Chapter

Supporting Named Entity Recognition and Document Classification for Effective Text Retrieval

Philippe Tamla, Florian Freund and Matthias Hemmje

Abstract

In this research paper, we present a system for named entity recognition and automatic document classification in an innovative knowledge management system for Applied Gaming. The objective of this project is to facilitate the management of machine learning-based named entity recognition models, that can be used for both: extracting different types of named entities and classifying text documents from different sources on the Web. We present real-world use case scenarios and derive features for training and managing NER models with the Stanford NLP machine learning API. Then, the integration of our developed NER system with an expert rule-based system is presented, which allows an automatic classification of text documents into different taxonomy categories available in the knowledge management system. Finally, we present the results of two evaluations. First, a functional evaluation that demonstrates the portability of our NER system using a standard text corpus in the medical area. Second, a qualitative evaluation that was conducted to optimize the overall user interface of our system and enable a suitable integration into the target environment.

Keywords: named entity recognition, document classification, rule-based expert system, social network, applied gaming, knowledge management system

1. Introduction

The European research project Realizing and Applied Gaming Ecosystem (RAGE) is an innovative online portal and service-oriented platform for accessing and retrieving reusable software components and other related textual documents from the Web, such as research publications, source code repositories, issues, and online discussions. RAGE is used to support software reuse in the domain of applied gaming. Applied games (AG) or serious games (SG) aim at training, educating and motivating players, instead of pure entertainment [1]. RAGE supports the integration with various social networks like Stack Exchange ("Hot questions"), or GitHub ("Build software better"). For instance, RAGE includes facilities to connect with the Stack Exchange REST API which enables an easy import of online discussions into its ecosystem. RAGE users can easily import multiple discussions from, for instance, the Stack Overflow social site,

describe them with further meta information, classify them using an integrated taxonomy management system, and then finally retrieve useful information with faceted search that enables drilling down large set of documents. Currently, the classification of text documents into existing taxonomies in RAGE is done manually. The user has to, first, analyze the content of each document manually to understand the context in which this document is used. This is done by consulting the title and description of each imported document, as well as, analyzing all related meta-information (like keywords and tags), which are associated with this document. Once done, the user has to search for taxonomies that may be used to classify the imported document based on its content and metadata. This process can be very hard and requires the full attention of the user because he or she needs to consult the document and taxonomy each time manually. With a large number of documents and multiple hierarchical taxonomies, it can be very time-consuming to classify documents in RAGE.

To solve this problem, *Named Entity Recognition (NER)* is generally applied because it can extract various knowledge contents (like named entities) from natural language texts [2]. The extracted knowledge content can then be used to automate the process of classifying text documents from various domains on the Web, using, for instance, an expert rule-based system. NER has been widely used to recognize named entities in medical reports [3], news articles [4], and software web documents [5, 6]. Techniques for NER vary from rule-based, over machine learning (ML), to hybrid methods. But, ML-based NER methods are more efficient on Web contents, because they include statistical models that can automatically recognize and classify named entities from very large and heterogeneous contents on the Web. The training of a machine learning-based NER model is however very challenging. It requires, besides very good programming knowledge, dealing with different technologies and pipelines for text analysis, natural language processing (NLP), machine learning and rule-based operations [7]. Errors in the initial stages of the pipeline can have snowballing effects on the pipeline's end performance. Therefore, facilitating the development, management, and execution of all necessary NER related tasks and pipelines will, not only reduce the effort to train new NER models but also contribute to optimizing the performance of the whole system.

The goal of this research project is to develop and integrate a named entity recognition system into the RAGE ecosystem. The efficient integration of a NER system into the RAGE ecosystem will not only facilitate knowledge discovery (efficient extraction and analysis of named entities and their interrelationships), but also, enable an automatic classification of text documents into the existing taxonomies of the RAGE ecosystem.

After reviewing and comparing common systems and tools for named entity recognition and document classification, we present real-world use case scenarios and derive features for training and managing NER models with the Stanford NLP machine learning API. Then, the integration of our NER system together with the Drools expert rule-based system is presented, allowing an automatic classification of text documents into different taxonomy categories available in the knowledge management system. Finally, the results of a cognitive walkthrough are shown, serving as a qualitative evaluation and the optimization of the user interface and enabling a suitable integration into the target system.

2. State of the art and related work

2.1 Rage

As stated earlier, the RAGE social platform can be used to import questions from the Stack Exchange platform and other text documents from the Web, which

generally consist of a title, a description, and other metadata. RAGE includes a *taxonomy management system* that serves at organizing and categorizing these documents into existing, hierarchical taxonomies found in its ecosystem. *Taxonomy* is the practice and science of classifying things and concepts including the principles underlining such classification [8]. It is used in RAGE to support faceted browsing, which is a technique allowing users to drill down their large number of search results, enabling faster information retrieval. However, it is hard to classify documents with multiple taxonomies. The user can easily mix up one with another while analyzing and classifying a document into multiple hierarchical taxonomies. Each individual document (including its metadata like title, description, tags) have to be analyzed each time manually in order to understand the context in which the document is used, before making a proper classification into the existing taxonomies. This process can be very challenging and time-consuming, especially with multiple documents and various taxonomies having complex hierarchical structures. To fulfill the requirements of the project, a very desirable goal would be to develop and integrate a named entity recognition system into RAGE that can automatically recognize and classify various kinds of named entities from the multiple social networks connected with the ecosystem. Then, to apply an expert rule-based system that will enable an automatic document classification by reasoning about the extracted named entities, the hierarchical taxonomies and other textual features found in RAGE textual documents.

2.2 Named entity recognition techniques

NER techniques generally include handcrafted rules or statistical methods that rely on machine learning (ML) [2], or even a combination of those. A NER technique is denoted as *rule-based* or *handcrafted* if all the parameters (including rules) that are used to identify and categorize named entities are defined manually by a human. *Machine learning* based techniques will use a computer to estimate those parameters automatically [7]. Existing ML techniques include *supervised learning* (parameter estimation is based on already annotated data), *semi-supervised learning* (parameter estimation uses only a small set of annotated data), and *unsupervised learning* (does not use annotated data for estimation). Most popular machine learning systems are relying on Conditional Random Fields (CRF), the state-of-the-art statistical modeling method for sequential text labelling [9]. CRF has been widely used with machine learning to support different NLP tasks, such as, *part-of-speech tagging* [10], *sentence splitting* [11] and *NER* [12]. Developing a machine learning-based NER system is however very challenges and requires a lot of data for model training. Often, *gazetteers* (dictionaries of specific named entities) are introduced as additional features to recognize unknown named entities - words that were not used in the training process. Likewise, *regular expressions* can be applied to optimize ML models, because they detect more complex named entities like compound words [13].

Many factors can influence the performance of a NER system, such as a) The *language*. Some NER systems were developed for one specific language like English. b) The *named entity type*. For instance, the class of a datetime can be easily found if it only contains absolute dates (2003; 6.2.2005, April 5, 2011), but it can be difficult to detect relative dates (next Saturday, in December). c) The *domain* of the processed texts (corpora). If a classifier was trained using juristic texts, it will be difficult for this same classifier to deal with material originated from bioinformatics. The standard measures for evaluation machine NER systems are *precision*, *recall* and *F1* for this task. Recall is the ratio of correct annotated NEs to the total number of correct NEs. Precision is the ratio of correct annotated NEs to the total number (correct and incorrect) of annotated NEs. F1 score is calculated from precision and

recall and describes the balance between both measures. Most NER tools have functions to calculate precision, recall and F1 from a set of training and testing data.

