Data-Driven Requirements Engineering

Using Natural Language Processing to Automatically organize a large collection of User Feedback

Universiteit Utrecht

Thesis for the master program Business Informatics 2017-2018

Student: Kariem Slegten

Studentnumber: 5767202

First supervisor: Dr. Fabiano Dalpiaz

Second supervisor: Dr. Basak Aydemir

Case Company: Centric B.V.

Company Supervisors: Hans Hoeijmakers & Frank Stegeman

Date: 2-7-2018

Abstract

Traditional Requirement Engineering (RE) has relied heavily on the communication between stakeholders. Today, the increasing size and usage of software products has complicated this process. This gives rise to a trend of a more data-centered approach to RE where user data is collected and elicited on a larger scale. This research uses Natural Language Processing (NLP) techniques to automatically organize a large collection of textual user feedback in order to help requirements analysts cope with vast amounts of information. We use several combinations of the Part-of-Speech tags Nouns, Verbs, Named-Entities and Adjectives and combine these with a supervised and an unsupervised clustering algorithm. We are interested in the difference in effect between algorithms that generate a fixed number of clusters and those that determine an optimal number of clusters, we use the K-Means and Meanshift clustering algorithms for this purpose. We test the use of NLP and clustering by conducting an experiment with requirements analysts in a large software development company. We use two cluster-evaluation metrics and several non-parametric tests. The obtained results, although preliminary seem to indicate that a combination of nouns and Named Entities is most informative for requirements analysts, yet no statistical evidence is found. We designed and tested an early prototype of a dashboard that enables requirements analysts to navigate a large collection of user feedback that has been automatically organized using POS tags and clustering. A preliminary evaluation with experienced analysts shows that the dashboard can effectively be used to explore a corpus of user feedback and group textually related items and it is stated how the dashboard could benefit from additional interactive features.

TABLE OF CONTENT

List of Figures

List of Tables

CHAPTER 1. INTRODUCTION AND PROBLEM STATEMENT

1.1 REQUIREMENTS ENGINEERING AND ELICITATION

Software projects often fail because misinterpreted requirements lead to software that is unequipped to handle the demands of users. It is critical for developers to have an optimal set of requirements so they can build software that is best equipped to meet the needs of their users and stakeholders. The better a requirement reflects the actual needs of the stakeholders, the better a software system will be perceived by these stakeholders. Requirements Engineering (RE) is the set of activities concerned with identifying and communicating the purpose of a software system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies (Nuseibeh & Easterbrook, 2000).

RE is often referred to as the most important phase in software development because errors produced at this stage can be very costly if undetected until a later stage (Chakraborty, Baowaly, Arefin, & Bahar, 2012). RE is a discipline that takes real-world problems and brings them into the solution space of a software developer. These requirements can be defined as verbalizations of decision alternatives regarding the functionality and quality of a system (Aurum & Wohlin, 2003). Bridging this gap is difficult and there are many ways to tackle this problem. It is an important topic of interest for the scientific community and business alike.

Both researchers and practitioners know well that requirements are not simply out there to be captured or collected; rather, *elicitation* is necessary, a complex process involving multiple stakeholders who need to be questioned and triggered to be as specific as possible about their wishes and needs and state the core of their need instead of communicating their envisioned solution. Techniques for requirements elicitation are derived mostly from the social sciences, organizational theory, group dynamics, knowledge engineering, and very often from practical experience (Zowghi & Coulin, 2005). The most common practices are prototyping, requirements workshops, interviews, brainstorming and observation. From a systematic literature review of elicitation techniques it is concluded that conducting interviews is considered to be most effective and techniques such as card sorting, ranking and thinking aloud tend to be less effective than conducting interviews (Davis, Dieste, Hickey, Juristo, & Moreno, 2006). The variety of business processes, development methods and software products account for the large variety in requirement elicitation techniques.

Some well-documented issues in RE are a lack of user involvement, which can lead to requirements that are not representative for the majority of the user base, little room for requirements negotiation and lacking incentives for stakeholders to participate. These can lead to inaccurate requirements that can ultimately cost a software company money. Recent trends promote the direct involvement of stakeholders, most notably users. Some techniques promote the construction of a crowd of stakeholders that actively participate in the RE process (Groen et al., 2017; Snijders et al., 2015). Others focus on the topic of data-driven requirements engineering, the science of eliciting requirements from data provided by users of software systems (Maalej, Nayebi, Johann, & Ruhe, 2015), where the emphasis is on the data that a user produces and the way requirements analysts can benefit from it.

Several studies have highlighted the importance of user feedback for the success of software products (Hu & Liu, 2004; Palomba et al., 2015; Panichella, Sorbo, & Guzman, 2015). Feedback which users enter themselves is called explicit feedback. Some say that explicit feedback is all observable behavior exhibited by users, like time spend on

a website or number of clicks (Jawaheer, Weller, & Kostkova, 2014). Users can be prompted to give feedback on software directly or they can use other platforms to discuss or comment. Platforms often used to give feedback are app stores, social media channels and fora. When analyzing such feedback it is important to identify stakeholders and their intent. The concepts of user involvement, requirements elicitation and requirements engineering are depicted in Figure 1. This is an example of a method where a crowd of users is actively engaged in the RE process.

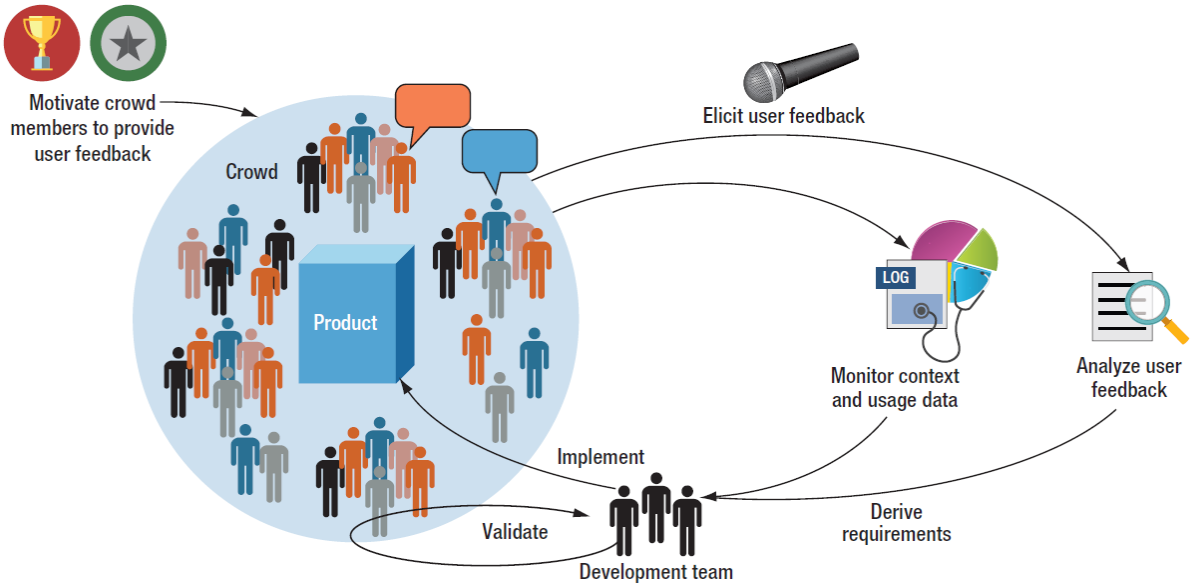

FIGURE 1: THE RELATIONSHIPS AMONG THE ASPECTS OF CROWD-BASED REQUIREMENTS ENGINEERING (GROEN ET AL., 2017)

Quantifying requirements with explicit data such as usage data or performance logs can help to prioritize requirements. It also helps to identify elements in software that are used more than others and require more attention from the software development team. The addition of implicit data such as bug reports or interaction patterns to user feedback contributes to improving the understanding of the circumstances under which the users submitted the feedback (Maalej et al., 2015; Pachidi, Spruit, & Van De Weerd, 2014). Finding the right way to combine implicit and explicit feedback is a challenge for most software companies and often privacy concerns play a key role in this.

1.2 Problem Statement

Traditional requirement elicitation techniques tend to exclude direct user feedback but rather focus on intuition, experience, the needs of their primary stakeholders or on relational schemas (Aurum & Wohlin, 2003; A. M. Davis, 2003). While these techniques proved to be effective for small-scale systems, the increase in size of software products complicates the RE process immensely. There is a need for new methods to tap into the potential that user data provides to elicit and specify requirements on a larger scale. The automation of user feedback is already commonplace for apps with a lot of daily reviews such as Facebook or WhatsApp but not much work has been done with *regards to traditional software development*.

Recent studies have shown that user feedback is a potentially valuable source of information for software developers. Users state their opinions, voice concerns and share their experiences with software products. Comments and reviews must be classified, summarized and filtered before they can be used as input for requirements analysis. RE is traditionally a stakeholder-centric, intuition- or rationale-based decision process but there is a shift towards group-based, mass-driven and user-centered decisions (Maalej et al., 2015) that are based

on real time analysis of a broad range of information sources (Nayebi & Ruhe, 2015). This is called *a data-driven approach* to RE. Data-driven RE can help requirements analysts identify, prioritize and manage requirements coming from a large group of users.

There is no such proven method to convert user feedback directly to requirements and infer context to these requirements because of the wide variety of software requirements and user feedback. Finding a right combination of Natural Language Processing (NLP) techniques, software usage loggers and requirements management tools is essential in creating a solution that can be practically adopted by a software company. Standardization of such tools and methods is still far away, however NLP techniques can be used to identify issues from unstructured texts and give requirements analysts insights into the most relevant topics that surround their product. Similarly, software usage loggers can help to identify critical parts of the software that are used most frequently and prioritize requirements accordingly. Requirement management tools are readily available for software companies and need to be integrated with data-management tools to streamline the flow of information from user to developer.

In this thesis a method to incorporate user feedback into the RE process is introduced based on *free-text input* from users. It uses NLP techniques to analyze user feedback and give requirements analysts insight into the needs of their customers. By extracting POS tags and clustering feedback items decisions regarding requirements can be improved by automatically organizing large amounts of user feedback. This allows requirements analysts to discover what their users are talking about and what the relation and text of the feedback is in regard to other items of user feedback. We examine eight different combinations of POS tags and clustering algorithms to determine which concepts of user feedback are most important in RE. Also, we designed a dashboard that automatically organizes users feedback based on these techniques and tested its effectiveness with professionals in a case company study.

This thesis is structured as following: Chapter 2 stipulates the research approach, Chapter 3 contains a literature study of related work, Chapter 4 details the design of the study, Chapter 5 describes the artifact that was created Chapter 6 outlines the experiment that tests the performance of several combinations of POS tags and clustering algorithms, Chapter 7 shows the results, in Chapter 8 we draw our conclusions and in Chapter 9 we stipulate the projects limitations and outline possible future work.

CHAPTER 2. RESEARCH APPROACH

2.1 RESEARCH BASELINE

The ever-increasing scale of software products and services creates the need for automated techniques to process user feedback. Most forms of user feedback are written in natural language text and are unstructured by nature and therefore difficult to analyze. The larger the user base of a software product is, the greater the need for automated analysis of their needs. This is most evident in popular apps such as Facebook Messenger, WhatsApp and Google Maps that receive hundreds of reviews per day, far too many to read manually.

Advances in NLP allow for large quantities of feedback to be processed automatically and can give software companies an edge. By extracting issues, features or opinions about aspects of software, developers and requirements analysts can make informed decisions about which features to implement or how to write their requirements. Eliciting, prioritizing and managing requirements automatically from analyzed feedback remains an issue because it is difficult to extract the right information from unstructured text. Computers are getting better at understanding natural language but it is still difficult to extract the right information. This research strives to use NLP techniques to improve the RE process and to provide requirements analysts with a tool to explore *large amounts of unstructured textual feedback*.

Most studies concerning extracting requirements from user feedback look at unstructured reviews, often from sources such as app market places (Palomba et al., 2015; Vu, Nguyen, Pham, & Nguyen, 2015). Issues in the domain largely revolve around identifying informative reviews (Hu & Liu, 2004; Maalej, Kurtanovi, Nabil, & Stanik, 2016) and classifying sentences according to intent (Palomba et al., 2015; Panichella et al., 2015; Schaufelbühl et al., 2017; Sorbo et al., 2016). Good work has been done identifying the most relevant reviews or sentences from developers using clustering or topic modelling (Chen et al., 2014; Fu et al., 2013).

Research that use POS tags and document clustering are mostly found in experimental linguistic papers that experiment with document clustering techniques and data preprocessing. We see that POS tags, among other Word-Net based measures, are used to improve word-sense disambiguation prior to performing clustering. The results show that POS tags contribute to an increase in clustering effectiveness because they decrease the chance of words being wrongly classified, a feat that is shown to improve even more by the inclusion of synonyms and hypernyms(Wang & Hodges, 2006). However, another experiment that uses the annotated Reuters news corpus and tests the effectiveness of document clustering using different layers of annotations has concluded that using POS tags to disambiguate the words leads to a decrease in performance because the POS tags only disambiguate the cases where different word classes are represented by the same stem, e.g., the noun ‘run’ and the verb ‘run’ and that by splitting them, the weight of their common concept is reduced(Sedding & Kazakov, 2001). Then, there are papers that use POS tags to improve document clustering performance in non-English languages(Liu, Yu, Deng, Wang, & Bian, 2010; Rosell, 2009). In this research however, we use POS tags as a format to identify words that are relevant for software development so that we can generalize across domains and languages. The main scientific contribution of this thesis are the findings about the POS tags that are most relevant for software development, the effect of these POS tags on the clustering of a large set of user feedback and the combination of these methods in an effective tool for requirements analysts.

2.2 OBJECTIVES

While the research in 2.1 gives a good indication of the current direction of the field, they mostly lack software development specific content. Therefore, the objective of this research is to identify types of words that are valuable for the RE process and test methods to use automatically analyze large amount of user input in a software development setting. The underlying objective is to provide requirements analysts with a way to navigate large amounts of unstructured textual user feedback. The dashboard produced by this research enables them to identify the most critical issues raised by their customers.

2.2.1 RESEARCH QUESTIONS

The main research question is formulated as follows:

Main research question: *What are effective automated techniques that can assist requirements analysts in organizing large sets of user feedback?*

In order to answer this question, we pose the following sub questions:

RQ1: *What information contained in user feedback is interesting for requirements analysts?*

A literature study is conducted to illustrate common concepts extracted from user feedback using NLP techniques, we survey the field and highlight research that forms the basis for this thesis. We are also interested in the opinion of practitioners and therefore have conducted a study at a case company where we use real user feedback of two software products. Interviews with requirements analysts and an experiment testing which types of words are interesting for this group illustrate the type of information requirements analysts look for in user feedback.

RQ2: *Which techniques can we use to automatically organize unstructured user feedback?*

For this RQ we examine the possibilities of automated techniques to handle user feedback with a literature study and we conduct two sessions with product managers at the case study company to determine relevant business needs and match these to automated techniques. With these techniques, software professionals can benefit from automatically organized sets of user feedback. We created a dashboard to test the techniques used in this thesis and to provide an illustration of the intended use in practice.

RQ3: *To what extent is our method useful for requirements analysts?*

> ***RQ3.1:*** *What type of concepts are most informative when analyzing user feedback?*
>
> ***RQ3.2:*** *What algorithms are most effective at grouping related user feedback and at separating unrelated user feedback?*
>
> ***RQ3.3:*** *What is the perceived value such a method for requirements analysts?*

To test this, we have designed an experiment that tests the effectiveness of extracted POS tags and clustering techniques. We have also performed an evaluation with product managers at the case study company.

2.3 STUDY DESIGN

The study is broken down into smaller steps according to the design science engineering cycle(Wieringa, 2014) as shown in Figure 2. This method has five steps:

- **Problem investigation**, this phase is used to set a diagnosis or evaluation of the problem. This is done by describing the problem according to stakeholders, goals and phenomena that accompany a problem.

- Treatment design, in this phase a solution is designed to best fit the problem described in the first phase. This solution is often an information science artifact.

- Treatment validation, the design is tested for its effects in the context in which it is intended to be used. The design decisions are evaluated and trade-offs are made explicit.

- Treatment implementation, the implementation in a real-world setting is described and executed in this phase.

- Implementation validation, the effects of the artifact and its implementation are measured against the problem as described in the first phase.

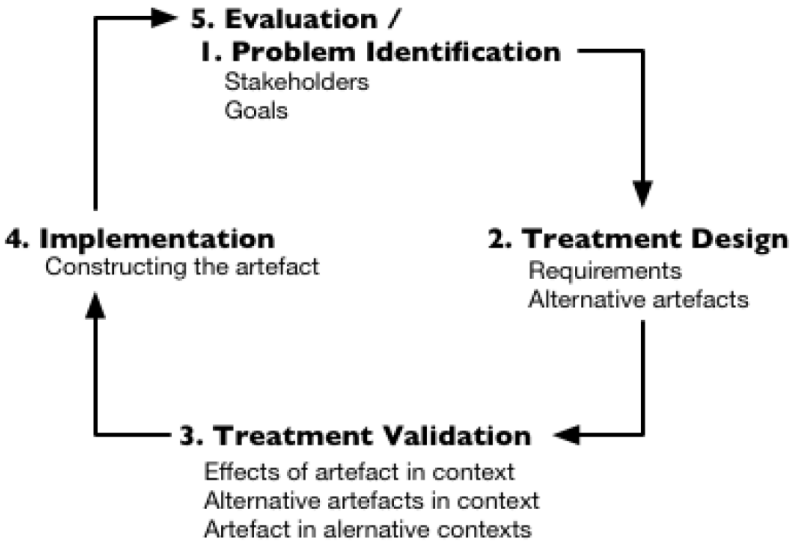

FIGURE 2: DESIGN SCIENCE CYCLE (WIERINGA, 2014)

Of these steps, only the problem investigation, treatment design and treatment validation are relevant in this context. This is because we conducted a case study and use the artifact created primarily to test the effects of selected concepts and techniques in a business setting. Therefore, we are not interested in the implementation of the dashboard, although we do briefly describe how this would be made possible in section 5.4.

To accurately describe the problem, we conduct a literature study on how user feedback is used with regards to RE and on how large sets of user feedback are handled in this context. Also, in the use case study this theoretical base is compared to practice by interviewing software professionals and defining use cases for the problems they face when handling user feedback. This phase answers RQ1 and is described in Chapters 3 – 4.2.

In the treatment design phase, the selected techniques are elaborate and mapped to parts of the problem and we explain the dashboard and the rationale behind design decisions. Several techniques are evaluated on their merits and matched to the context of the use case study. This is mostly described by RQ2 and is covered in 3.4 and chapter 5.

To validate the solution, an experiment is performed that tests the effectiveness of the techniques and concepts used. Also, a walkthrough of the dashboard is performed with professionals to validate the design. This is covered in RQ3 and is described in Chapters 6 and 7.

2.4 Scope and case study

This thesis is conducted as part of a case study that was performed at Centric, a software company that mostly supplies Dutch municipalities. We are presented with a structured set of user feedback on two Human Resource (HR) software products. These products are used by companies and municipalities to keep track of their employees' progress and finances. The feedback is provided by system administrators and is inserted into the system by helpdesk employees. It is categorized according to priority, impact, urgency, intent and type. Such a structured dataset of feedback presents some interesting research topics. We exclude comparing human generated classification to machine classified entries based on textual features, there is no classification in this research as this classifying items of feedback into categories does not allow us to examine which words are interesting for our participants pool. Although comparing automated classification to human generated classification is an interesting research topic, the structure of pre-defined classes is too rigorous for us to evaluate the factors that influence the classification. Clustering is a better method for this purpose.

The focus of this research is on providing requirements analysts with methods to improve the quality of their requirements. In the context of the case study company, requirements analysts are mostly product managers, who have a finals say in the inclusion of requirements, but these are also scrum masters, team leads and business analysts. The main tasks of a requirements analyst are to analyze, document, validate or manage software or system requirements and take into account possibly conflicting requirements from various stakeholders (Kotonya & Sommerville, 1998). The group that would benefit most from the dashboard are product managers because they have to asses a larger part of the product, and therefore the feedback over the product, than analysts that only work on certain parts of it. These are the people that are most likely to make decisions about which features to implement in software development and are most likely to benefit from a tool that enables them to explore the information contained in user feedback.

That means that discovering or examining bugs is out of scope for this project. Also out of scope is the building of a knowledge base using NLP techniques, such a knowledge base could be used by helpdesk and support employees and customers. This is a closely related subject, building on the same techniques as used in this research but does not directly cater to software development and is left out of scope accordingly. For more on this, see the future work section in 9.2.

Part of the appeal of this research is situating our methods in an actual software development company. It is therefore necessary to stipulate that the case study company represents a production company of about 5000-7000 employees, distributed over three countries and active in many domains of IT. The Dutch language of the feedback does not have an effect over the generalizability of the results because POS tags are language independent, this is also explained as a validity threat in 9.1 Limitations.

CHAPTER 3. LITERATURE STUDY ON REQUIREMENTS ENGINEERING, NLP, FEEDBACK AND CLUSTERING

In this chapter we state the results from our literature study on requirements engineering, NLP, feedback and clustering. For the literature study we applied both forward and backward snowballing. It is split up into four sections: Requirements engineering, User feedback, Using natural language processing on user feedback and Clustering techniques. The aim of the literature study is to identify current trends in RE, define user feedback and their use in software development and describe the state of the art in NLP as pertaining to software development. Many papers come from the conferences on Advances in Natural Language Processing and the conference on Requirements Engineering. 3.1 outlines current literature about requirements, in 3.2 user feedback is discussed, 3.3 details related work on feedback that uses natural language processing, 3.4 deals with clustering and a summary can be found in 3.5.

3.1 REQUIREMENTS ENGINEERING

As stated before, Requirements can be defined as verbalizations of decision alternatives regarding the functionality and quality of a system (Aurum & Wohlin, 2003). Requirements can be well-specified documents, the popular format of user stories and many other formats. Requirements Engineering (RE) is the scientific discipline that is concerned with refining stakeholder requirements for software systems into specifications that are passed on to software designers.

RE is a fundamental activity in software development and needs to be sensitive to how people perceive and understand the world around them, how they interact and how the sociology of the workplace affects their actions. And successful software systems always evolve as the environment in which these systems operate changes and stakeholder requirements change. Therefore, managing change is a fundamental activity in RE (Nuseibeh & Easterbrook, 2000).

Requirements can range from high-level abstract services or constraints to detailed technical specifications to which a software system must adhere. It has become widely accepted to divide requirements into functional and non-functional requirements, although the latter is described as quality requirements more often in recent scientific publications. The two types are defined as followed (Wiegers & Beatty, 2013):

- **Functional requirements:** These are requirements that developers have to implement in order for the intended users of the system to accomplish their tasks. They make up the functional parts of the software
- **Non-Functional requirements:** These specify which conditions have to be met for the software to be distributed instead of functionalities that the user may notice more easily. Matters such as portability, security and robustness are described in non-functional requirements.

