Semi-automated Extraction of New Product Features from Online Reviews to Support Software Product Evolution

A Thesis submitted to Auckland University of Technology
in partial fulfilment of the requirements
for the Degree of Master of Computer and Information Sciences

Phonephasouk Volabouth
2017

School of Engineering, Computer and Mathematical Sciences

Abstract

Throughout its lifetime, a software product is expected to continually improve and evolve to ensure its efficiency, utility and desirability to users, both existing and potential. This involves incrementally developing new product features and improvements that are then integrated into the existing product and released to users as a new product release. Deciding on which features to include in the next product release is part of the release planning process. This involves identifying new potential features, filtering and prioritising them for the next release. A readily available resource for finding these candidate release requirements is users' online reviews and feedback of the product, based on their experiences of using the product. It is common to find users suggesting new product features to meet their specific needs in these reviews. Sorting through such users' feedback manually can be very time-consuming, however, often requiring a large volume of unstructured and noisy data to be sifted through for suggested features. In this thesis, a novel method of extracting potential new features from online user reviews is proposed, using semi-automated machine-based analysis of the online reviews. In this research, a semi-automated tool is implemented in Python by extracting, classifying and ordering requirements of the product users from their online reviews. The approach incorporates a sequence of text processing and classification techniques. Online reviews of two software products, Jira and Trello, are used as cases for the feasibility test and optimisation of the process proposed. The main process includes firstly using a supervised machine learning approach to distinguish sentences in the reviews that represent new features, from other aspects of the reviews. Then, after a de-duplication process, multiclass multi-label classification is applied to the remaining set of candidate release requirements to group them into functional groups. This grouping of the extracted features provides a structure for the product manager to subsequently analyse and prioritise the extracted candidate features. The target categories are manually defined from knowledge of the product functionality. Several techniques for extracting the features are explored, and the optimal models for classification are identified based on precision, recall and F-measure metrics. Several iterations of experiments are conducted with control variables such as n-grams, sentiment analysis and training sample size. Three learning algorithms are

used in the experiments such as Naïve Bayes (with multinomial and Bernoulli variants), Support Vector Machines (with linear and multinomial variants) and Logistic Regression. Semantic similarity measures are used in the pre-categorization for the reduction of sentence duplication. The ranking of user requirements by category and date is performed by term-weighted frequency, which reflects frequently mentioned features by app users. In the evaluation of classifications, our best models show good results of performance based on measures of precision, recall and F-measures. In binary filtering, linear SVM with the input of review sentence and sentence sentiment, using 4-gram technique and validating by 10-fold is the best model in filtering sentences of feature requirements with a precision of 90.9%, recall of 91.1% and F1 of 91.0%. The corresponding model also performs well with the simulation of input data by splitting the experimental dataset and by an external dataset. For multiclass multi-label classification, Linear SVM with 5-fold validation and unigram technique shows the top performance on 5-class classification with 86% precision, 78% recall and 82 % F-scores on the experimental dataset and 89% precision, 81% recall and 83% F-scores on the simulated dataset. Overall, the research conducted has confirmed the feasibility of semi-automated extraction of candidate release requirements from a large volume of unstructured and noisy online user reviews. Furthermore, specific machine-based analysis techniques have been evaluated and recommendations are made based on commonly used metrics. This proposed process can now be further expanded and extended to usefully support future product release planning.

Keywords: software requirement, feature request, feature ranking, word frequency, machine learning, SVM, classification, categorization, semantic similarity, sentence analysis, online review, application evolution, product enhancement

Contents

List of Figures

List of Tables

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Auckland University of Technology

July 2017

Phonephasouk (Noi) Volabouth

Acknowledgements

This Master's research had been conducted at Auckland University of Technology (AUT), Auckland, New Zealand, under the supervision of Jim Buchan as primary supervisor, Professor Stephen MacDonell as secondary supervisor, and Dr Sherlock Licorish as technical advisor.

Firstly, I would like to acknowledge Jim Buchan for guiding me throughout this master's research. I thank him for giving me the freedom to work on this thesis. Though busy, he always provided me with advice, support and encouragement. I am grateful for his devotion. Also, I would like to thank Professor Stephen MacDonell. His invaluable support and suggestions were much appreciated. My gratitude further goes to Dr Sherlock Licorish who provided those state-of-the-art academic articles which were very beneficial to this thesis.

Finally, my thanks extend to my parents and my brother for unconditional love and encouragement. My gratitude is to the faith they have given me. Without them, I would not have been able to get this far.

Chapter 1

Introduction

1.1. Background and Motivation

Throughout its lifetime, a software product is expected to continually improve and evolve to ensure its efficiency, utility and desirability to users, both existing and potential (Galvis Carreño & Winbladh, 2013; Pagano & Maalej, 2013; Shah & Pfahl, 2016). This involves incrementally developing new product features and improvements that are integrated and released to users, often at some regular frequency, the product release cycle. Which features and enhancements to include in future releases is an enduring challenge, however. It can be a complex process which includes gathering, analysing, synthesising, and prioritising competing ideas for new features to decide which should in the next release, and which should be left for subsequent releases. The work in this thesis is aimed at supporting the gathering of potential new features and grouping them into a structure to assist with analysis and prioritisation. Ideas for new product features could come from a diverse range of sources including feedback from current users and reviewers, competitive market forces, Product Owners, and innovation opportunities created by new technologies or ways of working. The research in this thesis focuses on supporting the identification of new product features from user reviews, specifically those electronically available online. The notion of feedback from current users as a source of ideas for fixing bugs and enhancing product features is common in software development and makes sense since software applications that create value to the end-user by meeting their needs and expectations are more likely to thrive in that market. With the uptake of social media and online reviews, the vendor-driven, manual gathering of user feedback through survey forms, questionnaires, interviews or observations has now been augmented with online feedback channels driven by the users themselves. Online user reviews, comments and feedback provide a low effort, always-available mechanism for end-users to suggest ideas for new product features and is therefore a potentially rich source of requirements for future product releases. While this online user feedback is relatively simple for product suppliers to access, and often there is a large number of participants, it is less straightforward to translate this corpus of online data representing the end-users' voice into a useful set of potential release requirements for prioritisation. The large volume of information, its lack of structure, as well as the diverse quality and relevance of the feedback can make manual analysis of this data a time-consuming and costly-undertaking. A significant part of

the effort is identifying those parts of the users' comments that relate to a new requirement from among other content in the review. For example, some users might complain about existing features or want to share their experience on the usage of software, intermingled with constructive comments about possible product enhancements. Addressing this challenge in filtering useful information from those online comments which might contain a mixture of both informative and less useful data (noise) is the aim of this research. In this thesis, we explore, propose and evaluate some semi-automatic techniques for supporting the extraction of candidate release requirements from such a body of noisy online user feedback, reducing the manual effort.

With an increase in the use of the Internet, the diversity of online information sources on products have significantly increased. In addition to the original software product website that might enable users to comment through the application, there exist online discussion boards and forums on different sites. For example, expert software reviewers post their feedback comments of an application when it is released. Or, a software application can be "strategically" reviewed by competitors of similar type to compare the positive and negative aspects of its features and performance. While these online information sources on product feedback have proliferated, there is no consistent standard and the quality and structure can vary greatly. The process of extracting new features proposed in this thesis aims to be able to use this diversity of sources, although a specific popular product review website, https://www.g2crowd.com, is used in this research as an example of unstructured, noisy data contributed by both product review experts and end-users.

Having argued that the manual extraction of useful product feedback information from the large volume of noisy online data can require a lot of effort and resources, but that such information is potentially high value for software evolution and improvement, the mechanism for automating this process is now discussed. In the research proposed, we aim to be able to filter this high volume of online user feedback data faster than manual processing, with minimum human intervention.

One fruitful avenue of emerging research is the use of text processing techniques for the extraction of valuable information from a corpus of electronic text (Genc-Nayebi & Abran, 2017; Martin, Sarro, Jia, Zhang, & Harman, 2016; Shah & Pfahl, 2016).

Text processing is the exploitation of textual content, usually in the form of unstructured natural language, with the intention to retrieve meaningful information either by statistical or machine learning methods, for further analysis, evaluation, reporting and visualization. Text processing tasks, in general, include information extraction, categorization, classification, clustering, sentiment analysis or opinion mining, summarization and entity relationship analysis. It involves the application of techniques for information retrieval, natural language processing and lexical analysis such as data mining, pattern recognition and predictive models. Text processing enables the extraction and analysis of documents in an automated way, to discover useful trends and filter informative data hidden within large volumes of textual data. For example, it has been used to support business problem-solving by providing valuable knowledge and insights, derived from textual content such as electronic documents, emails and social media forums.

With an increase in the popularity of mobile applications, many studies have been heavily focused on extracting online information to serve the evolution of mobile applications. This includes the application of text processing for data classification, clustering, topic identification, or summarisation. Research using these techniques have had mixed success and include tools to extract feedback summary, user sentiments, user requirements, software defects, and evaluation methods (Genc-Nayebi & Abran, 2017; Martin et al., 2016; Shah & Pfahl, 2016).

Although fewer in number than studies automating the analysis of mobile application user reviews, there is some research applying text processing techniques to non-mobile software product reviews. For example, for supporting software development and evolution, text processing techniques are used to extract user requirements from expert reviews (Bakar, Kasirun, Salleh, & Jalab, 2016), Q&A software community (Xiao, Yin, Wang, Yang, & Chen, 2015), and software product reviews (Jiang, Ruan, Zhang, Lew, & Jiang, 2014). Mostly relying on text clustering, these approaches, so far, have produced results with relatively low precision (roughly within 50%-70%) (Bakar et al., 2016; Jiang, Ruan, Zhang, et al., 2014; Xiao et al., 2015). The aim of this thesis is to improve on these results by firstly evaluating alternative techniques for text processing and then applying the best of these (in terms of Precision, Recall and F-measure) to our proposed process. This thesis is the initial part of a broader research project, which has as its main goal an alternative approach to support

decision making in the evolution of software products in general, by exploiting the potentially diverse range of available resources through the use of optimised text processing techniques. This includes information from diverse sources such as online user and expert reviews, information on competitor products, as well as ideas from employees of the product vendor such as the Product Owner, customer services, marketing and technical staff. The scope of this thesis is restricted to a preliminary study of online user product reviews with the purpose of extracting and grouping software feature requirements reflected in user comments available online.

1.2. Research Questions and Research Approach

To achieve the aim of this thesis, the main research question (RQ) is proposed and further divided into three sub-questions (SQ1 to SQ3):

RQ: How can modern text processing and machine learning techniques be used to support the extraction of a set of candidate requirements from large volumes of online user product reviews?

> **SQ1:** What is a suitable text analysis process to achieve the extraction of new features from such a large corpus of noisy reviews?
>
> **SQ2:** What specific text-processing and machine learning techniques are candidates for the proposed feature extraction process?
>
> **SQ3:** Which of the candidate techniques perform the best in terms of Precision, Recall and F-Measure in this context?

To address the research questions, online reviews of two software products, Trello and Jira, from a popular software product review website, https://www.g2crowd.com, is used as text data sources. Based on a review of relevant research papers, a high-level process for extracting new features is designed (RQ1). Candidate machine learning and text analysis methods, corresponding to this process, are identified from relevant research papers (RQ2). These methods are then evaluated by applying different combinations to extract potential new product features from the reviews and calculating the Precision, Recall, and F-measure

metrics of the results of the different approaches. The combination that produces the best measures answers RQ3.

1.3. Scope

The research in this thesis is part of a wider research project, which aims to develop an approach to automate the dynamic creation of a structured list of candidate requirements for future product releases using a machine-based analysis and categorise a stream of relevant electronic data from a number of sources. We focus on proposing and evaluating techniques using online sources of user reviews only, with a view that the design can be expanded easily to process the other data sources mentioned.

For an evaluation of proposed techniques, measures for evaluating text analysis are used to assess the performance of experiments. Precision, Recall and F-measures are selected, as well-established evaluation methods designed for machine learning approaches, which provide per-class assessment of learning models (Chen, Lin, Hoi, Xiao, & Zhang, 2014; Galvis Carreño & Winbladh, 2013; Guzman & Maalej, 2014; Panichella et al., 2015).

The current approach is based on supervised machine learning and is semi-automated in the sense that it still requires some human intervention. A more automated approach using unsupervised learning is expected for future research, particularly as part of the wider research project.

For the data source of this research, an open-source website of online reviews of software products Trello and Jira, has been used. These two software products are selected as a sample case study representing other online product review sources. They are products with regular releases and with a large number of user and expert reviews, and they include “noise” as well as suggestions for new features. This ensures a sufficient amount of data for testing and evaluating the proposed analysis techniques and models.

This research focuses on online reviews in English content, whereas those reviews in foreign languages other than English, are beyond the scope of this study.

Furthermore, this research is meant to extract solely product requirements as mentioned by users, while other potentially informative data like bug reports are considered as noise in the training of machine learning models.

The prioritisation of extracted requirements in this thesis is restricted to the term-weighted frequency of extracted candidate requirements. However, more sophisticated models for feature prioritisation that might require larger and more varied datasets and appropriate comparison techniques should be further studied and assessed to enrich the extracted information regarding candidate requirements for product evolution.

Additionally, the visualization of extracted product requests is proposed for the future research project, where a sliding window illustrating the software product requirements by a period is recommended.

1.4. Example Use Scenario

Consider a development manager of a software vendor, whose main responsibilities are to manage the development of three cloud-based applications. These applications, as ready-to-use and efficient platforms, have attracted many online customers around the world. The applications are English-based. They enable users to post comments and feedback for checking the products' rating. On the discussion boards, users are free to interact and share their experiences on the use of applications; they can have their questions answered or have their problems addressed. However, after a while, it can be noticed that the online comment boxes are overly active when a new version of cloud applications has been released. Some users have complained about its utility, calling for some missing functionalities. Others have faced some kinds of bugs on versioning consistency; they cannot reach the data they used to in the previous version. To address this, the software manager gets a few people to manually summarise the feedback and comments within the current release of the applications, but it is a very difficult and time-consuming task because the number of comments is overwhelming. At the same time, some important customers have started a couple of external forums discussing the performance of the cloud applications. To deal with this, the app manager requires the development team to work on what customers are discussing on these forums.

The workers have complained that they are overloaded and required the team to be doubled. This would be costly to the company. The software manager hesitates to make a decision, weighing the cost and the benefit of gaining information from customers' feedback. By applying the current proposed approach which is aimed to filter and sort informative feature requirements from those raw reviews, the software manager can obtain his required information under a reasonable time and cost of operations.

1.5. Contribution

In this thesis, we propose and evaluate a tool to automate the process of requirements extraction from online comments to support the decisions related to release planning. While the text analysis and machine learning (ML) techniques used are not new, the process sequence and their adaptation to the application context are novel and grounded in experimental evaluation.

By initially focusing on online reviews, the contribution of this research is made to the feasibility of the research project to facilitate relevant decision makers on software improvement in future releases with the exploitation of existing online resources in an automated way. This study can be used as a platform for future research regarding the application of natural language processing techniques to extract product requirements for software development planning. This research also demonstrates the feasibility of the current approach which enables the potential for other researchers to extend the study. The contribution for practitioners is related to demonstrating the feasibility of a data-driven mechanism to support decisions about release planning. This research further helps amplify the involvement of end-users and reviewers whose voices, though critical and influential, might otherwise be ignored in the evolution of a product.

1.6. Organization

For the purpose of this thesis, the organization of our research is outlined as follows. In Chapter 2, relevant studies to our research are examined. First, the use of text processing techniques and methods to support the improvement of mobile

applications such as feature classification, clustering, ranking and comparison is explored. Then, some approaches for the development of general software applications are examined. Furthermore, similarity measures are studied to detect the likelihood between strings and semantics to deal with text duplication. At the end, our reviewing of the literature is summarized to select techniques and methods to conduct our research experiments.

In Chapter 3, the design and implementation of this research setting along with the justification are described. All techniques and methods used in the research experiments are discussed. Firstly, different techniques for data/text pre-processing are examined, after which is a discussion of attribute extraction and selection methods. At the next step, the examination of data classification using supervised machine learning algorithms is made. Methods for evaluating our classifications are justified, before the technique for ranking of extracted features from classifications is investigated.

In Chapter 4, we investigate the outcomes of research implementation and discuss the findings. First, the dataset and classification methods are evaluated. Next, the results of classification which include binary classification for filtering feature requirements and multiclass multi-label classification as categorization process are presented. After that, the outputs of feature ranking per defined categories are examined. In the last section, the results and findings from our experiments are associated with relevant literature.

Finally, in Chapter 5, we conclude the research experiments which are promising with reasonably effective performance and results. In addition, limitations and threats to the validity of our research approach are discussed. The chapter is ended with a consideration of potential future research.

Chapter 2

Literature Review

The objective of this thesis is to propose and evaluate solution for retrieving useful information to support decisions about the evolution of software applications in future releases based on online feedback from end-users. The proposed approach involves the use of semi-automated text processing techniques to online feedback. To build a foundation in the field of text processing, in this chapter, we examine recent relevant studies, whose approaches have embarked on the application of text processing techniques to support software development engineering. We seek to gain an understanding of current problems and existing solutions regarding the design, process and validation, so that optimal techniques and methods for this research can be synthesized and implemented.

The organization of this chapter is as follows. In Section 2.1, we explore the use of text processing in support of mobile applications, which is divided into approaches for classification, clustering, ranking and comparison of mobile application features. The techniques addressing the solutions for non-mobile applications are described in Section 2.2. In Section 2.3, we examine the studies of similarity measures as a method for detecting and reducing duplication in textual content, both by string and semantic likelihoods. In the final section, Section 2.4, we summarize all techniques from our reviewing of relevant literature for the selection and manipulation in this research design and implementation.

2.1. Text Analysis for Mobile Application Support

Mobile applications (hereinafter referred to as ‘mobile apps’) are distributed through online app stores and users frequently write reviews through the reviewing process on those app stores. The application of text analysis techniques to these online mobile app reviews is a popular research topic, with a number of utilisation in the app development process. There are several relevant studies focusing on extracting useful insights from mobile app reviews and descriptions, which are available on websites, social media pages and mobile communities. With the increasing use of mobile and smart devices in daily life, the development of mobile apps is at the very centre of interest. Users are encouraged to provide further feedback comments, besides simple star ratings, after usage of the apps as per their preferred platforms. This volume of textual descriptions is increasing every day according to the

popularity of mobile apps, which have become a big data source for mining useful information and knowledge for both consumers and developers for improving the apps in the future. In this section, we explore relevant backgrounds and techniques for mining mobile app reviews in terms of feature classification, clustering, ranking and comparison, in order that suitable techniques can be selected and adapted to the current research's setting of general software products.

2.1.1. Feature Classification

Guzman and Maalej have presented a fine-grained method for feature extraction and a fine-coarse method for summarising the topics for extracted features (Guzman & Maalej, 2014). With a large amount of mobile app reviews, the proposed methods can be used to retrieve and categorize features that reflect the quality of specific mobile apps, which is beneficial to provide app developers information regarding user requirements, sentiments and experience for enhancement in the next release. In the study, users' feedback data of specific mobile apps from two different platforms: Google and Apple were exploited. Data processing was made at sentence level in two separate stages. First, the data were pre-processed for feature extraction using POS tagging, stop word removal and word lemmatization techniques on Natural Language Toolkit libraries (NLTK). The terms associated with informative features were then extracted by a NLTK collocation algorithm and WordNet synonym dictionary in the form of bigram collocations (common phrases of two words). The second processing stage was to analyse the sentiments of review documents (or sentences) by using SentiStrength sentiment extraction tool. This tool enabled the identification of sentiment (positive or negative) of each sentence by assigning scores to sentiment words (usually adjectives). At processing stages, a fine-grained summary which included key features and their sentiment scores were listed. Finally, Latent Dirichlet Allocation (LDA), a topic modelling algorithm, was applied to the feature list to group and assign topics to the document features. This learning approach was evaluated by precision, recall and F-measures where true set samples were manually arranged by human labours. The performance of the machine for fine-grained feature extraction with topic modelling by average was not as high (59% precision, 51% recall and 55% F-score); however, the generated summary of one specific app achieved very satisfactory validation scores (91% precision, 73%

recall and 81% F-score), while the others were relatively low. This might be explained by the variant quality of review text on different categories of mobile apps such as game, social media and file management.

Processing text documents at sentence level is appropriate for the type of research in this thesis, since it is better suited to the removal of irrelevant information (noise) in the data, compared to working at the document level. This approach helps to minimize the chance of overlooking important new feature requirements. Using bigrams might be suitable for mobile reviews which tend to be short with limited space on mobile device screens, but with longer review text, for example online software reviews where users can write as long a comment as they want, bigrams are less likely to be as effective. For example, sentences with many key features (noun phrases) and sentiment terms (adjectives) can be sorted incorrectly or get confusing using bigram techniques.