2.2.1 Comparison of NER tools

Many tools have been proposed in the literature for named entity recognition. We need to review and compare them to enable a suitable integration into our target system. Therefore, we introduce the following **selection criteria**: a) the selected tool should not be limited to a specific type of text or knowledge domain b) should include a rich set of NLP features (including *NER, POS, Tokenization, Dependency Parsing, Sentiment Analysis*), c) must be stable, extendable, distributed as opensource, and should have an active community of developers. Our solution is designed to classify a relatively small amount of data. The RAGE contents have a limited size and do not consist of many gigabytes of data. Therefore, we prefer to achieve good results with a high level of accuracy and do not need a very fast classification process which often results in lower accuracy.

Our tool comparison is based on the work of Pinto [14]. According to our selection criteria, we exclude from our comparison non-opensource tools, tools without NER support, and those focusing only on specific data. To compare state-of-the-art tools, we added SpaCy, Spark NLP and Stanza to our list, because these tools arose in the last view years and may be relevant in our work.

GATE ANNIE[1] is a more general solution for various NLP tasks. It was first developed to help software engineers and researchers working in NLP but has been optimized to a more powerful system with an integrated user interface, which supports different data preprocessing tasks and pipeline executions. GATE is distributed with an integrated information extraction system called ANNIE that supports NER and many other NLP tasks. ANNIE relies on the JAPE[2] specification language, which provides finite state transduction over annotations based on regular expressions. Using the GATE interface, users can capture the provenance of machine and human-generated annotated data to create new metrics for NLP tasks like named entity recognition. Additional metrics for more specific scenarios can be added, but this requires an existing implementation in the RAGE architecture, which introduces the overhead of familiarization with the entire GATE architecture.

The **Natural Language Toolkit (NLTK)**[3] is a Python library that supports most of the common NLP tasks. It was launched in 2001 under the Apache license. Each NLP task is performed by an independent module and it is possible to train an own model for NER. The main disadvantage is that it lacks support for dependency parsing and an interface for the standard Universal Dependencies[4] dataset is missing.

Apache OpenNLP[5] is written in Java and based on machine learning. Launched in 2004 and licensed under the Apache License, the software supports NER and many NLP tasks. But it lacks support for dependency parsing.

The **Stanford CoreNLP**[6] is a Java-based tool suite from Stanford University that was launched in 2010. It supports all relevant NLP tasks, including NER and dependency parsing. CoreNLP can train new NER models independently from the data types, languages, or domain. Its API includes more than 24 different

[1] https://gate.ac.uk/ie/annie.html

[2] https://gate.ac.uk/sale/tao/splitch8.html

[3] https://www.nltk.org/

[4] https://universaldependencies.org/

[5] https://opennlp.apache.org/

[6] https://stanfordnlp.github.io/CoreNLP/

annotators for text annotation, regular expressions and language processing tasks. These annotators can be easily combined and executed sequentially in different pipelines. A REST service interface is also available, which can be used by other external systems for different NLP tasks execution. Thus, CoreNLP may be easily integrated with a rule-based expert system to support the automatic document classification in RAGE. Finally, the training of NER models is very flexible and customizable. CoreNLP includes nearly 100 parameters for CRF-based model training and performance fine-tuning, including other options for adding gazette lists that can recognize unknown named entities. CoreNLP is licensed under the GPLv3 and has a very big active community. Thus, state-of-the-art NLP methods and algorithms are permanently developed and integrated into the software.

Stanza[7] is a Python Library, developed by Stanford University as a possible successor for CoreNLP. It was launched in 2019 under the Apache license. Even the system is rather new it supports many features needed in our work, only sentiment analysis is missing. The ML models trained by CoreNLP are not directly supported in Stanza and need to be trained again. Stanza brings a client to connect to the CoreNLP server, so it is possible to use CoreNLP features over this interface, which increases the complexity. **SpaCy**[8] is one of the newer systems for NLP that was launched in 2015. It is written in Python and was published under the MIT license. It is used to produce software for production usage, which should be easy to use and fast. SpaCy supports most of the common NLP features, including dependency parsing and features for training custom models for NER. But it lacks support for sentiment analysis. The main disadvantage for our purpose is, it focuses on fast classification, which leads to a lower accuracy compared to other systems. **Spark NLP**[9] is one of the most recent NLP tools that was released in 2017. It is a library build on top of Apache Spark and TensorFlow. It supports Python, Java and Scala and focuses the usage in production systems. It has more dependencies to get it up and running compared to other systems, due to the Apache Spark architecture. The supported NLP features include all relevant features, including dependency parsing and the training of a custom model for NER. Due to its young age, the community is not as big and active compared to others. On Stack Overflow, only a few number of questions are tagged with "johnsnowlabs-spark-nlp", while the "stanford-nlp" tag has more than 3000 questions. We decided to use the Stanford CoreNLP suite for our project. CoreNLP is the only NLP software which met all our requirements. The competitors may be better or faster in one or another subtask, but overall CoreNLP seems to be the tool with the best mix of all required features. Especially the rich feature set in combination with an active and living community is a huge advantage of Stanford CoreNLP, compared to the other solutions.

2.3 Rule-based expert systems

Expert systems are rapidly growing technology of Artificial Intelligence (AI) that use human expert knowledge for complex problem-solving in fields like Health, science, engineering, business and weather forecasting [15–17]. An expert system represents knowledge solicited by a human expert as data or production rules within a computer program [17]. These rules and data can be used to solve complex problems. For instance, a rule-based classification system can be applied to classify text documents into organized groups by applying a set of linguistic rules.

[7] https://stanfordnlp.github.io/stanza/

[8] https://spacy.io/

[9] https://nlp.johnsnowlabs.com/

The rules will instruct the system to use semantically relevant elements of the document and its contents to identify useful categories for automatic classification [18]. Over the last decades, many expert systems have been proposed but essentially all of them are expressed using IF THEN-like statements which contain two parts: the conditions and the actions. In the mathematical sense, a rule can be defined as X == > Y, where X is the set of conditions (or antecedent) and Y is the set of actions (or the consequent). Rules are used to represent and manipulate knowledge in a declarative manner, while following the first-order logic in an unambiguous, human-readable form, and at the same time retaining machine interpretability. Rule-based systems generally include a "production memory" which contain a set of rules that are matched against facts stored in the "working memory" of an "inference engine" [40].

The **C Language Integrated Production System (CLIPS)** is a public domain software tool for building expert systems. It was developed by the NASA in 1985 [19]. It has become one of the most used RBES in the market because of its efficiency and portability [20]. CLIPS was written C, and for C programming. But, it is now incorporating a complete object-oriented language for writing expert systems, called COOL. COOL combines the programming paradigms of procedural, object-oriented and logical languages. While CLIPS can separate the knowledge base (the expert rules) from its inference logic, it is not that user friendly in the formulation of rules like many other systems [19].

Ten years after CLIPS, the **Java expert System Shell (JESS)** was launched by Ernest Friedman-Hill of Sandia National Lab [19] as a Java-based implementation of the CLIPS system. It supports the development of rule-based expert systems that can be tightly coupled to Java code and is often referred to as an expert system shell [21]. JESS is compatible with the CLIPS rule language, but a declarative language (called JessML) is also available for specifying rules in XML. JESS is free to use for educational and governmental purpose, but it is not an opensource software. There is no free source code under any available license[10].

The **Drools** expert system is an opensource software that was first developed by Bob McWhiter (in 2001), and later on, absorbed by the JBoss organization (in 2005). Drools is based on Java and its rule definitions rely on IF...THEN statements which are easier to understand than the syntax provided by CLIPS and JESS. Drools rules can be also specified using a native XML format. The rule engine essentially is based on the Rete algorithm [22], however, extended to support object-oriented programming in the rule formulation. Drools is available under the Apache Software Foundation's opensource license [23]. Because its easy and far more readable rule syntax, Drools has been widely used as an expert system in various domains [6]. Therefore, we chose Drools to enable an automatic document classification in the RAGE ecosystem.

3. System design

Our system design relies on the user-centered design (UCD) approach by [24], which has proved to be very successful in the optimization of the product usefulness and usability [25]. Applying the UCD to design a system includes: a) understanding the context in which users may use the system, b) identifying and specifying the users' requirements, c) developing the design solutions, and finally, d) evaluating the design against users' context and requirements.

