conceptual interoperability constraints, COINs, automatically in the text of API documents. In our research, we followed a systematic empirical-based methodology that has two main advantages: (1) tracing and verifying documented results among two research phases, and (2) repeating the defined activities in our protocol by other researchers in order to address researcher bias threat to validity. To achieve our research goals, we explored the potentials of using ML and NLP, built a classification model, and conducted explorative experiments. Here we offer a summarized answer for our two main research questions:

RQ1: What are the observed patterns in specifying the conceptual interoperability constraints COINs in the NL text of API documentation?

Answer: Through our observation during the classification process, we found some patterns. For instance, the sentences of type Not-COIN (which represent about 34.3% of the total COINs in the corpus) contain significantly many technical keywords, and they include a text explaining technical and practical details about services/systems. For COINs of type dynamic (which comprise about 28.3%), have some specific patterns. For example, we found that, they include a description of the activity or the flow of operations. Additionally, they contain many conditional sentences such as (IF/Then) sentences that are used to clarify the expected results of the specific input. For COINs of type semantic (which constitute approximately 27.4%), have some distinctive patterns. For example, the sentences describe the purpose or goal of the service or activity. On the other hand, there are more than 48% of the semantic COINs that contain special terms. Thus, we classified them into three lists: 1) Output/Input verbs. 2) Supporting verbs. 3) Admission verbs. As shown in Appendix (**Table 34**, **Table 35** and **Table 36**).

We have also achieved additional related findings in this matter that we formulate in the following questions and answers:

*Question A***:** Where could we find the COINs in the API documents, (i.e., in which sections or paragraphs)?

Answer: To the best of our knowledge and according to our reviews of the API documents, we conclude that, the COINs fundamentally exist in specific paragraphs such as abstract - Introduction - Overview - Conclusion – Summary. In light of the fact that these paragraphs are rich within concepts and abstract level of knowledge that are needed for analysts and architects to cover within their conceptual interoperability analysis. However, COINs rarely exist in the paragraphs or sections that describe technical information such as method description and code examples.

Question B: To which extent are API documents similar in terms of structure and format?

Answer: We have noticed that there are differences in the composition of the API documentation. For instance, some of these documents are subject to the special format and structure. Thus, we found that some API documents such as GoogleMaps is technical oriented and serves developers rather than software archi-

tects. Therefore, mining COINs in these type of documents would be tough. While other documents like AppleWatch, SoundCloud and Eclipse serve developers and architects as they have an organized structure and format of the information with proper balance between concepts and technicalities.

Regarding to the second defined research question for our research; we summarize its answer as below.

RQ2: How effective and efficient would it be to use Natural Language Processing (NLP) along with Machine Learning (ML) technologies to automate the extraction of COINs from the text in API documentations?

Answer: Initially, when using ComplementNaïveBayes algorithm for classifying Seven-COIN and Two-COIN. We got encouraging results with F-measure of 70.0%, which is a quite good result. On the other hand, when classifying only Two-COIN, the results get better with F-measure of 81.9%, which is about 11.9% higher than the accuracy of classifying Seven-COIN. It is self-evident because classifying multi-classes requires more holistic data, while classifying data of less classes will show better results if we have the same volume of data.

Similar as the first research question we have achieved further findings related to RQ2 that we formulate in the following questions and answer.

Question C: What is the best text classification algorithm for identifying the COINs in API documents?

Answer: In fact, for each problem domain, there is a different text classification algorithm. In particular, after our experiments that we have performed, we came up with that, *ComplementNaiveBayes* achieved the best performance in terms of accuracy in classifying the COINs.

Question D: How can the classification model be used in practice?

Answer: In fact, a simple and practical mechanism to use such model is through utilizing a plugin tool that works through a web browser. More specifically, we have developed the classification model using Weka API and Java API together. However, the user interface is designed as a chrome plugin, which can be used easily and instantly, and again the accuracy of our tool depends on the accuracy of the model that we have designed.

Question E: What can improve efficiency?