The various steps and procedures associated with RE differ slightly depending on the required task, domain or the researcher, as many different models have been presented (Nuseibeh & Easterbrook, 2000; Sommerville, 2012; Thayer, Bailin, & Dorfman, 1997). Abstracted to a degree, they are:

- **Elicitation,** where the needs and preferences of stakeholders are specified.
- **Analysis,** requirements are identified and conflicts are resolved.
- **Specification,** requirements are documented and described in more detail.
- **Validation,** checking if documented requirements are consistent and meet the needs of stakeholders.

Also, several framework have been proposed, such as the one in Figure 3(Pohl, 2010). In which system context, core activities and requirements artifacts are depicted as the central concepts in RE which their most important parts highlighted. Also, there are two cross-sectional activities that significantly influence the RE process at all levels. These are validation and management.

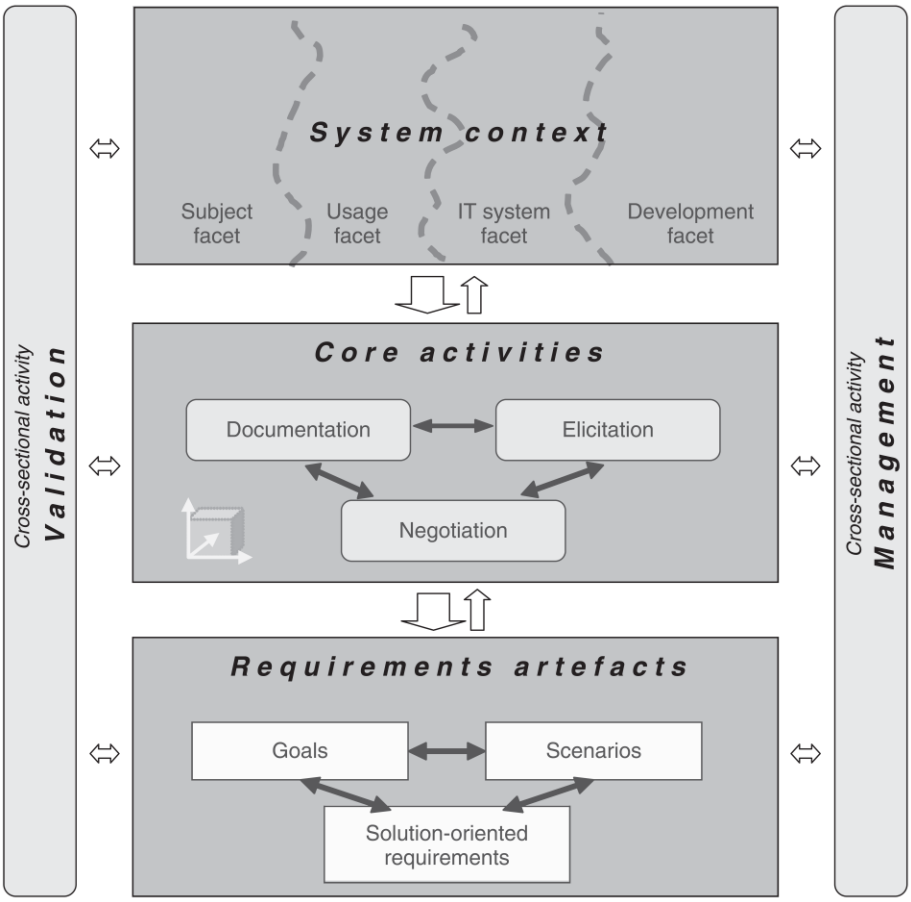

FIGURE 3: REQUIREMENTS ENGINEERING FRAMEWORK(POHL, 2010)

3.2 USER FEEDBACK

There is an important distinction to be made concerning user feedback. Users can choose to submit feedback to a software company, this is called explicit feedback, or the company can collect data on the user, which is called implicit feedback. Explicit feedback is often taken from online portals or through questionnaires or service desks and the while the use for automated analysis techniques increase with a more rigorous structure, the expressiveness of the user, and therefore the value of the feedback can suffer. Collecting implicit user feedback

can help requirements analysts to prioritize and contextualize feedback and therefore make decisions based on data. Some examples for both types of feedback are shown in Table 1.

Explicit	**Implicit**
Reviews	Usage data
Bug Reports	Click streams
Comments	User Metadata
Feature requests	Motion Capture
Rating	Error reproduction

TABLE 1: EXPLICIT VS IMPLICIT DATA

3.2.1 EXPLICIT USER FEEDBACK

User feedback in plain text, or natural language, can be considered as an informal description of a need or issue of a stakeholder. Users can be prompted to give their opinion on certain elements of the software or the product as a whole. Explicit feedback is feedback that the user deliberately communicates to the software vendor or company. As a rule of thumb for software companies, this type of data collection is expensive, limited in coverage, and subject to selection biases since users decide whether to participate or not (Fox, Karnawat, Mydland, Dumais, & White, 2005).

Recently, crowd participation has entered the domain of RE. Crowd participation is different from crowdsourcing requirements because it does not delegate micro-tasks to an anonymous crowd but rather engages user to share their opinions in order to deliver better software products to them. This approach is called CrowdRE (Groen et al., 2017) and the principal objective is to derive requirements for software from a large user base that might not be able to influence software development otherwise. This large base of users can be incentivized to participate through gamification, where they are invited to contribute and rate contributions of others. A recently developed platform for this is Refine (Snijders et al., 2015) where users can suggest ideas, comment and rate on these and earn points for a leaderboard. The analysis of user feedback plays a central role in these methods and often text mining techniques are used.

3.2.2 IMPLICIT USER FEEDBACK

Usage data, or system-user interaction data, consists of temporal sequences of events that took place while the users were interacting with the system (El-Ramly & Stroulia, 2004). These kinds of sequences are called interaction traces. Interaction traces tell developers something about possibly interesting patters of user activities and how the system support these activities. These patterns can be used for a variety of engineering tasks and software companies often track these patterns through specialized tools. Typically, interaction traces contain interesting patterns of user activities and the usage of the system in support of these activities can be used for various engineering tasks.

Although the benefits of using usage data are well recognized, translating prioritized usability problems or redesign proposals is a subject less documented in literature. A systematic study was conducted by (Følstad, Law, & Hornbæk, 2012) who conclude that there is a discrepancy between research and practice and that the field should explore more light-weight usability tracking tools. Software companies today often build in their own trackers or use industry standard tools like Microsoft Application Insights. (Pachidi et al., 2014) developed the Usage Mining Method which extracts usage data from users profiling, clickstream analytics and classification analysis. From a literature study and outcomes of a case-study they conclude that the desired outcomes of usage data mining

should be statistical summaries of user behavior, factors that influence customer decisions, user profiles and most frequent navigation paths.

Some feedback channels allow the addition of multimedia feedback descriptions such as screenshots and audio recordings. These are unstructured by nature and are difficult to analyze. Combinations of feedback are called Multi-modal feedback (Morales-Ramirez, Perini, & Guizzardi, 2015). The authors developed a ontological representation for user feedback that describes the types and attributes of feedback. A uniform definition of feedback can help researchers and companies to develop new methods for using user feedback to aid software evolution. An interesting method to prioritize end-user feedback with multi-media files is to define a domain-specific dictionary and use text mining to determine whether terms from this dictionary are present in the feedback or in the multimedia files attached to the feedback (Gartner & Schneider, 2012). Priority can then be derived by weighting the number of keywords and their entropy. This can be an interesting way of quickly prioritizing feedback without extracting requirements from it.

3.3 USING NATURAL LANGUAGE PROCESSING ON USER FEEDBACK

An interesting categorization to start this chapter with are the types of NLP tools with regards to RE as proposed by (Berry, Gacitua, Sawyer, & Tjong, 2012). There are basically four types of tools available for requirements analysts that want to use NLP for different purposes, these and some examples are shown in Table 2. The dashboard produced in this research is used to identify key abstractions from the document by presenting requirements analysts with clusters of feedback that are related. The literature described in this chapter is either of the same category, uses the same techniques or is used to illustrate the progression of the field of NLP and RE towards a more data-driven approach. By reviewing additional related works, the dashboard is enriched by context and design decisions become more apparent.

Purpose	**Example**
Find defects and deviations from good practice; find weak sentences and ambiguity.	(Lucassen, Dalpiaz, van der Werf, & Brinkkemper, 2016; Wilson, Rosenberg, & Hyatt, 1997)
Generate models from textual descriptions.	(Popescu, Rugaber, Medvidovic, & Berry, 2007; Robeer, Lucassen, van der Werf, Dalpiaz, & Brinkkemper, 2016)
Infer links between requirements and other development artifacts.	(Huffman, Alex, Senthil, & Sundaram, 2006; Palomba et al., 2015)
Identify key abstractions from documents	(Gacitua, Sawyer, & Gervasi, 2010; Hu & Liu, 2004)

TABLE 2: THE FOUR TYPES OF NLP TOOLS (BERRY ET AL., 2012)

Many researchers have extracted some form of information from user feedback using NLP. Most have done so using publicly available datasets of reviews in app stores. Table 3 shows an overview of relevant work that uses user feedback and utilizes NLP to help facilitate software development or show how to extract interesting textual information from other sources of feedback that could be used in RE.

Type of feedback	Technique		Output	Source
	Extraction	Processing		
Movie reviews	Features. Negative and positive opinions.	Linear Conditional Random Fields.	Correlation between entities and opinions.	(Li et al., 2010)
App reviews	Classifier to remove 'non-informative' reviews.	Clustering through topic modelling.	Groupings of the most informative reviews for app developers	(Chen et al., 2014)
App reviews	Sentiment analysis.	Latent Dirichlet Allocation(LDA).	Micro-, Meso- and Macro level analysis review and relation to rating.	(Fu et al., 2013)
App reviews	POS tags to extract keywords.	Ranking through sentiment analysis.	Sentiment towards keywords over time. Most relevant reviews.	(Vu et al., 2015)
App reviews	Text- and sentiment analysis.	Classifiers. Taxonomy of user intent.	Textual features and intentions of reviews.	(Panichella et al., 2015)
Developer discussions	Linguistic pattern mining.	Taxonomy of developer intent.	Classified sentences.	(Sorbo et al., 2016)
Online product reviews	POS-tagging, frequent feature analysis.	Feature-based summarization.	Summarized reviews.	(Hu & Liu, 2004)
App reviews	AR-Miner. Commits mined from change log.	Text similarity computation.	Features extracted from user reviews. Changes in code.	(Palomba et al., 2015)

TABLE 3: OVERVIEW OF LITERATURE CONCERNING AUTOMATED PROCESSING OF USER FEEDBACK

One of the issues with using user reviews as a source of information is the unstructured nature and information classification problem. To sort out which reviews are more informative than others, AR-Miner (Chen et al., 2014) cuts the text down to a sentence-level granularity and classifies these into 'informative' or 'non-informative' according to the perceived intent. Intent is measured using a set of rules that developers consider as constructive, such as requests to add or modify features vs questions and inquires. They use a semi-supervised machine learning algorithm, the Expectation Maximization for Naïve Bayes, to classify sentences into the two categories. They use topic modeling to create a structure in the informative reviews. This is done by assigning each review or sentence a number of topics that the review entails, this is perceived by the authors as better suited than clustering, where each instance belongs to a single category. Finally, they use a ranking method to rank topics and instances according to importance and visualize these for developers.

An interesting study that builds on AR-Miner is the work by (Palomba et al., 2015), which takes as input reviews for released apps and extracts issues and links these to source code changes and monitors these for developers. They use AR-Miner to extract informative reviews and from these they identify issues and commits that can be of use for the released version of the app in question. From the changelog of the released version all relevant commits are mined and subsequently linked with issues and commits from the dataset of reviews through several parameters, such a textual similarity or matching GUI-related terms. Finally, these issues, commits and links are monitored through a report which helps developers track changes.

An approach to classifying feedback in the form of user reviews is the classification framework proposed by (Panichella et al., 2015) which uses a taxonomy of user intent that might be useful for developers. They classify sentences in user reviews of apps using text analysis, NLP and sentiment analysis, most interestingly by matching sentences to a list of 246 linguistic patterns that describe user intent. They experiment with a range of settings and algorithms using the WEKA-tool to conclude the J48 algorithm is most efficient for their research. AR-miner is used here to filter out non-informative reviews as well. Their conceptual model is shown in Figure 4.

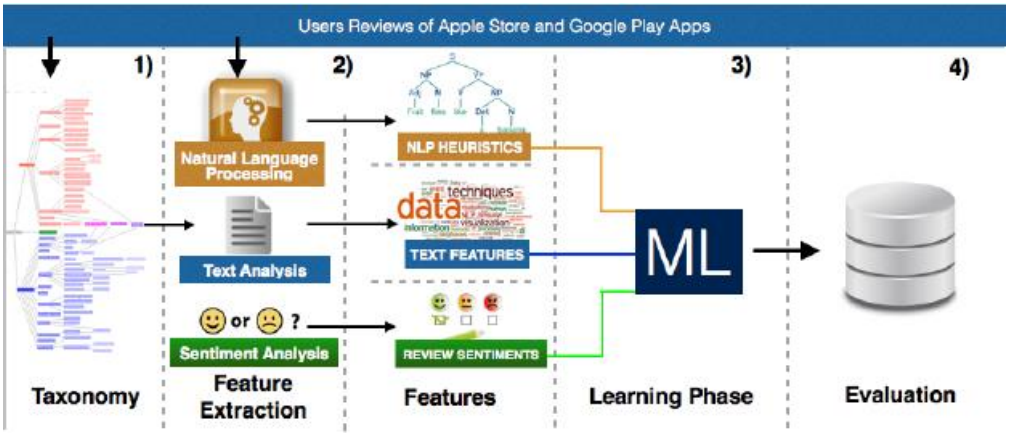

FIGURE 4: NLP ARCHITECTURE FOR THE AR-MINER TOOL (PANICHELLA ET AL., 2015)**.**

A way for developers to actively search for user feedback pertaining to the issue they are currently working on is a tool called MARK (Vu et al., 2015), which is available online[1]. It searches for keywords that developers can search for and gives a historical sentiment analysis as well as a ranking of most relevant reviews. Besides some creative pre-processing techniques, see Figure 5, a ranking method is introduced which determines relevance by calculating a contrast-score for keywords that weights the positive and negative sentiment associated with the word.

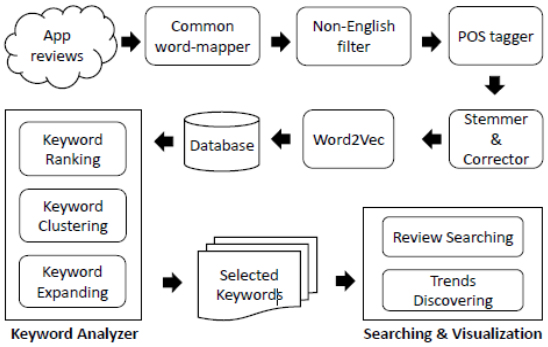

FIGURE 5: PREPROCCESSING STEPS FOR MARK (VU ET AL., 2015)**.**

[1] http://useal.cs.usu.edu/mark

Most sources of user feedback are reviews that come from app marketplaces because they are easy to mine and publicly available. It is also interesting to look at developer communication and how valuable information can be extracted from this. (Sorbo et al., 2016) identified six categories of developer intent which they matched with 231 linguistic patterns often used by developers to classify sentences according to their intent and value. An online tool to evaluate developer emails was released online[2].

Another source of input for research on text mining are movie reviews. Features, objects and opinions are extracted using Condition Random Fields, a technique that models dependencies between nodes in a parse tree. When these are all mapped, a structured summary of the review is generated which can be converted to XML (Li et al., 2010).

Twitter is also used as a source of information relevant to software companies in science (Guzman, Alkadhi, & Seyff, 2016). The large number of tweets makes it an interesting potential source of requirements. In this study annotators classified 1000 tweets pertaining to software products into categories of content, relevance and sentiment. Two machine learning algorithms, C4.5 and Support Vector Machines, were used to classify the rest of their 10,986,494 tweets with a precision ranging between 0.78 and 0.54.

The importance of a human analyst in refining the data produced by data-mining tools, and further guiding and tuning the data-mining process was argued by (Hayes, Dekhtyar, & Sundaram, 2005). It is typical of data-mining tools not to produce perfect results i.e., both precision and recall are never 100%. Such results may create negative ripple effects when utilized to help automate a desired task. To deal with this problem, the authors suggested that only refined results obtained by an analyst, and not those directly produced by a data-mining tool, should be made available to other tools or humans.

3.4 Clustering techniques

The use for clustering techniques when applied to user feedback has two uses (Villarroel, Bavota, Russo, Oliveto, & Penta, 2016): (i) requirements analysts having hundreds of items of feedback in a category could experience information overload, negating any positive advantages achieved by any classification that might have proceeded the clustering, this effect is also one of the main drivers to use automated techniques in RE (ii) information about the number of users or size of the bug or feature already represents valuable information for requirements analysts because it allows them to prioritize or scope. Given these uses in RE, this chapter briefly describes the workings of clustering, the relationship with other data mining techniques, the techniques used in this research and a few interesting ways in which clustering has been used in an RE context.

Data clustering is the practice of repartitioning unlabeled data points into joint or disjoint groups of clusters. Typically, the clustering is considered more successful when the points in the same cluster are similar and the points in different clusters are dissimilar. If the data points being clustered are documents, we refer to the process as document clustering (Wang & Hodges, 2006). Documents can be clustered based on their contents, title or meta-data. In this thesis, items of feedback are considered documents and they are clustered based on word-co-occurrence. For the sake of brevity, this chapter will not describe traditional clustering as performed on large data sets with data points but only on document clustering techniques, which use different distance metrics. For an example of the output of clustering performed on requirements, see Figure 6.

[2] http://www.ifi.uzh.ch/seal/people/panichella/tools/DECA.html

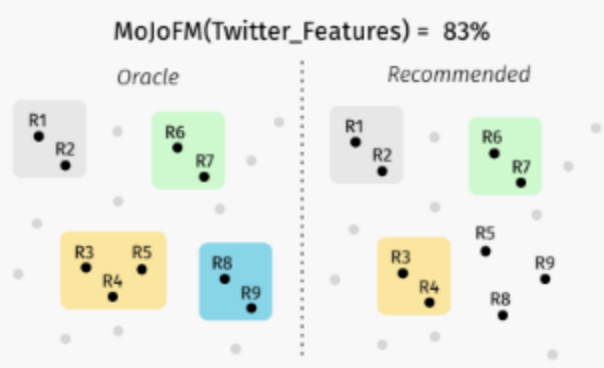

FIGURE 6: EXAMPLE OF K-CENTROID CLUSTERING OF REQUIREMENTS (VILLARROEL ET AL., 2016)

There are many different clustering algorithms, but a broad distinction can be made between algorithms that perform hierarchical clustering and centroid-based clustering. The former technique organizes data into a hierarchy that facilitates browsing. In Figure 7, part of a dendrogram is presented in which the hierarchy of some of the items of feedback from our research dataset can be seen. The parent-child relationship among the nodes in the tree can be considered as topics and subtopics in a subject hierarchy (Fung, 2002). The latter technique is based on the idea that a centroid, or a central point in a group, can represent a cluster. The k-means algorithm is the most known of this family. After selecting k initial centroids, each data point is assigned to a cluster based on a distance measure, then k centroids are recalculated. This step is repeated until an optimal set of k clusters are obtained based on a criterion function (Luo, Li, & Chung, 2009). Cluster algorithm performance often depends on the task at hand and there is no universally applicable algorithm. Data characteristics such as volume, density and dimensionality as well as factors such as computing power play a role in algorithm selection. A good review of document algorithm performance which compares hierarchical document clustering to centroid-based clustering has found centroid-based algorithms performs as good or better than hierarchical document clustering techniques in addition to having a better run-time (Steinbach, Karypis, & Kumar, 2000).

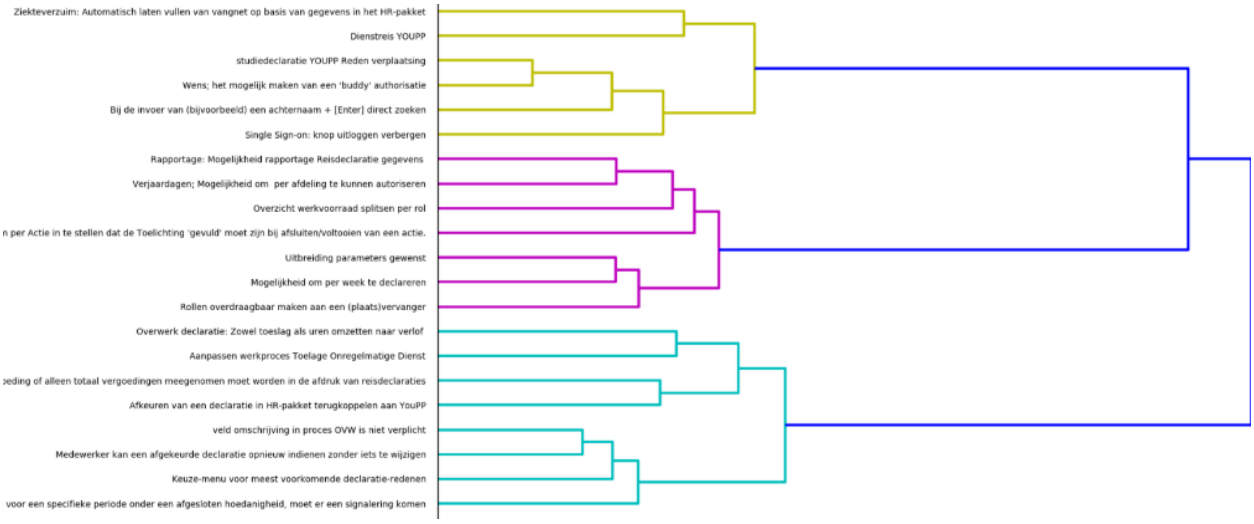

FIGURE 7: AN EXAMPLE OF HIERACHICAL CLUSTERING[3].

[3] Created from the dataset of the case company, using the scipy.clustering.hierachy dendogram in python.

When discussing the notion of supervised or unsupervised clustering, it is important to make a distinction first between clustering and classification. In classification, data points are categorized according to a set of categories provided by humans. It is the task of the classification method to classify documents or data points to these categories accordingly. This is not be confused with supervised or unsupervised clustering. The k-means algorithm for instance is an example of a supervised clustering algorithm, where the human gives a number of clusters to be generated. With unsupervised clustering, the algorithm creates a number of clusters based on the characteristics of the data. A final distinction that has to be made is the difference between hard and soft clustering. This difference is created by the fact that data can be clustered in one or more clusters, a single cluster is hard clustering and more than one indicates that the data is soft clustered (Duda, R. O., Hart, P. E., & Stork, 2012).