[10] https://jess.sandia.gov/jess/FAQ.shtml

Our system allows any user (experts or novice developers) to customize and train a machine learning-based NER model in their domain of expertise. In the target system, the user starts with a named entity recognition definition, which is a set of parameters and configuration steps to train a named entity recognition model using machine learning. With the support of the system, the user can upload a text corpus, define the named entity categories, and the named entity names (including their related synonyms) based on the requirements of the target domain. Then, he/she can customize all the conditional random fields and optimization parameters used to train a model with machine learning. The information about the NE categories, the NE names, and their related synonyms is used for the automatic annotation of the text corpus, using the BIO annotation mechanism which is integrated into our system. This is very useful because machine learning-based NER systems generally require a lot of annotated data for model train. However, while the system is able to suggest a first annotation of the text corpus, which can then be used for training and testing, it is necessary for the user to customize the testing data to avoid overfitting issues which may lead to very poor quality of the trained model [7]. Once a NER model is trained, the user can finally use it to construct flexible rules (by referring to the extracted named entities in the text) for automatic document classification in various domains. These rules are business rules and are constructed using a rule-based expert system. They will be used to represent and manipulate knowledge in a declarative manner using a set of WHEN ... THEN statements in a human-readable form. The next sections will now provide an overview of relevant use cases and describe the overall architecture of the system.

3.1 Use case

Our use case diagram in **Figure 1** describes all tasks for a user to create a NER model definition, train a model, manage it, and finally use the trained model to support automated document classification in RAGE. We call our system the Standford Named Entity Recognition and Classification (SNER), as it relies on Standford NLP for NER, and Drools for document classification. Our actor is a

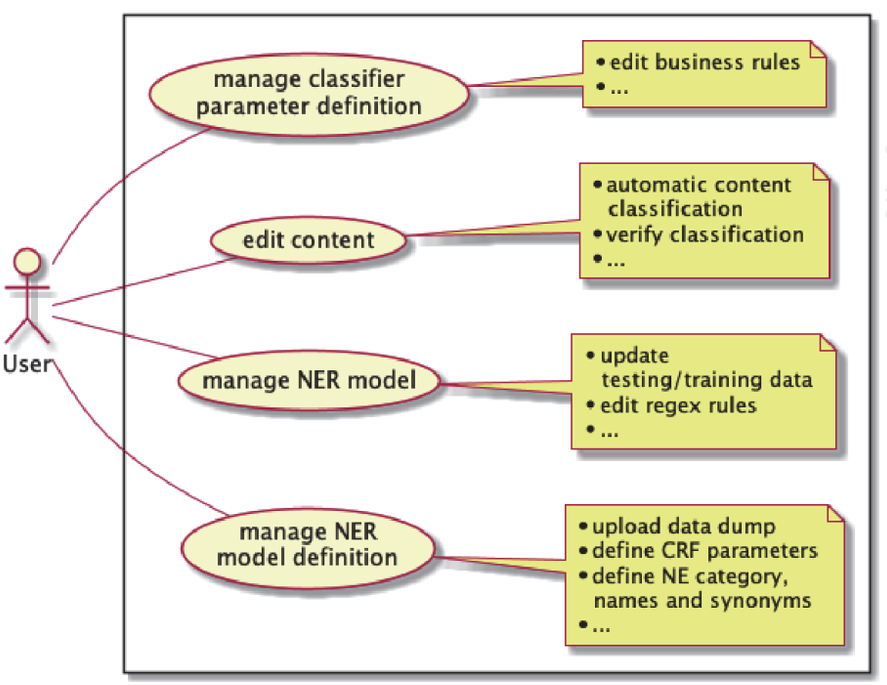

Figure 1.
SNERC use case.

registered and logged-in user in KM-EP. There are four main actions that can be executed by the user: **1) "Manage NER model definition"**. This includes uploading a data dump for use in the target domain, defining the corresponding NE categories, names, and synonyms, customizing CRF and performance parameters, adding regular expressions to identify complex named entities (like Java 11.0), preparing the NER model, which includes features for the automatic annotation of the text corpus and the splitting of the annotated text into testing and training data. Finally, training the NER model using CronJobs and the Stanford NLP machine learning API. **2) "Manage NER model"**. This includes dealing with the management of the created NER models, reviewing the performance indicators like precision, recall and F1, edition and deletion of NER models, and upload of already existing NER models in the system. **3) "Manage classifier parameter definition"**. This action deals with adding, editing or deleting business rules that are used for classifying text documents into existing taxonomies. To create new rules, the user can select the taxonomies and NER models that are relevant for its specific domain. 4) The **"Edit content"** action describes the steps, where a KM-EP content is edited and the automated classification suggestion is retrieved, supervised and saved.

3.2 Taxonomies in serious games development

Our system is developed to enable automatic document classification into hierarchical taxonomies. Since, our research is applied to the domain of serious games development, we need to review existing taxonomies and find out, which ones may be useful to validate our approach. We can refer to our previous study about software search during serious games development [26] to figure out which taxonomies may be relevant for the domain of serious games. In this research [26], we applied the LDA statistical topic modeling to automatically discover 30 topics about serious games development, from which the following belong to the most popular ones: *Programming and Scripting Language, 3D-Modeling, Game Design, Rendering, Game Engines, Game Physics, Networking, Platform*, and *Animation*. We can now review the current state-of-the-art in taxonomies for serious games and select a list of taxonomies to be used in our proof-of-concept.

Taxonomies in serious games have many aspects and dimensions. Most relevant taxonomies for our work are related to 1) *Game genre*, 2) *programming languages*, 3) *video game tools*, 4) *machine learning algorithms*, and 5) *video game specification* and *implementation bugs*. Many researchers have proposed different hierarchical taxonomies in the domain of serious games. Their main objective was to elucidate the important characteristics of popular serious games and to provide a tool through which future research can examine their impact and ultimately contribute to their development [27]. Our first classification taxonomy reflects the *game genre* [GEN], as it is one the basic classification schemes proposed by researchers in the classification of serious games [27–30]. A serious game can be classified based on the market [GEN/MAR](e.g. Education, HealthCare, Military), the game type [GEN/TYPE](board-game, card-game, simulation, role-playing game, toys, etc) or the platform [GEN/PLA] in which the game runs (Browser, Mobile, Console, PC) [27]. Many Stack Overflow discussions are already tagged with specific words like "education", "board-game", "simulation", "console". Therefore, we want to classify SG-related discussions in the game genre dimension. Second, our analysis of SG-related online discussions in Stack Overflow has revealed that developers of serious games are generally concerned with finding ways to implement new features using a specific programming language (or scripting) language [LANG]. So, a taxonomy in the programming language dimension is essential. To classify programming languages, we refer to Roy's work [31] and use the *programming paradigm* as the main

attribute in our work. We focus on serious game development, where existing game engines and tools for classic video game development are used, and we want to classify the Stack Overflow posts in this way. Third, [30] proposed a lightweight taxonomy to standardize the definition of common *tools*, *development environments* [TOOL/IDE], and game engines [TOOL/ENG] that are used for game development. We can use this taxonomy as a classification scheme for the Stack Overflow posts. Fourth, another aspect is *machine learning [ML]*, the most trending aspect in serious games development. Machine learning is one of the main techniques used in reusable software components [32] and for creating intelligent learning systems. For instance, pedagogical systems use observational data to improve their adaptive ability, instead of relying on theoretical guidelines [33]. This motivates us to integrate a machine learning-based classification scheme in our work. [34] created such a scheme and gave a brief overview of state-of-the-art machine learning algorithms. We will use this in our work for classifying posts in the machine learning dimension. Our final dimension is regarding video game bugs [BUG]. As shown in our study, one of the main concerns of serious games developers (like most of the software developers) is to find solutions to fix their bugs, whether during the design or implementation of their games. [35] developed in 2010 a taxonomy for video game bugs, which differentiate between specification bugs [BUG/SPEC] and implementation bugs [BUG/IMP]. A specification bug is generally referring to a wrong requirement in the game design document. This may refer to missing of critical information, conflicting requirements, or incorrectly stated requirements. A bug in an implementation is an error found in any asset (source code, art, level design, etc.) that is created to make the specification into a playable game [36]. A failure in an implementation is generally a deviation of the game's operation from the original game specification [35].