One other point that should be noted is that no date or time of comments are considered in the research reviewed. The chronological date can be important for developers to prioritize the features for enhancement by the next release. Also, regarding the use of review sentiments, not all relevant sentences present both key features and sentiment terms. These sentences (for example those with neutral sentiments) might be eliminated or ignored through the analysis process.

Therefore, in this thesis, we commit to experiment at the sentence level. Besides bigrams, other n-grams techniques (1-grams, 3-grams, 4-grams) might be used in this research to test the performance of classifications. In addition, review dates are included in this thesis's experiments. For sentiment analysis, instead of using sentiment terms, we rely on sentiment scores, where all documents cannot be ignored, whether they are of positive (> 0), negative (< 0) or neutral sentiments ($= 0$).

Maalej and Nabil have found in their study that the use of both metadata and review text can boost the results of mobile app review classification (Maalej & Nabil, 2015). This study focused on categorizing app reviews into four pre-labelled groups of 'bug reports', 'feature requests', 'user experiences' and 'user ratings', by proposing supervised learning methods for binary and multiclass classification. The authors

experimented on three classification algorithms, Naïve Bayes, Decision Tree and Maximum Entropy, with which a different combination of input data components (pure review text and review metadata like verb tense, rating and sentiment) were analysed. The sample reviews for this study were randomly selected from three mobile apps each, from two platforms (Google Android and Apple iOS) (totally 6 apps). The data pre-processed techniques used included stop word removal and lemmatization. Input data were review text, rating, sentiments of reviews and verb tense of review text, in which a combination was interchangeably tested. Classification results were compared to the manual text process using precision, recall and F-measures. From the experiments, it can be suggested that binary classification perceived more accuracy than multiclass classification (precisions and recalls by class ranging from 71%-97% for binary, compared to roughly 50%-60% for multiclass). Also, among the three proposed classifiers, Naïve Bayes performed the best, particularly when the combined text and metadata were used (Maalej & Nabil, 2015).

Extending this study, the bigram technique was applied which apparently helped boost the performance of proposed classifiers, especially when the bigram inputs were used in combination with lemmatization, the percentage or precision surged up to 92% and the recall could reach as high as 99% for a specific class (averaging from 70%-99% by class) (Maalej et al., 2016).

The two aforementioned studies have proposed a similar approach to the current thesis, where different machine learning techniques (for binary and multiclass classifications) have been applied to mobile app reviews and evaluated by precision, recall and F-measures. Similarly, we also use text processing techniques like bigrams to boost the performance of binary and multiclass classifications. The results of evaluation of this study might be useful to compare with the current thesis's which conduct a similar type of experiments.

Similar results have been suggested from a study by Panichella et al., which introduces a learning approach using a combination of natural language processing (NLP), sentiment analysis and text analysis techniques for analysing mobile apps reviews (Panichella et al., 2015). This study also experimented on classifying mobile apps reviews at sentence granularity into multiple classes of features, namely

information seeking, information giving, feature request and problem discovery. They test different classifiers such as Bayes, SVM, Logistic Regression, J48 and ADTree and with different training and testing sample scales. The study argued that J48 classifier with NLP + sentence sentiment and 60% data for training offered the best classification results (84.5% precision, 84.7% recall and 83.8% F-measures).

It is interesting that applying review text and metadata such as document sentiments can help increase the performance of some experimented classifiers. However, it would have been useful to further test this with other machine leaning classifiers such as SVM, or n-grams techniques, to ascertain whether it can improve the results of classification. In addition, the injection of training dataset for machine learning techniques by dividing the percentage of review data might offer comparable results of classification to the current thesis's experiments.

2.1.2. Feature Clustering

In addition to the extraction of information regarding users' requests and complaints, there are studies focusing on automating the grouping or categorizing of those extracted features, which target to support the development of mobile apps.

A semi-supervised approach for generating and categorizing features to support the evolution of mobile apps which meets users' needs has been introduced in (Galvis Carreño & Winbladh, 2013). In this study, information retrieval techniques for topic modelling and sentiment analysis were employed to extract user feedback for improving software quality in a forthcoming version. It was important to identify user requirements among different versions of software applications which were directly correlated to the defects of applications to ensure users' satisfaction and loyalty in mobile apps. Feedback comments of both free and commercial Android apps were used in the experiments. The data were pre-processed by tokenizing words, removing punctuation, symbols and spaces, and lowercasing words. Next, most common words (stop words) were removed and short sentences with less than three words were considered as non-informative and eliminated to reduce noise. The data were then converted to vectors to generate the document corpus in the form of bags of sentences and word lists. Three classification methods, namely manual classification, K-medians with Jaccard similarity coefficient between pairwise

sentences and ASUM were used to automatically analyse and identify topics of pre-processed data. K-medians was a clustering algorithm similar to K-means, but the medians of vectored data were used instead of the means to define clusters. Aspect and Sentiment Unification Model (ASUM) was an extension of the topic modelling algorithm, Latent Dirichelt Allocation (LDA) combining with sentiment analysis. Manual classification was performed by three software-specialized experts to generate true sets for evaluation. To make the results more comparable, the numbers of clustering centroids (k) were selected to match with manual clustering. The reports were generated and the results of each classifier (manual and semi-automated) were compared to the generated true sets. The results showed that the proposed ASUM was the most promising classifier, outclassing both manual and K-medians approaches, with the proportion of precision up to 95%, recall 77% and F-score 84% by average, particularly when the number of cluster centroids was high (k = 150).

In another study, App Review Miner (AR Miner), a comprehensive framework to help application developers analyse information from mobile app reviews has been proposed in a study by Chen et al. (2014). The proposed framework included five main tasks: raw review pre-processing, extraction, grouping, ranking and representation of informative reviews from mobile app feedback. First, raw data (review text, user rating and review time stamp) were retrieved from mobile app reviews. This study describes a case study of Android applications from Google Play Store. The retrieved raw data were converted to sentences using a sentence splitter provided in the LingPipe library and pre-processed by word tokenization, non-alphabetic removal (numbers, special characters and symbols), lowercasing, white space removal, stop word removal and word stemming. This study used sentence-level granularity as it was more convenient and accurate to distinguish informative and non-informative reviews. Next, at the extraction step, Expectation Maximization for Naïve Bayes (EMNB) was selected as a semi-supervised learning algorithm. EMNB was used for classifying review text as “informative” or “non-informative”; in other words, it filtered out non-informative reviews. The benefit of using EMNB was that it only required small amounts of labelled data to train the classifier. Also, EMNB provided probability scores to be used in the ranking step. The topic modelling algorithm, Latent Dirichlet Allocation (LDA) was chosen to categorize

similar informative reviews at the grouping stage. Topic modelling was suitable to sort reviews into overlapping groups where one review could belong to more than one group. In the grouping experiments, it was suggested that LDA outperformed Aspect and Sentiment Unification Model (ASUM) in terms of generating topics and grouping review text. Next, for the ranking step, a ranking model based on feature weights of each group and group instances was proposed. The model assigned a score of importance to each group to order the groups; also, in every group, group instances were sorted using the same algorithm. Time stamps and ratings were associated in the calculation of important scores. Finally, the group scores (of the top ten groups) were plotted in a radar chart with group labels to visualize the featured reviews by group and weighted by level of importance. Extracting and grouping steps of the framework were evaluated using precision, recall and F-measures. In the extracting step, a case study app review had the average F-score of above 75% in filtering informative features. In grouping evaluation, LDA was compared with ASUM; where LDA clearly had a better performance than ASUM. In the ranking stage, Normalized Discounted Cumulative Gain (NDCG) scores and Recall (Hit-Rate) were used as validation measures. The performance of the proposed raking model was compared to true ranking data available from the case study applications. The result showed that LDA still performed better (hit-rate = 70% and NDCG-top ten = 55%).

In this thesis, we focus on a similar approach of semi-supervised learning, where manual classification is low. It is useful to experiment with clustering techniques, since it can lead to an approach to extract and categorise the relevant feature requests of software products that minimises human intervention. However, it can be noted that in terms of using k-means, k-median, LDA and ASUM algorithms, the number of the centroids of clusters (k) can be very difficult to define, if analysis experts are not familiar with the data. Nevertheless, it can be useful to test these topic modelling and clustering techniques and, at the same time, to explore more automated ways for information categorisation, especially in the future research project. In addition, we aim to follow a similar path of exploiting feedback from commercial software, as it tends to be informative in terms of feature requests and complaints when cost is involved. Using a combination of commercial and open-source apps also widens the analysis domain, broadening the potential applicability contexts.

2.1.3. Feature Ranking

After identifying potential features for future releases, the decision on which features should be included in the *next* release, and which will be left for future ones is made. Therefore, this section investigates techniques and methods proposed in literature to prioritize features in mobile app development and that might be adaptable to the objective of this thesis.

Text mining techniques have been presented to explore different insights associated with feature requests in Android Operating System (OS) reviews and topics of defect requests (Lee, Licorish, Savarimuthu, & MacDonell, 2016). In this experiment case, they extracted feature requests, as labelled in Android OS feedback community, ordered by the release dates of Android OSs. Data preparation was conducted by removing irrelevant HTML tags and any non-English (foreign) characters. The data were next pre-processed using POS tagging and n-grams techniques. Pointwise mutual information (PMI) was then applied to indicate interconnections among requested features for graph-based analysis using Social Network Analysis model (SNA). In the SNA graph-based model, nodes represent features of interests, while PMI values were used to identify how different nodes were interconnected by edges. The SNA model in this study enabled different measures of relationships of features by nodes and edges, regarding the degree of connection (popularity), the level of importance and the level of influence, where all measures in the experiment were ranged as high, medium and low. The degree of popularity was defined by the total number of individual edges connected to the nodes (features). This measured how popular a specific feature was by considering the degree of connection with other related features. In other words, this measure showed the feature requests for enhancement that were discussed more frequently in the review text. The level of importance in SNA was indicated by the closeness or distance between the nodes. The distance of two features determined the level of importance the features had for each other. A case study of this work showed that mostly-mentioned features were not always subject to the same level of importance. Level of influence was defined by clustering coefficients which represented the inter-correlation between two feature nodes. It measured the influence or the correlation between features. Dealing with highly-connected features could be more complicated, and, in turn, more cost,

time and labour-intensive, because making changes to such features could affect all the other related features.

Overall, this study showed an interesting use of text processing techniques to support developers of applications in prioritizing features for enhancement by considering different measures of feature request analysis.

In order to prioritize mobile app features extracted from users' reviews to support developers on the improvement of the application, different approaches, namely individual attribute-based ranking, weighted ranking and regression-based approaches have been introduced (Keertipati, Savarimuthu, & Licorish, 2016). The individual attribute-based approach prioritized proposed features by focusing on one single attribute, for example poor rating or the most-frequently mentioned feature (AR-miner (Chen et al., 2014) was an example of this type of approach), but not both at the same time. This could offer a quick overview of feedback features by individual attribute of interest. It was also useful in case there were limited data available. The downside of this approach was that considering only one attribute could sometimes overlook other important aspects when more than a single attribute played a role. To deal with this limitation, a weighted approach was introduced. This approach enabled the integration of more than one attribute in ranking associated reviewing features which could be conducted by considering weighted scores for each interested feature. In this approach, weighted values which showed the level of importance of each attribute ranging within [0, 1] should be assigned before the ranking scores could be computed (for example, assigning 60% importance to feature frequency, and the rest 40% to rating). Alternate to this, there was also a regression-based ranking model which allowed the integration on relevant attributes. The regression-based model was a data-driven approach to indicate relevant attributes to focus on. This approach relied on a regression model to calculate the correlation between focused attributes (or dependent variables). The ranking was identified by the correlation coefficient of each variable and their level of significance.

To sum up, these studies regarding feature ranking and prioritisation offer several different techniques and approaches for the prioritisation of extracted features of interest. In this thesis, we test on term-weighted ranking techniques of product

requirements by using term frequency, with the restriction of the research timeframe, to point out the feasibility for the application of advanced prioritisation approaches in the future research project.

2.1.4. Feature Comparison

Beyond the scope of dealing directly with mobile apps features, researchers have put further efforts into how these features have been responded to, in each release or development cycle of software features. Studies proposing different methods to compare user requests over a period are to be subsequently examined.

Based on the AR-Miner model (Chen et al., 2014), an approach called CRISTAL (Crowdsourcing RevIews to SupporT Apps evoLution) to compare mobile app reviews posted by users to change logs for app development over app releases has been proposed (Palomba et al., 2015). The purpose of this approach was to estimate if suggestions by reviewers were resolved or responded to in the new release of apps by manipulating links between user requests from the review comments and developers' reactions (issues and commits) appearing in the notes for source code change. The proposed method also associated user satisfaction toward the success of such mobile apps which reflected in the ratings provided by the same users before and after each release. This could be beneficial to track changes or updates of a mobile app before and after its release through crowdsourced components, which linked to qualitative improvement of the app. In the study, user reviews from Google Play Store were collected and classified as informative or non-informative reviews using AR-Miner software (Chen et al., 2014). On the other hand, source-code change logs were explored to retrieve the data regarding the issues and commits (reactions) based on release dates of applications. Then the data were pre-processed by splitting camel case and underscore terms, removing stop words and word stemming. Stop words in this experiment were identified by term entropy, which was the ratio of the number of occurrences of the term in a review and the total number of occurrences in all reviews. The entropy was ranged between [0, 1], where words with a higher number of entropy (over three quantile) were eliminated. Next, links between issues and review documents and commits were created by using Dice coefficient (with [0.1, 1.0] threshold value, $\lambda = 0.6$) as a similarity measure before reports regarding the relationship between reviews and changes committed and monitoring

components could be generated. The performance of the proposed approach was validated by manual true sets. By average among ten sample Android apps, the proportion of link generations was fairly promising with 77% in precision, 73% in recall and 75% in F-measure.

In another study, insights on how feature requests by Android OS users are associated over its lifecycle have been investigated (S. A. Licorish, Lee, Savarimuthu, Patel, & MacDonell, 2015). This study explored feature requests posted on the Android OS community over a period between Android OS release dates. The data were cleaned by removing HTML tags and foreign characters before descriptive statistical analysis could be performed to gain a quick overview of the data. Then, POS tagging and n-gram techniques were employed to extract requested features (unigrams of noun) and a pointwise mutual information (PMI) measure was applied to compare how those features were correlated to each other. The experiments observed top-20 features that were mostly mentioned in the requests for enhancement. It could be suggested from the results that over each period of Android OS versions, specific features were usually requested for improvement. Also, when the gap between version releases was lengthy, the number of requests for enhancement increased, as expected. Moreover, it was very likely that requests for enhancement often involved changes in many correlated features.

Extending this work, extracted top-20 feature requests have been examined in comparison with developers' notes (S. A. Licorish, Tahir, Bosu, & MacDonell, 2015) to see how these enhancement requests by Android OS users have been responded to by development teams. Release notes by developers which were available in Android OS community were retrieved and cleaned using the same techniques as the former study. To investigate the correlations between feature requests and responses, responded features which matched top-20 features in enhancement requests were observed. The results of the experiment appeared to suggest that developers of Android OSs tended to change the way they interacted with users' requests over the observed period in an upward trend, where user's requests for improvement were strongly correlated to developers' responses. In addition, over the OS lifetime, the number of requests for enhancement generally increased, particularly in some specific OS versions which showed that it took time

for developers to accommodate the changes according to user suggestions for enhancement. The study showed how users were passively involved in the development of a software application which were reflected in the success of Android OS systems.

These studies of mobile feature comparison have demonstrated useful techniques to link and compare mobile app feature reviews which are on users' side to app release notes which are on developers' side. It can be seen that besides the use of online app reviews, other online data sources are also studied in a support of mobile app enhancement. While the contexts of this research on mobile apps has a restricted data set and simpler extraction process compared to the context we are considering in this thesis, nonetheless, they provide a useful guidance to consider when selecting techniques for the wider research project discussed in section 1.3.

2.2. Text Analysis for Desktop Software Development Support

In addition to relevant studies regarding the use of mobile app reviews to support the app evolution, the current research on the application of text analysis and processing techniques in a non-mobile context is explored to gather candidate approaches for this thesis. The focus of this survey on non-mobile research relate to the aspect of data sources, techniques and methods for analysis and evaluation, and evaluation of results.

In a recent study, the application of information retrieval techniques has been proposed to support a new software production line by making use of online software reviews (Bakar et al., 2016). This study showed how key features could be extracted from online opinions regarding the existing software by using a semi-supervised approach. To retrieve data about software requirement specifications from user perspectives was an easy approach to access straightforward data that better matched the requirement of certain user groups. This was because user requirement information, especially that used by large software developers was not usually publicly accessible, but was usually discussed via web boards or other discussion blogs, for example, online repositories (Yu, Wang, Yin, & Liu, 2013) or Q&A software community (Xiao et al., 2015). Therefore, the extraction of software

features that users were enthusiastic about could be very useful for small software manufacturers or independent developers to produce a similar software application to meet the market needs. In the experiments, the authors relied on expert reviews of different educational software applications designed for children of different learning levels, since they were less likely to display noise, bias and sentiment, but provided more informative functional features and were well-structured. The data of this type were usually in the form of large-paragraph and long-sentence documents and the amount was not very large compared to online product reviews. The data sources used in the experiment were scraped from two independent websites (toptenreviews.com and superkids.com). After being scraped, the data were prepared at the pre-processing stage which included removing stop words, lemmatization, Part-of-Speech (POS) tagging, and duplicate words before translating into a term-document matrix by converting to vectors using Term Frequency-Inverse Document Frequency (TF-IDF) algorithms. Then Latent Semantic Analysis (LSA) with Singular Value Decomposition (SVD) which grouped similar documents based on the document matrix was applied to reduce the dimension of the matrix and, in turn, boost the performance of the classifiers. Three classifiers, namely K-means with Euclidean distance, Fuzzy C-means and Self-Organizing Map were used to automate clustering featured documents into categories. Also, key phrases were identified based on feature POS tags (bigrams and trigrams) and grouped by common phrase similarity using the Word Overlap method. The approach was finally evaluated by comparing the performance of humans (manual document clustering) by a truth dataset and machine automation using Precision, Recall and F-measures. The best result of the clustering validation showed an average of 62% precision, 82% recall and 70% F-measure where Fuzzy C-means outperformed K-means and SOM (Bakar et al., 2016).

In this study, the automated clustering of documents was not closely matched with the manual one in terms of the number of categories. Because machine clustering was based on probability, the number of clusters was smaller than the number obtained with the manual approach. Also, the classifiers were experimented on small-length datasets in analysis and evaluation. As a result, the retrieved result could be biased.

In a study by Xiao et al., data was retrieved from a popular Q&A software community site (stackoverflow.com) where either professional or amateur programmers and developers shared their knowledge and experience. The data was used to test the SVM classifier in terms of extracting requirement requests to support software developers (Xiao et al., 2015). The extraction of tagged questions along with related information could lead to useful requirements acquisition useful to the development of relevant software applications. The study focused on the categorization of questions tagged on the website whether it was a feature request or non-feature request. In this research setting, data were prepared by scraping all raw data including HTML tags. The raw data were then purged of irrelevant HTML tags, retaining only relevant text. The pre-processing steps in the experiments included text lowercasing, word tokenization, punctuation removal, stop word removal and word stemming. Also, the least common words (those appearing in the corpus less than three times) which would be regarded as trivial and irrelevant, were removed. Next, the pre-processed data were converted to vectors using a vector space model. Three classification algorithms including linguistic rules, pure Support Vector Machine (SVM) and SVM with requirement dictionary were chosen for the experiments and comparison. Linguistic rules which included a set of specific keywords were generated from the data tagged on the website as “feature requests”. TF-IDF technique was applied as a feature selection process for the SVM classifiers. SVM with requirement dictionary is the combination of SVM and requirement dictionary (keyword sets) to refine the performance of classification. It could be concluded from the results of the experiment that SVM (precision = 68.8%, recall = 75% and F-score = 71.7%) performed better than traditional linguistic rules (precision = 61%, recall = 55% and F-score = 57.8%). Also, when requirement dictionary was used, the classification performance slightly improved (precision = 72%, recall = 77% and F-score = 74.4%). The difference in the performances could be particularly seen when increasing the size of the testing samples (Xiao et al., 2015).

From this study, it can be observed that the use of linguistic rules or keyword sets might be domain-specific and labour-intensive as they relied on specific data existing in the documents to generate the rules. This suggests machine learning is a good candidate as an approach for text processing.