One of the first successful implementations of clustering applied to a large corpora of textual documents is the Scatter/Gather method (Cutting, 1992). This technique uses clustering to form clusters of The New York Times articles so that the user can select a sub-section of the corpus that he is interested in. The method then proceeds to re-cluster this subsection allowing the user to iteratively reduce the number of articles that he is interested in, creating an intuitive browsing experience. This has been proven to be effective when a formal search query is difficult to formulate.

A case where clustering has been applied to requirements, specifically to improve the test case prioritization process, has shown that clustering can successfully be applied in the RE domain. The techniques used are similar to the techniques used in this research, as words in requirements are treated as a bag-of-words, a term-document matrix is created and k-means is used to create the clusters. The initial clustering of requirements is improved by adding information about traceability, source code, code modification and client preferences, as shown in Figure 8. Although the effectiveness of the approach varies depending on the amount of clusters generated, the general conclusion is that by grouping related requirements, testers can more effectively manage their regression tests (Badwal & Raperia, 2013).

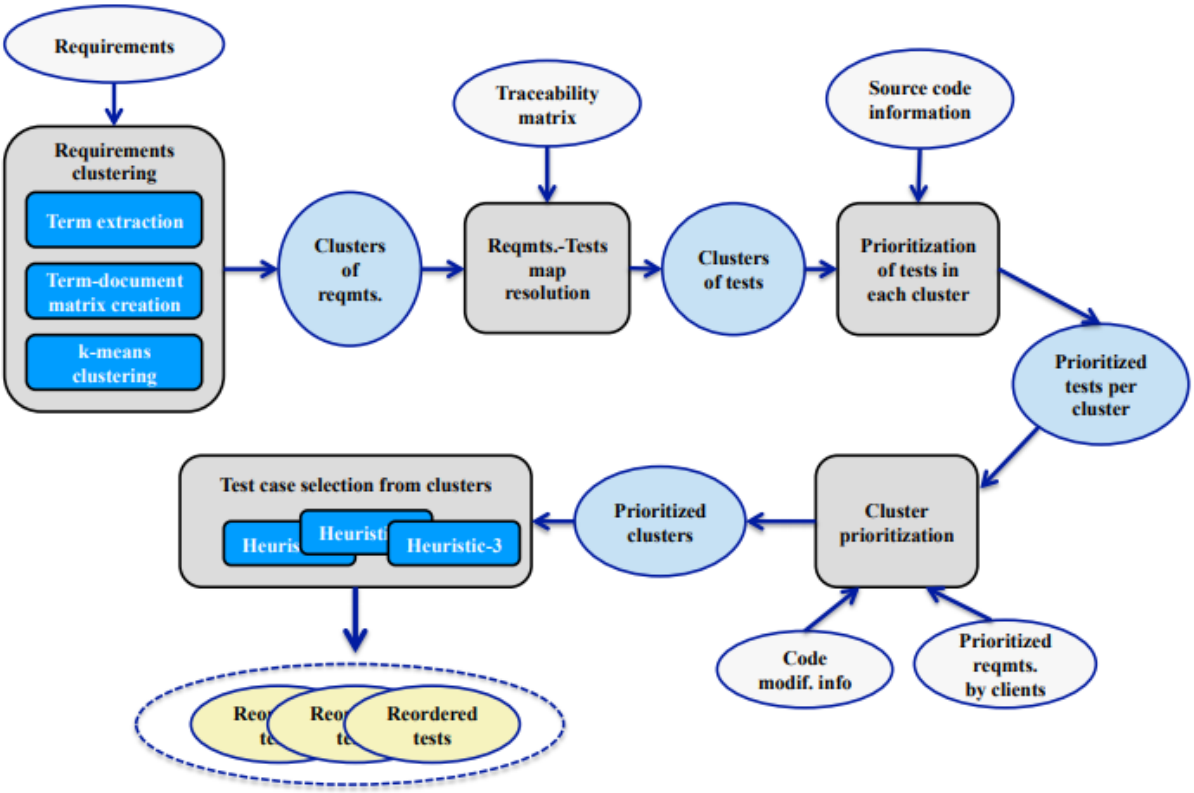

FIGURE 8: CLUSTERING APPLIED TO IMPROVE TEST CASE SELECTION OF REQUIREMENTS (BADWAL & RAPERIA, 2013)**.**

In another case where clustering user feedback can benefit the RE process, document clustering and text similarity measures are used and the authors conclude that it is possible to compare clusters based in textual features of reviews or other sources of feedback and rank and prioritize in this manner. This done by comparing the number of reviews in the cluster, the average rating in the cluster, the difference between the average rating of the cluster and the average rating of the app or by the number of different hardware devices in the cluster. To compare textual features they use text similarity measures that can rank clusters accordingly (Villarroel et al., 2016).

3.5 SUMMARY OF LITERATURE STUDY

In the first part of this literature review we define the field of RE and describe some features of requirements and the RE process. The various steps of the RE process are mentioned and the need for more expertise on the subject of RE is sketched. By examining the RE framework (Pohl, 2010) the conclusion is drawn that RE is a complex process, in which many stakeholders and influences play a part.

The role of user feedback in the RE process is discussed as a valuable source of information for requirements analysts. Feedback can be either be explicit or implicit. Explicit feedback is feedback that the user chooses to send, this is often textual. Implicit feedback is data collected on the user that can be used to improve the quality of the requirements.

To automatically process user feedback, many techniques have been used in the RE domain by using NLP. There is great variety in the type of feedback used, technique, output and purpose, a small slice of which has been described. But the main focus of this part of the literature study is to show the difficulties of analyzing large amounts of unstructured texts.

Also described are the workings and uses of clustering in RE. Several important distinctions about clustering are made as well as the difference between clustering and classification. Clustering has also been shown to be beneficial to readers when applied to large amount of texts or requirements. From the entirety of the literature study, the rationale for the design of the study becomes apparent. Because of the characteristics of the data and the problems faced by requirements analysts today, a combination of NLP and clustering is applied to a large set of user feedback to improve the analysis conducted by requirements analysts.

CHAPTER 4. RESEARCH DESIGN

4.1 OVERVIEW

In this chapter, the design of the study is explained. This is plotted against the design cycle (Wieringa, 2014) as described in 2.3 Study design. Figure 9 shows that three of the phases of the design cycle are used and that there is a distinction made between activities that are classified as scientific contributions and those that are performed in the context of the case study we performed at Centric. These are denoted in the swim lanes as 'Theory' and 'Practice'. Note that the dashboard design spans all three of the phases.

In the problem investigation phase, a literature study performed on the field of RE and NLP has produced a clear description and scoping of the research field. In this chapter, the characteristics of the data from the case study company are described and it is shown how several use cases for NLP in the context of the case study company have been formulated. The literature study, data characteristics and defined use cases have shaped the design of the artifact in the treatment design phase. And finally, our approach is validated in two ways: By performing an expert evaluation on the dashboard and by performing an experiment with students and professionals to test which concepts and techniques used in the dashboard are most efficient.

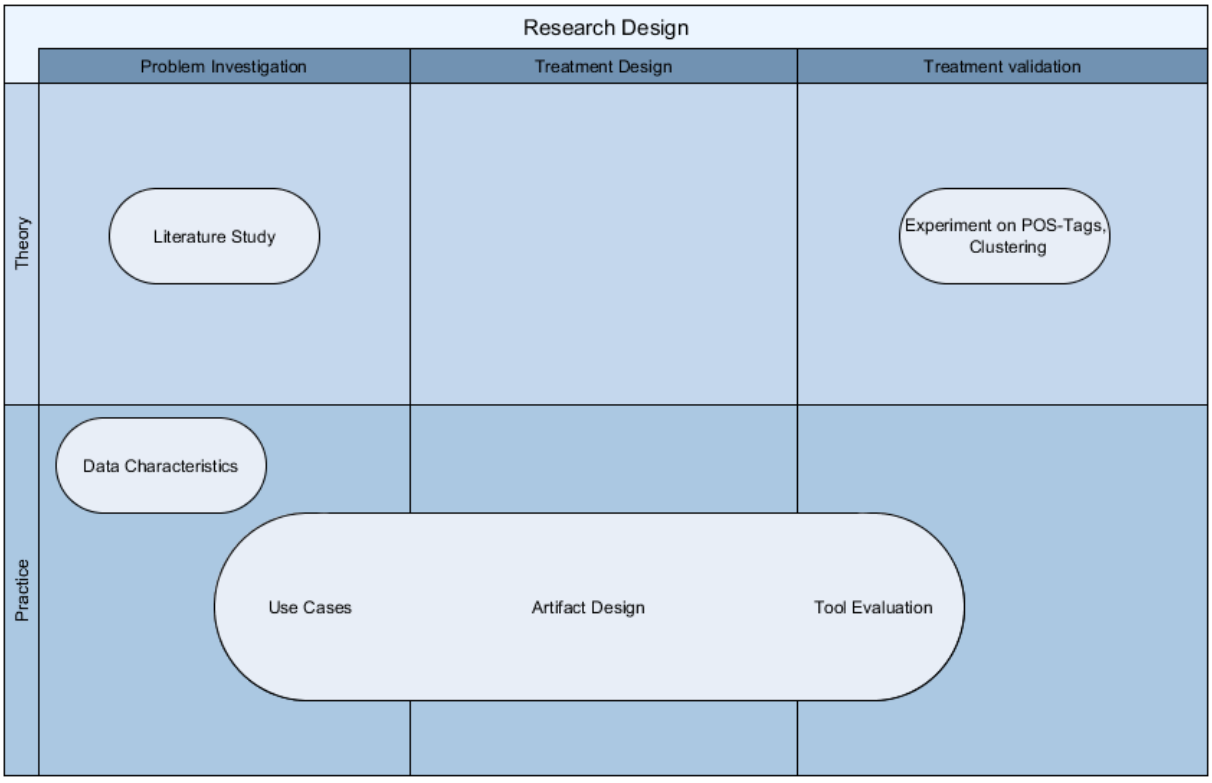

FIGURE 9: RESEARCH DESIGN

4.2 DATA DESCRIPTION

The dataset used in this research consists of 1,772 entries of customer feedback provided over two software products of Centric in the HR-domain. All entries are either requests or bugs and all are labeled as yet to be resolved. Table 4 describes the different features of the data. Note that these features are sometimes also classifications, as decided upon by the helpdesk employee of the case study company that submitted the item of feedback.

Feature	Description	Example
Nummer	Unique identifier.	'AF2305987'
Productgroup	Name of the software for which the entry is made.	'YouPP'
Type of report	The intent of the feedback as evaluated by the helpdesk. Included are only requests for change and incidents.	'Change'
Priority	Priority as indicated by the customer. Can either by Low, Medium or High.	'High'
Date	Date of original submission.	'16.01.2017'
Type of notification	The type of feedback submitted by the user. We included only wishes and mistakes.	'Wish'
Category	This is a feature that divides the entries into categories such as Customer/Internal/Budget, Must Have/Should Have/Could Have/Would Have or yet to be determined.	'Must Have'
Melder	The name of contact who submitted the entry.	'Lotte'
Organisatie	The name of the company or internal division that submitted the entry.	'Cafeteria administration Netherlands'
Korte_oms	The title of the entry.	'Show manager on form level'
Toelichting	The actual feedback, written in natural language text. This is the main area of focus for this research.	'In the form it is not stated clearly which employee is assigned to which manager.'
Oplossing	If provided, this entails the provided solution.	'Bug fixed at 09.00'
Releasenotes	Description of the release notes provided with the solution.	'Deployed under condition ..'
Testinstructies	Instructions for the testing of the provided solution.	'Test like this ..'
Aanvullende_info	Additional information.	'Corresponds to ticket nr Ga202394'
Configuratie_Item	The module or feature to which the entry is applicable.	'YouPP desktop manager'
Productlijn	The product line to which the entry is applicable.	'HR & Payroll'
Originele_melding	If the entry is manually linked by the helpdesk employee to an earlier entry, this field shows the unique identifier of the earlier entry.	'Ga202394'

TABLE 4: DATASET CHARACTERISTICS

4.3 USE CASES

After reviewing the literature and seeing the possibilities that the user feedback collected by the case study company presented, a session was held with of a Business Development Manager and a Project Manager from Centric. The goal of this session was to see if concrete use cases could be defined that utilized NLP to satisfy both the scientific contributions of this research as the practical applications that the case study company desired. These use cases form the basis of the dashboard design and are an interesting take on the possibilities of NLP and the needs of a software production company and are included in the thesis accordingly.

In this session, the professionals from the case study company were asked to rank a number of possible applications of NLP techniques in the context of their business. They were given a few scenarios and the following is the result of a refinement session:

- Developer wants to explore the issues surrounding a module.
    - This was rated as the most interesting use case and is therefore described in more detail in Table 5.
- Which items should go into the next sprint backlog?
    - What is the sentiment of the total text in the ticket?
    - What is the sentiment towards named entities, features and aspects?
    - Which aspects of features/modules are mentioned in the text?
    - What is the basis for a prioritization? This can be done through relative frequency, sentiment, relevance or explicit data.
- Developer decides to take care of a bug.
    - Find/suggest similar ones that already exist.
    - Suggest solutions from resolved tickets.
- Developer is writing a user story.
    - Is it possible to match a user story to incoming tickets?
    - Suggest possible semantic overlap between customer complaints and user stories.
- Customer files a complaint or bug.
    - Find/suggest similar tickets.
    - It is already solved?
    - Knowledge center.
    - What are the possibilities of developing a chatbot/natural language question answering?

A second session was organized after having evaluated the results of the first session and the goal was to match the use cases identified in the first session to the opportunities provided by NLP. Again the Business Development Manager and the Project Manager attended and it was concluded that exploring an issue or a range of issues is the most interesting use case and would be best suited in the context of this research. The concepts in Table 5 are the result of this session where considerable thought was given to possible use cases within the projects context and the techniques that could possibly be applied. These can be seen as a prelude to the dashboard where some of these techniques are put into practice.

Use case	Technique
Extracting features/modules/keywords	Named entity recognition, Entity extraction.
Determine which keywords/features/modules are mentioned frequently	Summarization, tf-idf.
Can we cluster tickets based on textual features?	Document clustering, K-Means.
Comparing generated clusters to predefined clusters	Variation of information, comparing two clusters on information.
What are the topics of the clusters?	Topic modelling: LDA.
What is the sentiment of the total text in the ticket (grouped by cluster)?	Sentiment analysis, Double Propagation.
Which tickets are relevant to my module?	Summarization of occurrences, ranking by tf-idf, LDA.
Is it possible to identify common occurrences of mentioned features across product lines?	Clustering, Keyword based search.
Is it possible to derive priority from text?	Sentiment analysis, tf-idf.
And eventually from data such as: value of customer, number of users, sentiment?	Combining data, feature weighting.

TABLE 5: DEFINED USE CASES AND MATCHING NLP TECHNIQUES

4.4 DASHBOARD DESIGN

As discussed in 3.3 and shown in Table 2, NLP tools have different uses in the RE process. In the context of this research a dashboard is created that aims to support requirements analysts in their work by helping them navigate large amounts of user feedback. By automatically organizing items of feedback through clustering, requirements analysts can evaluate the automatically generated clusters of feedback and decide on the value of these clusters with regards to the requirements extracted from the feedback. The dashboard also enables several combinations of POS tags to base the clustering on, so the requirements analysts can re-organize the collection of feedback based on nouns, verbs, Named-Entities or adjectives that the users who sent the feedback are using. To increase the precision and to give requirements analysts more information, a domain-specific dictionary is automatically generated and topic modelling is applied, these techniques and their uses are described in 5.2.

The input of the dashboard is purely text-based and its purpose is to explore what the users are actually saying and which other items of feedback are textually related to this. The dashboard is also designed with the notion in mind that in the future, a wide range of user feedback will be available, such as forums, social media channels and review platforms, and combining this data with sources such as business requirements, system and technical requirements, stakeholder preferences and requirements interdependencies into aggregated user data will require new data analytics tools and more sophisticated user feedback tools (Maalej et al., 2015). With this in mind, we designed a dashboard to test some of the concepts discussed in this research and to examine the viability of such a text-based user feedback browsing tool.

4.5 EXPERIMENT WITH POS TAGS AND CLUSTERING TECHNIQUES

To test the effectiveness of our approach we have designed an experiment. We want to test which concepts, or combinations of POS tags, and which clustering techniques are most effective when automatically organizing large amounts of user feedback. To test this, 10 participants are asked to do a pairwise evaluation of items of feedback. These pairs are carefully selected from the dataset so that all variables are tested equally.

These participants are split up into 5 IT-students and 5 professionals. We are interested in the differences in results when a group is more familiar with the domain. We are also interested which POS tag combination performs the best and we have selected 4 combinations to test. A supervised and an unsupervised clustering algorithm are used

to repartition the data after the POS tag combinations are extracted and we are also testing these approaches for any differences in performance. After the experiment, participants are interviewed and asked for their motivation when selecting related items.

The metrics we are using are intra-cluster relatedness, how similar are the items that are in the same cluster, and inter-cluster relatedness, how different are the items that are in a different cluster. Using these metrics, we can evaluate how well the different settings of our NLP and clustering techniques perform. To evaluate our results, we are dependent on several non-parametric tests.

CHAPTER 5. FEEDBACK VISUALIZATION DASHBOARD FOR REQUIREMENTS ANALYSTS

Dashboards are information management tools that display visual representations of data. They often feature charts, tables and gauges that track business performance. Dashboards can be connected to a plethora of data, such as real-time data, and are often customized to suit a specific business need or support an employee in a specific role. As explained in 4.4 Dashboard design, the purpose of the dashboard in this research is to provide requirements analysts with a means to *explore textual relations in a dataset of user feedback*. This chapter details the created artifact of this thesis. It shows the architecture, details the used techniques and provides rationale for design decisions that are made.

5.1 ARCHITECTURE

The logical architecture of the dashboard is shown in Figure 10. The figure shows that we extract items of feedback from servers at the case study company, perform several actions on them and present our analysis in a dashboard.

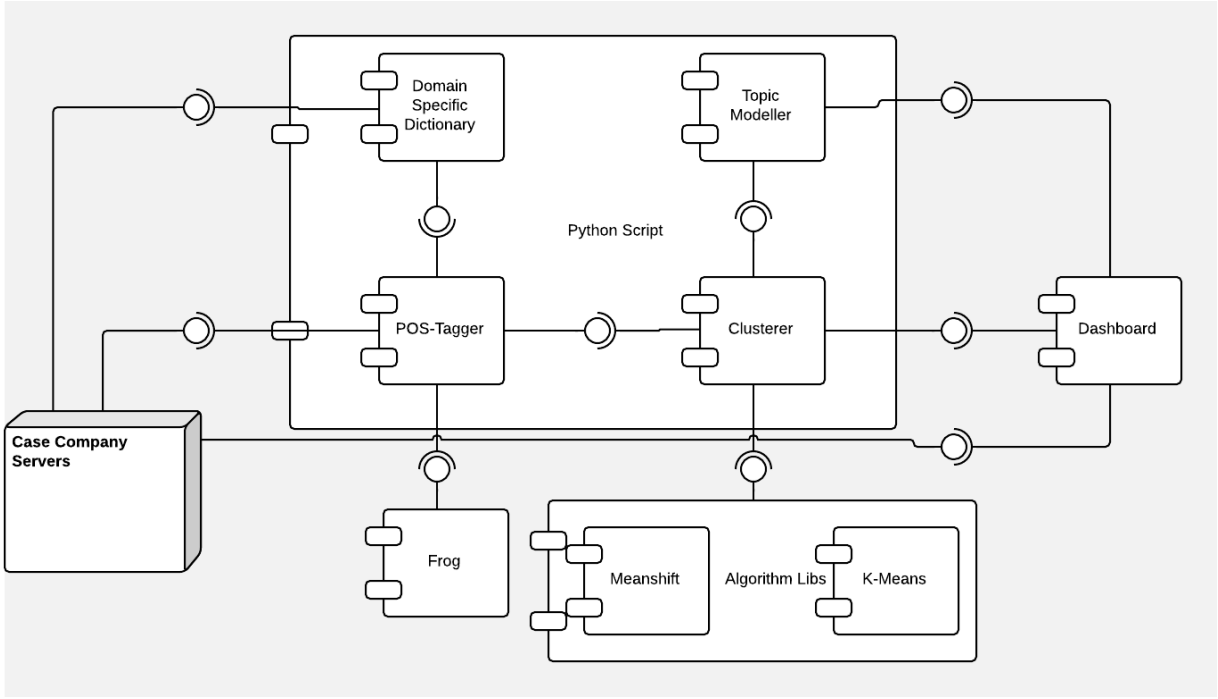

FIGURE 10: UML COMPONENT DIAGRAM OF ARTIFACT ARCHITECTURE

The items of feedback in the dataset contain bug reports, requests for change, feature requests and questions that customers supplied to the case study company. Only requests for change and feature requests are taken into consideration. The full text of the items are analyzed, all other categories described in Table 4 in section 4.4 Dashboard design are ignored. This is because we fully want to focus on automatically organizing sets of unstructured texts and the human generated classification does nothing to help us achieve this goal. After the

items of feedback are extracted, we use a POS tagger that relies on an NLP tool for Dutch and a domain specific dictionary.

These altered texts are assigned to clusters through two algorithms, K-means and Meanshift. Over these clusters we also perform topic modelling to identify the topics per cluster. The result of the clustering is displayed in a customized dashboard. With this dashboard, requirements analysts can see the clusters that are generated based on the POS tags that were selected and view the full text and other information about the feedback of their customers.

5.2 COMPONENTS

In this section the used techniques are explained in greater detail.

5.2.1. FROG

For parsing Dutch language we use Frog[4], a tool by Radboud Universiteit Nijmegen. The text in the dataset is split into sentences and tokenized. Every token is subsequently lemmatized and segmented into morphemes. A part-of-speech tag with corresponding confidence in the tag allows us to do a grammatical analysis of the token. The data is further analyzed by recognizing named entity types and base phrase chunks, both denoted in Bio-encoding[5] . Finally, a dependency graph shows the sentence structure and how the words in it are related. Frog performs better than other Dutch-language parsers (Van den Bosch, Busser, Canisius, & Daelemans, 2007) and has become the industry standard for mining Dutch texts.

5.2.2 POS TAGS

Using Frog, all words in the items are assigned a POS tag. Based on the desired POS tag that the user wants to filter on, all other words are removed and just these words remain in the items of feedback. We set the script to generate several combinations of POS tags, for instance if we were to filter on Nouns and Named Entities, all words that are not labeled as such will be removed from the item of feedback.