3.3 Drools extensions for document classification

This section presents our Drools extensions that is relevant to enable a flexible classification of text documents into the RAGE taxonomies. Our features extension rely on techniques for *Linguistic Analysis*, *Syntactic Pattern Matching* and *Document Structure Analysis*. Our classification system will be implemented as a standalone RESTful webservice so that it can be easily integrated within RAGE and any other external systems that may need to classify documents into predefined taxonomies.

Linguistic Analysis. We use the Stanford NLP API to support linguistic analysis in our System. Stanford NLP supports many NLP tasks like part-of-speech tagging (POS), tokenization, and NER. By analyzing specific part-of-speeches and recognizing various mentions of named entities discussion sentences, we can analyze the syntactic structure of each sentence. Then, we can refer to the **sentence components** (subject, predicate, object), the ***sentence form*** (whether it is *affirmative* [37] or *negative*), and the ***sentence mood*** (whether it is *interrogative* or *declarative*) to understand the structure of each sentence and derive its meaning. A similar approach was proposed by [37] for the classification of Stack Overflow discussions into software engineering-related facets, but this approach relied on hand-crafted rules for recognizing named entities in discussion posts. Instead of applying hand-crafted rules for NER, we will rely on our NER system to extract SG-related named entities (like game genres, programming languages, or game engines) from the existing text documents. To detect the *sentence form* and determine if a sentence is positive or negative, we will rely on the *StanfordNLP Sentiment Analysis API*[11], as it

[11] https://nlp.stanford.edu/sentiment/index.html

includes a machine learning-based API for this purpose. We will rely on regular expressions to determine the *sentence mood*. We will consider a sentence to be *interrogative*, if it contains a question mark, or if it starts with an interrogative word (what, how, why, etc.) (e.g. what is the best way to record player's orientation?), otherwise the sentence is *declarative*. Using our linguistic analysis features, we can understand the meaning of each individual sentence, and use this information to derive the semantic of a document. Then, it becomes easier to group documents having similar semantic into a single taxonomy.

Syntactic Pattern Matching. Research on web content mining has demonstrated that certain lexico-syntactic patterns matched in texts convey a specific relation [38]. Liu's study has revealed that many online questions belonging to similar topics have similar syntactic patterns. They found that many programming languages usually appear after a preposition, like **with** Java, **in** JavaScript. After carefully analyzing the title and description of some SG-related topics in Stack Overflow, we could easily observe similar behavior for game genres, game engines and tools, such as **for** educational games, **in** Unity 3D, **with** GameMaker, etc. Thus, the categories of a question can be derived based on the syntactic patterns of its sentences.

Table 1 shows the list of our syntactic patterns that can be used to classify Stack Overflow discussions into taxonomies of the RAGE system. Our syntactic pattern definition is based on a rich set of terms, term combinations, and standardized synonyms (**Table 2**), that we observed in various Stack Overflow discussions. Applying synonyms in our approach is very important to automatically detect name variations in text and enable a classification to perform better. For instance, we can use a pattern that includes the term "implement" and use the same pattern to identify texts that include the term "develop" or "build". To achieve this goal, we will need to create a domain dictionary with a set of semantic classes, each of which includes a standardized term and its synonyms [37].

For each parameter in our defined template shown in **Table 2**, and for each taxonomy and category that the template applies to, we will use a list of popular terms found in Stack Overflow to instantiate our template and created a semantic

Pattern	Description
PA	Entity or Term appears after a preposition
PB	Entity or Term appears before a preposition
SG	Entity or Term appears in the subject group
PG	Term appears in the predicate group
OG	Entity or Term appears in the object group
SA	The sentence is affirmative
SI	The sentence is interrogative
SP	The sentence is positive
SN	The sentence is negative
TT	Term combination <*term*1> <*term*2> appears in a sentence
TTSG	Term combination <*term*1> <*term*2> appears in the subject group
TTOB	Term combination <*term*1> <*term*2> appears in the object group
TTPB	Term combination <*term*1> <*term*2> appears before a preposition

Table 1.
List of syntactic patterns.

Taxonomy Category	Term	Term synonyms
Programming Language	< *implement* >	implement, develop, code, create, construct, build, set
Specification Bug	< *specify* >	design, require, define, determine, redefine
Implementation Bug Specification Bug	< *error* >	error, bug, defect, exception, warning, mistake
Game Engine	< *configure* >	configure, setup, adjust, adapt, optimize
—	< *howto* >	How to, How do (I,we), How can (I,we), How should (I,we)
... Bug	< *fix* >	fix, solve, remove, get rid of, eliminate

Table 2.
List of synonyms.

class with each term. We will rely on the WordNet API[12] to create semantic classes of candidate synonyms using standardized terms. When a new term is added, all its synonyms should be identified using WordNet and then considered for inclusion. By combining different terms and synonyms, we can discover a wide range of expressions and term combinations and phrases used in the majority of SG-related discussions. For instance, the term combination $<Best>$ $<Way>$ can be used to identify posts containing the expressions: "best way", "best strategy", "proper design", "optimal solution", etc. This will allow us to have a more generic syntactic pattern definition that can easily scale in different domains compared to [37]'s system (**Table 3**).

Document Structure Analysis. This feature is used to explore the structure of online text documents. We can refer to specific HTML elements to find out if a document contains a code snippets ($<code>$ ... $</code>$), bullet points ($
$ ... $
$), or even images ($<img/>$). Exploring the structure of online discussion can help us to classify documents into specific taxonomies like *Programming Languages* or *Video Game Bugs*. A quality study of Stack Overflow online discussion [39] has revealed that explanations (generally represented using bullet points in the question bodys) accompanying code snippets are as important as the snippets themselves. Also, existing survey research on document structure analysis has demonstrated that analyzing the hierarchy of physical components of a web page can be very useful in indexing and retrieving the information contained in this document [40]. For instance, if a Stack Overflow post, contains the word "bug" in its title, and one or more code snippets in its body, then it may be assigned to the *Implementation Category* of the *Video Game Bug Taxonomy*. Generally, such a discussion would include sentences like "How to **fix** my bug in ..." or "How can I

Pattern	Description
LS	Text contains multiple bullet points as HTML list
CS	Text contains one or multiple code snippets
IM	Text contains one or multiple images followed by a text description

Table 3.
Patterns for document structure analysis.

[12] https://wordnet.princeton.edu/

solve this issue... in my game" in its title or description body. Similarly, if a bug discussion includes terms like "requirement, design, or specification" in its title (e.g. I want to **fix** ... in my **specification**), with multiple bullet points in its description body, then it may indicate that the user is seeking help to solve an issue in a particular section of its design specification. In this case, the discussion post may be classified into the *Specification Bug* category of the *Video Game Bug Taxonomy*.

Our features extensions are very flexible and can be easily combined to construct even more complex rules in the Drools language. There is also no limitations for adding new extensions to document classification in our system (**Table 4**).

3.4 System architecture of SNERC

This section presents the system architecture of SNERC. Based on our use cases, we have defined 5 main components which will want to describe here (**Figure 2**).