One other study suggested that software features extracted from online feedback was evidently overlooked by some developers who relied solely on Software Requirement Specifications (SRSs) for future software releases (Jiang, Ruan, & Zhang, 2014). Some of these features could be have immediate economic benefits if they were introduced in the software improvement for the next release, since they came directly from current users. In this research, Jiang, Ruan, and Zhang presented an approach for user requirement analysis by combining opinion mining and document clustering techniques with a utility-oriented econometric model to indicate critical software features that were related to sales and, therefore, worthy of future improvement. To validate the proposed approach, a case study was conducted by collecting reviews of an antivirus software (Kaspersky Internet Security 2011) from the website, amazon.com. The data were prepared at the pre-processing stage which employed a syntactic relation-based propagation approach (SRPA) and pruning SRPA (P-SRPA). SRPA was a fine-grained feature selection technique based on a bootstrapping approach. This method relied on a rule-based strategy and document POS tags to identify the terms relevant to candidate features suggested by users. Next, heuristic rules were applied to locate and remove unnecessary noise from the data. In addition, to further refine candidate features, pruning techniques based on SVM were used in P-SRPA. The extracted features were then categorized using the K-means method to cluster similar features, before opinion ratings based on the features could be generated. Finally, econometric model-based requirements analysis was used to identify the level of utility of the software from rating scores, extracted features and categories. The result of the experiment showed that the P-SRPA technique for the extraction of candidate required features perform much better than SRPA in terms of precision, recall and F-scores. This demonstrated that P-SRPA was an effective method to deal with data with a large amount of noise. Also, when the candidate features and categories were manually checked by a sample group of developers, those software features were confirmed to be useful, where most features were matched with the actual Software Requirement Specifications, while some features were found to be ignored in new release evolutions of the software.

Extending this study, they proposed SRPA+ techniques where the S-GN method for document clustering, were applied. S-GN proved to out-perform two baseline K-means algorithms (J-Kmeans and S-Kmeans) on clustering online feedback of a

software Kaspersky Internet Security 2011 and a mobile app, TuneIn 3.6. Also, the proposed GN algorithm, in which the most appropriate numbers of clusters were automatically identified by machine, was less supervised when compared to K-means (Jiang, Ruan, Zhang, et al., 2014).

Overall, there are much fewer studies for non-mobile context compared to those for mobile applications. Of these studies, three different data sources were common: software expert reviews, Q&A comments and online reviews of one software application. Clustering techniques are mostly used by the studies, with one study focusing on a machine learning approach. Most studies rely on the similar methods of evaluation as this current research thesis by using precision, recall and F-measures, and the results of performance are not very promising because of clustering process. These studies show that there is a feasibility to apply text processing techniques in a more general content than the mobile app development. However, not as many studies as those for mobile apps, text processing techniques and algorithms have been proposed to support the development of software products, which might be explained by the high availability of large volumes of mobile app data. However, with an increase in the amount of general online product reviews, it is important to extend the research of text analysis techniques to a more general context, as proposed in this thesis's objective, to broaden the horizon of this current field of study.

2.3. Similarity Measures for Detecting Duplication

In a single online review, there is a high potential that the very same feature requirement can be mentioned more than once. This is especially likely when working at sentence level, as proposed in this thesis, because every reviewer tends to discuss the same feature requirement in more than one sentences. Therefore, metrics to recognize text similarity are proposed to avoid extracting features that are essentially the same. Such duplication problems not only hamper the quality of extracted data, but also affect the accuracy of analysis in a negative way. To tackle the problem, similarity metrics have been used by other researchers to determine the likelihood of pairs of text or objects being similar or distinct according to some similarity threshold measure. Similarity measures for textual content have been

studied as metrics for detecting duplication (Bilenko & Mooney, 2003; Ektefa et al., 2011; Hadžić, Sarajlić, & Malkić, 2016; M. Li, Wang, Li, & Gao, 2010) and data integration for clustering purposes (Huang, 2008; Lin, Jiang, & Lee, 2014). Similarity measures as previously proposed can be classified into two groups: string similarity measures and semantic similarity measures.

String similarity metrics focus on tokens or characters to distinguish a pair of documents. Token-based similarity measures differentiate two string documents by tokenizing them into words or tokens and comparing the document vectors based on functions of similarity. Examples of this type of metrics include cosine similarity, Jaccard coefficient and Q-grams (Elmagarmid, Ipeirotis, & Verykios, 2007; Ferro, Giugno, Puglisi, & Pulvirenti, 2010; Gong, Huang, Cheng, & Bai, 2008; Hadžić, Sarajlić, & Malkić, 2016). Character-based similarity measures consider the number of transformations between two strings to compare a pair of strings. They rely on the number of editing operations as metrics of string likelihood. Some commonly used character-based similarity measures are Levenshtein distance, Jaro distance metrics, edit distance and affine gap distance (Bilenko & Mooney, 2003; Elmagarmid et al., 2007; Hadžić et al., 2016; M. Li, Wang, Li, & Gao, 2010).

Bilenko and Mooney have presented a machine learning approach to detect duplication in documents by using machine-learning algorithms. The study used the databases of restaurants from two different sources and a collection of research paper citations retrieved from search engines. The purpose here was to integrate the databases that shared a number of similar record sets, where noise such as misspellings of characters and numbers, different letter cases and word orders was presented. Measures of similarity used in the experiments were affine gaps, edit distance and q-grams (vector-space distance with TF-IDF), and the learning algorithms were SVM and decision tree. Besides the use of solely similarity measures for identifying data duplication, similarity measures were applied to a training set for learning approaches. The experiments showed that when applying similarity measures (edit distance and q-grams) to the training set, the learning-based approach using SVM outperformed every other experimented approach by F-measures (Learned edit distance = 87%; Learned q-grams = 89%, by average of

multiple data fields). Nevertheless, to achieve the best result, a good training set was very important (Bilenko & Mooney, 2003).

Even though string similarity measures are relatively efficient to compare two documents that exist similar corpus of tokens, they can be difficult in distinguishing documents with similar meaning, but with distinct sets of corpus. In other words, string similarity measures are poor at differentiating synonymous words. To address this limitation, semantic similarity measures have been proposed. Semantic similarity measures rely on the semantic meaning of the concerned words to compare two strings for equivalence. Two popular groups of semantic similarity metrics are knowledge-based and corpus-based approaches.

Knowledge-based similarity measures are based on manually generated ontologies. In knowledge-based ontologies, every node represents a word concept, and each concept is linked by an edge. The closeness (path length) between these edges is used to determine the relationship or similarity between each word pair. This type of approach is quite time-consuming and costly to develop and maintain word ontologies, which makes them restricted in terms of domains. Frequently used ontologies of knowledge-based measures are WordNet (general domain) and Gene ontology (bioinformatics).

On the other hand, corpus-based similarity approaches are introduced to address the problem of domain-specific ontologies. Corpus-based similarity measures exploit statistical information of words appearing in large text corpora to assess the similarity of texts. The size of documents (number of words) to be compared can be a challenge using this type of approach. Because every word in the documents is required for comparison, it results in a complex and time-consuming process. However, applying techniques for reducing text dimensions might help dealing with the problem at a level. Widely used corpus-based similarity measures are Pointwise Mutual Information (PMI) and Latent Semantic Analysis (LSA) (Bongirwar, 2015; Christoph, 2015; Ektefa et al., 2011; Y. Li, McLean, Bandar, O'Shea, & Crockett, 2006).

Li and his team have introduced a distinctive approach to detect similarity between sentences, which is suitable to the de-duplication process in this thesis. The study has

offered ready-to-use techniques that were specifically designed to work with text similarity at sentence level. They relied on WordNet lexical database to compute word order similarity and retrieve word synonyms. Meanwhile, Brown corpus was used to calculate semantic scores of words. The order similarity and semantic scores were then used to determine final similarity between the texts. The performance of the proposed method was very close to that of human upper bound in measuring sentence similarity (0.816 for proposed algorithm and 0.825 for typical human) (Y. Li et al., 2006).

Ektefa and colleagues have proposed an interesting threshold-based method to compute the similarity of two strings by using both string and semantic similarity measures. In this work, Jaccard similarity metrics were applied in combination with Latent Semantic Analysis (LSA) to define the semantic similarity threshold and calculate the similarity score. The proposed combined technique offered the most effective results in identifying the similarity to solve the duplication in records in the sample database at very high rates, particularly when the technique was learned by SVM (over 98% of accuracy, precision, recall and F-measure) (Ektefa et al., 2011).

This proposed approach is very effective to reduce redundant data. However, as it involves a time-consuming training process, it was decided not to use it in this thesis, because of time constraints. The adaptation of this approach is left to the future research project. In this thesis, we select to apply the sentence similarity measures as proposed by Li et al., which is quicker and fits our research purpose and approach.

2.4. Summary

Having explored a number of relevant research papers, Table 2.1 shows the summary of related studies as well as techniques, methods and models that can be useful to the current research. This table provides a list of techniques that are potentially useful for the process of automatically extracting the candidate release requirements from online reviews. They form a basis for evaluating alternative designs of the process and process steps.

In terms of extracting features and discarding non-features from the reviews, some pre-processing techniques of the text are required to convert the raw textual review

into key sentences, phrases and words. The common pre-processing techniques for text processing identified in Table 2.1, include stop word removal, POS tagging, word stemming and lemmatization. TF-IDF, LDA and n-grams are frequently used for feature extraction and selection.

Next, after pre-processing the review text follows the process of text analysis. Here, the reviews are filtered for product requirements, before those requirements can be grouped by their feature functionalities. This requires the application of machine learning techniques for text classification. As seen in Table 2.1, classification techniques include machine learning algorithms such as SVM and Naïve Bayes are among those that perform well, while common clustering techniques are K-means, LDA and ASUM. In addition, there are some interesting papers that apply graph-based methods such as Self-Organizing Map and SNA to rank the application features and some propose the models to compare those features. These techniques offer the potential for future research in the research project.

The research in this thesis involves evaluating the efficacy of different techniques and therefore some evaluation metrics are necessary. Table 2.1 shows that, precision, recall and F-measures are among the most frequently used evaluation measures for clustering and classification techniques.

To deal with duplication, string (Jaccard metrics, affine gaps, edit distance, q-grams) and semantic (LSA, PMI) similarity measures are required. These techniques offer different aspects of likelihood. Sentence similarity measures which embark on semantic similarity among the processing text are very suitable for the experiments of this thesis.

With regard to research issues, it can be observed that text analysis related to reviews on the app stores, mainly Apple’s and Google’s, has accounted for the centre of most research, whereas reviews outside app store reviews has had little attention from researchers. This is not unexpected with such an increasing popularity of mobile app development as well as their utility. Thus, in this thesis, we aim to contribute to the analysis of application reviews more generally, from a variety of online and offline sources. This includes cloud-based products, mobile apps, and standalone software products. To this end, a number of challenges can be noted. First, a new dataset for

the experiments and analysis is required and there might be no ideal baseline to compare or evaluate our proposed methods and techniques. In addition, existing approaches are mainly designed for mobile app review analysis, of which texts tend to be short and straightforward; unlike other types of app reviews which are more likely to be long and complex in content. To sum up, we can outline the research approach of this thesis and of the future research project based on the reviewing of relevant literatures by pointing out some issues and gaps, as shown in Table 2.2. To cope with these challenges, we offer an alternative approach by adapting and synthesizing the existing techniques and methods to fulfil the research purpose under the scope of this thesis, to be discussed in detail in the subsequent chapter.

Table 2.1 Summary of Relevant Literature

N	Title of the paper	Reference (Author/ Publish)	Objective/ purpose	Data	Pre-processed /cleaning techniques	Feature extraction, selection	Classification/ Clustering algorithms	Evaluation measures/Results
Text analysis for mobile app development support								
Feature Classification								
1	How do users like this feature? a fine grained sentiment analysis of app reviews	Guzman, E., & Maalej, W. (2014)	Identify topic for feature request grouping	Online mobile application reviews (Google Play and Apple App Stores)	POS-tagging, stop word removal, word lemmatization	Bigram collocation from NLTK collocation and WordNet dictionary, sentiment analysis	LDA	Precision, Recall, F-measures (59%, 51%, 55% average, 91%, 73%, 81% best for a specific application)
2	Bug report, feature request, or simply praise? on automatically classifying app reviews.	Maalej, W., & Nabil, H. (2015)	Classification of user reviews into bug reports, feature requests, user experience and user rating	Online mobile application reviews (Google Play and Apple Apps Stores)	Stop word removal, lemmatization		Naive Bayes, Decision Tree and Maximum Entropy (Using different data component combination)	Precision, Recall, F-measures 76%, 74%, 75% average Naive Bayes with combined data (BOW, rating, sentiments)
3	On the automatic classification of app reviews	Maalej, W., Kurtanović, Z., Nabil, H., & Stanik, C. (2016)	Same as above	Same as above	Same as above	Bigrams	Same as above	Precision, Recall, F-measures 86%, 74%, 78% average Naive Bayes with combined data
4	How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution	Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C. A., Canfora, G., & Gall, H. C. (2015)	An approach to classify mobile apps features for future maintenance and evolution	Mobile apps reviews (Google Play and Apple Apps Stores)	Stop word removal, stemming and term frequency indexing (TF)	Sentiment analysis, text analysis and natural language processing techniques	Bayes, SVM, Logistic Regression, J48 and ADTree	J48- 84.5% precision,84.7% recall and 83.8% F-measures

Feature Clustering								
5	Ar-miner: Mining informative reviews for developers from mobile app marketplace	Chen, N., Lin, J., Hoi, S. C. H., Xiao, X., & Zhang, B. (2014)	A complete model to analyse mobile application reviews	Online mobile application reviews (Google Play Store)	Word tokenization, non-alphabetic removal, white space removal, stop word removal, word stemming	-	Expectation Maximization for Naïve Bayes LDA, ASUM topic modelling	Precision, Recall, F-measures (95%, 77%, 75% average) LDA (best)
6	Analysis of user comments: an approach for software requirements evolution	Galvis Carreño, L. V., & Winbladh, K. (2013)	Extract features for evolution in next version	Online mobile application reviews (Google Play Store)	Word tokenization, punctuation, symbol, space removal, lowercasing	-	Topic modelling K-medians, ASUM (LDA + sentiment analysis)	Precision, Recall, F-measures (95%, 77%, 84% average, k =150) ASUM (best)
Feature Ranking/Prioritisation								
7	Approaches for Prioritizing Feature Improvements Extracted from App Reviews	Keertipati, S., Savarimuthu, B. T. R., & Licorish, S. A. (2016)	Offer three approaches to rank/prioritize feature requests for mobile application development	Online mobile application reviews (Google Play Stores)	Word tokenization, POS tagging	N-grams (unigrams-nouns)	-	Pros and cons of Individual attribute-based, weighted and regression-based approaches
8	Augmenting Text Mining Approaches with Social Network Analysis to Understand the Complex Relationships among Users' Requests: A Case Study of the Android Operating System	Lee, C. W., Licorish, S. A., Savarimuthu, B. T. R., & MacDonell, S. G. (2016)	Approach to sort feature requests (by topic) by different prioritization measures	Online Android OS reviews by release version (date)	POS tagging	N-grams, PMI	SNA to interconnect request features	Discuss different interconnection measures to accommodate and prioritize request features
Feature Comparison								
9	They'll Know It When They See It: Analyzing Post-Release Feedback	Licorish, S. A., Lee, C. W., Savarimuthu, B.	Approach to analyse feature requests of a	Online Android OS reviews by	POS tagging	N-grams, PMI	-	Trend analysis of review information

	from the Android Community	T. R., Patel, P., & MacDonell, S. G. (2015)	system over its lifecycle	release version (date)				
10	On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios	Licorish, S. A., Tahir, A., Bosu, M. F., & MacDonell, S. G. (2015)	Compare feature requests and responses for system development	Online Android OS reviews by release version (date) Developers' release notes	POS tagging	N-grams, PMI	-	Insights to requests and responses for system development
11	User reviews matter! Tracking crowdsourced reviews to support evolution of successful apps	Palomba, F., Linares-Vasquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., & De Lucia, A. (2015)	A model, CRISTAL to compare mobile application requests and responses for development process improvement	Online mobile application reviews and release notes (Google Play store)	Using AR-miner	Using AR-miner	Using AR-miner	Precision, Recall, F-measures (77%, 73%, 75%)
Text analysis for non-mobile software development support								
12	Extracting features from online software reviews to aid requirements reuse	Bakar, N. H., Kasirun, Z. M., Salleh, N., & Jalab, H. A. (2016)	Extract software requirement specifications for developing similar software systems	Online 'expert' reviews for educational software	Stop word removal, lemmatization, POS-tagging, duplicate term removal	TF-IDF, LSA	K-means, Fuzzy C-means, Self-Organizing Map	Precision, Recall, F-measures (62%, 82%, 70% average) Fuzzy C-means (best)
13	Analysis of economic impact of online reviews: an approach for market-driven requirements evolution.	Jiang, W., Ruan, H., & Zhang, L. (2014)	To extract economically viable requirements that sometimes overlooked in software development process	Online reviews of antivirus software	POS-tagging	Rule-base extraction for correlated noun-adjective pairs (SRPA and P-SRPA (SRPA + noise pruning techniques), sentiment analysis	SVM, topic modelling	Precision, Recall, F-measures 63%, 84%, 71% average (P-SRPA > SRPA, manual)

14	For User-Driven Software Evolution: Requirements Elicitation Derived from Mining Online Reviews	Jiang, W., Ruan, H., Zhang, L., Lew, P., & Jiang, J. (2014)	To extract software requirements for software evolution	Online reviews of antivirus software and mobile application (Apple App Store)	Same as above	Same as above	SRPA+ with S-GN algorithm, K-means	Precision, Recall, F-measures 69%, 75%, 71% (S-GN > K-means)
15	Requirement Acquisition from Social Q&A Sites	Xiao, M., Yin, G., Wang, T., Yang, C., & Chen, M. (2015)	Extract software requirement requested to support software development	Online Q&A text	Lowercasing, word tokenization, punctuation removal, stop word removal, word stemming	TF-IDF	SVM, rule-based algorithms	Precision, Recall, F-measures (69%, 75%, 72%) SVM (best)
Similarity Measures for Detecting/Reducing Data Duplication								
16	Adaptive duplicate detection using learnable string similarity measures.	Bilenko, M., & Mooney, R. J. (2003)	A learning approach to identify string similarity	Restaurant databases and 5sets of collections of academic citations	-	-	Similarity algorithms: q-grams, edit distance, affine gaps Learning algorithms: SVM, Decision tree	F-measures Learned edit distance = 87%; Learned q-grams = 89%, average on multiple data fields
17	Sentence similarity based on semantic nets and corpus statistics	Li, Y., McLean, D., Bandar, Z. A., O'Shea, J. D., & Crockett, K. (2006)	An approach to detect semantic similarity in short text or sentence	Selected sentence pairs	-	-	Sentence similarity measures based on WordNet and Brown corpus	Compare performance by algorithms to human's Algorithm: 81.6% Human best: 82.5%
18	A threshold-based similarity measure for duplicate detection	Ektefa, M., Jabar, M. A., Sidi, F., Memar, S., Ibrahim, H., & Ramli, A. (2011)	A combined approach for string and semantic similarity measurement	Restaurant dataset	-	-	Similarity algorithms: LSA+ Jaccard similarity Learning algorithms: SVM (for validation)	Precision, recall, F-measures, accuracy (Combined learning algorithms, all measures >98%)

Table 2.2 Summary of Issues/Gaps from the Reviewing of Relevant Literatures and our Current and Future Research Approaches

	Current Issues /Gaps	**Propose in this thesis**	**Propose for future research project**
Feature Classification	• Only apply one specific n-grams to boost the performance of models • No combination of nice-performed classifiers like SVMs, Naïve Bays and tuning parameters like text sentiments, n-grams	• Working at sentence level to reduce overlooking important information and to be flexible to deal with both long and short text • Take into account of reviewing dates • Adapt binary and multiclass classifications for the extraction of requirement information • Use different classifiers: SVM, Naïve Bayes • Adapt tuning parameters to improve the performance of classifications: o n-grams: unigram, bigrams, trigrams and 4-grams o division of data percentage for training and testing of classifiers o use sentiment analysis/scores	
Feature Clustering	Offer automated approaches using ASUM, LDA, but need to consider efficiency and accuracy		Experiment to see the feasibility
Feature Ranking		Can only test on term-weighted ranking technique within this research timeframe	Test on graph-based techniques like Social Network Analysis model (SNA) to help represent the extracted information
Feature Comparison			Require more data sources to indicate the solved and pending feature requirements
Text analysis for mobile and non-mobile software development support	• Most studies focus on mobile content to improve mobile app development and improvement • Fewer studies use non-mobile content	Use general product reviews of users, not specific to mobile/specific types of software applications	Besides using user reviews, should be extend to other sources: • expert reviews • peer reviews • competitors' reviews • internal reviews • product release notes
Similarity measures		Apply the semantic similarity measures by Y. Li et al., 2006 which is more appropriate and ready-to-use	Adapt the more effective ones by Ektefa et al., 2011 which considers both string and semantic similarity measures

Chapter 3

Research Design and Implementation

In this chapter, we present how this research is designed and describe details for the implementation. The proposed feature extraction process, including the data gathering and pre-processing, is described and explained in detail in this chapter. The planned experimental evaluations of a number of text analysis and clustering methods and their variations for key steps in the process are also presented in this chapter.