5.2.3 DOMAIN-SPECIFIC DICTIONARY

To emphasize words that might be relevant for a requirements analyst, a domain specific dictionary is automatically created. This is created by removing stop words and such from all the texts in the corpus of feedback and counting their frequencies. The same approach is taken for a corpus of Dutch language texts that are unrelated to the domain (Sang & De Meulder, 2003). The relative frequencies of the words are compared to each other and a domain-specific dictionary is created containing the words that are used three times more in the items of feedback than in the general Dutch texts. This is shown in Table 6, where the ten words with the highest relative frequency are shown with the number of times they occurred in the dataset and in the Dutch language corpus.

Word[6]	**Occurrences in Dataset**	**Occurrences in Dutch language corpus**	**Relative frequency**
parameter	13	1	13.0
supervisor	20	2	10.0
percentage	7	1	7.0
add	6	1	6.0
join	6	1	6.0

[4] Dutch language tool for parsing and tagging Dutch language: https://languagemachines.github.io/frog/

[5] https://lingpipe-blog.com/2009/10/14/coding-chunkers-as-taggers-io-bio-bmewo-and-bmewo/

[6] Translated from Dutch

user	6	1	6.0
change	5	1	5.0
notification	5	1	5.0
configuration	5	1	5.0
table	5	1	5.0

TABLE 6: TEN WORDS IN THE DATASET WITH THE HIGHEST RELATIVE FREQUENCY

The domain-specific dictionary is used to multiply words after they have been selected by our algorithm. By multiplying words that are used more often in the domain than in regular texts, we believe that the items of feedback have a better chance of being clustered on the words that are meaningful to the requirements analysts.

5.2.4 CLUSTERER

Using the altered items of feedback, containing only the POS tags, we transform the set of items to a tf-idf matrix that maps words to documents. This vector-space model uses Euclidian distance as a measure, which is very common in document clustering. This is used as input for two clustering algorithms, a supervised K-means and an unsupervised Meanshift.

5.2.4 TOPIC MODELER

After the data has been tagged and clustered. We perform topic modelling over the generated clusters of tickets. We use a technique called Latent Dirichlet Allocation for this (Blei et al., 2003). This technique assigns topics to clusters and iteratively redistributes these words to identify the most important topics of the cluster. So for every generated cluster, we have a list of the most probable topics of the cluster. In the dashboard, these topics are visualized in a word cloud.

5.3 DESIGN

For displaying the clusters we use a custom package with PowerBI visuals usually used for displaying news articles. We see that it has several visualization that help visualize clusters, information and individual items in the clusters. For the design of the dashboard we look at the information seeking mantra as stated by (Shneiderman, 2003), which dictate that any good visualization of information must start with an overview first, zoom-and-filter second and details-on-demand third.

5.3.1 OVERVIEW

The first page of the dashboard, shown in Figure 11, displays the clusters generated to the right, several data-slicers to the left and several instances of the cluster below. This gives the user a complete overview of the data that is selected. As the user selects a cluster, the cluster becomes bigger than the other clusters and the data-slicers to the left change accordingly.

FIGURE 11: FIRST SCREEN OF THE DASHBOARD

5.3.2 Zoom-and-Filter

Using the slicers on the left hand side of the main page, the user can filter on the date on which the item of feedback was submitted and he can select which tickets to display in a list of titles. These measures help the user to guide its own analysis of the data. When a cluster has been selected, all the items in this cluster are displayed at the bottom and can be read in full from the dashboard.

5.3.3 On-demand-details

The user can also navigate to the second screen where a more detailed picture of the cluster is presented. In this screen, the user still sees the clusters, the words associated with the clusters and their relationship to other clusters. The new information that the user gains is the topics of the selected cluster, obtained through topic modelling, displayed in a word cloud. This allows the user to discover the topics per cluster. Also on this page, there is the possibility to read the items that make up the clusters in full. This is shown in Figure 12.

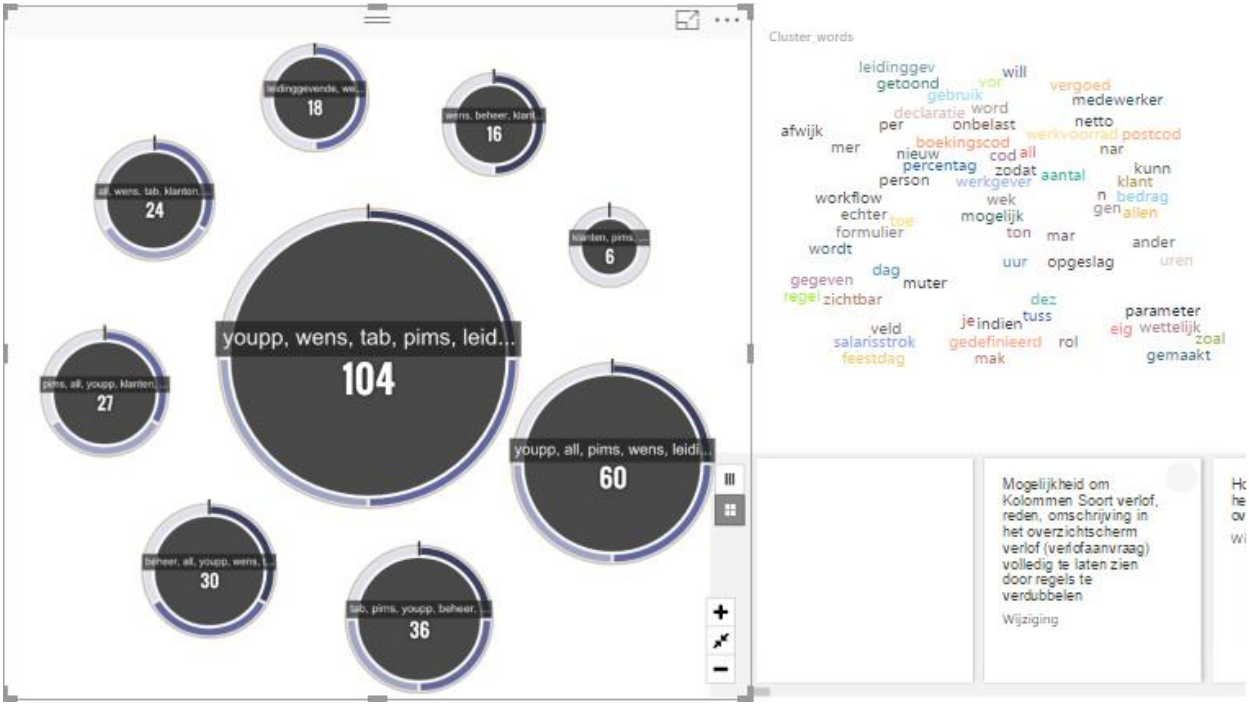

FIGURE 12: SECOND SCREEN OF THE DASHBOARD

5.4 USE IN PRACTICE

This dashboard is to be used by requirements analysts that have a large amount of user feedback at their disposal. Professionals can use the tool to browse large quantities of feedback and see automatically organized sets, distill the most important topics, see characteristics of the sets and review individual items in these sets. If we match the uses of the dashboard to the main activities of RE as shown in Figure 3, the activity that best suits the dashboard is elicitation because the tool is ideal of discerning the importance of issues or problems that users might have. The dashboard can also be used for documentation as it helps to build a rationale for RE decisions made by requirements analysts. The case can even be made that the dashboard also supports the two cross-sectional activities: validation and management, as the requirements analysts can show the results obtained with the tool to others, like management.

This is a prototype, running on a python script as an intermediate between the case study company database and the dashboard. In order for the tool to be used to increase its use, a connection between the two will have to be established to feed real-time data into the tool. A machine-learning implementation would also greatly benefit the tool, if an algorithm could track the use of the tool and the preferences of the user, the clustering results could be improved. The addition of interactivity, such as the user being able to add or remove words that are weighted more in the clustering process could also be a good option.

CHAPTER 6. EXPERIMENT DESIGN: ACCURACY OF POS TAGS AND CLUSTERING TECHNIQUES

To discover which combinations of POS tags and clustering algorithms perform the best, we have designed an experiment that measures the performance of these factors as scored by human evaluators.

6.1 EXPERIMENTAL SETUP

To answer our research questions and to validate the research done on POS tags and clustering algorithms, we design an experiment to measure the effectiveness of the clustering performed on a combination of these. This difficult because it requires some form of a ground truth set to evaluate performance, in most cases this is achieved by human evaluation. A nice take on this matter is: *"A particularly challenging problem is that of introducing human understanding, interpretation and biases (context) into the problem of visualizing and interacting with data arising in an information retrieval task (such as a web search). In such situations human judgment is the relevant standard for the measurement of the relevance of any clustering."* (Pfitzner, Leibbrandt, & Powers, 2009). With that in mind, we test the effectiveness of our approach by evaluating combinations of POS tags with different clustering algorithms with the help of participants. Depicted in Figure 13 is a graphical representation of the experimental setup and the labels on the arrows indicate in which section the relevant elements are explained.

The results of the clustering are scored by human evaluators who determine whether the clustering of tickets is relevant by means of pairwise comparisons. For this, sampling will be used to reduce the number of pairs to be examined. The evaluation is conducted with both professionals within the domain of HR software and IT-students because of the limited availability of the professionals and because we are interested in the type of words that each group finds important. Metrics for cluster analysis and non-parametric tests are used to evaluate the performance of the different settings of the experiment. After the participants completed their pairwise evaluation, we asked them questions to find out which words were most important to them.

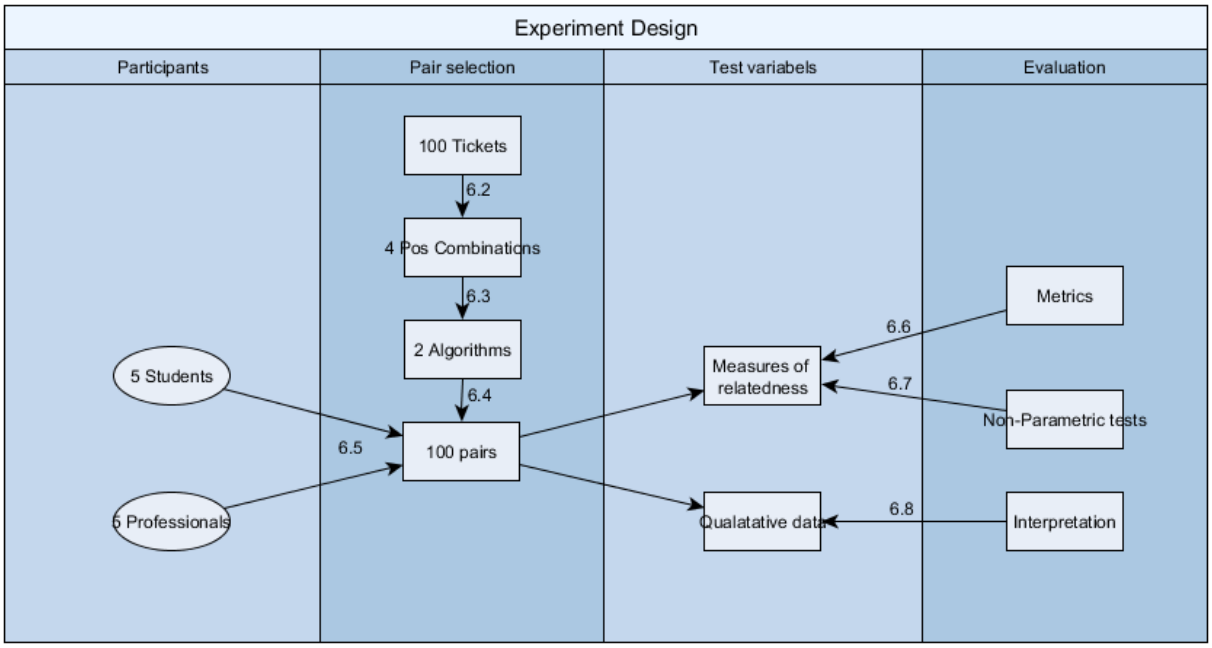

FIGURE 13: EXPERIMENTAL DESIGN

6.2 POS TAG COMBINATIONS

Determining which POS tags to include in the experiment is done manually. This means that the most informative POS tags are identified both in literature and in practice. We assume therefore that Nouns, Named Entities, Verbs and Adjectives are the most interesting POS tags to test, in Table 7 these concepts are expanded to include an example from the dataset and their relevance to RE is mapped. Visual inspection of the data, which has been validated by professionals from the case study company, confirmed that these words have the greatest information value. The combinations of these POS tags and their rationale are shown in Table 8. The words that are tagged with the POS tag that is selected are kept in this text; all other words will be removed. These texts are compared through the tf-idf matrix as described in 5.2.4 and clustered accordingly. The python code for this is in appendix I-A.

Concept	**Possible Values**	**Relevance to RE**
Nouns	‘functionality', 'competencies'	Extractions of concepts
Adjectives	‘heavy'	Indication of perceived value of concepts
Verbs	‘work', 'being'	Indicating uses of concepts
Named Entities	‘YouPP', 'Eindhoven'	Clustering on persons, location, products

TABLE 7: POS TAGS AND THEIR RELEVANCE TO RE

Combinations	Relevance to feedback on software	Rationale
Named Entities + Nouns	Filtering on customers or products combined with nouns, which usually indicate functionalities, could result in groups of feedback combination that are related in content or subject.	Both named entities and nouns can indicate the subject of an item of feedback. This is more true in systems for information retrieval and question answering, who are often designed to process information regarding named entities (Desmet & Hoste, 2014).
Named Entities + Nouns + Adjectives	Combining customers or products with functionalities and qualities can be a very effective way of organizing a dataset.	Adjectives can have many purposes, they are often considered as indicators of the sentiment of an opinion (Hu & Liu, 2004) or used to indicate the subjectivity of a sentence or review (Bruce & Wiebe, 1999). We include adjectives as a POS tag to group feedback based on common subjectivity, sentiment and the quality attribute that they convey.
Nouns + Verbs + Adjectives	By focusing just on functionalities, qualities and verbs it is possible to discover similar problems or comments at different customers or products.	
Verb + Noun	Extracting verbs and nouns gives information about system functionality and is often used in requirements extraction. Ideally, they indicate functionality and uses of these functionalities in software user feedback.	Verbs and nouns, especially when extracted in pairs, are indicators of the subjects and use of sentences or reviews. Extracting such pairs, ranking them on frequency, novelty or information gain and placing them in a requirements format is a good start to requirements generation (Bhowmik, Niu, Savolainen, & Mahmoud, 2015).

TABLE 8: COMBINATIONS OF POS TAGS

6.3 ALGORITHM SELECTION

We evaluate two clustering algorithms, each representing a family of approaches. We use the *supervised* algorithm K-Means, in which the user states as input how many K-centers the algorithm should use to execute the clustering. We use K-Means because of the wide-spread use in academia, ease of use and run-time efficiency(Steinbach et al., 2000). An advantage of unsupervised clustering is that it might yield clusters that are better suited to represent the diversity in the dataset because we do not specify the amount of clusters to be generated beforehand. This is why we also use Meanshift (Comaniciu & Meer, 2002), an unsupervised clustering algorithm where random cluster centroids are assigned and the algorithm further clusters the tickets according to the shift in cluster mean as more tickets are considered. Using these two algorithms gives us a baseline for our experiment. For a more detailed look in the python code, see appendix I-B and I-C.

6.4 PAIRWISE EVALUATION

We are evaluating a supervised and an unsupervised clustering algorithm, as explained in 6.3 Algorithm selection. The supervised algorithm K-Means is set to generate five clusters. We chose five because this showed to be a number of clusters where the items of feedback are often equally related upon visual inspection. Choosing a relative low number of clusters for the selected 100 tickets used in the experiment also greatly simplifies the execution of the experiment. The unsupervised clustering algorithm Meanshift has generated four, six, seven and nine clusters respectively. Considering the four different POS tag combinations that are selected to be evaluated:

1. Named Entities and Nouns
2. Named Entities, Nouns and Adjectives
3. Verbs and Nouns
4. Verbs, Nouns and Adjectives

A test dataset of 100 items of feedback has been analyzed and clustered in different ways. These 100 items are the most recent 100 tickets in the dataset for the YouPP software program. We use pairwise comparisons to evaluate the effectiveness of the methods used. This selection of 100 items already gives us (n * n-1) / 2 = 4950 possible combinations per distribution of clustering algorithm and POS tag and this number is also multiplied by 8, for the number of different distributions. Therefore, it is crucial that we apply sampling so that we arrive at a workable number of tickets.

To test these similarity metrics we divide all pairs in different areas based on their assigned cluster. We will separate pairs that are in the same cluster and those that are in a different cluster. For instance, if we have three clusters:

A = All pairs in cluster 1

B = All pairs in cluster 2

C = All pairs in cluster 3

D = A vs (B + C)

E = B vs (A + C)

F = C vs (A + B)

The two clustering algorithms and four POS tag combinations have rendered eight distributions of 4950 pairs of tickets, that is because we test the four different POS tag combinations with both algorithms. These are clustered in a different way in each distribution. The result of the POS-tagging, clustering and division into area's can be seen in Table 9, in the algorithm this is called a matrix. Note that every rom sums up to 4950 pairs.

Algorithm	Pos-Tag	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
K-Means	NE + N	210	210	78	528	66	868	643	602	1089	656								
K-Means	NE + N + ADJ	66	946	55	231	55	656	750	638	1061	492								
K-Means	V + N	595	171	153	105	78	1093	511	838	804	602								
K-Means	V + N + ADJ	105	91	120	153	666	788	738	379	788	1122								
Meanshift	NE + N	190	351	78	120	78	55	637	760	602	772	669	638						
Meanshift	NE + N + ADJ	55	45	66	15	136	28	66	78	55	431	490	367	202	692	370	683	533	638
Meanshift	V + N	210	78	1378	78	635	602	1239	730										
Meanshift	V + N + ADJ	36	465	28	78	91	91	55	446	918	437	303	544	740	718				

TABLE 9: PAIR DISTRIBUTION AFTER POS-TAGGING, CLUSTERING AND DIVISION INTO AREAS.

We want to test all distributions evenly and use the least amount of items of feedback. The selection algorithm can be found in appendix I-D and works as follows if we want to select 100 pairs for our experiment, in pseudo code:

Do 100 times:

1. *Select smallest area in new matrix*
2. *Check for preferred pair in corresponding area in old matrix*
    a. *If available →* ***select pair***
    b. *if not available →* ***select random pair***
3. *Add both item numbers to preferred pairs list*
4. *Find all area's that contain selected pair in old matrix*
5. *Move all 8 instances of the pair to the corresponding area's in the new matrix*
6. *Delete all instances of the pair in the old matrix*

We strive for an optimal balance and selecting pairs in this way guarantees:

1. All distributions are evaluated equally
2. All generated clusters have a similar amount of pairs that are in the same cluster and in a different cluster
3. We use a minimal amount of item of feedback in our experiment from the 100 items we clustered.

In Table 10 we see the distribution of pairs after we selected 100 unique pairs to be evaluated, every rom sums up to 100 pairs.

Algorithm	Pos-Tag	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
K-Means	NE + N	6	9	8	19	6	9	17	8	10	8								
K-Means	NE + N + ADJ	6	26	7	10	6	8	9	7	15	6								
K-Means	V + N	17	7	7	7	8	10	17	10	9	8								
K-Means	V + N + ADJ	5	6	6	9	17	13	12	11	11	10								
Meanshift	NE + N	8	7	8	7	7	7	9	14	8	11	7	7						
Meanshift	NE + N + ADJ	5	5	5	5	6	6	5	5	7	5	6	5	5	6	6	6	5	7
Meanshift	V + N	17	8	32	6	13	8	11	5										
Meanshift	V + N + ADJ	6	14	6	8	6	6	5	6	7	6	9	6	9	6				

TABLE 10: PAIR DISTRIBUTION AFTER SAMPLING

Table 11 is used to illustrate the number of tickets used by our algorithm when considering different numbers of pairs to be evaluated. So when we select 100 unique pairs of tickets, about 55 unique tickets are used in this distribution. The number of used items is an approximation of the number of used items because we randomly select a starting pair and multiple iterations of the algorithms yield a different number of unique used items accordingly, hence the tilde symbol used as a unary operator.

Number of used pairs	~Number of used items
20	24
50	45
100	55
200	80
400	87

TABLE 11: RELATION BETWEEN AMOUNT OF PAIRS USED AND ITEMS OF FEEDBACK

6.5 EXPERIMENTAL PROTOCOL

These 100 pairs are randomly split up into 5 sets of 20 pairs that are tested by our 10 participants so all tickets are scored both by our professional pool and our student pool. They rate each relation on a 4-point Likert scale from *not related at all*, *somewhat not related*, *somewhat related* to *highly related*. An example of the survey with dummy data can be found in appendix II. The participants will not be told that the pairs are clustered according to the co-occurrence of words or that these are determined using POS tags. This is to ensure that participant evaluate pairs according to their own notion of similarity. After the test is complete, we will discuss the results with the participants to determine the rationale behind their decisions. This could help us determine if there is room for improvement in both our NLP methods and our clustering decisions.

After the participants completed their evaluation, they are asked highlight their rationale for two pairs that they rated at *highly similar*, or if none exist *somewhat related*. Participants are specifically asked only to explain why they believe tickets to be related and not why they consider them to be not related. This is because we are interested in which words or sentences convinced them to rate the pair as related. We want to determine if our intuition matches our participants or if they focused on totally different concepts. This will help us identify correct strategies for future work in NLP and requirements engineering through qualitative research.

6.6 METRICS

The objective in clustering evaluation is to formalize goals in the assignment of clusters. Typically a goal in clustering in to reach a high level of intra-cluster similarity. When a cluster has a high level of intra-cluster similarity, all elements in the same cluster are similar to each other. Similar to this, when a cluster has a low level of inter-cluster similarity, all elements that are in a different cluster are dissimilar, as shown in Figure 14. This is called an internal criterion for the quality of clustering. The easiest way to evaluate clusters is to compare the results to a predefined truth set. As there is no truth set deductible for our dataset, we rely on human judgement to evaluate cluster scores.