Answer: We expect that classifying more sentences from other API documents and adding them to the ground truth (i.e. COIN Corpus) will increase the effectiveness of our proposed ML classification model.

10 FUTURE WORK

There is a window for more enhancements that can be performed in the future to our research and below we mention some of the most important ones.

1. Improving the performance of our ML classification model. Obviously, this requires us to train the classifier on more training data set, which in turn requires us extra effort and time to classify more sentences of API documents into the COINs' classes. As the volume of data plays a fundamental role in increasing the efficiency of the automated classifier [28].
2. The use of deep learning techniques: which is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by utilizing multiple processing layers, with complex structures [82] [83]. Which has proven their effectiveness and superiority in the field of classification of texts [84]. In Addition, using such techniques may help in comprehending the textual content, which is one of the most serious challenges in the ML area. However, it is relatively difficult to apply such techniques because of their need for high hardware resources and equipment with high specifications (e.g., high processor speed and memory size).
3. Preparing our tool for industrial usage by increasing its efficiency and supporting its work in different platforms (i.e., to be compatible with different web browsers and IDE environments).
4. Extending our tool with more features, so that it is able to find all COINs in an API document by segmenting it into sentences first, and then by classifying each sentence separately into the COIN classes.
5. Improving the current ML classification model to do self-learning through the feedback collected from users such as engineers and architects. This is expected to improve the accuracy of the classifier due to the added knowledge by the experts.

11 APPENDIX

11.1 Tables

A. The extraction data sheet used for collecting data.

Table 26. The extraction data sheet

Sentence id	Sentence	COINS Type	API Document

B. Part of Speech (POS) tagging [85]

Table 27. Part-of-speech tags used in the Penn Treebank[17] [86]

#	Tag	Description
1	CC	Coordinating conjunction
2	CD	Cardinal number
3	DT	Determiner
4	EX	Existential *there*
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, comparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NNP	Proper noun, singular
15	NNPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PRP	Personal pronoun
19	PRP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	Particle
24	SYM	Symbol
25	TO	to
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
29	VBG	Verb, gerund or present participle
30	VBN	Verb, past participle
31	VBP	Verb, non-3rd person singular present
32	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34	WP	Wh-pronoun
35	WP$	Possessive wh-pronoun
36	WRB	Wh-adverb

[17] Penn Treebank is a large corpus, approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, over 2 million words of text parsed for predicate argument structure, and 1.6 million words of transcribed spoken text annotated for speech disfluencies. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.8216 [86].

C. Top frequently terms

Table 28. Top 30 frequently terms used in "Dynamic" class

#	Co-occurrence	keyword
1	111	job
2	94	user
3	80	app
4	51	interface
5	46	client
6	46	create
7	46	method
8	45	notification
9	44	use
10	42	request
11	41	access
12	40	api
13	38	object
14	38	plug
15	38	value
16	36	watch-chain
17	35	specify
18	34	lock
19	33	time-share
20	32	schedule
21	30	include
22	30	set-up
23	29	run
24	28	property
25	27	application
26	27	data
27	27	platform
28	26	code
29	26	default
30	26	note

Table 29. Top 30 frequently terms used in "Semantic" class

#	Co-occurrence	keyword
1	84	user
2	78	plug
3	67	app
4	59	provide
5	50	platform
6	48	interface
7	46	job
8	45	extension
9	43	content
10	43	notification
11	40	application
12	39	use
13	38	work-in
14	36	person
15	36	return
16	33	display
17	32	request
18	30	api
19	30	object
20	29	allow
21	29	define
22	29	support
23	29	type
24	28	result
25	28	system
26	28	watch-chain
27	26	create
28	25	file
29	24	client
30	24	method