The organization of the chapter is as follows. In Section 3.1, we propose an overview for this research design, including frameworks for the design and implementation. In Section 3.2, we present our data collection. Data pre-processing is introduced in Section 3.3, which includes sentence tokenization, text pre-processing techniques, sentiment analysis and sentence similarity measures for duplication detection and reduction. Attribute extraction and selection are discussed in Section 3.4. Section 3.5 describes binary classification and multiclass classification. In Section 3.6, we examine model evaluation methods. The ranking of extracted feature requirements per category is discussed in Section 3.7. In Section 3.8, we provide a structured summary of the experimental evaluations of alternative text processing methods that could be used at different stages of the proposed feature extraction process.

3.1. Research Design

This research aims to support the development of software applications by providing information regarding software requirements to support decisions about release planning, with the utilisation of online product reviews. Our purpose is to develop a tool to automate the extraction of information by means of requirement identification (classification) and prioritization (ranking). In an effort to support the software development process by addressing a solution to our research questions, a framework for this research is constructed as shown in Figure 3.1. The data collection of software reviews posted by online users is practised on the assumption that those reviews could provide a reliable source of information and hereby be worthy to explore. The selection of a data source is made on the condition that it avails reviews of software applications related to software development with a sufficient number of reviews for the experiments. In addition, some manual screening to a number of

reviews is made to ensure the existence of useful information regarding product requirements, which are, in most cases, embroidered with unwanted data or noise. The website www.g2crowd.com and two applications, Jira and Trello, are selected as a sample case study, which are used to represent other user review data. Different text pre-processing and processing techniques are applied such as tokenization, lowercasing, stop word and punctuation removals, n-grams, sentiment analysis and data cross-validation.

In this research, a machine learning approach has been chosen for data classification over the traditional rule-based approach. Instead of developing linguistic rules which can be complicated and restricted to a certain domain, in this thesis, we choose to implement a machine learning approach, which is more generic. In machine-learning approach, solely a training dataset is required to expose to the machine, and based on the learnt knowledge, the prediction and classification of unknown data can be made.

Several models are constructed by adapting different text analysis techniques and evaluated over multiple iterations to compare results of experiments and identify the best ones. The construction of analysis models in this thesis is made from tuning parameters including n-grams, sentiment analysis, machine learning algorithms and cross-validation techniques, which are used to tune the performances of classification analysis. For the parameter of n-grams, 1-ngrams, 2-ngrams, 3-ngrams and 4-ngrams are interchangeably used along with the input review sentences. For sentence sentiment as tuning parameter, in this thesis, we rely on the input of review sentences alone and a combination of review sentences and their sentiments. For machine learning algorithms, Naïve Bayes, Support Vector Machines and Logistic Regression are experimented. Finally, for cross-validation, different proportions of training and testing datasets which include the use of 2-fold (with 20%, 33%, 50%, 67% of data), 5-fold, and 10-fold datasets for training the classification algorithms are experimented.

For the evaluation of analysis models, precision, recall and F-measures are used to test and assess the model performances and results. After the classification of product requirements, those feature requirements are prioritised by term-weighted frequency. However, this process of feature ranking is not evaluated in this thesis; it is proposed as a foundation for the future research project.

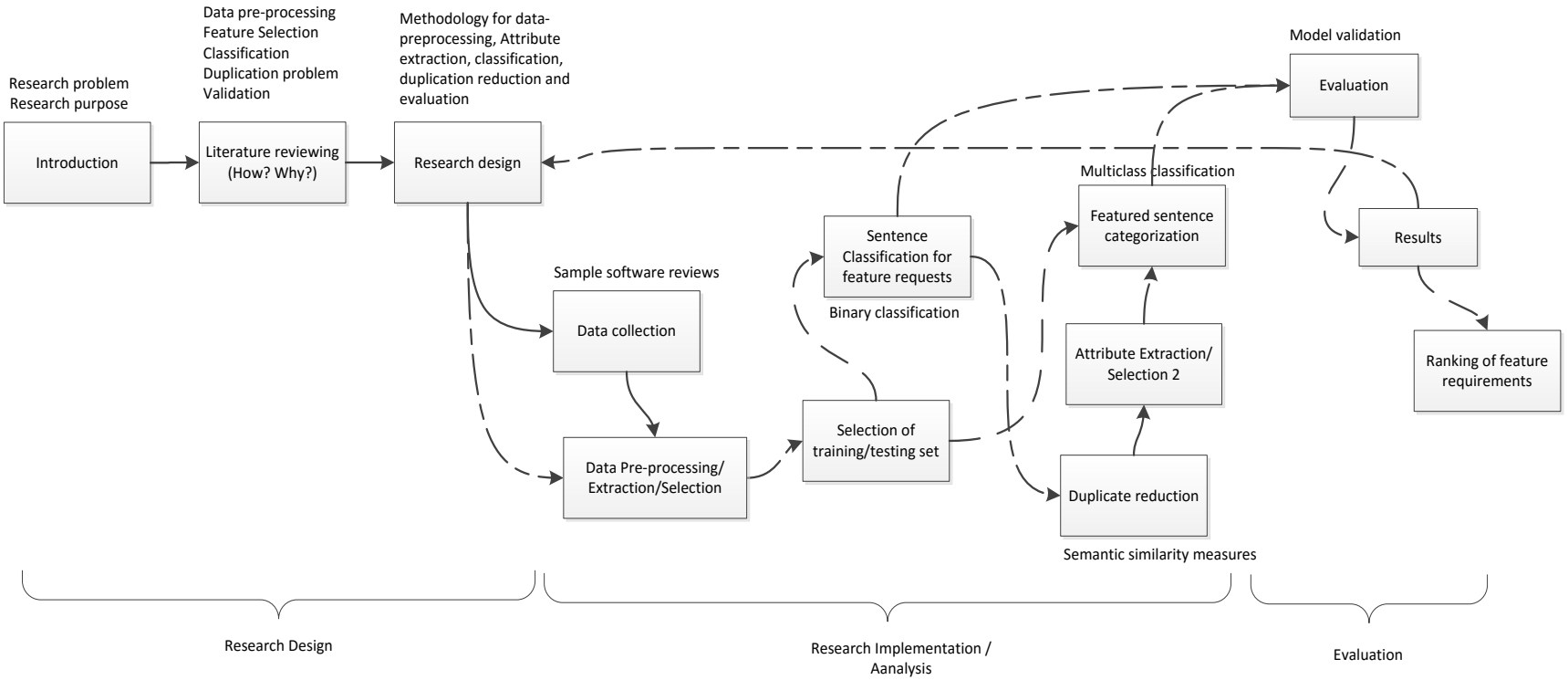


Figure 3.1 Current Research Framework

The details of the proposed feature extraction process are shown in Figure 3.2 below. The evaluation and validation activities of some alternative text analysis and classification techniques are also shown and highlighted.

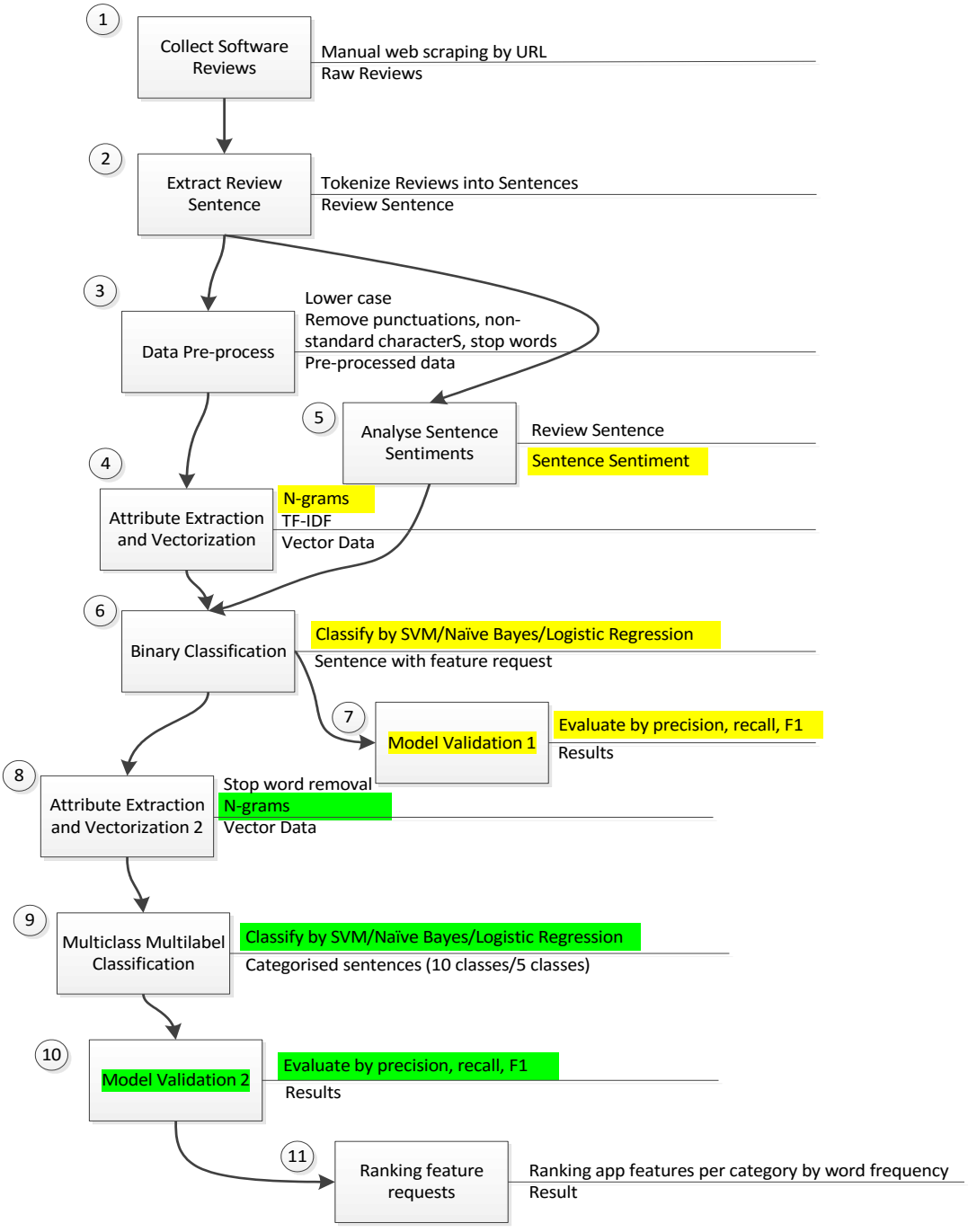


Figure 3.2 Framework for Research Analysis

To perform actual text analysis and processing, the system infrastructure used for the research implementation is shown in Table 3.1.

Table 3.1 System Infrastructure for Research Implementation

Programming Language	Python
Version	2.7
Developing Platform	Microsoft Windows 10
Modules for Text Analysis	nltk == 3.2.1; pandas == 0.18.1; scikit-learn == 0.17.1; numpy == 1.11.1; vaderSentiment == 0.5; textblob == 0.11.1; rake-nltk == 1.0.0
Data	Jira and Trello software reviews
Classifier	SVM, Naïve Bayes, Logistic Regression

3.2. Data Collection

The main sources of online reviews used in this thesis are those of two software products, namely Jira and Trello. Jira is a commercial software for software development project management, especially designed for agile software engineering teams, which helps users to plan and keep track of their development projects under real-time settings and provide reports on work progress as well as issues arising. Trello is a software of similar type offering both free and paid options to users. Reviews of a paid app are selected in this research to ensure informative feature requests for evolution (Galvis Carreño & Winbladh, 2013), and those for free apps are also used to help balance our data and reduce a potential bias in the proposed approach.

The data for this study's experiments are online reviews of Jira and Trello, retrieved from the website https://www.g2crowd.com. This website offers free software reviews for software systems of different ranges to help users select a suitable software for their business and application. These online comments are not only useful for software consumers, but also provide information to support software development and evolution (Galvis Carreño & Winbladh, 2013; Guzman & Maalej, 2014; Harman, Jia, & Zhang, 2012; Pagano & Maalej, 2013). Even though this research used reviews from one specific website, reviews from other data sources might also be applied under this proposed approach.

The collected data include the retrieved link (URL), app name, posting date/time, reviewer, rating, review title and review text. All reviews are in English and in the form of medium-to-long paragraphs, which are organized in a template of questions

and answers on dynamic web sources. The review text contains multiple sentences which describe software applications and features at the experience of users. All these review comments, including questions and answers, are retrieved to ensure that any important information regarding app feature requests are not ignored or missed.

To scrape the collection of our targeted software reviews, a manual web scraping tool is developed using the Python modules, ‘Requests’, ‘Cfscrape’, ‘Beautiful Soup 4’ and ‘Selenium’. This is based on Python 2.7.12, which is used throughout the experiments in this thesis. Our data scraper is designed to parse all targeted pages based on relevant web tags and attributes, such as with the attributes of ‘review body’, ‘heading’, ‘review by’, and ‘time’. The collected raw reviews are stored in a SQLite database.

Besides the main data collected from the website mentioned above, we also use some external data as a simulation dataset. The purpose of this is to test the performance of our classification models when different datasets are inputted. This dataset is manually selected from online reviews of similar software applications such as Asana, Wrike, Slack, and Jira, but from different websites, namely getapp.com and trustradius.com.

3.3. Data Pre-processing

Before processing and analysis, raw reviews are needed to be normalized at the pre-processing stage. First, is to prepare the data into a ready-to-use format and dimension for further analysis. Second, is to reduce some unneeded noise which can negatively affect the process of analysis and evaluation. In this research, the pre-processing techniques that are used include sentence tokenization, lowercasing, word tokenization, non-standard word and punctuation removal, stop word removal, short sentence removal, word stemming and lemmatization, sentiment analysis and sentence similarity measures. This section, we explore an overview of each technique.

3.3.1. Sentence Tokenization

The first pre-processing step is to tokenize each review comment into sentences. Sentence tokenization is a task to segment text which, in general, is multiple

sentences or paragraphs, into a token of sentences. In a short review, sentence tokenization might not be necessary because reviews are not very long and less complicated. However, to mine long review text, as in this research, it is better to work at sentence level. Breaking down a large document into tiny segments can facilitate the extraction of feature requirements and categorization, because the large text might contain so many software features which bring in confusion and complication in terms of processing. Tokenizing documents into sentences helps to minimize the chance of missing or overlooking any important information or ideas provided by users. Also, it is more effective dealing with noise and duplications when working with smaller segments of text. However, there is also a disadvantage of working at sentence granularity. Some sentences might be meaningless if they are separately investigated, since they should be coherent to their neighbouring sentences to convey the intended ideas. In this research, this problem is tackled by using a de-duplication technique, as explained in Section 3.3.9.

In this current study, a python module, Punkt sentence tokenizer in Natural Language Toolkit (NLTK) library has been used as a tool for sentence segmentation. Punkt tokenizer module comes with a pre-training dataset (which might also be inputted manually), which makes it ready-to-use. Also, the module allows the user to manually add a list of abbreviated words, like i.e. and e.g., that should not be marked as the end of a sentence. The tokenized sentences are stored separately in a SQLite attribute from the original raw reviews in the database to be easy for the next processing stages and, by the way, original comments can still be referred to in case of necessity.

3.3.2. Word Lowercasing

In text processing, same words of different letter case are considered as different. For example, the words ‘Networking’ and ‘networking’ are interpreted as two different words by the machine, while they should not be. Such a difference in word letter case can hamper the classification process. As a result, all words are changed to the same lower case before the process of analysis in this research, using the Python text attribute function, text.lower().

3.3.3. Word Tokenization

The next pre-processing step is to tokenize the sentences into words. Like sentence tokenization, word tokenization is the separation of text or a document into a list of words. Word tokenization is necessary before word stemming or lemmatization techniques can be applied. Segmenting sentences into words helps in the process of classifications and ranking of feature requirements which rely on certain featured keywords. Word tokenization module in the NLTK library has been used to tokenize sentences into words in this study.

3.3.4. Non-standard Character and Punctuation Removal

Non-standard characters include special characters and non-English characters. In this research, all non-standard words are eliminated to increase the performance of classifiers, since in most cases, these characters only bring about noise. In some context, removing punctuations might change the literal meaning of the text. However, this is not the case in this research because these characters are unnecessary in terms of text processing after the sentence tokenization, and the original text can always be referred to when necessary. The punctuations to be removed are “”!"#$%&\’()*+,-./:;<=>?@[\\]ˆ_‘{|}˜. The elimination of punctuations is conducted after all sentences are completely tokenized to avoid the confusion of sentence separators (.?!). In this study, we use RegexpTokenizer, a regular expression tokenizer module in the NLTK library for removing punctuations and non-standard characters.

3.3.5. Stop Word Removal

Stop words are very common words in English that do not contain important or meaningful features of a document, but only represent noise or irrelevant aspects for the analysis. When they are removed, significant candidate features are still presented. Removing stop words not only helps reduce noise in a document vector but also decreases the dimension of vector space which, in turn, improves the performance of classification. Some examples of stop words are ‘the’, ‘that’, ‘which’. In text processing, stop words are usually in the form of a word list where words can be flexibly added or removed as necessary. In this research, scikit-learn Python modules (Sklearn) for machine learning is used for removing stop words. In

the Sklearn, stop words are eliminated in the module pipeline, before the classification process. The stop word list used in this context is modified from http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words. The list is Sklearn built-in, but it is modifiable.

In this current study, three different sets of stop words are applied in the pre-processing stage for classifications. This is because some specific sets of words are required in different stages of classifications and analysis. Firstly, in binary classification, which is the filtering of 'non-feature requirements' out of 'feature requirements', our binary classifier needs to learn some keywords such as 'should', 'add', 'improve'. These words, though they are very common words, are important keywords for the binary classification. They are, therefore, retained in the document for processing, instead of being eliminated. Secondly, in multiclass classification, general standard stop words can be used, as this stage only focuses on classifying sentences with software features into their labels or categories. In addition, for the ranking of feature requirements, another stop word list which comes with the module for feature ranking is used (see Section 3.7 for the detail on feature ranking). In all cases, abstract nouns like the names of the apps (Jira, Trello) are included in the stop word list, as they only presented noise and are not necessary in the process of classification. All stop word lists used can be found in Appendix C.

3.3.6. Short Sentence Removal

Short sentences with less than three words, for example, 'Poor software.', 'Fantastic!", are eliminated in the process of data retrieving, since they are non-informative and presented noise which impede the performance of classifications.

3.3.7. Word Stemming and Lemmatization

Grammatically, words in English are in different forms and parts of speech, depending on the content they sit in, such as ease, easy, easily, easier. Also, there are words or terms in the same family which present similar meanings such as nation, nationalize, nationality. To represent these words of the same families and different forms in text processing requires a different index, which unnecessarily increases the dimension of documents in a vector space and affect the performance of classifiers. Stemming and lemmatization address this problem by producing the origin of these

words to retain them to their same feature set. Both techniques help decrease the dimension in the term-document matrix and boost the efficiency of classifiers.

Lemmatization is to group words of different forms and families, so that they can be analysed as a single word base or dictionary form, known as a lemma. For example, a group of words “see, saw, seeing, seen” would be lemmatized into “see” as its lemma.

Stemming is the process of chopping off prefixes and/or suffixes (affixes) from words in similar forms and families to a single word stem or word root. For instance, a group of words “ease, easy, easily, easier” might be stemmed to a root, “eas”.

In some cases, stemming words can be confusing, as the words do not appear in interpretable forms. Taking the example of lemmatizing the verb “see” above (see, saw, seeing, seen), the stem of this word group can just become “s” which does not make any sense, or they might not be able to stem at all. This is because the stemming technique only deals with a word, while discarding its context. It hence cannot distinguish the words with different meanings depending on their parts of speech or a dictionary. For this reason, using word stemming can decrease the accuracy of document representation in some applications. However, for the same reason, word stemming can be cut off at a much faster rate when compared to lemmatization, which makes stemming among one of the popular pre-processing techniques.