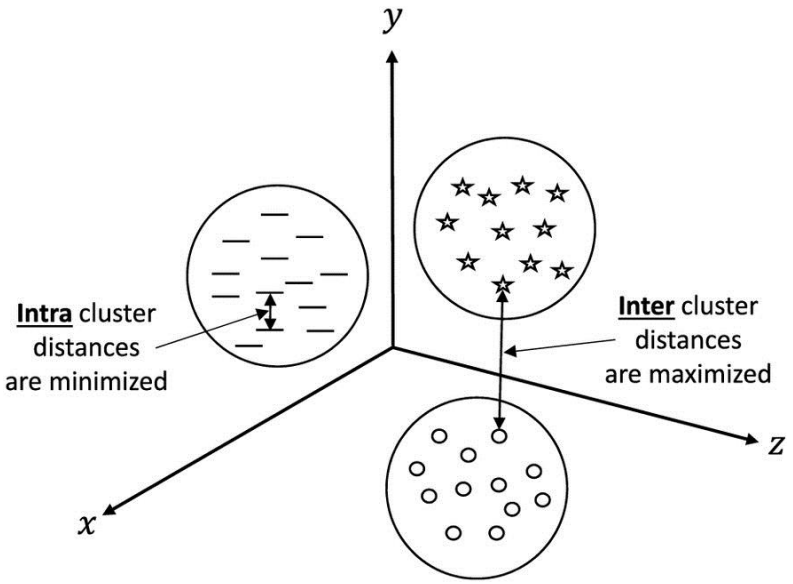

FIGURE 14: DISTANCE METRICS FOR CLUSTER EVALUATION (GALVAN-NUNEZ & ATTOH-OKINE, 2016)**.**

6.7 STATISTICAL EVALUATION

Statistical evaluation of the experiment is based on a few assumptions and with this experiment setup it has some inherent flaws. The data obtained from this experiment is considered as ordinal data because it is essentially Likert-type data in which the distances between answers are not even or measureable, statistics such as the average or standard deviation are not relevant because of this. Non-parametric tests are the best fit in this instance. We use two, as described in 6.7.1 and 6.7.2.

The last assumption that we make over the data is that the data comes from independent samples. This assumption is made because the samples do not influence each other. The data does come from the same basic 100 pairs and is multiplied in the following manner:

1. Both students and professionals rate the same 100 pairs.
2. Each pair is in 8 different distributions.

This means that 100 unique pairs translate to (100*2*8=) 1600 unique data points. The fact that we only used 10 participants to generate these points is another reason for us to choose non-parametric tests. We must be careful in drawing conclusions from these statistical tests however, as each of the 8 distribution have the same pairs and the distribution of ratings when we combine the pairs that are clustered in the same cluster and those clustered in a different cluster. The differences between distributions only arise when we separate these pairs. This complicates testing for statistical significance greatly as the ratings in the distributions are nearly identical.

6.7.1 WILCOXON RANK SUM TEST

This non-parametric test evaluates the sum of the ranks assigned to the data points. If we compare two distributions, it ranks all entries and gives either a positive or negative rank in the list and orders the list accordingly. The output is stated as an N = .., W = .., and p-value, where N = the number of data points used in both distributions as (distribution A * distribution B), W = the number of data points in one distribution that are expected to have a positive rank when compared the other distribution and the p-value is the confidence of the test in that number. Generally speaking, $\frac{N}{W} \cong 0.5$. We take the p-value as a measure of variance in the distributions. We do not use the Wilcoxon Signed-Rank Test because the data points in our dataset are not paired.

6.7.2 KRUSKAL-WALLIS TEST

The Kruskal-Wallis test is the non-parametric equivalent of the ANOVA-test in that is compares multiple distributions at the same time. It does so by comparing the means and testing with traditional statistics such as the chi-squared, degrees of freedom and the p-value whether one of the distributions differs significant statistically. We use this to determine the effect of POS tags on ratings. Like with the Wilcoxon Rank Sum Test, we interpret the p-value as a measure of variance.

6.8 QUALITATIVE DATA

After participating in the pairwise evaluation, each participant is asked to highlight the words or sections that led them to grade a pair as *'Highly related'*. The participant is asked to highlight a maximum of two tickets as this would give a representation of the type of words that they deemed most important. The participants are also asked about their rationale. The results of this additional testing can complement the results obtained by the main experiment and it could provide some interesting information about the decisions made by the participants. It can also be used to illustrate the differences between the students and the professionals participating in the experiment.

CHAPTER 7. RESULTS

This chapter consists of the results of the experiment in 7.1 and the results of the dashboard evaluation in 7.2. In Section 7.1.1 we describe the characteristics of the results obtained from our experiment, in 7.1.2 we discuss the ratings given by the professional participants, in 7.1.3 we discuss the ratings given by our student participants, in 7.1.4 we present the evaluation of the clustering algorithms, in 7.1.5 the performance of the POS tag combinations are discussed, in 7.1.6 we discuss the results of the qualitative analysis interviews with our participants and in 7.2 we evaluate the dashboard. All results and corresponding R-code can be found in appendix III.

7.1 EXPERIMENT RESULTS

7.1.1 DATASET CHARACTERISTICS

If we examine the totals of the evaluations given to the pairs by our test subjects, we see that 48% of the pairs are rated as *Not Related at all* and if we include *Somewhat not Related*, this figure grows to 68% of our data being regarded as *Not Related*. It is important to take this into account, as our metrics are dependent on correctly grouping or separating tickets and this disparity skews ours results. It is more likely that our clustering successfully separated tickets that are unrelated when over two-thirds of the pairs are unrelated. This imbalance can be seen in Figure 15. While the figure shows the total amount of tickets, the distributions for professionals and students follow the same pattern.

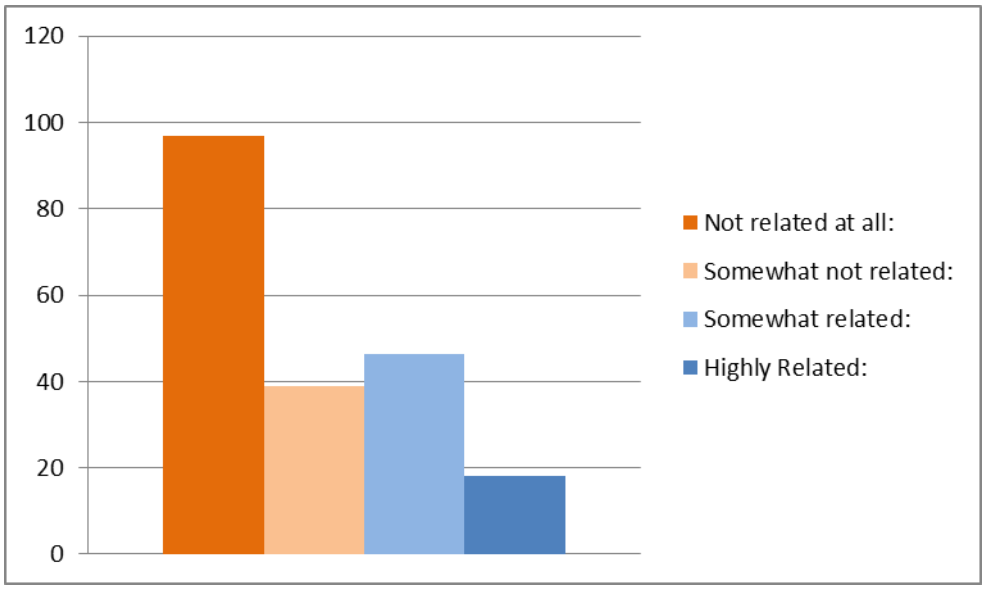

FIGURE 15: DISTRIBUTION OF RATINGS ACROSS DATASET.

We perform a Wilcoxon Ranked Sum test over the distribution of the ratings. We combine the pairs that are in the same cluster and the pairs in different clusters. We test the effect of the clustering on the ratings. The Wilcoxon Ranked Sum Test returned N=639991, W=397300 and the p-value is < 2.2e-16 so we can consider the two samples from a different population. From this, we can conclude that there is a statistically significant difference between pairs clustered in the same cluster and in a different cluster. Note that for all our test, we use a significant p-value of 0.05.

If we split the data by pairs that are in the same or in a different cluster and we run the Wilcoxon Ranked Sum Test on the difference in rating between students and professionals given to pairs in the same cluster, we see that N=160930, W=86388 and the p-value is 0.06284. If we run the same test on the pairs from a different cluster, the results are N=158530, W=72691 and the p-value is 0.02134. The W-score is the sum of ranks. If we were to draw random samples, how many of them would be greater in population A then in population B. When W is the half of N, the populations are generally more similar. We see in these scores and p-value that students and professionals tend to agree more when rating pairs that our algorithms have placed in the same cluster and tend to disagree more on the rating towards pairs in a different cluster. This is the information obtained from the test, even though no statistical significance has been found.

7.1.2 PROFESSIONALS

When examining the results for professionals, we look at pairs grouped in the same cluster and different clusters separately, as we are interested in the performance of the clustering algorithms and POS tags concerning grouping and separating tickets. In Figure 16 we can see how professionals rated tickets in the same cluster. We see that the in three combinations of the K-Means algorithm that are more than 50% related, while Meanshift only has one distribution that breaks 50%. Still, all distribution are between 42% and 56% related.

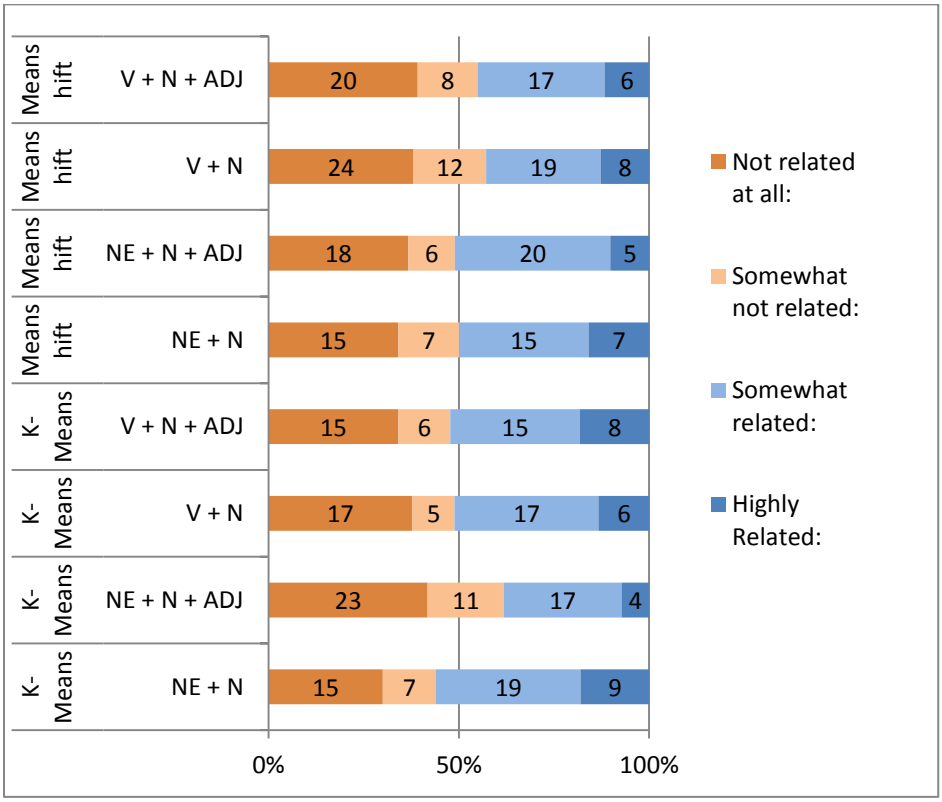

FIGURE 16: RATING BY PROFESSIONALS FOR PAIRS IN THE SAME CLUSTER.

The distribution for pairs in a different cluster shows a different picture in Figure 17. The objective here is to separate tickets so we see that a high level of unrelated pairs means that the distribution is successful. The most effective at this is the Meanshift algorithm combined with the Verbs and Nouns combination, which returned only unrelated tickets in 92% of the cases. The worst performance is the K-means algorithm with Named Entities, Nouns and Adjectives with 80%.

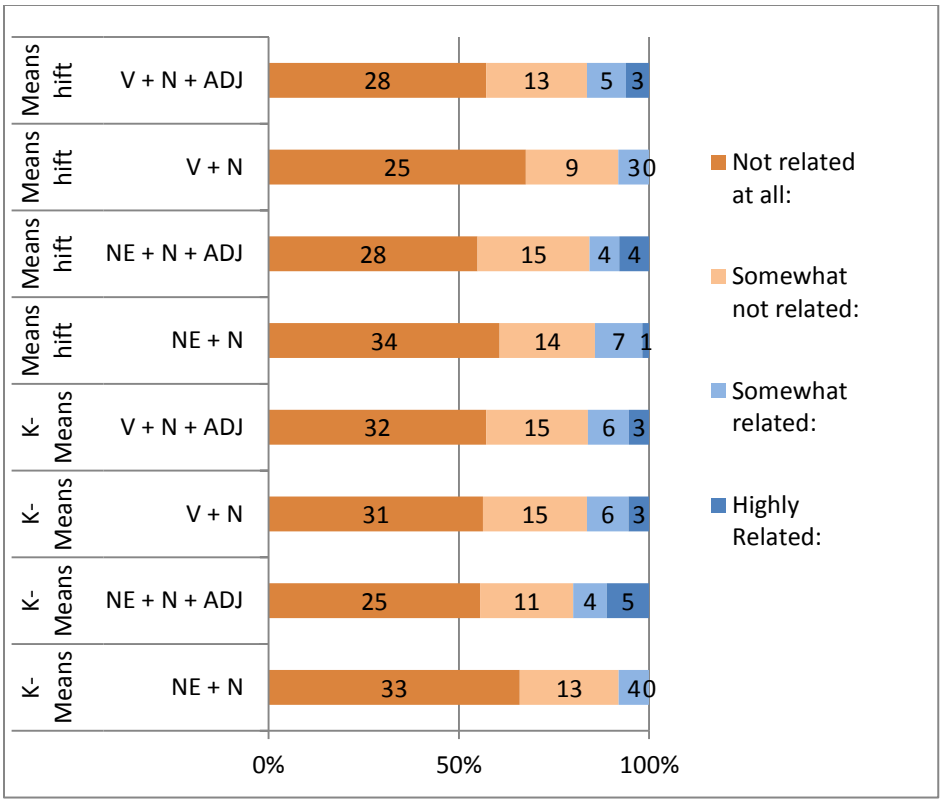

FIGURE 17: RATING BY PROFESSIONALS FOR PAIRS IN A DIFFERENT CLUSTER.

In order to test the ratings given by professionals and students alike, we split the data on pairs clustered in the same cluster and in a different cluster because if we would combine the two, each distribution would be identical. We use the Wilcoxon Ranked Sum Test again to test the effect of the clustering algorithm on the rating given by professionals to pairs in the same cluster. We see that N=40158, W=20538 and the p-value is 0.6771. When we run the test on the pairs in a different cluster, we see that N=39758, W=20134 and the p-value is 0.802. This indicates that the algorithm used was more of a factor for professionals when rating pairs in the same cluster than pairs in a different cluster, as the distributions are less uniform in the first category.

We use the Kruskal-Wallis test to test the four distributions of POS tags to determine for professionals how much they were influenced by these factors when rating pairs. We again apply the split on same or different cluster. For pairs that are in the same cluster, the test returns a chi-squared value = 2.6495, df = 3 and p-value = 0.4489. For pairs in a different cluster, chi-squared = 2.5193, df = 3 and p-value = 0.4718. This means that there is little difference between the effect of the POS tags on the ratings of professionals given to the pairs in the same or in a different cluster.

7.1.3 STUDENTS

The results for students sketch a similar image, in Figure 18 we see that all distributions are close together but unlike ratings given by the professionals, none of the distributions are more than 50% related. The Meanshift algorithm with the Named Entity and Nouns POS tag combination scores best at 47% related. There is also less variation in the ratings given to different distributions.

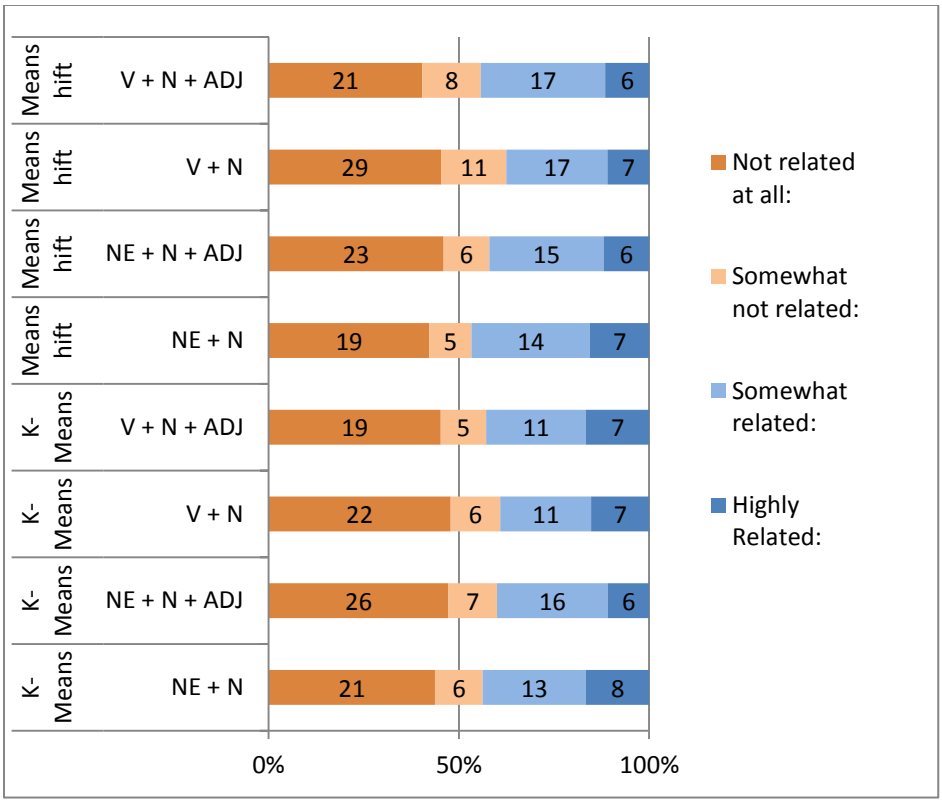

FIGURE 18: RATINGS BY STUDENTS FOR PAIRS IN THE SAME CLUSTER.

If we examine how students rated pairs that are from different clusters, we see that they rated these pairs as more related than the professionals, as shown in Figure 19. The scores range from 66% to 78%, compared to 80% to 92% for professionals. This indicates that students disagree with our clustering algorithms more than our professional test subjects. As with the professionals, the Meanshift algorithm with Named Entities and Nouns performs best.

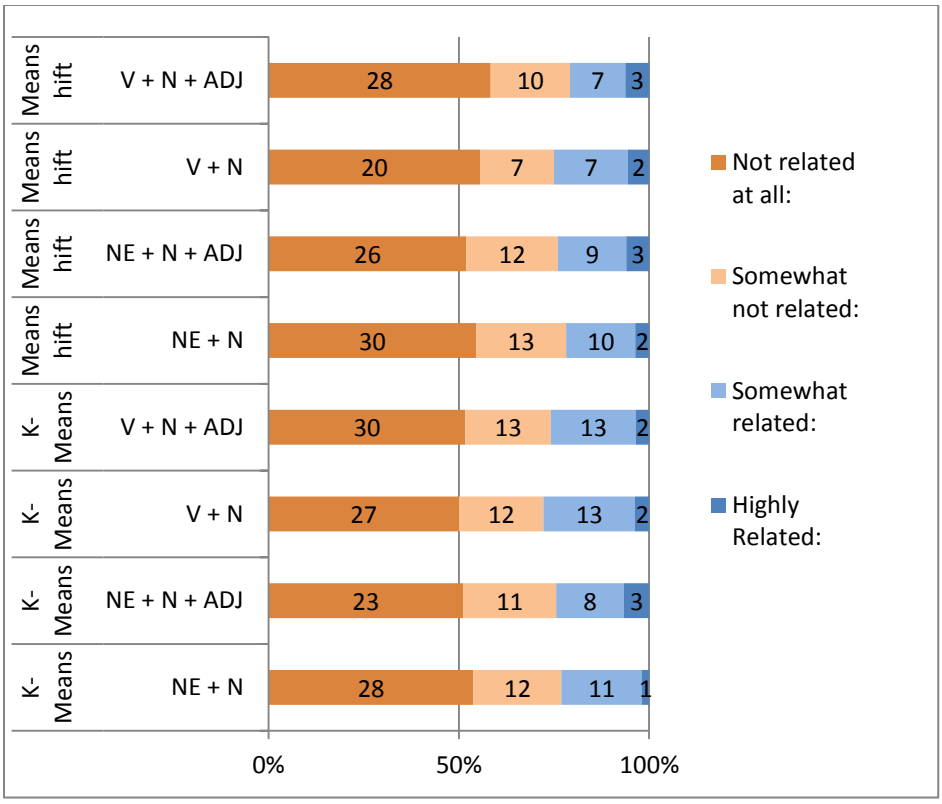

FIGURE 19: RATINGS BY STUDENTS IN A DIFFERENT CLUSTER.

After performing the Wilcoxon Ranked Sum Test, we see that the effect of the algorithm over the rating given by students to pairs in the same cluster is N=40301, W=19988 and the p-value = 0.8825. Over the pairs in a different cluster, the statistics are N=39501, W=20400 and the p-value = 0.5338. This large difference indicates that the algorithm used was more of a factor for students when rating pairs a different cluster than pairs in the same cluster. We can therefore conclude that the selected algorithm has more effect on students when separating pairs and more effect on professionals when grouping pairs.

If we look at the results of the Kruskal-Wallis Test, we see that the effect of the POS tags on the ratings of students that rated pairs in the same cluster, we see that chi-squared = 1.0217, df = 3 and p-value = 0.796. For pairs in a different cluster, this is chi-squared = 2.6495, df = 3, p-value = 0.4489. This means that, contrary to the ratings given by professionals, there is an effect of the POS tags on the ratings of student, most notably in pairs from a different cluster.

7.1.4 CLUSTERING ALGORITHMS

We evaluate the relative performance of the algorithms (supervised vs. non-supervised clustering) in two ways. First, we calculate the metrics as shown in Table 12 and Table 13. We see in the tables that the clustering algorithms scored virtually identical on the metrics.

Algorithm	Intra-cluster relatedness	Inter-cluster relatedness
K-Means	0,454	0,797
Meanshift	0.449	0,817

TABLE 12: ALGORITHMS PRECISION BASED ON *SOMEWHAT RELATED + HIGHLY RELATED*.

Algorithm	Intra-cluster relatedness	Inter-cluster relatedness
K-Means	0,143	0,552
Meanshift	0.124	0,573

TABLE 13: ALGORITHM PRECISION BASED ON *HIGHLY RELATED*.