Table 30. Top 30 frequently terms used in "Structure" class

#	Co-occurrence	keyword
1	20	interface
2	18	content
3	17	app
4	17	contain
5	16	collection
6	16	type
7	14	person
8	13	include
9	13	object
10	12	plug
11	11	file
12	11	platform
13	10	implement
14	10	property
15	9	bundle
16	9	user
17	8	class
18	7	application
19	7	controller
20	7	distribution
21	7	extension
22	7	watch-chain
23	6	data
24	6	eclipse
25	6	note
26	6	represent
27	6	separate
28	6	work-in
29	5	button
30	5	create

Table 31. Top 30 frequently terms used in "Syntax" class

#	Co-occurrence	keyword
1	15	calculate
2	15	route
3	13	indicate
4	10	specify
5	9	user
6	8	prefer
7	7	conversation
8	6	avoid
9	6	direction
10	6	transit
11	5	person
12	5	travel
13	4	character
14	4	element
15	4	mode
16	4	set-up
17	4	time-share
18	4	xml
19	3	activity
20	3	app
21	3	application
22	3	call
23	3	collection
24	3	eclipse
25	3	information
26	3	key
27	3	language
28	3	manifest
29	3	platform
30	3	plug

Table 32. Top 30 frequently terms used in "Quality" class

#	Co-occurrence	keyword
1	6	user
2	4	direction
3	4	provide
4	3	access
5	3	api
6	3	note
7	3	result
8	3	token
9	2	application
10	2	availability
11	2	bicycle
12	2	cancel
13	2	cause
14	2	content
15	2	display
16	2	fail
17	2	include
18	2	integrate
19	2	javascript
20	2	language
21	2	lead-in
22	2	malicious
23	2	match
24	2	oauth
25	2	optimize
26	2	parse
27	2	performance
28	2	platform
29	2	plug
30	2	presence

Table 33. Action Verbes

#	Co-occurrence	Verb
1	46	create
2	44	use
3	42	request
4	41	access
5	38	plug
6	34	lock
7	30	include
8	30	set-up
9	29	run
10	25	start
11	22	call-up
12	20	redirect
13	19	register
14	19	track
15	18	run-up
16	17	add
17	17	update
18	15	acquire
19	15	avoid
20	14	call
21	13	perform
22	12	return
23	12	store
24	11	implement
25	11	install
26	10	build-up
27	10	launch
28	10	receive
29	10	search
30	9	connect
31	9	determine
32	9	flow
33	9	list
34	9	load
35	8	execute
36	8	initiate
37	8	select
38	8	send
49	8	share

Table 34. Output/Input verbs

#	Verb
1	access
2	display
3	download
4	fetch
5	notify
6	read
7	recall
8	receive
9	recover
10	response
11	retrieve
12	return
13	select
14	send
15	share
16	submit
17	upload

Table 35. Supporting verbs

#	Verb
1	support
2	provide
3	Suggest
4	give
5	propose

Table 36. Admission verbs

#	Verb
1	allow
2	enable
3	admit
4	let
5	give
6	grant
7	permit
8	facilitate
9	authorize
10	prevent
11	stop
12	avoid

Table 37. Defined Stopwords

#	Stopword
1	!
2	#
3	#for
4	$
5	$
6	*
7	/
8	@
9	+
10	a
11	about
12	above
13	all
14	also
15	an
16	and
17	another
18	any
19	any
20	anyone
21	are
22	as
23	at
24	b
25	be
26	but
27	by
28	c
29	etc
30	everyone
31	here
32	in
33	into
34	is
35	it
36	its
37	like
38	no
39	not
40	now
41	of
42	often
43	on
44	only

45	or
46	other
47	others
48	our
49	over
50	re
51	s
52	s
53	such
54	t
55	that
56	the
57	their
58	them
59	then
60	there
61	therefore
62	these
63	they
64	this
65	those
66	to
67	up
68	us
69	was
70	we
71	were
72	which
73	who
74	will
75	with
76	within
77	x
78	y
79	yours
80	z

12 Bibliography

[1] H. Abukwaik , M. Naab and D. Rombach, "A Proactive Support for Conceptual Interoperability Analysis in Software Systems," in *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, Montreal, 2015.