NER Model Definition Manager manages all the necessary definitions and parameters for model training using machine learning. It includes 3 main classes. The first two, Named Entity Category and Named Entity, hold information about the domain-specific named entities names and categories. The third class, NERModelDefinition, is used to stored data like the model name, text corpus,

Pattern Matching	Taxonomy Categories	Examples
PA (SG || OG) && SA	LANG, GENRE, ...	<**How to**> to do animation with <**unity3d5.2**> An <**Educational Game**> for learning prog. Language.
(TT && SI) || PA	SPB	It might be an issue in the <**game**> <**design**> spec.
PB && CS	IMB	I am using a nstimer and it has a <**bug**> with my game loop <**code**> ... </**code**>

Table 4.
Pattern matching rules for matching stack overflow discussion posts.

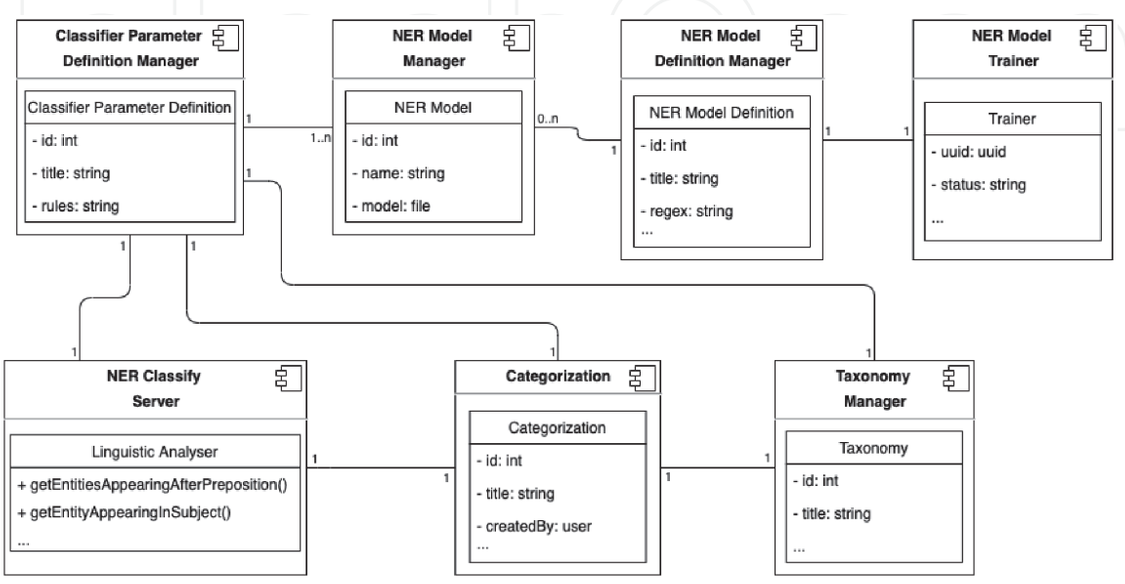

Figure 2.
Model of the conceptual architecture.

gazette lists, and regex. We use the Stanford RegexNER API to construct and store complex rules, as they can easily be combined with already trained models.

NER Model Trainer is our second component that is used to prepare a NER model. This includes the automatic annotation of the domain text corpus (or data dump) based on the previously defined NE categories, NE names and synonyms. Our system is also able to split the annotated text corpus into testing and training data. The testing data, however, needs to be reviewed by a human expert and uploaded again to avoid overfitting, and thus a realistic calculation of precision, recall and F1 scores. When this is done, the NER Model Trainer component can execute the task for training a NER model using jobs and the Stanford CoreNLP. As the NER Model Trainer is written in Java and KM-EP is a PHP project, we designed it as a separate REST service component. This has further advantages. First, the service can be developed independently and does not affect KM-EP. Second, this service can be used separately from KM-EP as it is defined as a REST API. Other external systems will just need to define the input data in a JSON format and send them via an HTTP REST call to this service. The NER Model Trainer has a class called *NER Model Definition* which represents the corresponding GUI components in KM-EP. The Trainer class is used to control the training process.

NER Model Manager. This component is very straightforward since it only serves the storage of the trained NER models into the KM-EP filesystem so that they can be used by other systems like a linguistic analyzer or our document classification system. If a model is prepared with a NER Model Definition, users can update the created testing and training data within the NER Model Manager to get better Precision, Recall and F1 scores. Also, the created Stanford Regex NER rules can be edited and updated. It is also possible to upload a StanfordNLP NER model that was trained with another system and use it in KM-EP. **Figure 3** shows an example of a recognized named entity with the NER Model Manager.

Classification Parameter Definition Manager. This component is used to manage and store business rules in KM-EP. To construct business rules that mention named entities and can be used to classify documents into existing taxonomy categories, the design of the "Classification Parameter Definition Manager" component needs to include links to the "NER Model Manager", "Content Manager" and "Taxonomy Manager" of KM-EP. We use the *Simple Knowledge Organization System (SKOS)* as the unique connection between our business rules and the taxonomy categories found in KM-EP. Even each taxonomy category in KM-EP has a SKOS persistent identifier representing the category.

NER Classifier Server. The NER Classify Server is our last component. It is developed as a standalone RestFul service to classify documents into taxonomies. To execute a document classification, the NER Classify Server needs information about the document (title, description, tags), the Drools rule, and references about the NER models, so that named entities can be used in the rule formulation. This information is sent to the server from KM-EP in a JSON format. With the provided document data and the references to the NER models, the server can now execute the NER, perform the synonym detection (with WordNet), and execute Linguistic Analysis, and Syntactic Pattern Matching on the Document structure and content. This analysis is done in the "classify()" method of a Java object, called Document. The analysis result is then stored into the properties of this object and can be used

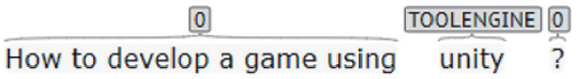

Figure 3.
Example of a recognized named entity.

during the execution of Drools rules. The following code snippet shows the implementation of our Document.classify() method.

```
Server
  Document
    title
    description
    tags
    ...
    classify()
      LinguisticAnalyzer.check(sentence)
        detectNamedEntities()
        detectSynonyms()
        appearsAfterPreposition()
        appearsBeforePreposition()
        isAffirmative()
        appearsInSubject()
        isSentencePostive()
      DocumentStructureAnalyzer(text)
      hasCodeSnippet()
      hasBulletPoint()
      hasImages()
```

3.4.1 System service implementation

To make the features of our implemented REST services available to the various KM-EP components, we created two new services in KM-EP. These services are used as an adaptor between KM-EP and its objects and our developed REST services. Each service bundles the features of the corresponding REST service and is connected with the KM-EP PHP API. The big advantage of relying on this service-based architecture is that, if we decide to change or update our REST APIs, we will only need to change the KM-EP services and leave their underline implementations untouched.

NER Model Trainer Service. The NER Model Trainer Service of KM-EP is used to connect with the NER Model Trainer REST service. As already discussed in the previous sections, this component includes the creation of a NER Model preview, the preparation of a NER Model and model training. Because the NER Models are created using the NER Model Trainer component, they need to be downloaded from there into KM-EP and deleted afterwards.

Classifier Service. The Classifier Service of KM-EP is used for the communication between KM-EP and the NER Classify Server REST service. To handle the automatic document classification, we first need to manage the NER Models using the NER Classify Server. Then, the Classifier Service of KM-EP can trigger the execution of the operation for adding or deleting NER Models by calling the NER Classify Server. Furthermore, the Classifier Service will be able to trigger the automatic classification of documents to be suggested to the user.