Second, we perform a Wilcoxon Signed-Ranks test over the distribution of the ratings. We separate the pairs that are in the same cluster and the pairs in different clusters. We split the data on algorithm to compare the effect that the algorithms have on the distribution of ratings. The Wilcoxon Signed-Ranks test over the pairs clustered in de same cluster returned that the results cannot be considered statistically significant with N=160930, W=81118 and p-value of 0.8336. The Wilcoxon Signed-Ranks test over the pairs clustered in a different cluster returned that the results cannot be considered statistically significant with N=158530, W=81214 and a p-value of 0.5036. From this, we can conclude that there is no statistical difference between K-Means and Meanshift as clustering algorithms. We can say however that the algorithms differ more when it comes to separating tickets than grouping tickets because the p-value, taken as a measure of variation, is lower when comparing the separation of tickets than grouping tickets.

7.1.5 POS TAGS

If we calculate the defined metric for the different POS tags we see that the combination of Named Entities and Nouns has the best score in both intra-cluster relatedness and inter-cluster relatedness. These are shown in Tables 14 and 15. Clearly, the combination of nouns and Named Entities scored the best on all metrics.

POS tag	Intra-cluster relatedness	Inter-cluster relatedness
NE + N	0,492	0.831
NE + N + ADJ	0,426	0,791
V + N	0,422	0,802
V + N + ADJ	0,46	0.801

TABLE 14: INTRA- AND INTER-CLUSTER RELATEDNESS OF THE POS TAG COMBINATION BASED ON *SOMEWHAT RELATED + HIGHLY RELATED*.

POS tag	Intra-cluster relatedness	Inter-cluster relatedness
NE + N	0,166	0.587
NE + N + ADJ	0,1	0,534
V + N	0,129	0,566
V + N + ADJ	0,143	0.559

TABLE 15: INTRA- AND INTER-CLUSTER RELATEDNESS OF THE POS TAG COMBINATION BASED ON *HIGHLY RELATED*.

A Kruskal-Wallis test has been performed over the different POS tag combinations. For the pairs clustered in the same cluster, we see that the chi-squared value is 3.2679, df = 3 and the p-value is 0.3521. For the pairs clustered in different clusters we get a chi-squared value of 1.8721, df = 3 and a p-value of 0.5994. Based on these tests, we can state that there is no statistically significant difference between the ratings of the pairs and the used POS tag.

There is however basis for a Post-Hoc Test over the pairs in the same cluster. We pick this distribution to test because it has the lowest p-value and therefore yields the best chance of getting interesting results. We use the Dunn Test with multiple Kruskal-Wallis tests to determine which of the distributions is most deviant from the rest. It uses the Kruskal-Wallis test and compares all combinations of the four distributions, normalizes the p-values according to the Benjamini-Hochberg adjustment(Benjamini & Hochberg, 1995) and ranks the combinations according to their adjusted p-value. This adjusted p-value indicates under which p-value each comparison would be deemed statistically significant in regards to the entirety of the distributions. The Z-score is also calculated, this score indicates how many standard deviations the projected mean is from the group mean. From these results, shown in Table 16, we can conclude that the Named Entities and Nouns distribution differ most from the Named Entities, Nouns and Adjectives distribution. Also, the comparison between the Named Entities and Nouns distribution and the Verbs and Nouns distribution show significant differences.

Comparison	**Z-score**	**P-value**	**Adjusted P-value**
(NE + N) - (NE + N + ADJ)	16.483.138	0.09928828	0.5957297
(NE + N) - (V + N)	13.797.139	0.16767475	0.5030243
(NE + N + ADJ) - (V + N)	-0.2933375	0.76926420	0.7692642
(NE + N) - (V + N + ADJ)	0.6601919	0.50913069	0.6109568
(NE + N + ADJ) - (V + N + ADJ)	-0.9745531	0.32978196	0.6595639
(V + N) -(V + N + ADJ)	-0.6985280	0.48484704	0.7272706

TABLE 16: POST-HOC TEST FOR THE POS TAG COMBINATIONS IN THE SAME CLUSTER

7.1.6 QUALITATIVE ANALYSIS

After rating the pairs, the participants were asked to re-examine two pairs that they marked as *Highly Related*, if none existed, the were asked to re-examine pairs marked as *Somewhat Related*. They were asked to highlight words or sections that led to believe the tickets are related. No further instructions were given to them. In Figure 20 we see that professionals mostly looked at Nouns and Named Entities. They stated that the same nouns appearing in both texts mostly led them to believe the tickets were related. But some of them also recognized Named Entities such as names of software modules and systems that they know are related or in the same category. In a few cases, they evaluated a pair as being related by stating they addressed the 'same type of error', using adjectives and nouns but not using any words that are the same.

Some other examples of how professionals related tickets to each other are that they could connect verbs to functionality, they connected identification numbers to software packages and they were able to match nouns to verbs, *'making adjustments'* and *'mutation'* for instance.

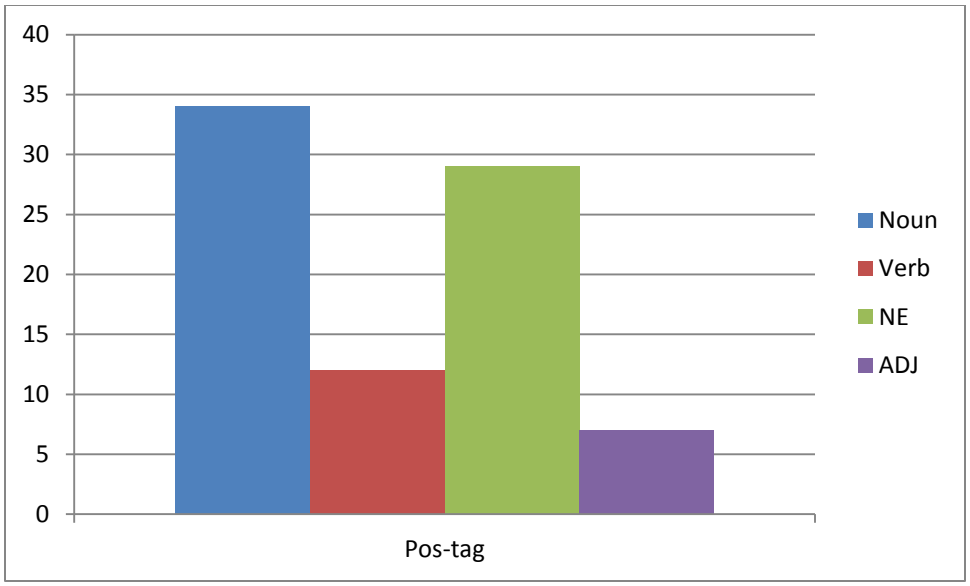

FIGURE 20: TAGGED POS TAGS BY PROFESSIONALS

The words that students identified are much more skewed towards Nouns, as shown in Figure 21. Most of them stated that they only looked at similar nouns that appeared in both texts, although some of them identified a 'same type of error'. It must be noted here that the students had a harder time understanding most of the text because they are reading actual bugs and complaints about software products directed at software professionals without much context.

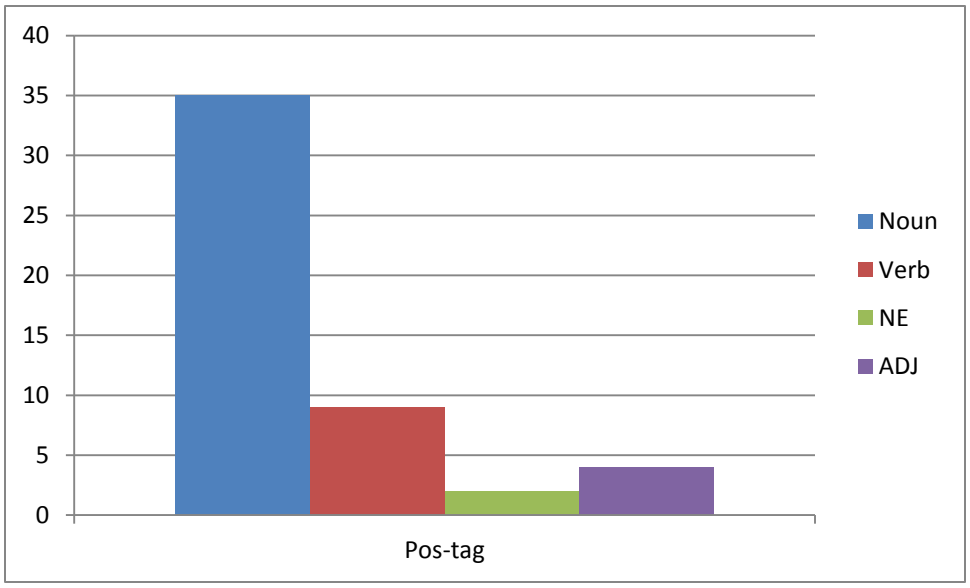

FIGURE 21: TAGGED POS TAGS BY STUDENTS

7.2 Dashboard evaluation

In this section we describe the rationale behind the dashboard evaluation and present the findings of a cognitive walkthrough performed with two professionals at the case study company.

7.2.1 Usability evaluation

The evaluation of the dashboard is conducted because we are interested in knowing if the design principles we used and the techniques applied can potentially help requirements analysts in navigating large amounts of user feedback. The best way to evaluate a tool would be through a heuristic evaluation (Nielsen & Molich, 1990), where experts first assess the tool based on a few actions which they have to conduct and evaluate. Second, the expert rates the tool based on a list of usability heuristics appropriate for the tool and give violations of these heuristics a rating of seriousness. After the evaluation concludes, the scores are measured for agreement between the experts and improvements for the tool are formulated.

However, since our dashboard is not designed to be a fully functioning user interface but rather a dashboard that allows browsing a collection as its lone purpose, we believe that a cognitive walkthrough will suffice. In a cognitive walkthrough, experts are not asked to rate an interface based on usability heuristics but rather perform a task or answer questions from the perspective of the user. In this case, the cognitive walkthrough was performed at the case study company with two requirements analysts at the same time. They had the opportunity of browsing a collection of 1124 items of feedback from a single software product that they were familiar with. They were able to do this for 20 minutes and were presented with different amounts of generated clusters. The task they were given was to see if the dashboard allowed them to browse the data more easily and they were asked about their opinion regarding design decisions.

7.2.2 Cognitive walkthrough

The experiences and comments of the two requirements analysts can be categorized in three sections.

Ease of use

Neither of the participants was familiar with PowerBI dashboards, the enabling technology for generating interactive charts, yet they found it easy to navigate between the two pages. Furthermore, the interactive visuals were positively rated, especially the circles that represent the clusters that change color and size when the participants hoovered their mouse over them. A little more difficult was changing the dashboard to display a different result of the clustering as these had been prepared in advance by the author and had to be loaded into the dashboard, which means connecting the visualizations to another pre-loaded dataset. The participants agreed that there has to be a connection with the company database in place with some form of Extract-Transform-Load procedure.

Task Execution

The most amount of clusters generated was 15, so that means that the cluster often contained more than a hundred items of feedback, which are considered mostly unrelated, as shown in Figure 15. This complicated the task for the participants as they were able to see the main words and topics of the clusters, which helped them to grasp the main themes that the submitters of the feedback were talking about, but they were unable to drill down to a more concrete topic or problem.

The participants suggested a filtering mechanism, where the user can decide which, and how much, items of feedback to load into the dashboard. They proposed to use the human generated classification of this, as described in Table 4. The hierarchical clustering as shown in Figure 7 could also be an option.

INFORMATION PRESENTATION

The participants said that they would have liked to see more information on the screen, such as the job titles of the senders or which of their customers submitted the feedback. They saw the value in the word cloud of cluster topics but they found this to be too generic and they recommended just giving the top five most important topics with an attached score.

CHAPTER 8. CONCLUSION

Based on the results of the literature study, the use case study, dashboard evaluation and experiment, we can draw the following conclusions about our research questions:

Main research question: *What are effective automated techniques that can assist requirements analysts in organizing large sets of user feedback?*

We used a combination of POS tags and clustering techniques to automatically organize feedback items and presented these clusters to professionals in the HR software domain. We created a dashboard to visualize clusters and this allowed us to obtain preliminary insights into the perceived usefulness of these concepts and techniques in a corporate setting. The results show that the dashboard is received well but it requires more interactivity and/or machine learning techniques to make it truly useful in the requirements engineering domain. The effectiveness of the POS tags and clustering techniques, as well as a comparison between the preferences of professionals in the domain of HR software and those of IT-students are tested in an experiment. We see that the measured differences between supervised and unsupervised clustering methods are negligible in the small-scale experiment that we conducted. As far as the POS tags are concerned, our preliminary results seem to indicate that a combination of Named Entities and Nouns perform the best for the purpose of relating feedback items out of the combinations used in this research although no statistically significant level was reached.

RQ1: *What information contained in user feedback is interesting for requirements analysts?*

From our review of the literature in 3.3, we have determined that user feedback is often mined for features, opinions, relationships and sentiment. Most of these are based on online app reviews given by users, but this thesis is situated in the software development domain and therefore is a great opportunity to study user feedback in a traditional software development environment. When discussing possible use cases for automatically analyzing or organizing this feedback with requirements analysts, their focus was mostly on identifying gaps in their own analysis, exploring issues over the whole dataset and adding prioritization to sprint backlog items based on textual features. Automated ways to inform their customers were also explored. The information need of a requirements analyst is dependent on the maturity of the RE process. From our experiment, we can state that professionals that are familiar with the software mostly look at Nouns and Named Entities, this is confirmed by the metrics that we used in the experiment and the interviews we conducted. The results that we obtained cannot be considered statistically significant however. From the interviews conducted with participants after the experiment, we can conclude that professionals familiar with the domain are much more likely to relate documents to each other based on Named Entities compared to students. They also used verbs and adjectives more often than the student participants.

RQ2: *Which techniques can we use to automatically organize unstructured user feedback?*

Automatically organizing large amounts of user feedback through clustering is beneficial for requirements analysts because it can help the analyst browse the dataset with more ease and therefore there is less chance of an information overload. It also gives more information about the size, scope and context of an item of user feedback. Clustering is used in this thesis combined with POS tags to test the effect of these tags on the clustering, we tested the effectiveness of two clustering algorithms when presented with items of feedback that contained only the

selected POS tags. We tested a dashboard that uses clustering to organize a dataset of user feedback with professionals familiar with the domain. Our preliminary results seem to indicate that clustering items of feedback improves the analysts' understanding of the dataset as a whole but that the additions of machine learning techniques show promise to improve this process.

RQ3: *To what extent is our method useful for requirements analysts?*

RQ3.1: *Which concepts are most informative when analyzing user feedback?*

From the four combinations of POS tags that we used, the preliminary results show that Named Entities and Nouns perform the best. This indicates that when requirements analysts are organizing sets of user feedback, these are the concepts they are looking at, and so should an automated method accordingly.

RQ3.2: *What algorithms are most effective at grouping related user feedback and at separating unrelated user feedback?*

For this RQ we used a supervised and an unsupervised algorithm, K-means and Meanshift respectively. From our preliminary results, there is no statistical difference between the two as far as performance in the experiment is concerned. The real distinction between the two is in practice, as the unsupervised method would give users of a feedback-browsing tool information about the optimal number of clusters while the supervised method gives users control over this process.

RQ3.3: *What is the perceived value such a method for requirements analysts?*

We can state that the scientific community is actively researching ways to automate the processing of feedback and is looking for tools or ways to do this and this method contributes towards this cause. As far as the requirements analysts that have participated in this research, we can state that they were interested in such a solution but that it holds little practical use at the moment. For its use to increase, the dashboard will have to be further developed.

CHAPTER 9. LIMITATIONS & FUTURE WORK

In this chapter we discuss the threats to validity of the research we conducted and we present the most interesting avenues for future work.

9.1 LIMITATIONS

To discuss any limitations of this research we use the widely accepted types of validity threats: conclusion, internal, construct and external (Wohlin et al., 2012). This is important because empirical research is subject to a multitude of possible threats and validity analysis should be conducted by every researcher, preferably through a common terminology (Feldt & Magazinius, 2010).

9.1.1 CONCLUSION VALIDITY

Conclusion validity is concerned with the relationship between treatment and outcome and is important when dealing with statistically significant results. As discussed in section 6.7, the experimental design is such that we test multiple factors based on 100 pairs but all distributions are therefore so closely related that we rarely achieve statistical significance. Therefore, p-values are often used as measures of variety and conclusions are based on adjusted p-values. This is the main conclusion validity threat, but others could also play a role.

For instance, we base our conclusions on a very small sample size of n=10 participants. In subsequent studies, this number should be increased. Also, because the participants gave their answers on a Likert-type scale we are left with non-parametric tests. These are not as powerful as parametric tests and they lack extensive descriptive statistics such as mean and standard deviation, making it harder to draw solid conclusions.

9.1.2 INTERNAL VALIDITY

This validity threat is concerned with the causal relationship between the factors in the experiment and the outcome. We are interested if there are ant factors in the research that might have influenced the outcome. There are a couple of arguments here. The selection of participants was not random for instance. The professionals were selected because of their availability and familiarity with the subject matter, which strengthens their contribution to the dashboards evaluation but it might influence their rating of the relatedness of items of feedback. The students participating in this experiment are selected because of their knowledge of IT, as these items of feedback can be quite technical and a better understanding could lead them to make more informed decisions because they are somewhat familiar with the terminology. Still, they have similar backgrounds in Information Science and should be noted for that may introduce a bias.

For the dashboard evaluation, there could have been a control group testing how to navigate the collection of feedback without the use of the dashboard. But noting that these items of feedback are electronic documents in a system, the comparisons would inadvertently be diverted towards a comparison of two tools to read these items. Also, this would be a daunting task for the participants with no clear objective or benefit. Lastly, there is no way to measure to what extent the results of the dashboard evaluation are attributed to the visualizations of PowerBI or to the visualization concepts described in this thesis.

There is also the risk of the researcher influencing the participants during the experiment, which we have tried to mitigate by not telling participants what to highlight or what was expected of them during the qualitative part of the experiment.

9.1.3 External validity

External validity threats shall be mitigated to improve the generalizability of the research. The first threat that would come the mind in the Dutch language of the items of feedback, but because we use POS tags and not actual words we can consider the language as a non-factor. However, it means that the classifier trained on the Dutch training set cannot be used on an English test set. The fact that these are the results of just a single case study is a more valid argument. If more reliable conclusions have to be drawn, additional case studies will have to be included.

9.14 Reliability

This threat concerns the consistency and repeatability of the research. Of which we can say that all methods used and procedures followed have been well described in the thesis but because of the varied nature and characteristics of datasets of user feedback, and exact replication looks to be difficult. This is enhanced by the inclusion of the dashboard in the research, which is even more difficult to emulate in subsequent studies because a similar artifacts will have to be produced. Also, the dataset is not public and no comparable datasets containing detailed bugs, feature requests and complaints are available online to the best knowledge of the authors.

9.2 Future work

In this section we outline a few future research opportunities stemming from this thesis. In the spirit of the projected paradigm shift towards a more data-centered approach to RE (Maalej et al., 2015), we see the need for more automated tools that process feedback. These studies could benefit from the lessons learned in this thesis about directed clustering, specifically targeted at types of words interesting for requirements analysts in a software development environment. Although our results prove to be preliminary a case study involving multiple companies could give us definite answers. Apart from increasing the scale of the case study, it could also be worthwhile to test these concepts in other domains of software engineering.

As far as the dashboard is concerned, a few additions would greatly improve the effectiveness of the approach. First, a direct connection to a data repository of feedback that streams feedback directly into the dashboard increases the use for requirements analysts. Giving the user control over which data is clustered and analyzed and using human or automatically generated classification for this could provide additional value. This measure of interactivity could be improved by letting users give priorities to certain words that they find important and re-cluster items of feedback.

REFERENCES

Aurum, A., & Wohlin, C. (2003). A . Aurum and C . Wohlin , " The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process ", Information and The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process. *Ist*, *45*(14), 945–954.

Badwal, J., & Raperia, H. (2013). Test Case Prioritization using Clustering. *2013 IEEE Sixth International Conference*, 488–492. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6569743

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society B*. https://doi.org/10.2307/2346101

Berry, D., Gacitua, R., Sawyer, P., & Tjong, S. F. (2012). The case for dumb requirements engineering tools. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7195 LNCS*, 211–217. https://doi.org/10.1007/978-3-642-28714-5_18

Bhowmik, T., Niu, N., Savolainen, J., & Mahmoud, A. (2015). Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering. *Requirements Engineering*, *20*(3), 253–280. https://doi.org/10.1007/s00766-015-0226-2

Blei, D. M., Edu, B. B., Ng, A. Y., Edu, A. S., Jordan, M. I., & Edu, J. B. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, *3*, 993–1022. https://doi.org/10.1162/jmlr.2003.3.4-5.993

Bruce, R. F., & Wiebe, J. M. (1999). Recognizing subjectivity: A case study in manual tagging. *Natural Language Engineering*, *5*(2), 187–205. https://doi.org/10.1017/S1351324999002181

Chakraborty, A., Baowaly, M. K., Arefin, A., & Bahar, A. N. (2012). The Role of Requirement Engineering in Software Development Life Cycle. *Journal of Emerging Trends in Computing and Information Sciences*, *3*(5), 723–729.

Chen, N., Lin, J., Hoi, S., Xiao, X., & Zhang, B. (2014). AR-miner: mining informative reviews for developers from mobile app marketplace. *Icse*, (2014), 767–778. https://doi.org/10.1145/2568225.2568263

Comaniciu, D., & Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(5), 1–37. https://doi.org/10.1109/34.1000236

Cutting, D. R. (1992). Scatter / Gather : A Cluster-based Approach to Browsing Large Document Collections 1 Introduction 2 Scatter / Gather Browsing.

Davis, A., Dieste, O., Hickey, A., Juristo, N., & Moreno, A. M. (2006). Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. *Proceedings of the IEEE International Requirements Engineering Conference*, 176–185. https://doi.org/10.1109/RE.2006.17

Davis, A. M. (2003). The art of requirement triage. *Computer*, *36*(3), 42–49. https://doi.org/10.1038/nchem.1109

Desmet, B., & Hoste, V. (2014). Fine-grained Dutch named entity recognition. *Language Resources and Evaluation*, *48*(2), 307–343.

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification.* John Wiley & Sons.

El-Ramly, M., & Stroulia, E. (2004). Mining software usage data. *Proceedings of 1st International Workshop on Mining Software Repositories (MSR'04)*, 64–8. Retrieved from http://msr.uwaterloo.ca/papers/ElRamly.pdf

Feldt, R., & Magazinius, A. (2010). Validity Threats in Empirical Software Engineering Research - An Initial Survey. *Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering*, (August), 374–379. Retrieved from http://robertfeldt.net/publications/feldt_2010_validity_threats_in_ese_initial_survey.pdf%5Cnhttps://www.researchgate.net/profile/Ana_Magazinius/publication/221390199_Validity_Threats_in_Empirical_Software_Engineering_Research_-_An_Initial_Survey/links/53

Følstad, A., Law, E., & Hornbæk, K. (2012). Analysis in practical usability evaluation: a survey study. *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems - CHI '12*, (April 2015), 2127. https://doi.org/10.1145/2207676.2208365

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating Implicit Measures to Improve Web Search. *ACM Transactions on Information Systems.*, *23*(2), 147–168.