[2] H. Abukwaik, M. Abujayyab, S. R. Humayoun and D. Rombach, "Extracting Conceptual Interoperability Constraints from API Documentation using Machine Learning," in *The 38th International Conference on Software Engineering (ICSE 2016) Companion*, TX,USA, 2016.

[3] IEEE standard computer dictionary. A compilation of IEEE standard computer glossaries - IEEE Std 610, Library of Congress Catalog Number 90-086306, 1990.

[4] ISO/IEC 2382:2015(en)- Information technology -Vocabulary - Terms and definitions.

[5] " C4ISR Interoperability Workig Group: Levels of information systems interoperability (lI-SI). Technical Report, Department of Defense (1998)".

[6] B. J. Powers, "A multi-agent architecture for NATO network enabled capabilities: enabling semantic interoperability in dynamic environments (NC3A RD-2376)," in *Service-Oriented Computing: Agents, Semantics, and Engineering*, Springer, 2008, pp. 93-103.

[7] "Extending the levels of conceptual interoperability model," in *Proceedings IEEE Summer Computer Simulation Conference. IEEE CS Press (2005)* , 2005.

[8] H. Abukwaik, D. Taibi and D. Rombach, "Interoperability-Related Architectural Problems and Solutions in Information Systems: A Scoping Study," in *Software Architecture: 8th European Conference, ECSA 2014*, Vienna, 2014.

[9] P. Jackson and I. Moulinier , Natural Language Processing for Online Applications: Text retrieval, extraction and categorization - Second revised edition (Natural Language Processing), 2nd ed., John Benjamins Publishing Company, 2007, pp. 2-3.

[10] L. Shane and H. Marcus, "A Collection of Definitions of Intelligence," in *Proceedings of the 2007 Conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006*, 2007.

[11] R. Grishman, Computational linguistics: an introduction, Cambridge University Press, 1986, pp. 4-5.

[12] D. Marneffe and C. Manning, *Stanford typed dependencies manual*, 2008.

[13] Manning and D. K. Christopher, "Natural Language Parsing," in *Advances in Neural Information Processing Systems 15: Proceedings of the 2002 Conference*, 2003.

[14] D. Marneffe, M. Catherine, MacCartney, Bill , Manning and Christopher , "Generating typed dependency parses from phrase structure parses," in *Proceedings of LREC*, vol. 6, 2006, pp. 449-454.

[15] K. Madani, A. D. Correia, A. Rosa and J. Filipe, Studies in Computational Intelligence, vol. 577, Springer; 2015 edition (October 16, 2014), pp. 3-5.

[16] J. S. Gero and S. Hanna, Design Computing and Cognition '14, Springer; 2015 edition (April 17, 2015), 2015, p. 477.

[17] C. Bird, T. Menzies and T. Zimmermann, The Art and Science of Analyzing Software Data, Morgan Kaufmann; 1 edition (September 15, 2015), pp. 499-500.

[18] "Stanford Dependencies," The Stanford Natural Language Processing Group, 03 April 2016. [Online]. Available: http://nlp.stanford.edu/software/stanford-dependencies.shtml. [Accessed 03 April 2016].

[19] F. L. Gaol and Q. V. Nguyen, "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, Stroudsburg, PA: Association for Computational Linguistics, 2005, pp. 363-370.

[20] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms, 1st ed., Cambridge University Press; 1 edition (May 19, 2014).

[21] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms, 1st ed., Cambridge University Press, 2014.

[22] M. Maharasi and N. A. Sophia, "A Survey of Text Categorization and its Various," *(IJCSIT) International Journal of Computer Science and Information Technologies,* vol. 6, no. 3, 2015.

[23] S. Fabrizio, "Text Categorization," 2005.

[24] K. P. Murphy, "Naive bayes classifiers," *University of British Columbia,* 2006.

[25] Leung and K. Ming, "Naive bayesian classifier," *Polytechnic University Department of Computer Science/Finance and Risk Engineering,* 2007.