3.5 Proof-of-concept

After presenting our major use cases and showing details about our implemented components, we can now present a common use case scenario where Stack Overflow discussions about SG topics can be classified in RAGE. With an existing NER model in the system, a classification parameter definition can be created with the

Classification Parameter Definition Manager component to classify discussion texts into taxonomies of the system. For instance, there may be a Stack Overflow post like this in RAGE:

> Title: "bug in my game loop"
> Keywords: "cocoa-touch, nstimer"
> Description:
> "I am making a game on xcode 5. I am using a nstimer in C\# and there may be a bug in my game loop. Can you help me please. All help is great. <code>...</code>"

According to our previous definition, we can create Drools rules to automatically classify this document into *Video Game Bug* and *Programming Language* taxonomies. First, we will start with the creation of a "Classification Parameter Definition", where we select the desired taxonomy and NER models for named entity extraction. Then, we will construct our classification rules using the WHEN ... THEN syntax provided by Drools. Based on the selected taxonomy, the NER models, and our rich set of features extensions, we can easily refer to specific named entities (like C# (LANG), cocoa-touch (TOOL)) in our rule definitions and perform *Linguistic Analysis*, *Syntactic Pattern Matching*, and *Document Structure Analysis* on the document. **Figure 4** shows an example of such classification rules in the Drools language.

- Lines 6–7 (of rule 1) refer to our *WordNet* integration to detect if the term "bug" (or one of its synonyms) is included in the discussion title. Line 9 analyzes the document structure to identify if the post description includes a code snippet. Because both conditions are true, the document is automatically assigned to the *Implementation Bug* of the *Video Game Bug* taxonomy.

- Line 19 (of rule 2) checks the syntax of the post description to identify if a named entity of type LANG appears after a preposition. Since it is true, the post is assigned to the C# category of the *Programming Language* taxonomy.

To make it easier for the user to test the created rules, we implemented a form to test the developed rules. The user can input some text, execute the classification parameter definition and see a classification report with the results of the annotation and classification process. There is also a visualization of the NLP features

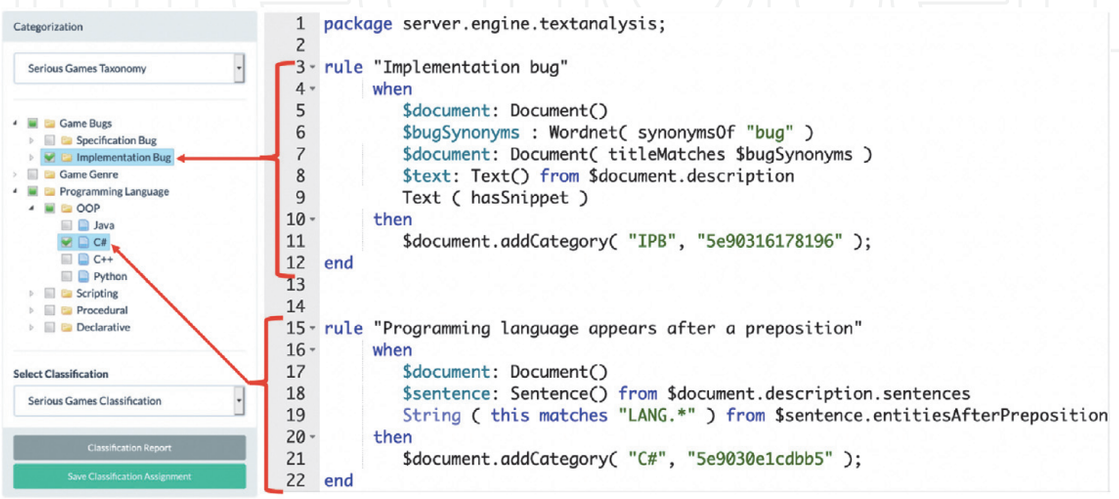

Figure 4.
Selected categories and their rules.

detected by Stanford CoreNLP which is based on CoreNLP Brat[13]. The reports include the following information:

A list of persistent identifiers of the **detected categories**, an area for the detected **sentences** with the results of the Stanford CoreNLP features, representation of detected **Parts-of-Speech**, **detected NEs**, detected **basic dependencies** and the detected **sentiment**. For further analysis the original Stanford CoreNLP output is also available in JSON format in the GUI.

4. Evaluation of SNERC

In the last chapter, we have described the implementation of our SNERC system, and presented a proof-of-concept scenario, where a machine learning NER model is used to support a rule-based classification of Stack Overflow discussions into taxonomies used in the domain of serious games. The concepts, models, designs, specifications, architectures, and technologies used in chapter 3 has demonstrated the feasibility of this prototype.

Now, we need to evaluate our developed system and prove that it is usable, useful, effective, efficient, etc. Therefore, this chapter presents different evaluations, that we conducted to evaluate different aspects of SNERC. There are several evaluation methods that can be used to evaluate software systems.

Our first evaluation is introduced to test the functionality of our NER system, as it the basic component used for NE recognition and classification, and also for supporting automatic document classification in RAGE. Thus, we use a standard text corpus to train a set of NER models and compare our evaluation values with another system, that is also based on Stanford CoreNLP. We use a text corpus of the medical area to demonstrate cross-domain portability of our approach. *Precision*, *recall*, and *F1* are also applied in this evaluation, as they are the standard evaluation parameters for comparing machine learning-based NER models.

Our second evaluation relies on the "Cognitive Walkthrough" [41] approach, which is a usability inspection method for identifying potential usability problems in interactive systems. This approach focuses on how easy it is for a user to accomplish a task with little or no formal instruction or informal coaching. We have used this method to identify possible issues in the SNERC user interface, while working through a series of tasks to perform NER and classify textual documents using business rules.

4.1 Comparison with a standard corpus

In this section, we describe the functional evaluation of our Stanford-based NER system and demonstrate the reproductivity of our approach in the medical research area. Thus, we refer to different text corpus previously used in the medical domain to train NER models with our system. Then, we compare our training result with another Stanford-based NER system applied on the same data set. Our system is compared with the work of [42], where various NER models for discovering emerging named entities (eNEs) were trained and applied in a medical Virtual Research Environments (VREs). As stated in Section 2.2, eNEs in medical environments are new research terms, that are already in use in medical literature, but are widely unknown by medical experts. The automatic recognition of eNEs (using

[13] https://github.com/stanfordnlp/CoreNLP/tree/master/src/edu/stanford/nlp/pipeline/demo

NER methods) can make them easily usable in Information Retrieval by search queries or indexing of documents.

4.1.1 Data preparation and system setting

Duttenhofer [42] used the Stanford CoreNLP for model training with the following data sets to train NER models in the medical context.

- CoNLL2003 ("english-training-data.txt"): a reference data set used to evaluate NER systems dealing with English documents.
- The NE dictionary Medical Subject Headings (MeSH) ("training-data.txt"). A dictionary (or thesaurus) of standard medical terms.
- User Relevance Feedback(URF) ("urf1.txt, urf2.txt, urf3.txt"). A set of known emerging Named Entities (eNEs) provided by experts in the medical field.

Data sets from CoNLL2003 and MeSH were selected and combined with three different variants of URF data sets. The following listing shows the parameters used for model training using Stanford CoreNLP.

```
map=word=0,answer=1
maxLeft=1
useClassFeature=true
useWord=true
useNGrams=true
noMidNGrams=true
maxNGramLeng=6
useNeighborNGrams=true
usePrev=true
useNext=true
useDisjunctive=true
useSequences=true
usePrevSequences=true
useTypeSeqs=true
useTypeSeqs2=true
useTypeySequences=true
wordShape=chris2useLC
```

These parameters describe the methods and features required for training NER models using the machine learning-based system available in Stanford CoreNLP (see chapter 2.2). These parameters include:

- map: describes the data format of the training data. The data must be separated using tabs. Column 0 must include the word (or NE), and column 1 the corresponding label used to annotate this NE.
- maxLeft: The number of words to be used as contextual feature for observing words on the left of the current word during the model training [6].
- useClasses: The "NE class" should be used as an additional feature during training.
- useWord: Each "word" of the text corpus should be used as an additional feature during training.

- useNGrams: Derive features from N-grams, such as Substrings of the word
- Other features includes are used for word shape like useTypeSeqs (for upper/lower case), useTypeSeqs2, useTypeySequences. WordShape defines the word share function to be used (here "chris2useLC'").