In the current study, the NLTK modules PorterStemmer and WordNetLemmatizer are used for stemming and lemmatization respectively. The stemming technique is used for binary classification, since the dimension of documents before classification is quite large, while the meaning of the text is not mainly considered. It is more appropriate and faster to stem the documents. For multiclass multi-label classification, lemmatization is applied because it is more suitable for the duplication reduction and classification stages. Both stages are based on similarity of features (noun phrases) where the meaning of the documents is important.

Besides those techniques mentioned above, misspelt words are ignored in this research, since it is unpredictable what the reviewers want to convey. Though it is a possibility to use a text correction tool, applying auto-correction to a document might completely change the meaning of the document. Document classification is

the process of probability. To predict the misspelt words would add another prediction to the process.

3.3.8. Sentence Sentiment Analysis

Sentiment analysis is a text analysis technique to identify sentiments or emotions of text, which reflect the attitude of a writer regarding a certain topic. In sentiment analysis, the polarity of a document is determined: whether it has 'positive', 'negative' or 'neutral' sentiment. According to two of the studies in our literature review, using sentiment data in combination with review text can increase the efficiency of learning classifiers in document classification (Maalej & Nabil, 2015; Panichella et al., 2015). This current study is thus to experiment with applying review sentences in combination with their sentiments as the input for the binary classification of feature requirements.

To achieve this, a Python module called VADER-Sentiment-Analysis or vaderSentiment (Gilbert, 2014) is applied as a tool for sentiment analysis of review sentences. VADER-Sentiment-Analysis is a rule-based module for analysing sentiments designed for social media comment analysis. It provides sentiment scores for documents or sentences within the range of $[-1; 1]$, from completely negative (-1) to completely positive polarity (1). In this research, we identify sentence sentiments based on the calculated polarity scores: $score < 0$ for negative, $score = 0$ for neutral, and $score > 0$ for positive sentiment sentences.

3.3.9. Duplication Reduction by Semantic Similarity

In this thesis, we consider a problem of duplicate features because it is not necessary and can be confusing to display redundant feature requirements at the same time. In our database, the problem of identical data fields can be effectively delimited, both by manual coding when data are updated into the tables and by using primary keys, to get rid of data redundancy. However, the solutions are unable to address the problem of duplication when data are semantically similar. For example, a product reviewer potentially mentions one feature repeatedly in a single review which includes multiple sentences by using different words with similar meaning. To cope with this, similarity measures for semantic words and/or sentences by Y. Li et al. (2006) are applied to reduce duplicate feature requirements. This handy technique

helps locate the sentences that contain similar meaning. Using this technique also helps reduce the weakness of working at sentence granularity as mentioned in Section 3.3.1, because sentences with coherent meaning are unified as one unit and stored in a separate table for further processing in this research. The Python codes are adapted from https://github.com/sujitpal/nltk-examples/blob/master/src/semantic/short_sentence_similarity.py, which offer the full scripts along with descriptions regarding the implementation of semantic similarity measures.

However, the application of similarity measures is a very time-consuming process. To detect the semantic likelihood of sentences, it is required to compare every pair of sentences specifically by selected keywords based on relevant word corpora and ontologies. Therefore, this current study is subject to apply the similarity metrics for reducing duplicate data on feature requirements by every single review, which is generally composed of a set of few sentences. This process is performed after the binary classification to the sentences with feature requirements where the data size is reduced to keep the process as minimal as possible.

3.4. Key Attribute Extraction and Selection

3.4.1. Data Vectorization with TF-IDF

After the sentences are prepared at the pre-processing stage, it is necessary to transform those sentences or documents into numbers or indices, which the machine can understand and then classification algorithms can be applied. This process of conversion is known as document representation. Selecting a feature to represent a document is typically performed by using words or terms in the document, which is known as Bag-of-Word (BoW) approach. The most common BoW representation is Vector Space Models (VSMs) where documents are specified by vectors (of words) in a vector space. In a VSM, a set of documents (n) of which a document (d) consists a set of terms (w) can be represented by a vector of n-dimensions:

$$d_j = (w_{1,j}, w, \dots, w_{i,j})$$

where $j \in \{1, \dots, n\}$ and $w_{i,j}$ is a value (or weight) given to the term i in document j. When document vectors are joined, they form a term-document matrix with an element $w_{i,j}$. Vector space model relies on the weight of terms appearing in a document to represent the content of the document. Documents that have represented vectors close to each other in a vector space are considered to have similar content.

Term frequency and inverse document frequency (TF-IDF) is a commonly used weighting scheme in text analysis. The term frequency scheme weights the importance of a document by how often the words occur within a document (local weights). Where $n_{i,j}$ is the number of times the term i occurs in the document j, term frequency can be calculated by

$$tf_{i,j} = \frac{n_{i,j}}{\sum_i n_{i,j}}$$

Inverse document frequency considers the importance of terms in a set of documents (global weights). Terms occurring frequently in many documents are generally less important to indicate the topic of document sets.

$$idf_i = \log\left(\frac{n}{n_{word(i)}}\right)$$

where n is the total number of documents in the document set (corpus) and $n_{word(i)}$ is the number of documents in which the term i occurs.

TF-IDF combines term frequency (local document weights) with inverse document frequency (global document weights) as the following example formula,

$$w_{i,j} = tf_{i,j} \cdot idf_i$$

The TF-IDF scheme can be made of many different variants besides this present example.

In this thesis, instead of using simple BoW representation which only considers term frequency within the same document, we use TF-IDF which are more inclusive with term weight frequency in both documents and whole document sets to format our review sentence dataset in the data vectorization. This helps balance the important of keywords throughout the experiment document dataset. We rely on Sklearn modules for vectorization, where plain sentences can be directly inputted. TfidfTransformer

module in Sklearn is used to generate TF-IDF for formatting the experimental sentences for classification. Also, when having data represented in the form of vectors, the DataFrame module in the Pandas Python library for data analysis is used to temporarily store pre-processed data and extracted key attributes for analysis. This module allows a reference to original review sentences by their relative indices.

3.4.2. N-grams

Document tokenization generates key attributes of words with their frequency of occurrence. In many cases, a group of words or tokens might be more precise at representing a specific document attribute. For example, “New Zealand” is better to go together as the same token than separating into two words of “New” and “Zealand”. This representation is known as an n-gram, where n is the number of tokens that come together in one specific attribute. N-gram techniques allow grouping of words which potentially represent the same key attribute in a document for analysis. In n-grams, when the value of n is higher, the frequency of the document attributes is lower. The most typical n-grams are unigrams, bigrams and trigrams (n = 1, 2 and 3 respectively).

Using n-grams can help increase the performance of classifications because our feature requirements might be represented by a group of collocations or n-grams (Guzman & Maalej, 2014; Maalej et al., 2016). As a result, in this research, we test n-gram techniques as one of our tuning parameters with different ranges, unigrams (n-gram (1,1)), bigrams (n-gram (1,2)), trigrams (n-gram (1,3)) and 4-grams (n-gram (1,4)) by using n-gram modules in Sklearn. Modules for data analysis in the Sklearn library allow the generation of n-grams at the data vectorization process, where the number of tokens can be flexibly selected by a range: n-grams (m, n), where m is the lower bound of token number and n represents the upper bound of token number. For example, using n-grams (1, 3) enables the generation of phrase of one to three words or tokens.

3.5. Document Classification

3.5.1. Machine Learning Approach

Machine learning is a field of artificial intelligence, of which a computer's learning process is driven by exposing a machine to new knowledge or experience. Basically, there are two main types of machine learning algorithms: supervised and unsupervised. In a supervised learning approach, classifiers learn knowledge from known values, and based on this, the prediction of unseen items can be accomplished. In other words, machines are presented with example data which are fully labelled. From this, a function for prediction (known as a classifier) is identified, and by using this function, labels can be assigned to unknown instances. Figure 3.3 shows the diagram of supervised learning methods. This type of learning is generally known as 'classification'. Supervised learning algorithms widely used are Naïve Bayes and Support Vector Machines (SVM).

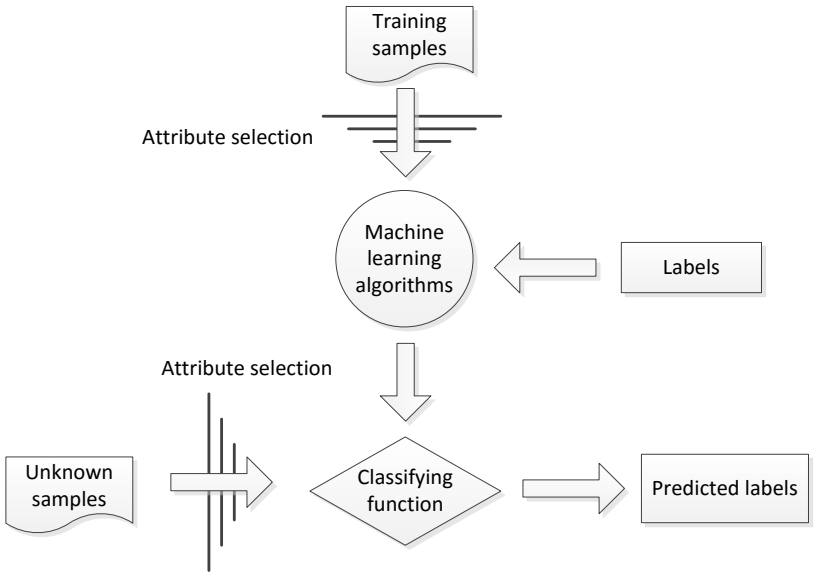


Figure 3.3 Supervised Learning Process

On the other hand, unlike supervised learning methods, unsupervised learning does not rely on any training examples, but data are classified based on natural patterns and structures of the data themselves such as similarity of words or word semantics. Without any known labelling assignment, the quality of this type of learning is difficult to identify. Some common uses of unsupervised classification include text clustering, feature extraction and estimation of density. Examples of unsupervised

learning algorithms are k-means and density-based algorithms. The process of unsupervised learning is shown in Figure 3.4.

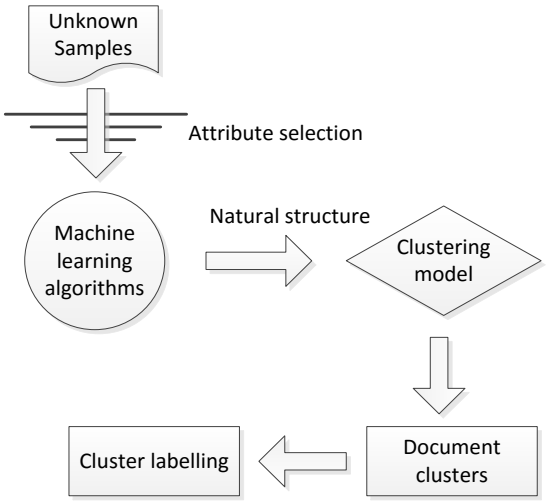


Figure 3.4 Unsupervised Learning Process

In this thesis, we adapt supervised learning approach to be semi-supervised, considering the fact that supervised learning usually offers higher accuracy and more efficient than the unsupervised. In our semi-supervised learning approach, we aim to reduce the intervention of humans by reducing the human input as much as possible. We experiment this by dividing the size of dataset injected into our models' learning process, taking into account different data sizes and effectiveness in the performance of our classification models.

3.5.2. Type of Classification

Classification involves the assigning of labels to unknown samples, which can be distinguished into different types depending on the aspects of classification. In binary classification, two classes are presented in the classifying process of unlabelled instances. For instance, emails are classified as spam or not spam, or a disease is grouped as transmittable or non-transmittable. Multiclass classification is the assignment of an unknown item to one of multiple classes (more than two), which requires a more complex learning process. For example, news can be categorized by columns of world, politics, business, sports. In multi-label classification, each unknown object might be assigned to at least one (or more) labels or classes. For example, music can be sorted by genres such as blues, punk and jazz, and the same music might also be categorized by album, year or artist.

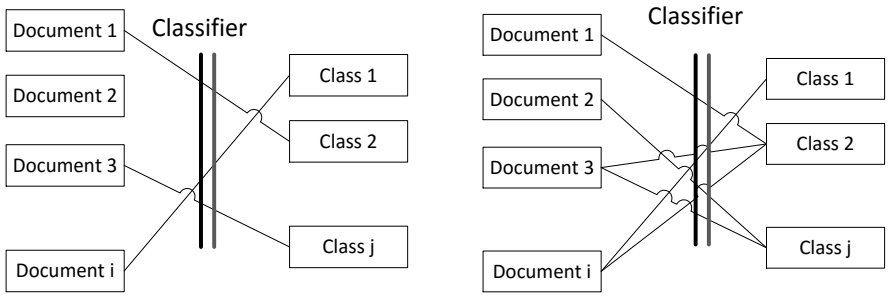


Figure 3.5 Multiclass Classification (left) and Multi-label Multiclass Classification (right)

Binary classification can perform faster than multiclass classification, as it deals with less iterations and involves a less complex process. Also, the results of prediction of the former are usually more promising.

In this thesis, we aim to identify and group sentences associated with feature requirements in online product reviews, which include two stages of classification. Binary classification is used to distinguish pre-processed reviewing sentences into two groups of non-feature and feature requirements (Classes 0 and 1). This is to filter only the sentences that include product feature requirements and ignore the unneeded non-feature-requirement sentences.

For clarification of the binary filtering process, feature requirements (for software product improvement) in this thesis are defined as “the addition of new functionality or the change of existing functionality of a software application to improve the performance of the application by increasing its efficiency, usability, usage and desirability to both existing and potential users”.

The second classification stage is multiclass multi-label classification. In the experiments, those sentences with feature requirements are further categorized to 10 categories (multiclass): board management; user management and permission; customization; infrastructural design; configuration and maintenance; data import, export and integration; mobile services; notification; navigation and search; and others, detailed as shown in Table 3.2. In addition, it is assumed that every sentence can possibly be composed of at least one or more features. In other words, every feature-requirement sentence can belong to more than one category, which entails multi-label classification. Therefore, a combination of multiclass and multi-label classification is used in this research. Categories for feature sentences used in our experiments are adapted from actual features of relevant applications (Jira and

Trello) available online at https://jira.atlassian.com/projects/JRA?selectedItem=com.atlassian.jira.jira-projects-plugin:components-page for Jira and https://trello.com/b/nC8QJJoZ/trello-development-roadmap for Trello. As both applications are of the similar software type, and the same categories are all eligible. For other types of products, classification categories might be different from the current study's, but the same models can be applied after they are trained with the data of new categories.

Table 3.2 Category for Multiclass Multi-label Classification

Class	Category for Classification	Description	Example sentences[1]
0	Board management	Sentence to describe operations on information boards such as workflow management, stories project backlogs, tickets	- Sorting sprints and backlogs by users assigned is an obvious miss. - Agile board doesn't have an ability to move multiple tickets to the next column at once.
1	User management and permission	Sentence used to describe how permission of usage is managed and approved	- The way that you have to set individual permissions is cumbersome. - I wish there were more and more specific levels of permissions in regards to editing and creating issues.
2	Customization	Sentence involved how flexible users can manipulate/ customize the app to meet their requirement and usage	- The issue collector should be fully customizable as well. - I wish you could create custom fields and make certain fields required.
3	Infrastructural design	Sentence related to specific designs which require enhancement	- Having each link open within the app is probably the most frustrating, god-awful thing about JIRA. - It would be nice to have a "Close" option available at every state instead of having to pre-define all the states to be able to transition to the "Closed" state.
4	Configuration and maintenance	Sentences related to how the app can be setup, configured, updated and maintained	- Setup can be complex. - Updates sometimes are difficult to install.
5	Data management, visualization and integration	Sentence to describe data sharing, import, export, reporting, backing up and integrating	- I'd like to see more integration features for marketing specific users. - Import/export to excel, word format required.
6	Mobile services	Sentence related to mobile features and usability	- Mobile site also not so great, lot more need to be improved. - When I'm trying to view a Jira ticket on mobile it's become a real headache.
7	Notification	Sentences about system notification and reminders	- Emails- way too many; I want emails I just don't want too many.

[1] Example sentences here are extracted from pure online product reviews; the content might contain mistakes in English grammar, misspelling, etc.

			- It would also be useful to have a settings that controls the types of notifications that get sent.
8	Navigation and search	Sentence related to page navigation, data filtering and search	- I have a hard time finding the status of issues if it isn't emailed to me. - Searching for an existing ticket can be quite difficult at times.
9	Others	Sentence of other issues than described above such as app speed, scalability, inconsistency, functions or features existing in other certain apps but not available in the current	- With its standard offering, once you reach million of record, it will start to run slow. - Difficult to scale up with a large team. - It is very focused on agile development, which is great for development teams but for our company we have to use a different project management software for the other teams (marketing, sales etc...).

3.5.3. Classification Algorithms

Based on the reviewing of relevant literature, two popular algorithms which offer promising performance ratings on app review classification are Naïve Bayes and Support Vector Machines (SVMs). Thus, these two classifiers are the candidates for experiments of classification in this research. Additionally, for comparison, Logistic Regression is selected because of its easy-to-implement aspect and effectiveness in performance. In this section, some backgrounds regarding these three classification algorithms are examined.

3.5.3.1. Naïve Bayes

Naive Bayes is a simple machine learning classifier which relies on conditional models of probability. It relies on the assumption that all document vectors for classification are naively independent.

In Naïve Bayes, a conditional probability model can be defined as follows:

Given n documents or instances for classification, represented by a vector $x = (x_1, x_2, \dots, x_n)$, where x_n is an independent variable and probabilities of class C_k given x: $p(C_k \mid x_1, x_2, \dots, x_n)$, based on Bayesian's theorem, the conditional probability can be calculated by:

$$p(C_k|x) = \frac{p(C_k)\, p(x|C_k)}{p(x)}$$

where k is the number of possible classes, $p(C_k|x)$ is a posterior probability of target class C_k given the classifier x instances, $p(C_k)$ is prior probability of class, $p(x|C_k)$

is a likelihood probability of classifier given class, and $p(x)$ is a prior probability of classifier.

Using the assumption of naïve independent instances that the probability of a specific instance i:

$$p(x_i|x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = p(x_i|C_k)$$

for every i, the conditional probability can be rewritten as:

$$p(C_k|x) = \frac{p(C_k)\prod_{i=1}^{n} p(x_i|C_k)}{p(x)}$$

Naïve Bayes classifiers can be made of different distribution variants. Two common variants of Naïve Bayes are Multinomial Naïve Bayes and Bernoulli Naïve Bayes.

Multinomial Naïve Bayes is defined by:

$$p(x|C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!}\prod_i {p_{ki}}^{x_i}$$

Bernoulli Naïve Bayes is defined by:

$$p(x|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}\,(1 - p_{ki})^{(1-x_i)}$$

Based on probability, the Naïve Bayes classifier can perform very efficiently and it is easy to implement. It also requires a small amount of training data, but provides a good classification result, especially for binary classification. However, the downside of this classifier is that it uses the assumption of document independency, which is not always the case in text categorization. It can perform very poorly if classification documents are more dependent (Khan, Baharudin, Lee, & Khan, 2010).

3.5.3.2. Support Vector Machines (SVMs)

Support Vector Machines (SVMs) is a supervised machine learning algorithm which is based on defining a hyperplane to optimize the boundary between two classes of classification instances. Vectors used to generate the hyperplane are called support vectors. With an input vector $\vec{x}$, a hyperplane is identified by a decision function $\vec{w} \cdot \vec{x} + b$, where $\vec{w}$ is a weighted vector (perpendicular vector) and b is bias (distance from the origin).