[26] H. N. Vladimir N. Vapnik AT&T Bell Labs, The Nature of Statistical Learning Theory, New York, NY: Springer-Verlag New York, Inc., 1995.

[27] Tong, Simon and Koller and Daphne, "Support Vector Machine Active Learning with Applications to Text Classification," *J. Mach. Learn. Res.,* vol. 2, pp. 45--66, 2002.

[28] M. Banko and E. Brill, "Scaling to very very large corpora for natural language disambiguation," in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, Toulouse,France, 2001.

[29] C. Silva and B. Ribeiro, Inductive Inference for Large Scale Text Classification: Kernel Approaches and Techniques, Springer, 2009, pp. 26-27.

[30] A. Gelbukh, F. C. Espinoza and S. N. Galicia-Haro, "Feature Selection Based on Sampling and C4.5 Algorithm to Improve the Quality of Text Classification Using Naïve Bayes," in *Human-Inspired Computing and its Applications: 13th Mexican International Conference on Artificial Intelligence, MICAI2014, Tuxtla Gutiérrez, Mexico, November 16-22, 2014. Proceedings, Part 1*, Mexico.

[31] R. S. Kumaran, K. Narayanan and J. N. Gowdy, "Language modeling using independent component analysis for automatic speech recognition," in *Signal Processing Conference, 2005 13th European*, IEEE, 2005, pp. 1--4.

[32] W. Chu and T. Y. Lin, Foundations and Advances in Data Mining (Studies in Fuzziness and Soft Computing), Springer; 2005 edition (October 26, 2005), pp. 225-226.

[33] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for IDF," *Journal of Documentation,* vol. 60, p. 2004, 2004.

[34] M. Radovanović and M. Ivanović, "Text mining: Approaches and applications," *Novi Sad J. Math,* vol. 38, no. 3, pp. 227-234, 2008.

[35] M. J. Michael P. Oakes, Ed., Quantitative Methods in Corpus-Based Translation Studies: A Practical Guide to Descriptive Translation Research (Studies in Corpus Linguistics), John Benjamins Publishing Company (March 20, 2012), 2012.

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111-3119.

[37] "Feature selection," Stanford University, [Online]. Available: http://nlp.stanford.edu/IR-book/html/htmledition/feature-selection-1.html. [Accessed 03 April 2016].

[38] G. Forman, "An Extensive Empirical Study of Feature Selection Metrics for Text Classification," *J. Mach. Learn. Res,* vol. 3, pp. 1289-1305, 01 March 2003.

[39] S. Zhou, K. Li and Y. Liu, "Text categorization based on topic model," *International Journal of Computational Intelligence Systems,* vol. 2, no. 4, pp. 398-409, 2009.

[40] A. Navot, R. Gilad-Bachrach, Y. Navot and N. Tishby, "Is Feature Selection Still Necessary?," in *Proceedings of the 2005 International Conference on Subspace, Latent Structure and Feature Selection*, Bohinj, Slovenia, Springer-Verlag, 2006, pp. 127-138.

[41] A. Arora, A Selective-phrase-based Preprocessor for Improved Spam Filtering, 2008, pp. 19-20.

[42] Q. Wu, L. Wu, G. Liang, Q. Wang, T. Xie and H. Mei, "Inferring Dependency Constraints on Parameters for Web Services," in *Proceedings of the 22Nd International Conference on World Wide Web*, Rio de Janeiro, Brazil, International World Wide Web Conferences Steering Committee, 2013, pp. 1421-1432.

[43] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney and A. Paradkar, "Inferring Method Specifications from Natural Language API Descriptions," in *Proceedings of the 34th International Conference on Software Engineering*, Zurich, Switzerland, IEEE Press, 2012, pp. 815-825.

[44] F. Melvin, First-order logic and automated theorem proving, 2nd ed., Springer-Verlag New York, Inc., 1996.