We use the same list of parameters for training the three models (classifierURF1, classifierURF2 and classifierURF3) initially developed in [42] (see **Table 5**). For testing these models, we use the data set ("test-document-with-O-and-NE-and-eNE-replaced1.tok"), which is an update version of the MeSH data set used by Duttenhofer.

4.1.2 Model training with SNERC

To train the same models developed by Duttenhöfer [42], we first defined three "NER Model Definitions" in our SNERC system. The data sets used in [42] are already annotated, thus, there is no need to upload a new data dump or use our automatic annotation tool to generate training and testing data. Also, we skipped the step for cleaning up the data dump (removal or HTML tags, code snippets, URLs, etc.). We continued by adding all the parameters for model training in the tab "Training Properties", where each of them can be easily changed, if needed. Then, we clicked on "Prepare NER Model" in the tab "Train Model" to prepare our models. Our model preparation function generated three documents representing the prepared models, which we renamed to remain consistent with our input data. The input documents used for training in Duttenhofer ("training-data.txt, english-training-data.txt, urf1.txt, urf2.txt, urf3.txt") were combined and uploaded to the respective prepared models. Then, we uploaded an annotated document "'*test-document-with-O-and-NE-and-eNE-replaced1.tok*"' for testing to the generated models. Finally, the training process was triggered using job. **Figure 5** shows the final result of our trained models using SNERC, which also displays the evaluation values precision, recall and F1 (**Table 6**).

4.1.3 Result

Table 7 shows the evaluation values of our trained models and the comparison with the system of Duttenhofer [42]. We have used a text corpus previously used in

Classifier	Precision	Recall	F_1
classifierURF1	93,52%	55,96%	70,02%
classifierURF2	98,92%	75,90%	85,89%
classifierURF3	97,18%	95,29%	96,22%

Table 5.
Evalutation results of Duttenhöfer [42].

ID	Name	Training date	Precision	Recall	F1	Status
44	cerc_classifier_urf1	20.08.2020 08:17:08	93.52%	55.96%	70.02%	OK, view log
45	cerc_classifier_urf2	20.08.2020 08:17:11	98.92%	75.9%	85.89%	OK, view log
46	cerc_classifier_urf3	20.08.2020 08:17:15	97.18%	95.29%	96.22%	OK, view log

Figure 5.
SNERC evaluation of Duttenhofer trained models.

Generated model names	Renamed models	Text corpus
d3dbc3839dx	*SNERC_classifier_urf1*	training-data.txt, english-training-data.txt, urf1.txt
x5dhgfb33gh	*SNERC_classifier_urf2*	training-data.txt, english-training-data.txt, urf2.txt
bc8ac12fgdb	*SNERC_classifier_urf3*	training-data.txt, english-training-data.txt, urf3.txt

Table 6.
Generated classifier names and text corpus for training.

		Duttenhoefer			SNERC		
Classifier	**Entity**	**P**	**R**	**F1**	**P**	**R**	**F1**
classifierURF1	NE	93,52%	99,51%	96,42%	93,52%	99,51%	96,42%
	eNE	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	total	93,52%	55,96%	70,02%	93,52%	55,96%	70,02%
classifierURF2	NE	98,50%	97,04%	97,77%	98,50%	97,04%	97,77%
	eNE	100,00%	48,73%	65,53%	100,00%	48,73%	65,53%
	total	98,92%	75,90%	85,89%	98,92%	75,90%	85,89%
classifierURF3	NE	98,47%	95,07%	96,74%	98,47%	95,07%	96,74%
	eNE	95,57%	95,57%	95,57%	95,57%	95,57%	95,57%
	total	97,18%	95,29%	96,22%	97,18%	95,29%	96,22%

Table 7.
Comparion of evaluation values (precision, recall, F1) between SNERC and Duttenhofer system.

the medical area to train three different NER models and show the cross-domain portability of approach. As it can be seen, all the models trained with SNERC have the same evaluation values as in the reference work, since both systems are relying on Stanford CoreNLP for machine learning-based NER. We also note, that all the evaluation values shown in picture 5 are automatically computed by SNERC and can be read in the log output function of the "NER Model Definition Manager component" (see Section 3.4). This feature is always available and can used by a user to check the performance of a model during the preparation or training process.

4.2 Cognitive walkthrough

After we implemented SNERC, it is needed to prove the usability of the system. There are several evaluation methods available to perform this task. Automated and formal methods are testing a system with a computer program, based on a formal specification, or with formal models. As it is difficult to create such a specification or model, we will not use one of these methods. Other methods like empirical methods involve a crowd of potential users of the system, which will perform common tasks in it. Such an evaluation is very resource-intensive and therefore not appropriate to our purpose. Informal methods are based on the knowledge and experience of the evaluating persons. It is known, that these methods create good results and detect many problems in a given system. On the other hand, they are not very difficult or expensive to implement, so they may be a good approach for our project. One of these informal inspection methods is the "Cognitive Walkthrough" [41], where a group of experts simulates a potential user of the system. The group navigates the system and tries to perform the typical steps to achieve the results a user tries to get. Potential problems and defects are documented and solved.

Afterwards, the cognitive walkthrough may be repeated. We chose the cognitive walkthrough as an appropriate evaluation method for our system.

Our evaluation was performed in two steps. First, we performed a cognitive walkthrough in a collaborative meeting with three experienced experts: **Expert 1** is a very experienced professor and since many years Char of Area of Multimedia and Internet Application in the Department of Mathematics and Computer Science at FernUniversität in Hagen. **Expert 2** is a PhD, significantly responsible for the concept and design of KM-EP. **Expert 3** is a PhD student, researching in the area of serious games and named entity recognition.

First, the menu structure of SNERC was navigated exploratively, to simulate the navigation of a potential user in the system. Then each SNERC component was tested. Finally, the creation of an automated classification was evaluated. Within these steps, there were overall eight defects detected, which needed to be fixed. Then, a second evaluation was performed. We extended the expert group by two new evaluators: **Expert 4** is a PhD student, researching in the medical area and emerging named entity recognition. **Expert 5** is a PhD student, researching in the area of advanced visual interfaces and artificial intelligence.

Within the second cognitive walkthrough all typical steps where performed, as a potential user would do it. There were no further defects detected. Expert 4 pointed to the problem of unrealistic performance indicators due to overfitting. This could be disproved with the possibility to supervise and edit the automatically generated testing data within the NER Model Manager. A further note was, SNERC may not be suitable to deal with huge data sets, because of its web-based GUI architecture. As KM-EP does not deal with such huge data sets this is not a real problem for our approach.

We saw the informal evaluation method lead to many results with a limited amount of time and resources. Nevertheless, an empirical evaluation with a bigger group of potential users should be done, to prove the usability and robustness of the system further.

5. Conclusion and final discussion

In this research, we presented a system for named entity recognition and automatic document classification that was integrated into an innovative Knowledge Management System for Applied Gaming. After presenting various real-word use case scenarios, we demonstrated, that it is possible to support users in the process of automatic document classification by combining techniques, such as, semantic analysis, natural language processing techniques (like named entity recognition) and a rule-based expert system. Our NER system was validated using the standard metrics for machine learning models. We demonstrated the portability of this system by using standard text corpus for model training and testing in various domains. Our overall system consisting of both, the NER and document classification system, has been successfully integrated into the target environment and was validated using Cognitive Walkthrough. A future evaluation with a bigger group of potential users may help to gather further insights about the usage, usability and error handling of the entire system.

Author details

Philippe Tamla[1*†], Florian Freund[1†] and Matthias Hemmje[2]

1 Faculty of Multimedia and Computer Science, Hagen University, Germany

2 Research Institute for Telekommunikation and Cooperation, Dortmund, Germany

*Address all correspondence to: philippe.tamla@fernuni-hagen.de

† These authors contributed equally.

References

[1] David R, Sandra L. Serious games: Games that educate, train, and inform. ACM; 2005.

[2] Nadeau D, Sekine S. A survey of named entity recognition and classification. Lingvisticae Investigationes. 2007;30:3–26.