Fu, B., Lin, J., Li, L., & Faloutsos, C. (2013). Why people hate your app: making sense of user feedback in a mobile app store. *Proceedings of the 19th …*, 9. https://doi.org/10.1145/2487575.2488202

Fung, B. C. M. (2002). Hierarchical Document Clustering using Frequent Itemsets, (September), 1–59.

Gacitua, R., Sawyer, P., & Gervasi, V. (2010). On the effectiveness of abstraction identification in requirements engineering. *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010*, 5–14. https://doi.org/10.1109/RE.2010.12

Galvan-nunez, S., & Attoh-okine, N. (2016). Hybrid Particle Swarm Optimization and K -Means Analysis for Bridge Clustering Based on National Bridge Inventory Data, 1–6. https://doi.org/10.1061/AJRUA6.0000864.

Gartner, S., & Schneider, K. (2012). A method for prioritizing end-user feedback for requirements engineering. *2012 5th International Workshop on Co-Operative and Human Aspects of Software Engineering, CHASE 2012 - Proceedings*, 47–49. https://doi.org/10.1109/CHASE.2012.6223020

Groen, E. C., Seyff, N., Ali, R., Dalpiaz, F., Doerr, J., Guzman, E., … Stade, M. (2017). The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software*, *34*(2), 44–52. https://doi.org/10.1109/MS.2017.33

Guzman, E., Alkadhi, R., & Seyff, N. (2016). A Needle in a Haystack: What Do Twitter Users Say about Software? *Requirements Engineering Conference - RE '16*, 96–105. https://doi.org/10.1109/RE.2016.67

Hayes, J. H., Dekhtyar, A., & Sundaram, S. (2005). Text Mining for Software Engineering : How Analyst Feedback Impacts Final Results. *Proceeding MSR '05 Proceedings of the 2005 International Workshop on Mining Software Repositories*, 1–5. https://doi.org/10.1145/1083142.1083153

Hu, M., & Liu, B. (2004). Mining and summarizing customer reviews. *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD 04*, *04*, 168. https://doi.org/10.1145/1014052.1014073

Huffman, J., Alex, H., Senthil, D., & Sundaram, K. (2006). Advancing Candidate Link Generation for Requirements Tracing- the.pdf. *Transactions on Software Engineering*, 1–22.

Jawaheer, G., Weller, P., & Kostkova, P. (2014). Modeling User Preferences in Recommender Systems: A Classification Framework for Explicit and Implicit User Feedback. *ACM Transactions on Interactive Intelligent Systems*, *4*(2), 8.

Kotonya, G., & Sommerville, I. (1998). *Requirements engineering: processes and techniques.* Wiley Publishing.

Li, F., Han, C., Huang, M., Zhu, X., Xia, Y. Y.-J., Zhang, S., & Yu, H. (2010). Structure-aware Review Mining and Summarization. *Proceedings of the 23rd International Conference on Computational Linguistics*, (August), 653–661. Retrieved from http://dl.acm.org/citation.cfm?id=1873855%5Cnhttp://dl.acm.org/citation.cfm?id=1873781.1873855

Liu, Z., Yu, W., Deng, Y., Wang, Y., & Bian, Z. (2010). A feature selection method for document clustering based on part-of-speech and word co-occurrence. *Proceedings - 2010 7th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010*, *5*(August 2010), 2331–2334. https://doi.org/10.1109/FSKD.2010.5569827

Lucassen, G., Dalpiaz, F., van der Werf, J.-M., & Brinkkemper, S. (2016). Improving agile requirements : the Quality User Story framework and tool. *Requirements Engineering*, *21*(3), 383–403. https://doi.org/10.1007/s00766-016-0250-x

Luo, C., Li, Y., & Chung, S. M. (2009). Text document clustering based on neighbors. *Data & Knowledge Engineering*, *68*(11), 1271–1288. https://doi.org/10.1016/j.datak.2009.06.007

Maalej, W., Kurtanovi, Z., Nabil, H., & Stanik, C. (2016). On the automatic classification of app reviews. *Requirements Engineering*, *21*(3), 311–331.

Maalej, W., Nayebi, M., Johann, T., & Ruhe, G. (2015). Towards Data - Driven Requirements Engineering, 1–6.

Morales-Ramirez, I., Perini, A., & Guizzardi, R. S. S. (2015). An ontology of online user feedback in software engineering. *Applied Ontology*, *10*(3–4), 297–330. https://doi.org/10.3233/AO-150150

Nayebi, M., & Ruhe, G. (2015). Analytical product release planning. *The Art and Science of Analyzing Software*.

Nielsen, J., & Molich, R. (1990). Heuristic Evaluation of user interfaces. *CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (April), 249–256. https://doi.org/10.1145/97243.97281

Nuseibeh, B., & Easterbrook, S. (2000). Requirements Engineering : A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35–46).

Pachidi, S., Spruit, M., & Van De Weerd, I. (2014). Understanding users' behavior with software operation data mining. *Computers in Human Behavior*, *30*(November 2015), 583–594. https://doi.org/10.1016/j.chb.2013.07.049

Palomba, F., Linares-Vasquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., & De Lucia, A. (2015). User reviews matter! Tracking crowdsourced reviews to support evolution of successful

apps. *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*, 291–300. https://doi.org/10.1109/ICSM.2015.7332475

Panichella, S., Sorbo, A. Di, & Guzman, E. (2015). How Can I Improve My App ? Classifying User Reviews for Software Maintenance and Evolution, (2), 281–290.

Pfitzner, D., Leibbrandt, R., & Powers, D. (2009). Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, *19*(3), 361–394. https://doi.org/10.1007/s10115-008-0150-6

Pohl, K. (2010). The Requirements Engineering Framework. *Requirements Engineering: Fundamentals, Principles, and Techniques-Solution-Oriented Requirements*, 41–58.

Popescu, D., Rugaber, S., Medvidovic, N., & Berry, D. (2007). Reducing ambiguities in requirements specifications via automatically created object-oriented models. *Monterey Workshop*, (September), 103–124.

Robeer, M., Lucassen, G., van der Werf, J. M., Dalpiaz, F., & Brinkkemper, S. (2016). Automated Extraction of Conceptual Models from User Stories via NLP. *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*. https://doi.org/10.1109/RE.2016.40

Rosell, M. (2009). Part of Speech Tagging for Text Clustering in Swedish. *Proceedings of the 17th Nordic Conference of Computational Linguistics*, 150–157.

Sang, E. F. T. K., & De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. https://doi.org/10.3115/1119176.1119195

Schaufelbühl, A., Panichella, S., Ciurumelea, A., Schaufelb, A., Panichella, S., & Gall, H. (2017). Analyzing Reviews and Code of Mobile Apps for Better Release Planning Analyzing Reviews and Code of Mobile Apps for Better Release Planning, (December 2016). https://doi.org/10.1109/SANER.2017.7884612

Sedding, J., & Kazakov, D. (2001). WordNet-based Text Document Clustering. *Proceedings of the 3rd Workshop on RObust Methods in Analysis of Natural Language Data*, (2004), 104–113. https://doi.org/10.3115/1621445.1621458

Shneiderman, B. (2003). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *The Craft of Information Visualization*. https://doi.org/10.1016/B978-155860915-0/50046-9

Snijders, R., Dalpiaz, F., Brinkkemper, S., Hosseini, M., Ali, R., & ??z??m, A. (2015). REfine: A gamified platform for participatory requirements engineering. *1st International Workshop on Crowd-Based Requirements Engineering, CrowdRE 2015 - Proceedings*, 1–6. https://doi.org/10.1109/CrowdRE.2015.7367581

Sommerville, I. (2012). Software Engineering. Retrieved from https://books.google.co.uk/books?id=N69KPjBEWygC&pg=PA224&dq=black+box+testing&hl=en&sa=X&ved=0ahUKEwiL69mq1aDaAhWdF8AKHRMHBt0Q6AEIOTAD#v=onepage&q=black box testing&f=false

Sorbo, A. Di, Panichella, S., Visaggio, C. A., Penta, M. Di, Canfora, G., & Gall, H. (2016). DECA : Development Emails Content Analyzer, 1–4. https://doi.org/10.1145/2889160.2889170

Steinbach, M., Karypis, G., & Kumar, V. (2000). A Comparison of Document Clustering Techniques. *KDD*

Workshop on Text Mining, *400*(X), 1–2. https://doi.org/10.1109/ICCCYB.2008.4721382

Thayer, R. H., Bailin, S. C., & Dorfman, M. (1997). *Software Requirements Engineerings, 2nd Edition*. IEEE Computer Society Press Los Alamitos, CA, USA ©1997.

Van den Bosch, A., Busser, B., Canisius, S., & Daelemans, W. (2007). An efficient memory-based morphosyntactic tagger and parser for Dutch. *Selected Papers of CLIN 2007*, 99–114. Retrieved from http://eprints.pascal-network.org/archive/00003894/

Villarroel, L., Bavota, G., Russo, B., Oliveto, R., & Penta, M. Di. (2016). Release Planning of Mobile Apps Based on User Reviews. *Proceedings of the 38th International Conference on Software Engineering.*, 12–24.

Vu, P. M., Nguyen, T. T., Pham, H. V., & Nguyen, T. T. (2015). Mining User Opinions in Mobile App Reviews: A Keyword-based Approach. https://doi.org/10.1109/ASE.2015.85

Wang, Y., & Hodges, J. (2006). Document Clustering with Semantic Analysis. *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, *00*(C), 54c–54c. https://doi.org/10.1109/HICSS.2006.129

Wiegers, K., & Beatty, J. (2013). *Software requirements*. Pearson Education.

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. London: Springer Verlag.

Wilson, W. M., Rosenberg, L. H., & Hyatt, L. E. (1997). Automated analysis of requirement specifications. *Proceedings - International Conference on Software Engineering*, 161–171. https://doi.org/10.1109/ICSE.1997.610237

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. ., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*.

Zowghi, D., & Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches, and tools. *Engineering and Managing Software Requirements*, (July), 19–46. https://doi.org/10.1007/3-540-28244-0

APPENDIX I: SOURCE CODE

In these appendixes the python code for the experiment is given.

APPENDIX I-A: POS TAGGING

```
# # Processing text

# Import everything up front

# In[14]:

import numpy as np
import pandas as pd
import nltk
from bs4 import BeautifulSoup
import re
import os
import codecs
from sklearn import feature_extraction
import csv


# # Stopwords, stemming, and tokenizing

# In[15]:

#import two lists which have previously been extracted from the dataset: titles and con
tents
titles = open('titels.txt').read().split('\n')
#ensures that only the first 100 are read in
titles = titles[:100]

# the items of feedback are separeted by 'BREAKS HERE', so we split on this
toelichtingen = open('entries.txt', encoding="utf8").read().split('\nBREAKS HERE')
toelichtingen = toelichtingen[:100]

print(str(len(titles)) + ' titles')
print(str(len(toelichtingen)) + ' contents')


# In[16]:

# The contents of the texts have previously been POS tagged by the Frog tool and stored
 in 1.csv
f = open('1.csv')
csv_f = csv.reader(f,delimiter=';')

# Three functions are designed to filer on one, two or three POS tags.
def category_filter1(category):
    filtered_list = []
    for row in csv_f:
        split = row[4].split('(')
        if split[0] == category:
            filtered_list.append(row[1])
            filtered_list = [word.lower() for word in filtered_list]
            seen = set()
            unique_filtered_words = []
```

```
            for x in filtered_list:
                if x not in seen:
                    unique_filtered_words.append(x)
                    seen.add(x)
    return unique_filtered_words

def category_filter2(category, second_category):
    filtered_list = []
    for row in csv_f:
        split = row[4].split('(')
        if split[0] == category or split[0] == second_category:
            filtered_list.append(row[1])
            filtered_list = [word.lower() for word in filtered_list]
            seen = set()
            unique_filtered_words = []
            for x in filtered_list:
                if x not in seen:
                    unique_filtered_words.append(x)
                    seen.add(x)
    return unique_filtered_words

def category_filter3(category, second_category, third_category):
    filtered_list = []
    for row in csv_f:
        split = row[4].split('(')
        if split[0] == category or split[0] == second_category or split[0] == third_cat
egory:
            filtered_list.append(row[1])
            filtered_list = [word.lower() for word in filtered_list]
            seen = set()
            unique_filtered_words = []
            for x in filtered_list:
                if x not in seen:
                    unique_filtered_words.append(x)
                    seen.add(x)
    return unique_filtered_words

#categorized_words = category_filter1('SPEC')
#categorized_words = category_filter2('SPEC', 'N')
#categorized_words = category_filter3('ADJ', 'N', 'SPEC')


# In[17]:

# generates index for each item in the corpora
ranks = []

for i in range(0,len(titles)):
    ranks.append(i)

        print(ranks)


        # This section is focused on defining some functions to manipulate the tickets.
List of dutch stopwords is obtained through http://www.damienvanholten.com/blog/dutch-
stop-words/

        # Next I import the [Snowball Stemmer](http://snowball.tartarus.org/) which is a
ctually part of NLTK. [Stemming](http://en.wikipedia.org/wiki/Stemming) is just the pro
cess of breaking a word down into its root.

```

```
        # In[18]:

        stopwords = open('dutch-stop-words.txt').read().split('\n')


        # In[19]:

        # load nltk's SnowballStemmer as variabled 'stemmer'
        from nltk.stem.snowball import SnowballStemmer
        stemmer = SnowballStemmer("dutch")


        #
        # Below I define two functions:
        #
        # *tokenize_and_stem*: tokenizes (splits the texdt into a list of its respective words (or tokens) and also stems each token
        # *tokenize_only*: tokenizes the text only

        # In[20]:

        # here I define a tokenizer and stemmer which returns the set of stems in the text that it is passed

        def tokenize_and_stem(text):
            # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
            tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
            filtered_tokens = []
            # filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation)
            for token in tokens:
                if re.search('[a-zA-Z]', token):
                    filtered_tokens.append(token)
            stems = [stemmer.stem(t) for t in filtered_tokens]
            return stems


        def tokenize_only(text):
            # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
            tokens = [word.lower() for sent in nltk.sent_tokenize(text.decode('utf8')) for word in nltk.word_tokenize(sent.decode('utf8'))]
            filtered_tokens = []
            # filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation)
            for token in tokens:
                if re.search('[a-zA-Z]', token):
                    filtered_tokens.append(token)
            return filtered_tokens


        # Below I use my stemming/tokenizing and tokenizing functions to iterate over the list of synopses to create two vocabularies: one stemmed and one only tokenized.

        # In[13]:

        toel_tokenized = []
        for i in toelichtingen:
            allwords_tokenized = tokenize_only(i)
```

```
159.            print (allwords_tokenized)
160.            toel_tokenized.append(allwords_tokenized)
161.
162.        print(toel_tokenized)
163.
164.
165.        # In[ ]:
166.
167.        # I iterate over the texts and create a list that contains only the POS tagged w
    ords but also keeps the structure and order of the feedback items
168.        texts = toel_tokenized
169.        good_worden = categorized_words
170.        final_list = []
171.        for text in texts:
172.            filtered_text = []
173.            for sub_text in text:
174.                for word in good_words:
175.                    if word in sub_tekst and len(word) > 2:
176.                        filtered_text.append(word)
177.            filtered_text = ' '.join(filtered_tekst)
178.            final_list.append(filtered_tekst)
179.
180.        print(final_list[:10])
```

APPENDIX I-B: K-MEANS CLUSTERING

```
# In[ ]:

# in order to improve clustering results, stemming is used to reduce variability
totalvocab_stemmed = []
totalvocab_tokenized = []
for i in toelichtingen:
    allwords_stemmed = tokenize_and_stem(i)
    totalvocab_stemmed.extend(allwords_stemmed)

    allwords_tokenized = tokenize_only(i)
    totalvocab_tokenized.extend(allwords_tokenized)


# Using these two lists, I create a pandas DataFrame with the stemmed vocabulary as the
 index and the tokenized words as the column.

# In[ ]:

vocab_frame = pd.DataFrame({'words': totalvocab_tokenized}, index = totalvocab_stemmed)

print(vocab_frame)


# ## Tf-idf and document similarity

# In[ ]:

# a tf-idf matrix is created to measure document similarity
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.9, max_features=200000,
                                   min_df=0.1, stop_words = stopwords,
                                   use_idf=True, tokenizer=tokenize_and_stem, ngram_range
=(1,3))

get_ipython().magic('time tfidf_matrix = tfidf_vectorizer.fit_transform(eindlijst)')


# In[ ]:

terms = tfidf_vectorizer.get_feature_names()
print(terms)


# In[ ]:

# we use consine similarity as a distance measure
from sklearn.metrics.pairwise import cosine_similarity
dist = 1 - cosine_similarity(tfidf_matrix)


# # K-means clustering

# In[ ]:

# the number of generated clusters is set to 5
```

```
55.
56. from sklearn.cluster import KMeans
57.
58. num_clusters = 5
59.
60. km = KMeans(n_clusters=num_clusters)
61.
62. get_ipython().magic('time km.fit(tfidf_matrix)')
63.
64. clusters = km.labels_.tolist()
65.
66. print(clusters)
67.
68.
69. # In[ ]:
70.
71. from sklearn.externals import joblib
72.
73. joblib.dump(km,  'doc_cluster.pkl')
74. km = joblib.load('doc_cluster.pkl')
75. clusters = km.labels_.tolist()
76.
77. print(clusters)
78.
79.
80. # In[ ]:
81.
82. import pandas as pd
83.
84. texts = { 'title': titles, 'rank': ranks, 'content': toelichtingen, 'cluster': clusters
     }
85.
86. frame = pd.DataFrame(texts, index = [clusters] , columns = ['rank', 'title', 'cluster']
    )
87.
88. print (frame)
89.
90.
91. # In[ ]:
92.
93. frame['cluster'].value_counts()
94.
95.
96. # In[ ]:
97.
98. grouped = frame['rank'].groupby(frame['cluster'])
99.
100.        grouped.mean()
101.
102.
103.        # In[ ]:
104.
105.        # we create a print function to explore the clusters
106.        from __future__ import print_function
107.
108.        print("Top terms per cluster:")
109.        print()
110.        order_centroids = km.cluster_centers_.argsort()[:, ::-1]
111.        for i in range(num_clusters):
112.            print("Cluster %d words:" % i, end='')
113.            for ind in order_centroids[i, :6]:
```

```
                #print(ind)
                print(' %s' % vocab_frame.ix[terms[ind].split(' ')].values.tolist()[0][0
    ].encode('utf-8', 'ignore'), end=',')
            print()
            print()
            print("Cluster %d titles:" % i, end='')
            for title in frame.ix[i]['title'].values.tolist():
                print(' %s,' % title, end='')
            print()
            print()


        # In[ ]:

        #This is purely to help export tables to html and to correct for the 0 start ran
    k
        frame['Rank'] = frame['rank'] + 1
        frame['Title'] = frame['title']

        print(frame)


        # In[ ]:

        #export tables to HTML
        print(frame[['Rank', 'Title']].loc[frame['cluster'] == 1].to_html(index=False))
```

APPENDIX I-C: MEANSHIFT CLUSTERING

```
# In[1]:


# All package are imported
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.cluster import MeanShift, estimate_bandwidth
import matplotlib.pyplot as plt
from itertools import cycle


# In[ ]:

# import a dataset that contains only the selected pos-tags, transformed into a tf-idf matrix
data = pd.read_csv("Pos-tags.csv")

X= data
bandwidth = estimate_bandwidth()

# meanshift is run over the data
ms = MeanShift()
ms.fit(data)
labels = ms.labels_
cluster_centers = ms.cluster_centers_

print (labels)
print (cluster_centers)
n_clusters_ = len(np.unique(labels))
print("Number of estimated clusters:", n_clusters_)
```