[45] H. Zhong, L. Zhang, T. Xie and H. Mei, "Inferring Resource Specifications from Natural Language API Documentation," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, IEEE Computer Society, 2009, pp. 307-318.

[46] U. Dekel and J. D. Herbsleb, "Improving API Documentation Usability with Knowledge Pushing," in *Proceedings of the 31st International Conference on Software Engineering*, IEEE Computer Society, 2009, pp. 320-330.

[47] S. Forrest , S. Janice and S. I. K. Dag, Guide to Advanced Empirical Software Engineering, Springer; 2008 edition (October 26, 2007), 2007.

[48] R. Victor, Basili, G. Caldiera and H. D. Rombach, "The Goal Question Metric Approach," in *Encyclopedia of software engineering -2 volume set*, Wiley, 1994.

[49] R. K. Yin, Case Study Research: Design and Methods (Applied Social Research Methods), p. 46.

[50] R. Bakeman and J. M. Gottman, Observing Interaction: An Introduction to Sequential Analysis, 2nd ed., Cambridge University Press; 2 edition (March 13, 1997), 1997.

[51] B. Claudia, R. Mike and P. J. Gordon, "Automatic Grammar Rule Extraction and Ranking for Definitions".

[52] M. T. Goodrich, R. Tamassia and M. H. Goldwasser, Data Structures and Algorithms in Java 6th Edition, Wiley, 2014.

[53] S. Dumais, J. Platt, D. Heckerman and M. Sahami, Inductive Learning Algorithms and Representations for Text Categorization, New York, NY: ACM, 1998, pp. 148--155.

[54] Rish and Irina, "An empirical study of the naive Bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, 2001, pp. 41--46.

[55] F. Colas and P. Brazdil, "Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks," in *Artificial Intelligence in Theory and Practice*, vol. 217 , M. Bramer, Ed., Springer Science \& Business Media, 2006.

[56] J. Rennie, L. Shih, J. Teevan and D. Karger, "Tackling the Poor Assumptions of Naive Bayes Text Classifiers," in *In Proceedings of the Twentieth International Conference on Machine Learning*, 2003, pp. 616--623.

[57] A. McCallum and K. Nigam, *A comparison of event models for Naive Bayes text classification,* 1998.

[58] B. Pablo, G. Jose and P. Jose , "Improving the Performance of Naive Bayes Multinomial in e-Mail Foldering by Introducing Distribution-based Balance of Datasets,," vol. 38, pp. 2072--2080, March 2011.

[59] A. Roshani and D. PR, "Instance-based vs Batch-based Incremental Learning Approach for Students Classification," *International Journal of Computer Applications,* vol. 106, no. 3, 2014.

[60] B. Neeraj, S. Girja, B. Ritu and M. Manish, "Decision tree analysis on j48 algorithm for data mining," *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 6, 2013.

[61] A. Jehad, K. Rehanullah, A. Nasir and M. Imran, "Random forests and decision trees," *International Journal of Computer Science Issues (IJCSI),* vol. 9, no. 5, 2012.

[62] D. A. Freedman, Statistical Models: Theory and Practice., Cambridge University Press, 2009, p. 128.

[63] M. Mohri, A. Rostamizadeh and A. Talwalkar, Foundations of Machine Learning (Adaptive Computation and Machine Learning series), The MIT Press (August 17, 2012), 2012, pp. 129-131.

[64] N. S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, vol. 3, The American Statistician 46, p. 175–185.

[65] C. Silva and B. Ribeiro, Inductive Inference for Large Scale Text Classification (Kernel Approaches and Techniques), vol. 225, Berlin: Springer-Verlag, pp. 21-24.

[66] E. Alpaydin, Introduction to Machine Learning, MIT Press, 2004, pp. 330-331.

[67] S. Fabrizio, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1-47, March 2002.

[68] P. Buitelaar and P. Cimiano, Ontology Learning and Population: Bridging the Gap Between Text and Knowledge, IOS Press , 2008 , p. 138.