[3] K X, Zhou Z, Hao T, Liu W. A bidirectional LSTM and conditional random fields approach to medical named entity recognition. International Conference on Advanced Intelligent Systems and Informatics. 2017.

[4] Newman D, Chemudugunta C, Smyth P, Steyvers M. Analyzing entities and topics in news articles using statistical topic models. International conference on intelligence and security informatics. Springer; 2006.

[5] Ye D, Xing Z, Foo CY, Ang ZQ, Li J, Kapre N, editors. Software-specific named entity recognition in software engineering social content. IEEE; 2016.

[6] Tamla P, B¨ohm T, Hemmje M, Fuchs M, editors. Named Entity Recognition supporting Serious Games Development in Stack Overflow Social Content. International Journal of Games Based Learning; 2019.

[7] Konkol IM. Named entity recognition [PhD]. PhD thesis, University of West Bohemia; 2015.

[8] Sokal RR. The principles and practice of numerical taxonomy. Taxon. 1963; p. 190–199.

[9] Sutton C, McCallum A, Rohanimanesh K. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. Journal of Machine Learning Research. 2007;8:693–723.

[10] Gimpel K, Schneider N, O'Connor B, Das D, Mills D, Eisenstein J, editors. Part-of-speech tagging for twitter: Annotation, features, and experiments; 2010.

[11] Tomanek K, Wermter J, Hahn U. Sentence and token splitting based on conditional random fields. In: Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics. vol. 49. Melbourne, Australia; 2007. p. 57.

[12] Ritter A, Clark S, Etzioni O, editors. Named entity recognition in tweets: an experimental study; 2011.

[13] Nagy I, Berend G, Vincze V. Noun compound and named entity recognition and their usability in keyphrase extraction. In: Proceedings of the International Conference Recent Advances in Natural Language Processing 2011; 2011. p. 162–169.

[14] Pinto A, Gonc¸alo Oliveira H, Oliveira Alves A. Comparing the Performance of Different NLP Toolkits in Formal and Social Media Text. In: 5th Symposium on Languages, Applications and Technologies (SLATE'16); 2016. p. 16 pages. Artwork Size: 16 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany. Available from: http://drops.dagstuhl.de/opus/volltexte/2016/6008/.

[15] Koppich G, Yeng M, Ormond L. Document management system rule-based automation. Google Patents; 2009. US Patent 7,532,340.

[16] Awan MSK, Awais MM. Predicting weather events using fuzzy rule based system. Applied Soft Computing. 2011; 11(1):56–63.

[17] Abu-Nasser B. Medical expert systems survey. International Journal of Engineering and Information Systems (IJEAIS). 2017;1(7):218–224.

[18] Blosseville MJ, Hebrail G, Monteil MG, Penot N. Automatic document classification: natural language processing, statistical analysis, and expert system techniques used together. In: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval; 1992. p. 51–58.

[19] Velickovski F. Clinical decision support for screening, diagnosis and assessment of respiratory diseases: chronic obstructive pulmonary disease as a use case [PhD]. University of Girona; 2016. Accepted: 2017–11-27T07: 51:26Z Publisher: Universitat de Girona. Available from: https://dugi-doc.udg.edu/handle/10256/14611.

[20] Batista-Navarro RTB, Bandojo DA, Gatapia MAJK, Santos RNC, Marcelo AB, Panganiban LCR, et al. ESP: An expert system for poisoning diagnosis and management. Informatics for Health and Social Care. 2010;35(2): 53–63. Publisher: Taylor & Francis eprint: https://doi.org/10.3109/17538157.2010.490624. Available from: https://doi.org/10.3109/17538157.2010.490624.

[21] Friedman-Hill EJ. Jess, the java expert system shell. Sandia Labs., Livermore, CA (United States); 1997.

[22] Forgy CL. Rete: A fast algorithm for the many pattern/many object pattern match problem. In: Readings in Artificial Intelligence and Databases. Elsevier; 1989. p. 547–559.

[23] Cavalcanti YC, Machado IdC, Neto PAdMS, de Almeida ES, Meira SRdL. Combining rule-based and information retrieval techniques to assign software change requests. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ASE '14. Association for Computing Machinery; 2014. p. 325–330. Available from: https://doi.org/10.1145/2642937.2642964.

[24] Norman DA, Draper SW. User Centered System Design; New Perspectives on Human-Computer Interaction. L. Erlbaum Associates Inc.; 1986.

[25] Vredenburg K, Mao JY, Smith PW, Carey T. A survey of user-centered design practice. In: Proceedings of the SIGCHI conference on Human factors in computing systems; 2002. p. 471–478.

[26] Tamla P, B¨ohm T, Nawroth C, Hemmje M, Fuchs M, editors. What do serious games developers search online? A study of GameDev Stackexchange. vol. VOL. 2348 of PROCEEDINGS OF THE 5TH COLLABORATIVE EUROPEAN RESEARCH CONFERENCE (CERC 2019). CEUR-WS.ORG; 2019. Available from: http://ceur-ws.org/Vol-2348/paper09.pdf.

[27] RATAN RA, Ritterfeld U. Classifying serious games. In: Serious games. Routledge; 2009. p. 32–46.

[28] Buchanan L, Wolanczyk F, Zinghini F. Blending bloom's taxonomy and serious game design. In: Proceedings of the International Conference on Security and Management (SAM). The Steering Committee of The World Congress in Computer Science, Computer . . . ; 2011. p. 1.

[29] De Lope RP, Medina-Medina N. A comprehensive taxonomy for serious games. Journal of Educational Computing Research. 2017;55(5):629–672.

[30] Toftedahl M, Henrik E, editors. A Taxonomy of Game Engines and the Tools that Drive the Industry; 2019.

[31] Van Roy P, et al. Programming paradigms for dummies: What every programmer should know. New computational paradigms for computer music. 2009;104:616–621.

[32] Van der Vegt W, Nyamsuren E, Westera W. RAGE reusable game

software components and their integration into serious game engines. In: International Conference on Software Reuse. Springer; 2016. p. 165–180.

[33] Melo F, Mascarenhas S, Paiva A. A tutorial on machine learning for interactive pedagogical systems. International Journal of Serious Games. 2018;5(3):79–112.

[34] Dasgupta A, Nath A. Classification of Machine Learning Algorithms. International Journal of Innovative research in Advanced ngineering. 2016; 3(3).

[35] Lewis C, Whitehead J, Wardrip-Fruin N. What went wrong: a taxonomy of video game bugs. In: Proceedings of the fifth international conference on the foundations of digital games; 2010. p. 108–115.

[36] Varvaressos S, Lavoie K, Gaboury S, Hall'e S. Automated bug finding in video games: A case study for runtime monitoring. Computers in Entertainment (CIE). 2017;15(1):1–28.

[37] Liu M, Peng X, Jiang Q, Marcus A, Yang J, Zhao W. Searching StackOverflow Questions with Multi-Faceted Categorization. In: Proceedings of the Tenth Asia-Pacific Symposium on Internetware; 2018. p. 1–10.

[38] Liu B, Chen-Chuan-Chang K. Special issue on web content mining. Acm Sigkdd explorations newsletter. 2004;6(2):1–4.

[39] Nasehi SM, Sillito J, Maurer F, Burns C, editors. What makes a good code example?: A study of programming Q&A in StackOverflow. IEEE; 2012.

[40] Mao S, Rosenfeld A, Kanungo T. Document structure analysis algorithms: a literature survey. In: Document Recognition and Retrieval X. vol. 5010. International Society for Optics and Photonics; 2003. p. 197–207.

[41] Polson PG, Lewis C, Rieman J, Wharton C. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. International Journal of man-machine studies. 1992;36(5):741–773.

[42] Duttenhöfer A. Automated Feedback Based Emergent Named Entity Recognition (ENER) in medical Virtual Research Environments (VREs); 2020.