APPENDIX I-D: PAIR SELECTION ALGORITHM

```
# In[20]:

import math
import csv
import itertools
import random


# In[21]:

# we define a function to process csv files
def open_csv(filename):
    f = open(filename)
    csv_f = csv.reader(f, delimiter=',')
    ranks = []
    csvlist = []

    for row in csv_f:
        ranks.append(row[0])
        csvlist.append(row)
    return ranks, csvlist


# In[22]:

# all possible piar combinations are generated
def possible_pairs(ranks):
    pairs = itertools.combinations(ranks, 2)
    big_list = []
    for row in pairs:
        big_list.append(row)
    return(big_list)


# In[23]:

# we discover which possible piars are in the clustering results
def pair_clusters(big_list, csvlist):
    combined_list = []
    for row in big_list:
        row = list(row)
        for ticket in csvlist:
            if row[0] == ticket[0]:
                row.insert(1, ticket[1])
        for ticket in csvlist:
            if row[2] == ticket[0]:
                row.insert(3, ticket[1])
        combined_list.append(row)
    return(combined_list)


# In[24]:

# function that determines if a pair is the same cluster
def same_cluster(pair):
    if pair[1] == pair[3]:
        return(True)
    else:
```

```
        return(False)


# In[25]:

# These functions split the pairs into area's based on the number of their cluster and if they are in the same cluster or a different cluster.
# these functions have different sizes because of the different number of clusters that the meanshift algorithm can produce
def split_into_areas(combined_list):
    AreaA = []
    AreaB = []
    AreaC = []
    AreaD = []
    AreaE = []
    AreaF = []
    AreaG = []
    AreaH = []
    AreaI = []
    AreaJ = []

    for pair in combined_list:
        #print(pair)
        if same_cluster(pair) == True and pair[1] == '0':
            #combined_list.remove(pair)
            #del(pair[1])
            #del(pair[2])
            AreaA.append(pair)
            #combined_list.remove(pair)
        if same_cluster(pair) == True and pair[1] == '1':
            AreaB.append(pair)
        if same_cluster(pair) == True and pair[1] == '2':
            AreaC.append(pair)
        if same_cluster(pair) == True and pair[1] == '3':
            AreaD.append(pair)
        if same_cluster(pair) == True and pair[1] == '4':
            AreaE.append(pair)
        if same_cluster(pair) == False and pair[1] == '0':
            AreaF.append(pair)
        if same_cluster(pair) == False and pair[1] == '1':
            AreaG.append(pair)
        if same_cluster(pair) == False and pair[1] == '2':
                AreaH.append(pair)
            if same_cluster(pair) == False and pair[1] == '3':
                AreaI.append(pair)
            if same_cluster(pair) == False and pair[1] == '4':
                AreaJ.append(pair)
        return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ


    # In[26]:

    def split_into_areasNEN(combined_list):
        AreaA = []
        AreaB = []
        AreaC = []
        AreaD = []
        AreaE = []
        AreaF = []
```

```
            AreaG = []
            AreaH = []
            AreaI = []
            AreaJ = []
            AreaK = []
            AreaL = []

            for pair in combined_list:
                #print(pair)
                if same_cluster(pair) == True and pair[1] == '0':
                    #combined_list.remove(pair)
                    #del(pair[1])
                    #del(pair[2])
                    AreaA.append(pair)
                    #combined_list.remove(pair)
                if same_cluster(pair) == True and pair[1] == '1':
                    AreaB.append(pair)
                if same_cluster(pair) == True and pair[1] == '2':
                    AreaC.append(pair)
                if same_cluster(pair) == True and pair[1] == '3':
                    AreaD.append(pair)
                if same_cluster(pair) == True and pair[1] == '4':
                    AreaE.append(pair)
                if same_cluster(pair) == True and pair[1] == '5':
                    AreaF.append(pair)
                if same_cluster(pair) == False and pair[1] == '0':
                    AreaG.append(pair)
                if same_cluster(pair) == False and pair[1] == '1':
                    AreaH.append(pair)
                if same_cluster(pair) == False and pair[1] == '2':
                    AreaI.append(pair)
                if same_cluster(pair) == False and pair[1] == '3':
                    AreaJ.append(pair)
                if same_cluster(pair) == False and pair[1] == '4':
                    AreaK.append(pair)
                if same_cluster(pair) == False and pair[1] == '5':
                    AreaL.append(pair)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ,
    AreaK, AreaL


        # In[27]:

        def split_into_areasVNADJ(combined_list):
            AreaA = []
            AreaB = []
            AreaC = []
            AreaD = []
            AreaE = []
            AreaF = []
            AreaG = []
            AreaH = []
            AreaI = []
            AreaJ = []
            AreaK = []
            AreaL = []
            AreaM = []
            AreaN = []


            for pair in combined_list:
```

```
                #print(pair)
                if same_cluster(pair) == True and pair[1] == '0':
                    #combined_list.remove(pair)
                    #del(pair[1])
                    #del(pair[2])
                    AreaA.append(pair)
                    #combined_list.remove(pair)
                if same_cluster(pair) == True and pair[1] == '1':
                    AreaB.append(pair)
                if same_cluster(pair) == True and pair[1] == '2':
                    AreaC.append(pair)
                if same_cluster(pair) == True and pair[1] == '3':
                    AreaD.append(pair)
                if same_cluster(pair) == True and pair[1] == '4':
                    AreaE.append(pair)
                if same_cluster(pair) == True and pair[1] == '5':
                    AreaF.append(pair)
                if same_cluster(pair) == True and pair[1] == '6':
                    AreaG.append(pair)
                if same_cluster(pair) == False and pair[1] == '0':
                    AreaH.append(pair)
                if same_cluster(pair) == False and pair[1] == '1':
                    AreaI.append(pair)
                if same_cluster(pair) == False and pair[1] == '2':
                    AreaJ.append(pair)
                if same_cluster(pair) == False and pair[1] == '3':
                    AreaK.append(pair)
                if same_cluster(pair) == False and pair[1] == '4':
                    AreaL.append(pair)
                if same_cluster(pair) == False and pair[1] == '5':
                    AreaM.append(pair)
                if same_cluster(pair) == False and pair[1] == '6':
                    AreaN.append(pair)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ,
    AreaK, AreaL, AreaM, AreaN


        # In[28]:

        def split_into_areasVN(combined_list):
            AreaA = []
            AreaB = []
            AreaC = []
            AreaD = []
            AreaE = []
            AreaF = []
            AreaG = []
            AreaH = []


            for pair in combined_list:
                #print(pair)
                if same_cluster(pair) == True and pair[1] == '0':
                    #combined_list.remove(pair)
                    #del(pair[1])
                    #del(pair[2])
                    AreaA.append(pair)
                    #combined_list.remove(pair)
                if same_cluster(pair) == True and pair[1] == '1':
                    AreaB.append(pair)
                if same_cluster(pair) == True and pair[1] == '2':
```

```
                AreaC.append(pair)
            if same_cluster(pair) == True and pair[1] == '3':
                AreaD.append(pair)
            if same_cluster(pair) == False and pair[1] == '0':
                AreaE.append(pair)
            if same_cluster(pair) == False and pair[1] == '1':
                AreaF.append(pair)
            if same_cluster(pair) == False and pair[1] == '2':
                AreaG.append(pair)
            if same_cluster(pair) == False and pair[1] == '3':
                AreaH.append(pair)

        return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH


    # In[29]:

    def split_into_areasNENADJ(combined_list):
        AreaA = []
        AreaB = []
        AreaC = []
        AreaD = []
        AreaE = []
        AreaF = []
        AreaG = []
        AreaH = []
        AreaI = []
        AreaJ = []
        AreaK = []
        AreaL = []
        AreaM = []
        AreaN = []
        AreaO = []
        AreaP = []
        AreaQ = []
        AreaR = []


        for pair in combined_list:
            #print(pair)
            if same_cluster(pair) == True and pair[1] == '0':
                #combined_list.remove(pair)
                #del(pair[1])
                #del(pair[2])
                AreaA.append(pair)
                #combined_list.remove(pair)
            if same_cluster(pair) == True and pair[1] == '1':
                AreaB.append(pair)
            if same_cluster(pair) == True and pair[1] == '2':
                AreaC.append(pair)
            if same_cluster(pair) == True and pair[1] == '3':
                AreaD.append(pair)
            if same_cluster(pair) == True and pair[1] == '4':
                AreaE.append(pair)
            if same_cluster(pair) == True and pair[1] == '5':
                AreaF.append(pair)
            if same_cluster(pair) == True and pair[1] == '6':
                AreaG.append(pair)
            if same_cluster(pair) == True and pair[1] == '7':
                AreaH.append(pair)
            if same_cluster(pair) == True and pair[1] == '8':
```

```
                    AreaI.append(pair)
                if same_cluster(pair) == False and pair[1] == '0':
                    AreaJ.append(pair)
                if same_cluster(pair) == False and pair[1] == '1':
                    AreaK.append(pair)
                if same_cluster(pair) == False and pair[1] == '2':
                    AreaL.append(pair)
                if same_cluster(pair) == False and pair[1] == '3':
                    AreaM.append(pair)
                if same_cluster(pair) == False and pair[1] == '4':
                    AreaN.append(pair)
                if same_cluster(pair) == False and pair[1] == '5':
                    AreaO.append(pair)
                if same_cluster(pair) == False and pair[1] == '6':
                    AreaP.append(pair)
                if same_cluster(pair) == False and pair[1] == '7':
                    AreaQ.append(pair)
                if same_cluster(pair) == False and pair[1] == '8':
                    AreaR.append(pair)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ,
    AreaK, AreaL, AreaM, AreaN, AreaO, AreaP, AreaQ, AreaR


        # In[30]:

        # main function that reads csv files with the results of k-
    means clustering and returns pairs in their correct area for the
        def main(name_of_file):
            ranks, ticketlist = open_csv(name_of_file)
            all_pairs = possible_pairs(ranks)
            combined_list = pair_clusters(all_pairs, ticketlist)
            AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ = split
    _into_areas(combined_list)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ


        AreaA1, AreaB1, AreaC1, AreaD1, AreaE1, AreaF1, AreaG1, AreaH1, AreaI1, AreaJ1 =
    main('Ne + N.csv')
        AreaA2, AreaB2, AreaC2, AreaD2, AreaE2, AreaF2, AreaG2, AreaH2, AreaI2, AreaJ2 =
    main('Ne + N + ADJ.csv')
        AreaA3, AreaB3, AreaC3, AreaD3, AreaE3, AreaF3, AreaG3, AreaH3, AreaI3, AreaJ3 =
    main('V + N.csv')
        AreaA4, AreaB4, AreaC4, AreaD4, AreaE4, AreaF4, AreaG4, AreaH4, AreaI4, AreaJ4 =
    main('V + N + ADJ.csv')




        # In[31]:

        # these functions process the results of the meanshift algorithm
        def KmainNEN(name_of_file):
            ranks, ticketlist = open_csv(name_of_file)
            all_pairs = possible_pairs(ranks)
            combined_list = pair_clusters(all_pairs, ticketlist)
            AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ, AreaK,
    AreaL = split_into_areasNEN(combined_list)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ,
    AreaK, AreaL

        def KmainVNADJ(name_of_file):
```

```
            ranks, ticketlist = open_csv(name_of_file)
            all_pairs = possible_pairs(ranks)
            combined_list = pair_clusters(all_pairs, ticketlist)
            AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ, AreaK, AreaL, AreaM, AreaN = split_into_areasVNADJ(combined_list)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ, AreaK, AreaL, AreaM, AreaN

        def KmainVN(name_of_file):
            ranks, ticketlist = open_csv(name_of_file)
            all_pairs = possible_pairs(ranks)
            combined_list = pair_clusters(all_pairs, ticketlist)
            AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH = split_into_areasVN(combined_list)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH

        def KmainNENADJ(name_of_file):
            ranks, ticketlist = open_csv(name_of_file)
            all_pairs = possible_pairs(ranks)
            combined_list = pair_clusters(all_pairs, ticketlist)
            AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ, AreaK, AreaL, AreaM, AreaN, AreaO, AreaP, AreaQ, AreaR = split_into_areasNENADJ(combined_list)
            return AreaA, AreaB, AreaC, AreaD, AreaE, AreaF, AreaG, AreaH, AreaI, AreaJ, AreaK, AreaL, AreaM, AreaN, AreaO, AreaP, AreaQ, AreaR


        # In[32]:

        #the pairs for both the k-means and the meanshift and their pos tags are appended to the matrix
        matrix = []
        matrix.append(main('Ne + N.csv'))
        matrix.append(main('Ne + N + ADJ.csv'))
        matrix.append(main('V + N.csv'))
        matrix.append(main('V + N + ADJ.csv'))
        matrix.append(KmainNEN('NEN6.csv'))
        matrix.append(KmainNENADJ('NENADJ9.csv'))
        matrix.append(KmainVN('VN4.csv'))
        matrix.append(KmainVNADJ('VNADJ7.csv'))


        # In[33]:

        # all cluster numbers are deleted
        for r in range(len(matrix)):
            for c in range(len(matrix[r])):
                for p in range(len(matrix[r][c])):
                    del matrix[r][c][p][1]
                    del matrix[r][c][p][2]




        # In[34]:

        print (matrix)


        # In[36]:

```

```
403.        # these classes generate an empty maxtrix and populate this maxtrix according to
    our selection criteria
404.
405.        class Network(object):
406.
407.            def __init__(self, matrix, empty):
408.                self.area_names = []
409.                if empty:
410.                    self.areas = self.initialize_empty_network(matrix)
411.                else:
412.                    self.areas = self.initialize_network(matrix)
413.
414.
415.            def initialize_empty_network(self, matrix):
416.                areas = {}
417.                row_names = range(1, 10)
418.                column_names = [chr(i) for i in range(ord('A'), ord('Z') + 1)]
419.                for r in range(len(matrix)):
420.                    sub_area_names = []
421.                    for c in range(len(matrix[r])):
422.                        areas[str(row_names[r]) + column_names[c]] = []
423.                        sub_area_names.append(str(row_names[r]) + column_names[c])
424.                    self.area_names.append(sub_area_names)
425.                return areas
426.
427.
428.            def initialize_network(self, matrix):
429.                areas = {}
430.                row_names = range(1, 10)
431.                column_names = [chr(i) for i in range(ord('A'), ord('Z') + 1)]
432.                for r in range(len(matrix)):
433.                    for c in range(len(matrix[r])):
434.                        areas[str(row_names[r]) + column_names[c]] = []
435.                        for p in range(len(matrix[r][c])):
436.                            p1 = int(matrix[r][c][p][0])
437.                            p2 = int(matrix[r][c][p][1])
438.                            areas[str(row_names[r]) + column_names[c]].append((p1, p2))

439.                return areas
440.
441.
442.            def find_pairs(self, p1, p2):
443.                result = []
444.                for key in self.areas.keys():
445.                    if (p1, p2) in self.areas[key]:
446.                        result.append(key)
447.                return result
448.
449.
450.            def get_smallest_area(self, keys):
451.                smallest_area = ''
452.                smallest_len = float('inf')
453.                for key in keys:
454.                    if len(self.areas[key]) < smallest_len:
455.                        smallest_len = len(self.areas[key])
456.                        smallest_area = key
457.                return smallest_area
458.
459.            def add_pair(self, area, p1, p2):
460.                self.areas[area].append((p1, p2))
461.
```

```
462.
463.           def remove_pair(self, area, p1, p2):
464.               self.areas[area].remove((p1, p2))
465.               if len(self.areas[area]) == 0:
466.                   self.areas.pop(area)
467.
468.
469.           def count_pairs(self):
470.               return sum([len(self.areas[key]) for key in self.areas.keys()])
471.
472.
473.           def convert_to_matrix(self):
474.               result = []
475.               for row in self.area_names:
476.                   sub_result = []
477.                   for name in row:
478.                       sub_result.append(self.areas[name])
479.                   result.append(sub_result)
480.               return result
481.
482.
483.
484.
485.       class Experiment(object):
486.
487.           def __init__(self, matrix):
488.               self.old_network = Network(matrix, False)
489.               self.new_network = Network(matrix, True)
490.
491.
492.           def get_prefered_pairs(self, pairs, used_numbers):
493.               prefered_pairs = []
494.               for pair in pairs:
495.                   p1, p2 = pair
496.                   if p1 in used_numbers or p1 in used_numbers:
497.                       prefered_pairs.append((p1, p2))
498.               return prefered_pairs
499.
500.
501.           def select_pair(self, prefered_pairs, pairs):
502.               if len(prefered_pairs) > 0:
503.                   p1, p2 = random.choice(prefered_pairs)
504.               else:
505.                   p1, p2 = random.choice(pairs)
506.               return (p1, p2)
507.
508.
509.           def return_new_matrix(self):
510.               return self.new_network.convert_to_matrix()
511.
512.
513.           def main_experiment(self, iterations, print_output = False):
514.               used_numbers = set([])
515.               #number of experiment iterations (e.g. 200, 500, 1000)
516.
517.               for i in range(iterations):
518.                   #1) Find smallest non_empty area in the new matrix
519.                   smallest_area = self.new_network.get_smallest_area(self.old_network.
    areas.keys())
520.
521.                   #2) Get list with prefered pairs
```

```
                    prefered_pairs = self.get_prefered_pairs(self.old_network.areas[smal
    lest_area], used_numbers)

                    #3) Select a pair
                    p1, p2 = self.select_pair(prefered_pairs, self.old_network.areas[sma
    llest_area])

                    # ) Add numbers to the 'known' list
                    if p1 not in used_numbers:
                        used_numbers.add(p1)
                    if p2 not in used_numbers:
                        used_numbers.add(p2)

                    # ) Look for pairs in each area
                    areas = self.old_network.find_pairs(p1, p2)

                    for area in areas:
                        # ) Add area to new matrix
                        self.new_network.add_pair(area, p1, p2)

                        # ) Delete pairs from old matrix
                        self.old_network.remove_pair(area, p1, p2)

                    if print_output:
                        print ('Smallest area:', smallest_area + '.',)
                        print ('Selected pair:', p1, p2)
                        print ('\tFound in %i other areas' % (len(areas)))
                        print ('\tSize new network:', self.new_network.count_pairs())
                        print ('')


        def print_matrix_sizes(matrix, name):
            print ('*************' + name + '*************')
            print (''.ljust(5),)
            for i in range(ord('A'), ord('R') + 1):
                print (chr(i).ljust(5),)
            print ('')

            count = 1
            for row in matrix:
                print (str(count).ljust(5),)
                for column in row:
                    print (str(len(column)).ljust(5),)
                print ('')
                count += 1
            print ('\n')


        new_network = Experiment(matrix)
        new_network.main_experiment(100)
        new_matrix = new_network.return_new_matrix()

        print_matrix_sizes(matrix, 'Old matrix')
        print_matrix_sizes(new_matrix, 'New matrix')


        # In[ ]:
```

APPENDIX II: SURVEY WITH DUMMY DATA

1.

The support staff is excellent, they have helped me when I needed it the most, they have training videos and online chat. This software has been useful to cover the hiring process, the navigation is quite intuitive and its design very friendly

I would suggest that the options be modified so that more than one person can be added as an interviewer, since only the information can be sent to a person to conduct the interview

2.

It was a really great applicant tracking system that help us improve our ways and efficiency within the company recruiting tasks.

It was a really great goal tracking software, basically we always receive different people applying on our company with different skills and it was really annoying getting a big amount of papers and stuff and ending with just one new hire but now we just need them to reduce all that.

First of all I think they need some better publicity is not a cons about the software itself but they need to properly sale itself because I can't believe it took me so long to get the software.

To what extent are these tickets related?

Not related at all | Somewhat not related | Somewhat related | Highly related

APPENDIX III: RESULTS STATISTICAL TESTS IN R

wilcoxon rank sum test for two algorithms

Two data samples are matched if they come from repeated observations of the same subject. Using the Wilcoxon Signed-Rank Test, we can decide whether the corresponding data population distributions are identical without assuming them to follow the normal distribution.

The value `W = 81118` corresponds to the sum of ranks assigned to the differences with positive sign.

Students vs professionals

```
> wilcox.test(Rating ~ Profession, data = data.in)

        Wilcoxon rank sum test with continuity correction

data:  Rating by Profession
W = 86388, p-value = 0.06284
alternative hypothesis: true location shift is not equal to 0
```

```
> wilcox.test(Rating ~ Profession, data = data.out)

        Wilcoxon rank sum test with continuity correction

data:  Rating by Profession
W = 72691, p-value = 0.02134
alternative hypothesis: true location shift is not equal to 0
```

```
> wilcox.test(Rating ~ Profession, data = data.out, conf.int = TRUE)

        Wilcoxon rank sum test with continuity correction

data:  Rating by Profession
```

W = 72691, p-value = 0.02134

alternative hypothesis: true location shift is not equal to 0

95 percent confidence interval:

-4.969917e-05 -8.013020e-06

sample estimates:

difference in location

-2.666002e-05

In

```
wilcox.test(Rating ~ Algorithm, data = data.in)
```

Wilcoxon rank sum test with continuity correction

data: Rating by Algorithm

W = 81118, p-value = 0.8336

alternative hypothesis: true location shift is not equal to 0

students

```
> wilcox.test(Rating ~ Algorithm, data = data.in.students)
```

Wilcoxon rank sum test with continuity correction

data: Rating by Algorithm

W = 19988, p-value = 0.8825

alternative hypothesis: true location shift is not equal to 0

Professionals

```
> wilcox.test(Rating ~ Algorithm, data = data.in.professional)
```

Wilcoxon rank sum test with continuity correction

data: Rating by Algorithm

W = 20538, p-value = 0.6771

alternative hypothesis: true location shift is not equal to 0

Out

```
> wilcox.test(Rating ~ Algorithm, data = data.out)
```

Wilcoxon rank sum test with continuity correction

data: Rating by Algorithm

W = 81214, p-value = 0.5036

alternative hypothesis: true location shift is not equal to 0

students

```
> wilcox.test(Rating ~ Algorithm, data = data.out.students)
```

Wilcoxon rank sum test with continuity correction

data: Rating by Algorithm

W = 20400, p-value = 0.5338

alternative hypothesis: true location shift is not equal to 0

Professionals

```
> wilcox.test(Rating ~ Algorithm, data = data.out.professional)
```

Wilcoxon rank sum test with continuity correction

data: Rating by Algorithm

W = 20134, p-value = 0.802

alternative hypothesis: true location shift is not equal to 0

Kruskal wallis for four pos tags

In

```
> kruskal.test(Rating ~ Pos.Tag, data = data.in)
```

Kruskal-Wallis rank sum test

data: Rating by Pos.Tag

Kruskal-Wallis chi-squared = 3.2679, df = 3, p-value = 0.3521

Students

```
> kruskal.test(Rating ~ Pos.Tag, data = data.in.students)
```

Kruskal-Wallis rank sum test

data: Rating by Pos.Tag

Kruskal-Wallis chi-squared = 1.0217, df = 3, p-value = 0.796

Professionals

```
> kruskal.test(Rating ~ Pos.Tag, data = data.in.professional)
```

Kruskal-Wallis rank sum test

data: Rating by Pos.Tag

Kruskal-Wallis chi-squared = 2.6495, df = 3, p-value = 0.4489

Out

```
> kruskal.test(Rating ~ Pos.Tag, data = data.out)

        Kruskal-Wallis rank sum test

data:  Rating by Pos.Tag
Kruskal-Wallis chi-squared = 1.8721, df = 3, p-value = 0.5994
```

Students

```
> kruskal.test(Rating ~ Pos.Tag, data = data.out.students)

        Kruskal-Wallis rank sum test

data:  Rating by Pos.Tag
Kruskal-Wallis chi-squared = 0.40934, df = 3, p-value = 0.9383
```

Professionals

```
> kruskal.test(Rating ~ Pos.Tag, data = data.out.professional)

        Kruskal-Wallis rank sum test

data:  Rating by Pos.Tag
Kruskal-Wallis chi-squared = 2.5193, df = 3, p-value = 0.4718
```

Dunn Test with multiple Kruskal Wallis Tests

In

```
> PT = dunnTest(Rating ~ Pos.Tag,
+               data=data.in,
+               method="bh")    # Can adjust p-values;
```

```
> PT

        Dunn (1964) Kruskal-Wallis multiple comparison
    p-values adjusted with the Benjamini-Hochberg method.

          Comparison           Z    P.unadj     P.adj
1      NE + N - NE + N + ADJ  1.6483138 0.09928828 0.5957297
2             NE + N - V + N  1.3797139 0.16767475 0.5030243
3       NE + N + ADJ - V + N -0.2933375 0.76926420 0.7692642
4       NE + N - V + N + ADJ  0.6601919 0.50913069 0.6109568
5 NE + N + ADJ - V + N + ADJ -0.9745531 0.32978196 0.6595639
6        V + N - V + N + ADJ -0.6985280 0.48484704 0.7272706
```

Out

```
> PT = dunnTest(Rating ~ Pos.Tag,
  +               data=data.out,
  +               method="bh")    # Can adjust p-values;
> PT
        Dunn (1964) Kruskal-Wallis multiple comparison
    p-values adjusted with the Benjamini-Hochberg method.

          Comparison           Z    P.unadj     P.adj
1      NE + N - NE + N + ADJ -1.3552809 0.1753281 1.0000000
2             NE + N - V + N -0.6108544 0.5412960 0.8119439
3       NE + N + ADJ - V + N  0.7085387 0.4786108 0.9572215
4       NE + N - V + N + ADJ -0.8043182 0.4212133 1.0000000
5 NE + N + ADJ - V + N + ADJ  0.5700421 0.5686492 0.6823790
6        V + N - V + N + ADJ -0.1627296 0.8707313 0.8707313
```