In SVM, an ideal hyperplane is the one with the maximum margin width, which is used to determine the side on the plane (or which class) $\vec{x}$ lies. This is determined by:

$$|\vec{w} \cdot \vec{x} + b| = 1$$

if $\vec{w} \cdot \vec{x} + b \geq 1, \vec{x}$ is in class 1, $\vec{w} \cdot \vec{x} + b \leq -1, \vec{x}$ is in class 2

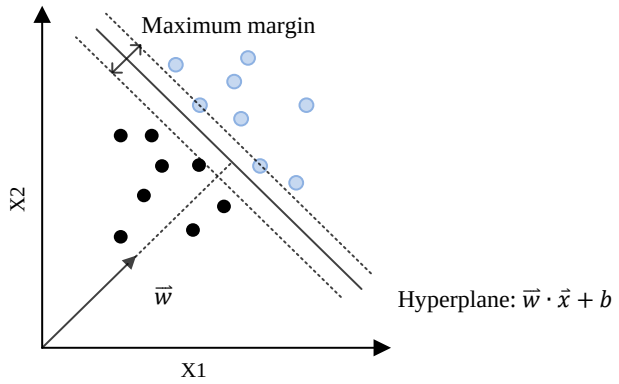


Figure 3.6 SVM Hyperplane

Basically, SVMs rely on a linear hyperplane to separate between labelled classes, which are known as linear SVMs. However, SVMs also support non-linear variants when a hyperplane cannot be fitted in classification (non-linear SVMs), by using kernel functions to plot input vectors into a different shape, which work better at determining the boundary between classes, for example:

Polynomial function with the polynomial degree d:

$$k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)^d$$

Gaussian radial basis function (RBF, with σ as the Gaussian kernel parameter):

$$k(\vec{x}_i, \vec{x}_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

Sigmoid kernel (Hyperbolic tangent):

$$k(\vec{x}_i, \vec{x}_j) = tanh(\alpha \vec{x}_i \cdot \vec{x}_j + c), \alpha > 0 \; and \; c > 0$$

The key advantage of SVM algorithms over other classifiers is kernel functions for classification which are not linear dependent. They offer a robust approach to generalize unknown samples by convex optimization. SVMs also offer high accuracy, even with noisy datasets and can handle data with large feature space. However, using kernel-based functions is, at the same time, a disadvantage. The

selection of parameters for SVM kernels functions can be difficult and this can produce poor results in classification. In addition, SVMs can be slow when dealing with a large amount of data (Auria & Moro, 2008; Karamizadeh, Abdullah, Halimi, Shayan, & Rajabi, 2014; Khan et al., 2010).

3.5.3.3. Logistic Regression (Maximum Entropy)

Logistic regression is a machine learning algorithm for binary-based classification and prediction, from which a regression model with categorical dependent variables is grounded. Logistic regression is basically a linear model, but a sigmoid function or logistic function is used in predictions, which is defined by:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $\sigma(z)\ \epsilon\ (0,1)$, as plotted in Figure 3.7.

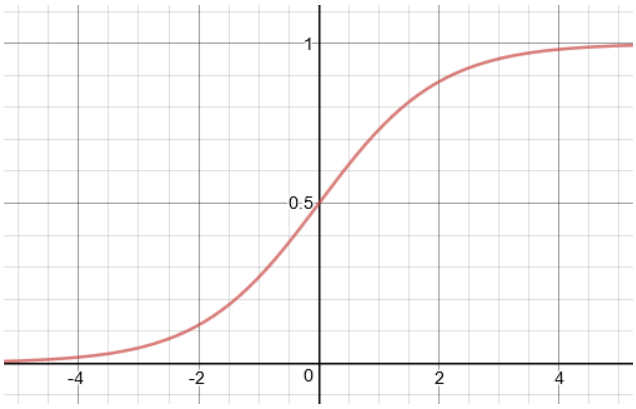


Figure 3.7 Logistic Function Curve

In logistic regression, z is a linear regression function of an independent (explanatory) variable X with the bias β_0 and the coefficient β_1. The function can be written as:

$$z = \beta_0 + \beta_1 X$$

The probability of predictions can, therefore, be obtained from the function:

$$P(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

The key advantage of the logistic regression algorithm is its simplicity in implementation, as it is based on the linear combination of parameters β_0 and β_1, which makes the model very efficient and reliable in terms of classification. On the

odd side, it solely relies on linear functions, so it might not fit to distinguish those datasets of non-linearity (Stoltzfus, 2011).

Supervised learning classifiers such as SVMs, Naïve Bayes and Logistic Regression are generally developed for binary classification. However, these classifiers can also be adapted for multiclass classification, which is conducted by iterating the training of classifiers to distinguish every example to a single class versus the rest instances. This type of classification is known as “one versus the rest” classification. One other way is to train the classifiers by every pair of classes. This is called “one versus one” classification. Usually, classification by “one versus the rest” class can be more difficult, and classification by “one versus one” class is simpler and faster, but the latter is more likely to lower effectiveness in performance.

In this thesis, we use Sklearn analysis modules for both classification stages. The classification models, Logistic Regression, Naïve Bayes classifier with Multinomial and Bernoulli variants, and SVMs with linear and RBF functions are all used in the experiments. For multiclass multi-label classification, an approach of “one versus the rest” is chosen because of its effectiveness.

3.6. Evaluation

3.6.1. Truth Set Creation

To train a classifier and compare the performance of classification, truth set data is necessary. In real world classification, a truth set is only required in the classifier training process. The creation of the truth set in this thesis has been conducted manually by a single entity, the author. There is an awareness that using human labour to generate a truth set can be subject to an inevitable bias, as humans can have different perspectives and interpretations towards the same data or topic. Thus, the following steps are conducted as an attempt to minimize the gap.

Because this study involves two stages of classification, the creation of true sets is required for both stages. For binary classification, tokenized sentences are read manually and classified into ‘sentence with feature requirements’ or ‘sentence with no feature requirements’. To rectify the manual classification, a clear definition of feature requirement is determined as a guideline, as in Section 3.5.2. The manual

classification is checked by the two-pass verification or double entry rule for data quality control. The manual classification has been conducted twice in different document sets by one entity over a different period and they are reconciled after an interleaving period. Where there exist any inconsistency or disputes in the two classifications, the definition is used to decide the truth class for the data sample.

Furthermore, the truth set of sentences with feature requirements obtained from the above process is used to generate a second truth set for multiclass multi-label classification. Parallel to binary classification, a clear definition of every category to be used as reference is developed (Table 3.2 in Section 3.5.2) for conducting manual classification. Again, the process is double performed, checked and finalized using the two-pass verification method to ensure the quality of the truth set and alleviate potential bias.

3.6.2. Evaluation Metrics

For implementing machine learning algorithms, it is important to evaluate the robustness of their performance. This process is a fundamental part to develop learning algorithms and compare candidate algorithms. The most common measures to assess learning algorithms are precision, recall and F-measures (Sokolova & Lapalme, 2009). Precision, recall and F-measures are designed for labelled classification. F-measures, the harmonic mean between precision and recall are especially for imbalanced datasets where the distribution of data per class is not normal.

Precision is a fraction of the number of documents correctly classified to the observed type, known as True Positive (TP) to the number of documents correctly classified to the observed type (TP) and the documents incorrectly classified to the observed type, False Positive (FP).

$$Precision = \frac{TP}{TP + FP}$$

Recall is a fraction of the number of documents correctly classified to the observed type (TP) to the number of documents correctly classified to the observed type (TP) and the documents incorrectly classified to the other type, False Negative (FN), but

actually they are in the observed type. In other words, recall is the actual accuracy rate per class.

$$Recall = \frac{TP}{TP + FN}$$

F-measure combines precision and recall measures where the relative importance (weight) of the two measures $\beta, \beta\epsilon]0,1]$ is considered.

$$F - measure = \frac{1}{\beta \frac{1}{Precision} + (1 - \beta) \frac{1}{Recall}}$$

When precision and recall are equally weighted $\beta = 50\%$, F-score can be obtained as:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{(precision + recall)}$$

Examples of the number of TP, TN, FP, FN in a confusion matrix can be seen in Tables 3.3 and 3.4.

Table 3.3 Confusion Matrix of TP, TN, FP, FN for Class 0 Data in Binary Classification

Class	0	1
0	#TP	#FN
1	#FP	#TN

Table 3.4 Confusion Matrix of TP, TN, FP, FN for Class 0 Data in Three-Class Classification

Class	0	1	2
0	#TP	#FN	#FN
1	#FP	#TN	#TN
2	#FP	#TN	#TN

In the current study, we use precision, recall and F_1, available in Sk-learn modules as measures for our classification model evaluation, because these metrics are common, easy to use and effective in terms of evaluating the performance of classification. The evaluation is conducted at both stages of classification (binary filtering and multiclass multi-label classification).

3.6.3. Cross-validation

Cross-validation is a method used to validate the effectiveness of a model in terms of generalizing independent data. Cross-validation involves the practice of partitioning datasets into corresponding subsets to validate the analysis model. To cross validate a machine learning classifier, it is necessary to have sample data to develop or train a classifier. This data set is known as a training set. Additionally, it is important to assess how effective the classifier can be used in prediction. The data used in this process should be an independent set for testing the classifier, which is called a validation set or a testing set. It is important to have a clear partitioning of a training set and a testing set to avoid the problems of overfitting and underfitting (Freitas, 2000). Overfitting happens when a classifier is developed with very high variants and can perform with unrealistically high accuracy for the testing set, but not effectively in generalization for new datasets. Underfitting occurs when an insufficient amount of data is used to train the classifier, which leads to the development of a classifier with high bias and returns a poor result of classification. The most common cross-validation methods are random subsampling and K-fold cross-validation.

3.6.3.1. Random Subsampling

In random subsampling, data are split into a fixed number of subsets. The classifier is trained and tested thoroughly with each subset where the number of samples are selected randomly without replacement. The measure of performance is given by the average of each training/testing loop. The advantage of this method is that it can minimize the bias in selecting training and testing data and it can be repeatedly conducted under the randomization. However, the limitation of this method is that because training and testing sets are dependent on the distribution of samples within every subset by randomization, it is possible that some samples can never be used in training or testing of classifiers, while the others might be selected repeatedly. For this reason and with our imbalance dataset, we need to look for the alternative to cross-validate our classification models.

3.6.3.2. K-fold Cross-validation

In K-Fold cross-validation, data are partitioned into k different folds of equal size. The subsamples of k-1 folds are used for training the classifier and the fold k is for the testing. The practice is applied to every fold of data until all is validated. The validation of classifiers is practised to every fold of data, totally k times. The performance of the classifier is average from all iterations of every data fold. Unlike the random subsampling method, in K-Fold cross-validation, all samples in the dataset are used thoroughly both for training and testing of the classifier.

Leave-one-out cross-validation is a k-fold cross-validation where $k = 1$. In this approach, every single sample of data is used for testing the classifier and all the rest is for the training. This is an exhaustive method of validation; it can be very time-consuming, especially when the data size is large.

Holdout method is another version of k-fold cross-validation where $k = 2$. This method simply divides a dataset into two blocking groups: a training set to train a classifier and a testing set to assess the trained classifier. The problem with this method is that when the data are sparse, it can be difficult to select the right portion of training and testing sets to generalize the classifier. Also, because it is heavily dependent on a single experiment of training and testing, the method can be biased.

Considering the advantages and disadvantages of this cross-validation technique, we conduct the experiments with cross-validation of 5 and 10 folds and using different proportions of training and testing datasets. For binary classification, we compare three training and testing proportions, 33%, 50%, 67% for training sets on 2-Fold cross-validation and k-fold cross-validation when $k = 5\ and\ k = 10$ by using cross-validation modules in Sk-learn library. For multiclass classification, as the size of our data is not very large in each category, the same modules are used with different training and testing proportions by 20%, 33% and 50%, along with 5-fold and 10-fold ($k = 5\ and\ k = 10$). In Sk-learn modules, the number of k in k-fold cross-validation and the proportions of training and testing sets can flexibly be selected.

3.7. Ranking of Feature Requirements

Beyond the process of categorization, the ordering of feature requirements is additionally performed. As discussed in Chapter 2, there are many techniques designed to sort textual documents by different purpose of utility. In this thesis, ranking of extracted feature requirements by term-weighted frequency distribution has been chosen (Chen et al., 2014; Keertipati et al., 2016; Lee et al., 2016). The selected technique fits very well to our purpose of ranking as well as the nature of our data (sentence granularity). It is applied to our experimented sentences by category to quickly illustrate the required features frequently mentioned by reviewers. The ranking stage is tested on generating the top-three-mentioned features per app and per category which are prioritized by frequency of words over the observed period.

To implement this data ranking, Rapid Automatic Keyword Extraction algorithm (RAKE) is used (Rose, Engel, Cramer, & Cowley, 2010). RAKE provides domain-independent extraction of keywords by weighted scores which are calculated from the ratio of the degree of word (word occurred in the keywords) to the frequency of words (the number of words co-occurred in the document corpus): $\deg(w)/freq(w)$. In RAKE, keywords are extracted by removing trivial words (stop words) and delimiting punctuations. The weighted score of each word is individually calculated before the total degree and frequency of word can be summed up by presented keywords. The keyword scores are then used to rank the documents which are the sentences with feature requirements in this thesis.

In this research, the technique is applied after the categorization of feature requirements to the data from each category and the SQL script is used to filter the data by app. Besides a date range is inserted in the script to enable information selection by a specific period. For the implementation of RAKE, source codes are adapted from this link, https://github.com/fabianvf/python-rake/blob/master/RAKE/RAKE.

3.8. Experimental Evaluation of Alternative Processing Techniques

There are a number of specific alternative text processing and categorising techniques that are used in different parts of the proposed process, as highlighted in Figure 3.2. Which techniques to use is decided by a series of experiments to evaluate the performance of the techniques with the data and context of the proposed process.

The experiments vary in the processing approach and tuning parameters as the independent variable and the performance measures of precision, recall and F-measure as the dependent variables. The control variables for each experiment include the hardware configuration and the input dataset. Table 3.5 summarises the experiments undertaken in different parts of the proposed process models.

Table 3.5 Experimental Summary

Process of Experiment	Processing Techniques evaluated
Binary filtering of feature/non-feature classification	
• Overall, 200 models are experimented with in the binary classification for filtering featured sentences	• Five machine learning classifiers: 1. SVMs with linear and multinomial variants (2) 2. Naïve Bayes with Bernoulli and multinomial variants (2) 3. Logistic Regression • Five cross-validation approaches: a) 2-fold cross-validation with 66%, 50% and 33% testing samples (3) b) K-folds of 5 and 10 (2) • Four attribute extraction techniques: a) unigram: n-gram (1,1) b) bigram: n-gram (1,2) c) trigram: n-gram (1,3) d) 4-gram (1,4) • Two sentiment analysis at sentence level: a) sentence only, b) sentence + sentiment
Multiclass multi-label classification for feature categorisation	
• Two separate experiments: - Classification by five feature categories - Classification by ten feature categories • Overall, 100 models are experimented with in the multiclass multi-label	• Five machine learning classifiers: 1. SVMs with linear and multinomial variants 2. Naïve Bayes with Bernoulli and multinomial variants 3. Logistic Regression • Five cross-validation approaches: a) 2-fold cross-validation with 50%, 33% and 20% testing samples,

classification for categorisation of feature requirement sentences	b) K-folds of 5 and 10 • Four attribute extraction techniques: a) unigram: n-gram (1,1), b) bigram: n-gram (1,2), c) trigram: n-gram (1,3) and d) 4-gram (1,4)

The proposed feature extraction process, including the data gathering and pre-processing, are described and explained in detail in this chapter. The planned experimental evaluations of a number of text analysis and clustering methods and their variations for key steps in the process are also presented in this chapter. In the next chapter, we present the results of these evaluations of different techniques at various stages of the process, as well as the results of applying the overall process on the online reviews of two case products.

Chapter 4

Results and Discussion

With the purpose of supporting product evolution by utilizing text processing techniques to filter useful information regarding software feature requirements from online reviews, the design and implementation of the experiments have been discussed in the previous chapter. In this chapter, we examine the outputs of implementation and discuss the findings. This chapter are outlined as: the evaluation of classification in Section 4.1, the results of classification along with the discussion of findings in Section 4.2, the ranking of extracted app features in Section 4.3 and some related work in Section 4.4.

4.1. Evaluation of Classification

4.1.1. Dataset

Before we explore the results of experiments and discuss the findings, a plot for our research can be briefly drawn as shown in Figure 4.1 below.

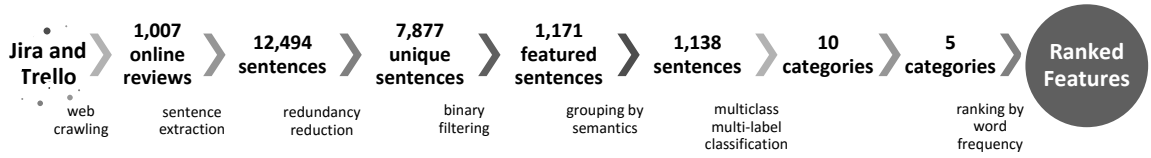


Figure 4.1 A Plot of Current Research

Our research experiments start from the selection of two sample software products (Jira and Trello) (Section 3.2). Raw reviews are manually collected from a sample website (www.g2crowd.com), which comprise of 1,007 app reviews. Most data are in 2015 to 2016 (roughly 90%). Each review record contains retrieved link, app name, reviewer, app rating, posting date and time, review title and review text. (See Appendix B, Figures B.1-B.4 for samples of collected data; the data in the experiments are stored in a SQLite database.)

In data pre-processing, retrieved reviews are segmented (tokenized) into 12,494 sentences (Section 3.3.1). In this process, the short sentence rule (Section 3.3.6) is applied. Next, tokenized review sentences are used to generate a truth set for binary classification, where Class 0 represented sentences with no feature requirements and Class 1 is for sentences with feature requirements. To summarize, the creation of a

truth set for binary classification generated 1,171 sentences with feature requirements.

In binary classification (Section 3.5.2), the input data are injected based on a set of unique sentences to avoid the problem of overfitting (Section 3.6.3) due to the use of repetitive input. This is because there are some duplications in our datasets. For example, every review is tagged by the same set of questions which are crawled repetitively by reviewers for the purpose of reference. The actual set of samples used in the experiments of binary classification is 7,877 unique sentences, of which 1,169 sentences are featured. It can be noticed that the distribution of the experimental dataset (Class 0 and Class 1) is considerably unbalanced; the experimental dataset is obviously skewed. The detail of sentence tokenization and truth set generation for binary classification can be found in Appendix B, Table B.1.

The truth set featuring sentences from binary classification (1,169 sentences) are further used to create the second truth set for multiclass multi-label classification. Before the classification, the sentences are verified for duplication by using semantic similarity measures (Section 3.3.9). The sentences, proposed by the same reviewer with a similar meaning are grouped together into one unit. At this process, the sentences with feature requirements are reduced to 1,138 (sentence groups). For more information on the distribution of datasets for multiclass multi-label classification see Appendix B, Table B.2.

In multiclass multi-label classification, those featured sentences are categorized into ten categories (Section 3.5.2). In addition, the experiments are extended on the categorization of data by reducing the number of categories to five classes which aimed to further test some of the well-performed classifiers from the 10-class classification process. Our proposed classification approach is multi-labelled, where one document (sentence) is eligible to more than one category or class.

Finally, in our study, we prioritize the documents from the categorization process by using the distribution of term-weighted frequency (Section 3.7). The results of this process are featured sentences which represent the features from each category and rank by their term-weighted scores.

4.1.2. Classification Models

Binary classification is the process of filtering sentences with feature requirements (Class 1) and discarding others with no feature requirements (Class 0), as detailed in Section 3.5.2. The experiments are conducted using three main classification algorithms (SVMs with linear and multinomial variants, Naïve Bayes with Bernoulli and multinomial variants and Logistic Regression), totally five classifiers (Section 3.5.3). Also, five cross-validation approaches are tested: 2-fold cross-validation with 66%, 50% and 33% testing samples, K-folds of 5 and 10. For tuning parameters, n-grams techniques such as unigram: n-gram (1,1), bigram: n-gram (1,2), trigram: n-gram (1,3) and 4-gram: n-gram (1,4) are under consideration. The use of sentiment analysis as input combinations are also experimented with (sentence only and sentence + sentiment). Overall, 200 models are experimented with in the binary classification for filtering featured sentences (Section 3.5.2).

Multiclass multi-label classification is the categorization of sentences with feature requirements into 10 pre-labelled categories (Section 3.5.2). Simultaneous to binary classification, multiclass multi-label classification relied on five classifiers from three classification algorithms (SVMs with linear and multinomial variants, Naïve Bayes with Bernoulli and multinomial variants and Logistic Regression). For cross-validation techniques, as the size of our sample dataset is tiny, the experiments are tested on 2-fold cross-validation with 50%, 33% and 20% testing samples, 5-fold and 10-fold validation. N-gram techniques including n-gram (1,1), n-gram (1,2), n-gram (1,3) and n-gram (1,4) are experimented. Unlike binary classification, sentence sentiments are not applied at this stage, as it is not appropriate to the purpose of categorization. The sentiment of a sentence might help boost the performance of binary classification, since the sentences with feature requirements tend to entail the appealing or feeling of reviewers towards the relevant apps' features, while in categorization, reviewers are less involved, but it matters to the labelling of samples to its categories, discarding what reviewers are thinking or feeling. Hence, in total, there are 100 models in the experiments of categorization.