[69] Z. Cai, Z. Li, Z. Kang and Y. Liu, "Complex Numerical Evaluation Measures," in *Computational Intelligence and Intelligent Systems (4th International Symposium on Intelligence Computation and Applications)*, Springer, p. 469.

[70] X. Guo, Y. Yin, C. Dong, G. Yang and G. Zhou, "On the Class Imbalance Problem," in *Fourth International Conference on Natural Computation*, Jinan, 2008.

[71] M. Melucci and R. Baeza-Yates, Advanced Topics in Information Retrieval (The Information Retrieval Series), Springer; 2011 edition (June 27, 2011), 2011, p. 64.

[72] G. A. Jivani, *A Comparative Study of Stemming Algorithms.*

[73] M. F. Porter, "An algorithm for suffix stripping. Program," *Program*, vol. 14, no. 3, pp. 130 - 137, 1980.

[74] B. J. Lovins, "Development of a stemming algorithm," *Mechanical Translation and Computational Linguistics*, vol. 11, pp. 22-31, 1968.

[75] "About WordNet," [Online]. Available: https://wordnet.princeton.edu/. [Accessed 11 02 2016].

[76] S. Scott and S. Matwin, "Text Classification Using WordNet Hypernyms," in *Usage of WordNet in Natural Language Processing Systems*, 1998, pp. 45--51.

[77] J. Kwak and H.-S. Yong , "Ontology Matching Based on hypernym, hyponym, holonym, and meronym Sets in WordNe," *International Journal of Web & Semantic Technology.*

[78] "Latest SOAP versions," W3C, [Online]. Available: https://www.w3.org/TR/soap/. [Accessed 06 April 2016].

[79] "Web Services Description Language (WSDL) 1.1," W3C, [Online]. Available: https://www.w3.org/TR/wsdl. [Accessed 06 April 2016].

[80] R. Reis, "Information Technology: Selected Tutorials," in *IFIP 18th World Computer Congress Tutorials*, Toulouse, France, 2004.

[81] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java," cs.waikato.ac.nz, [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/documentation.html. [Accessed 06 April 2016].

[82] L. Deng and D. Yu, Deep Learning Methods and Applications, Now Publishers Inc, 2014.

[83] J. Porter, Deep Learning: Fundamentals, Methods and Applications (Education in a Competitive and Globalizing World), Nova Science Pub Inc.

[84] M. Iyyer, V. Manjunatha, J. Boyd-Graber and H. Daum´e, "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*, Beijing, China}, 2015.

[85] "Alphabetical list of part-of-speech tags used in the Penn Treebank Projec," [Online]. Available: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html. [Accessed 07 03 2016].

[86] A. Taylor, M. Marcus and B. Santorini, *The Penn Treebank: An Overview*, 2003.

[87] L. Quoc V and T. Mikolov, "Distributed representations of sentences and documents," *arXiv preprint arXiv:1405.4053*, 2014.

[88] A. Singhal, "Modern Information Retrieval: A Brief Overview," *IEEE Data Eng. Bull.*, vol. 24, pp. 35-43, 2001.

[89] S. J., Gershman and B. Joshua, "Phrase similarity in humans and machines," 2015.

[90] A. Günter, R. Kruse and B. Neumann, Eds., KI 2003: Advances in Artificial Intelligence: 26th Annual German Conference on AI, KI 2003, Hamburg, Germany, September 15-18, 2003, Proceedings, Springer; 2003 edition (November 5, 2003), 2003, p. 454.

[91] M. Bryan, "Formal Definition of Semantic Concepts," [Online]. Available: http://www.is-thought.co.uk/concept.htm. [Accessed 23 2 2016].

[92] H. Amiri and T.-S. Chua, "Sentiment Classification Using the Meaning of Words," *AAAI Workshops; Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence,* 15 07 2012.

[93] C. Borg, M. Rosner and G. Pace, "Evolutionary algorithms for definition extraction," in *WDE '09 Proceedings of the 1st Workshop on Definition Extraction*, Stroudsburg, PA, USA, 2009.