4.2. Results of Classification

4.2.1. Binary Classification

In this section, we present the results of the binary classification. With a large number of proposed classification models, the results of binary classification performance are evaluated per classification algorithms and their tuning parameters in a search for the best-performing model in the experiments based on evaluation measures (precision, recall and F1 as detailed in Section 3.6.2) as follows. The detailed results of binary classification by average and by class performance can be seen in Appendix D, Tables D.1-D.10. The best outputs in each table are marked in bold.

Figure 4.2 shows the average proportion of performance measures by our proposed binary classifiers and cross-validation techniques. It is clear that SVMs (of both variants) are the top-performed classifiers with the highest average rates of precision, recall and F1 measures of above 88%. It is noticeable that when the size of the training set increase, the performance of classifiers also improves. The only exception is Naïve Bayes, where the recall and F1-scores fluctuate throughout the scales of training samples. A comparison of performance scores reveals that SVMs show relatively stable rates of performance, while in the other algorithms, the gap between the evaluation measures is generally larger. Linear SVM with a 33% testing set is the leading classifier in terms of performance scores.

The average performance scores offer us some hints of the best performed classification algorithms. However, this might not be sufficient for the judgement, because the performance by class might offer more significant insight to our classifiers. Hence, an investigation in class level for the data filtering process is further discussed next.

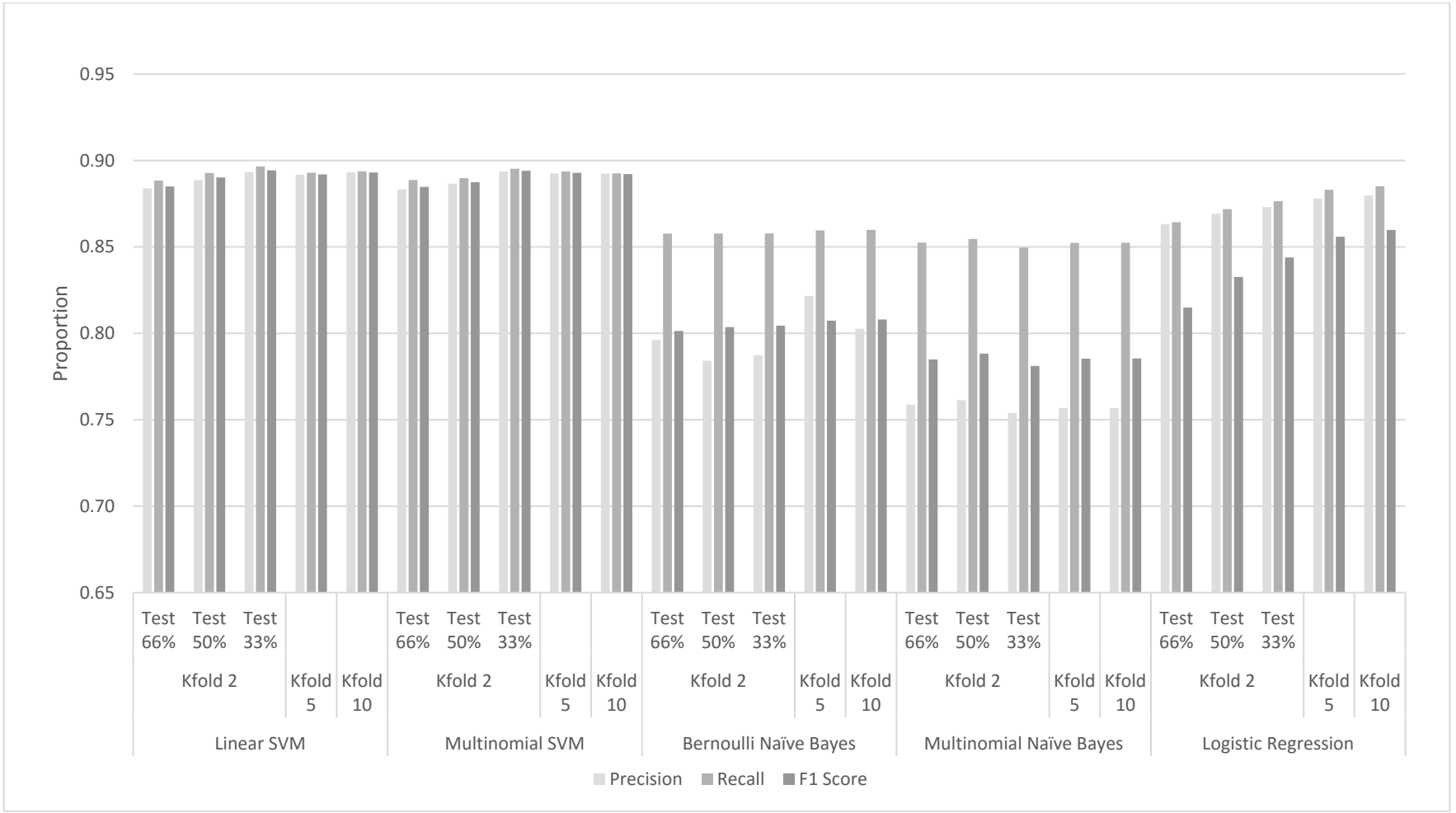


Figure 4.2 Average Performance of Binary Classification by Classifier

At class level, by average, similar improving trends in the classification performance can be seen with an increase in the training datasets (Figure 4.3). Also, it can be observed that the classification of data in Class 0 is more accurate than that in Class 1. All testing classifiers can correctly distinguish sentences with no feature requirements with the precision of no less than 85% and average recall and F-scores of at least 90%. With such a relative high-performance rate in Class 0, none of the same trends can be seen for the data classification in Class 1. Even though, in Class 1, the precision proportion of Logistic Regression is satisfactorily high, the recall and F-scores are comparatively poor. Naive Bayes are the most struggling algorithms to differentiate the data in Class 1. Meanwhile, SVMs shows more convincing results with the average precision, recall and F1 scores of between 55% and 69% (Figures 4.4, 4.5 and 4.6).

The justification of such classification trends might be explained by the skewness of our datasets, as the input data in Class 0 are overwhelmingly lopsided the data in Class 1. Because of this, more data in Class 0 are learned in the classification process, while Class 1 samples might not be sufficient to be distinguished. On the other hand, the quality of Class 1 samples might not be adequately rigid for the classification. All of these can result in such a difference in the performance measures at both classes.

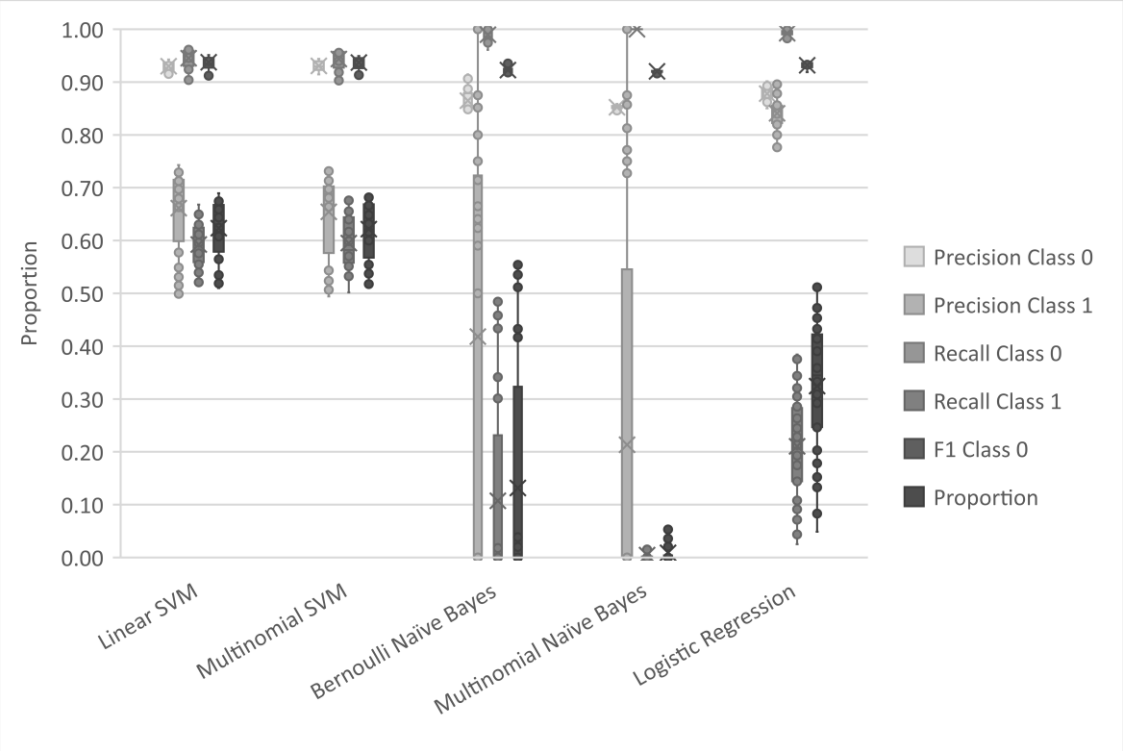


Figure 4.3 Average Precision, Recall, F1 Measures per Class for Binary Classification

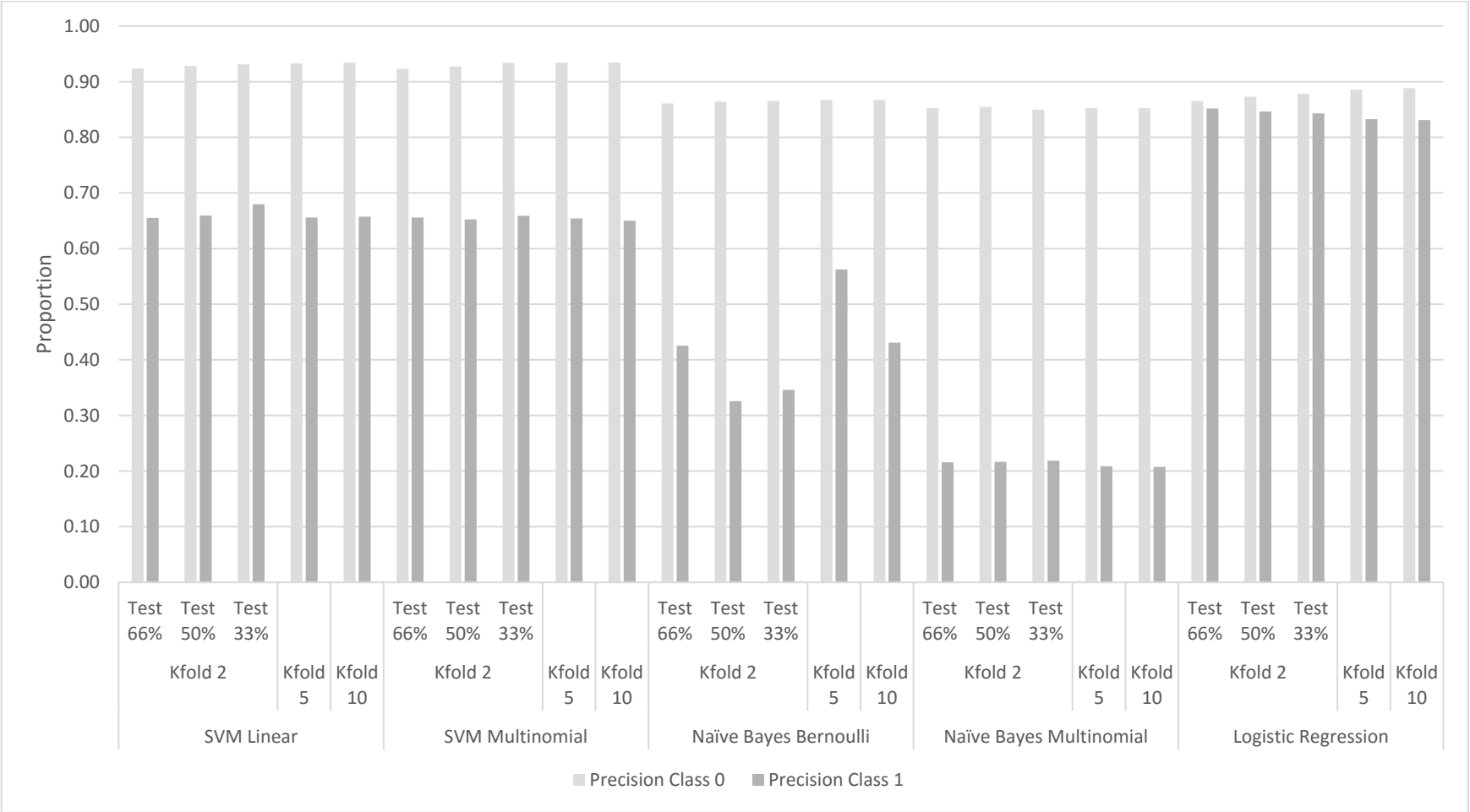


Figure 4.4 Average Precision per Class for Binary Classification by Classifier

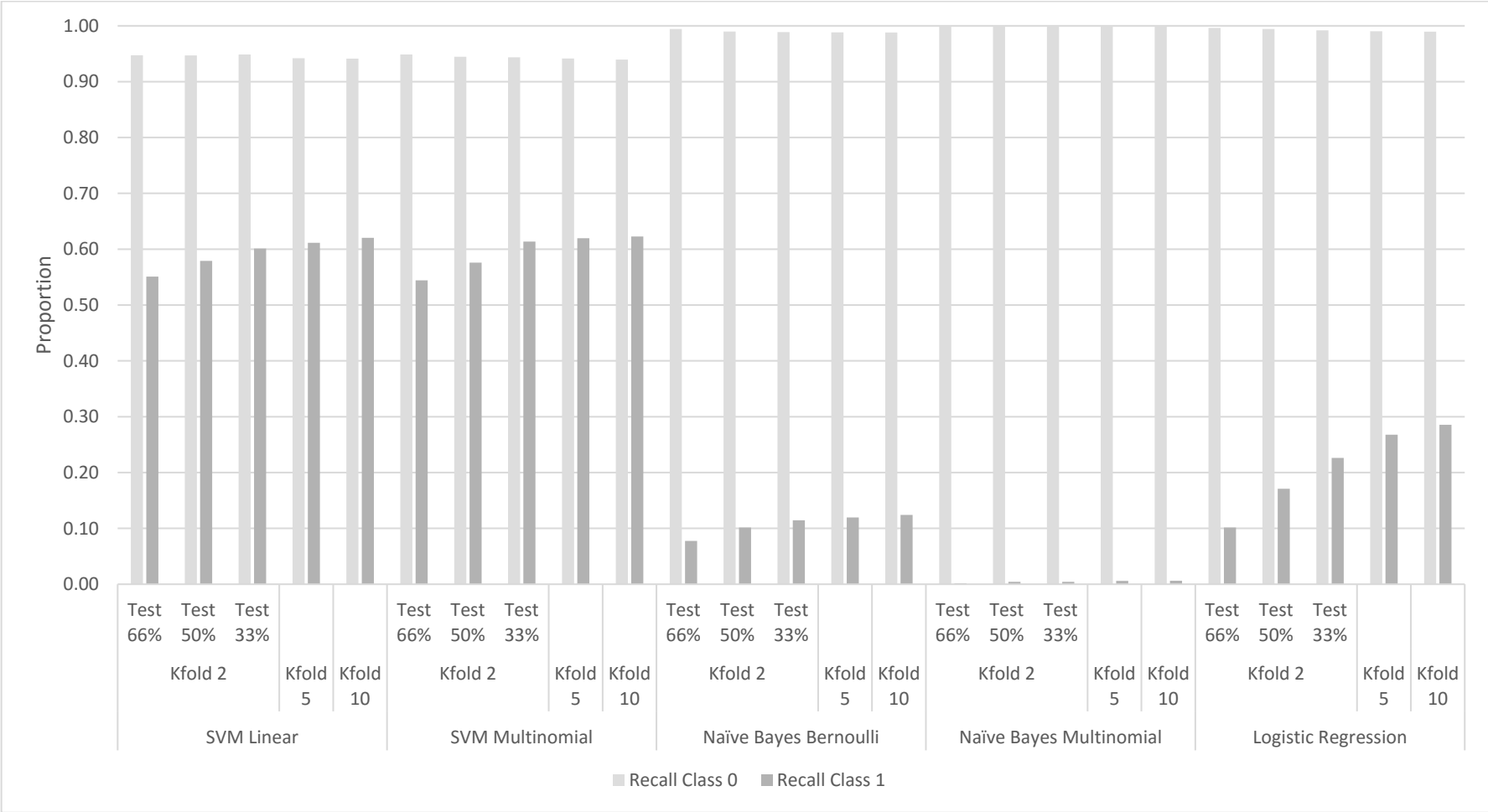


Figure 4.5 Average Recall per Class for Binary Classification by Classifier

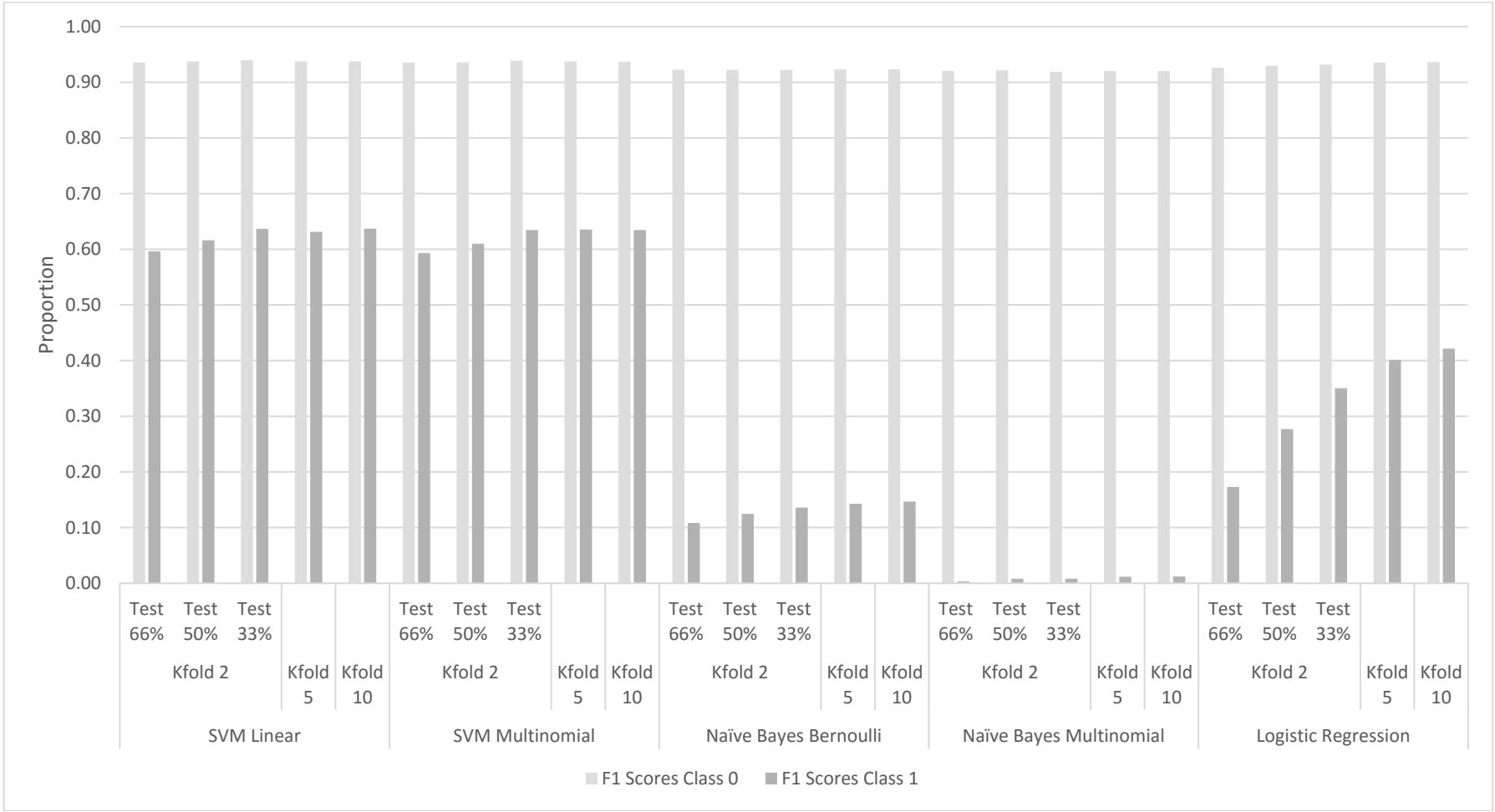


Figure 4.6 Average F1-score per Class for Binary Classification by Classifier

The average performances of classifiers on average and per class are explored in an analysis above, in which the top two classifiers with the most promising performance can be obtained, namely SVMs with linear and multinomial variants. Next, more detailed performance by tuning parametric variables (n-grams and sentiment analysis) for the classifiers is observed and analysed to narrow down our selection of the best model for the data filtering process. As more interesting results are seen (from above) when the size of training samples is bigger, our next observation is to focus on linear and multinomial SVMs with the testing size of 33% 2-fold, 5-fold and 10-fold cross-validation.

In Figure 4.7, it can be seen from the chart that when sentiment analysis of a sentence is applied (as input data) in combination with the sample sentence itself, the average precision, recall and F1of SVMs are improved slightly within the range of 0.1% to 2%. A similar pattern based on the same ground can be seen at class level as shown in Figure 4.8. In addition, it can be noticed that n-gram techniques help considerably improve the performance of SVMs, particularly when more tokens are applied (bigram: n-gram (1,2), trigram: n-gram (1,3) and quartet-gram: n-gram (1,4)) (Figure 4.9). The improvement of classification can also be seen in both classes (Figure 4.10).

As described, the top-leading models in binary classification can be identified, which are SVMs with sentence + sentiment, n-grams > 1 and 2-fold- 33% testing, K-fold 5 and K-fold 10, as shown in Figures 4.11, 4.12, 4.13 and 4.14. It is clear from Figure 4.11 that Linear SVM with n-gram (1,3), sentence + sentiment and K-fold 2 Testing 33% offers the highest rates of precision (91.1%), recall (91.5%) and F1 (91.2%) by average. However, on the class level, while the precision values in classes do not vary as much (Class 0 above 90% and Class 1 at roughly 70%), the recall rates are more competitive, especially at Class 1 which represents our targeted class for sentences with feature requirements. Recall rate (or per-class accuracy) of Linear SVM with n-gram (1,4), sentence + sentiment and K-fold 10 slightly outperforms all its counterparts, as well as the precision of Class 0 and F-scores of Class 1. At the same time, the average percentages of other measuring metrics are not distinctively different at the class level. For example, comparing Linear SVM and Multinomial SVM using the same top-leading criteria, the average performance scores have less than 1% deviation. The same pattern of tiny diversity is found in the class level by

the models (Figures 4.12, 4.13 and 4.14). As a result, it can be concluded that **Linear SVM with n-gram (1, 4), sentence + sentiment and K-fold 10** is the top model for culling sentences with feature requirements out of those with no feature requirements with the precision of 90.9%, recall of 91.1% and F1 of 91.0% (94.3%/71.3% precision, 95.3%/66.8% recall and 94.8%/69.0% F1 in Class 0/Class 1 respectively).

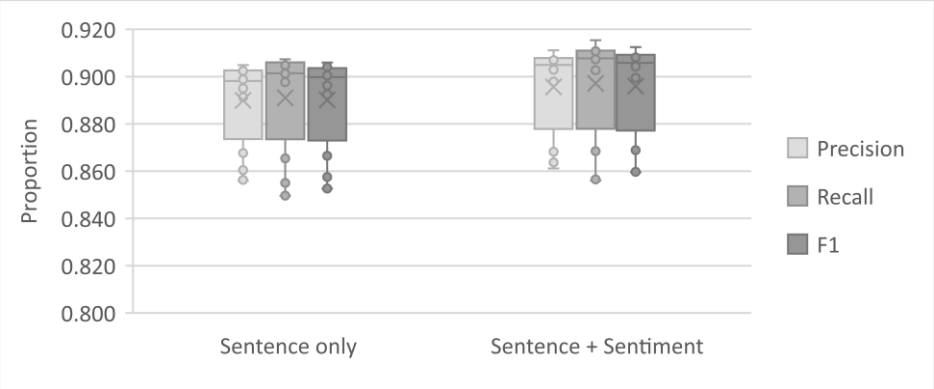


Figure 4.7 Average Precision, Recall, F1-Measure of SVMs by Sentence Only versus Sentence + Sentiment

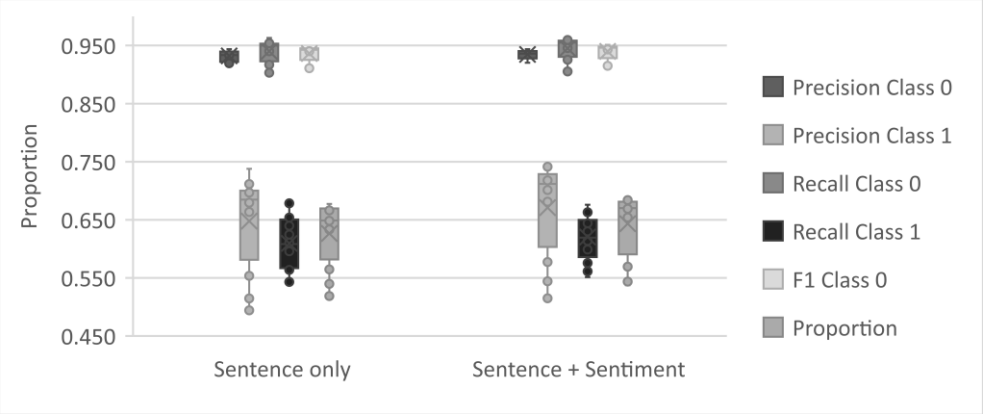


Figure 4.8 Average Precision, Recall, F1-Measure per Class of SVMs by Sentence Only versus Sentence + Sentiment

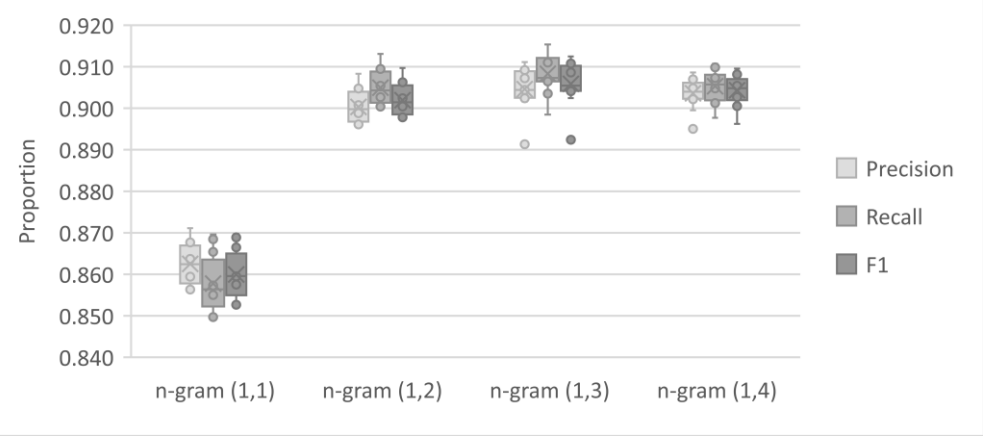


Figure 4.9 Average Precision, Recall, F1-Measure of SVMs by n-grams

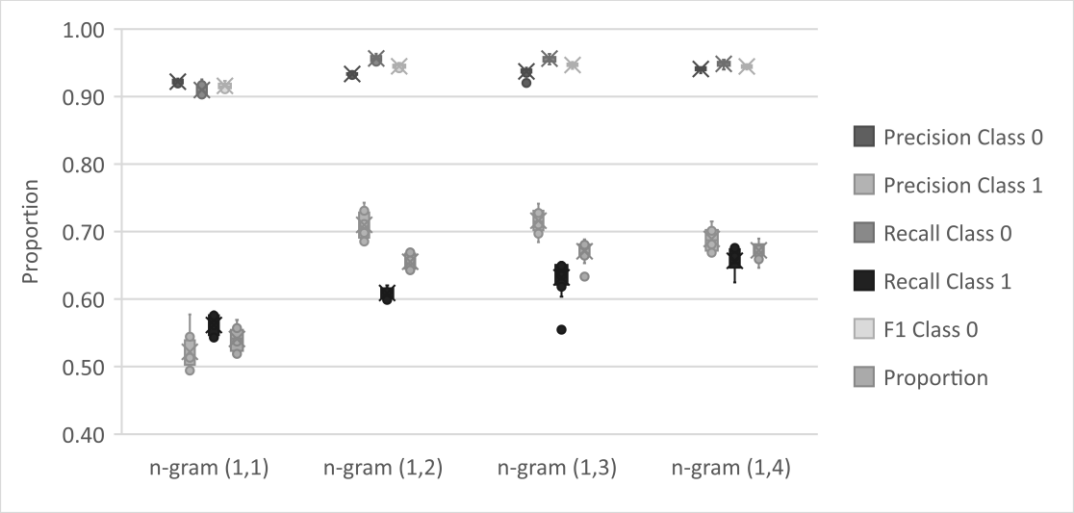


Figure 4.10 Average Precision, Recall, F1-Measure per Class of SVMs by n-grams

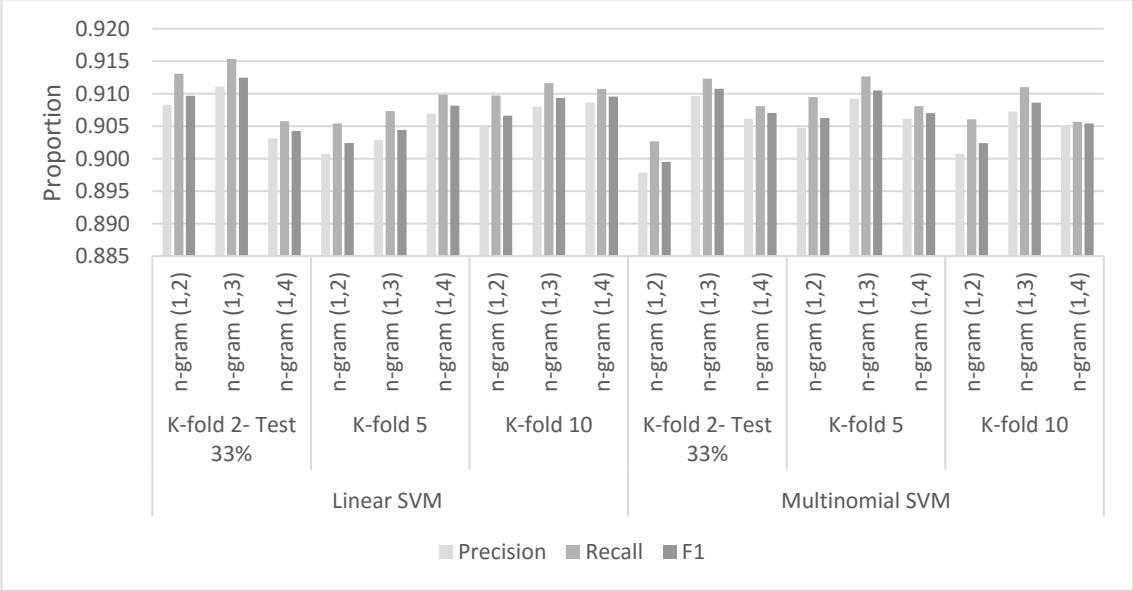


Figure 4.11 A Comparison of SVM Models for Binary Classification by K-folds and n-grams

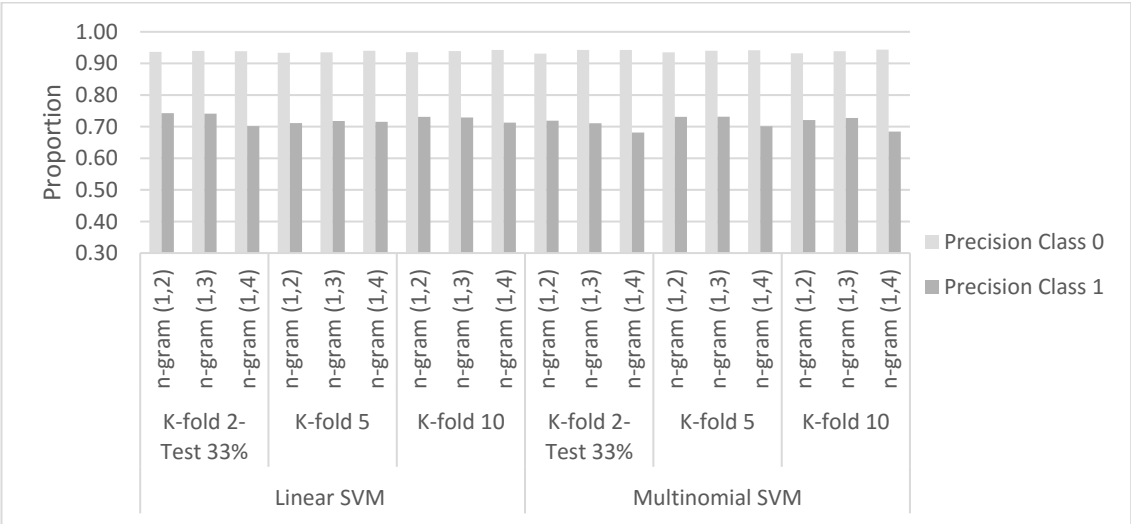


Figure 4.12 A Comparison of Precision per Class by SVM Models for Binary Classification by K-folds and n-grams

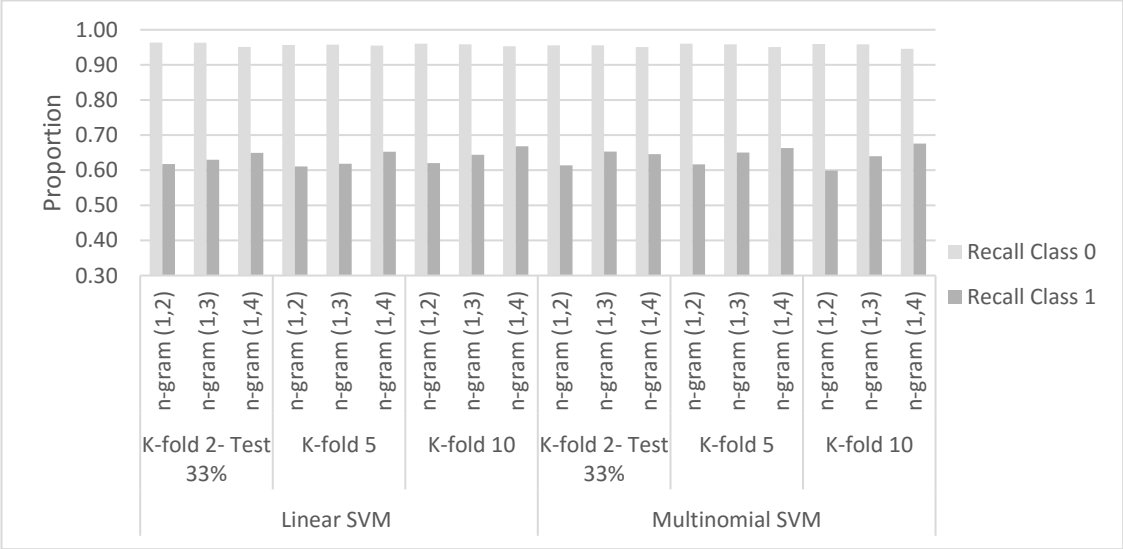


Figure 4.13 A Comparison of Recall per Class by SVM Models for Binary Classification by K-folds and n-grams

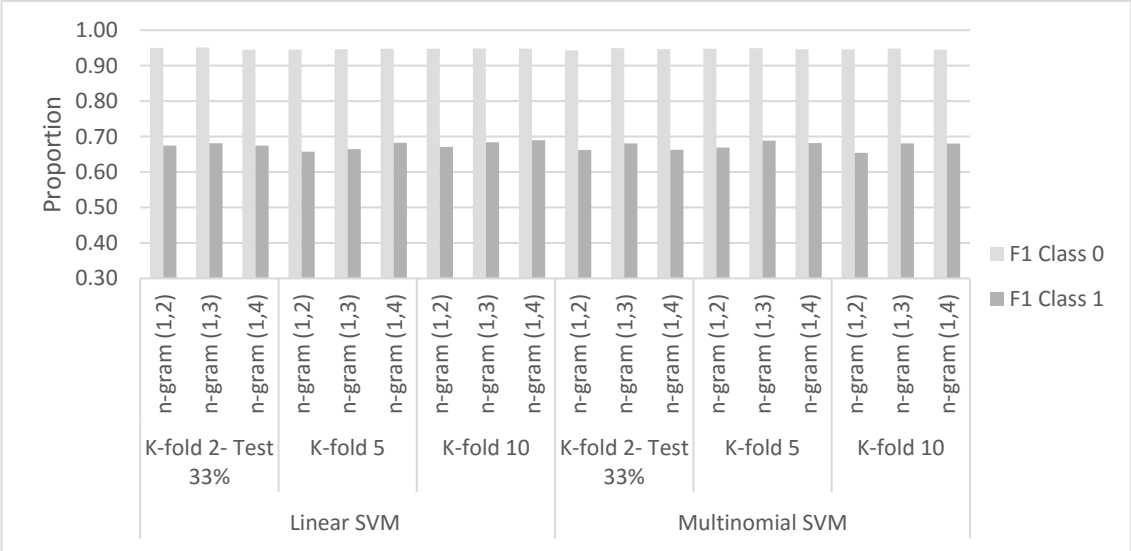


Figure 4.14 A Comparison of F1-score per Class by SVM Models for Binary Classification by K-folds and n-grams

To further test the effectiveness of our top classifier for binary classification, additional experiments are conducted by using two data simulation settings. The purpose of the first experiment is to evaluate the strength of our selected classifier under different input criteria, which aims to see if our classifier only performs well with any specific product reviews. To this end, our datasets are resized by using all samples from the Jira reviews as the training set (Class 0 = 3,778 samples and Class 1 = 701) with a few samples from Trello as a seed (Class 1 = 49) and most samples of Trello are used as the testing set. The justification of this is that Class 1 is our

targeted filtering class which presents the sentences with feature requirements and that the samples from this class are much smaller than Class 0. It is necessary to train the classifier with some seeded data (as comparative to the testing sample size in our original experiments of binary model selection, 33% testing) to boost the performance scores.

The results of the first simulation experiment on binary classification using the best-performed classifier, Linear SVM with n-gram (1, 4), sentence + sentiment and K-fold 10 with the input criteria above can be found in Tables 4.1 and 4.2 below. An analysis of binary classification results of the Trello dataset shows that our best classifier is effective at filtering the sentences with feature requirements for Trello under our experimental setting with a precision of 90.9%, recall of 89.7% and F1 of 90.2%. The recall score on Trello data simulation drops slightly from the original experiment (Precision 90.9%, Recall 91.1%, F1 91.0%), but the recall in Class 1 has slightly improved (71.4% > 66.8%).

Table 4.1 Performance of Linear SVM for Binary Classification of Trello Data

Classifier	Class	Precision	Recall	F1 Scores
Linear SVM, n-gram (1,4), sentence + sentiment, K-fold 10	Class 0	0.958	0.923	0.940
	Class 1	0.571	0.714	0.634
	Average	0.909	0.897	0.902

Table 4.2 Confusion Matrix of Linear SVM for Binary Classification on Trello Data

Class	Class 0	Class 1	#Sample
Class 0	2705	225	2930
Class 1	120	299	419

Besides, our top classifier which is trained from the above experiment is experimented with an external dataset in the second data simulation experiment. This is to evaluate whether our classifier is effective within the data domain of a specific site. To achieve this, external data simulation is manually selected from online reviews of similar apps to our baseline data (such as Asana, Wrike, Slack, and including Jira itself), but from different web sources (getapp.com and trustradius.com). The reviews on these sources are from actual app users of different levels of expertise such as system analysts, software engineers, developers, strategy planners and managers. Totally there are 224 sentences for simulation, where 50% is with feature requirements and the other half with no feature requirements.

The results of this experiment are seen in Tables 4.3 and 4.4. It can be seen that scores of precision, recall and F_1 are a little lower than those of the original experiment and the Trello dataset experiment above. However, the performance by average is very satisfactory, especially as the classifier achieves a moderately high per-class recall of 78.6% in Class 1 with this balanced dataset. It should also be noted here that the training set used in this experiment is mostly Jira samples with a small seed from Trello.

Table 4.3 Performance of Linear SVM for Binary Filtering on External Data Simulation

Classifier	Class	Precision	Recall	F1 Scores
Linear SVM, n-gram (1,4), sentence + sentiment, K-fold 10	Class 0	0.817	0.955	0.881
	Class 1	0.946	0.786	0.859
	Average	0.882	0.871	0.870

Table 4.4 Confusion Matrix of Linear SVM for Binary Filtering on External Data Simulation

Class	Class 0	Class 1	#Sample
Class 0	107	5	112
Class 1	24	88	112

From the experiments of data simulation, it can be said that our best classifier, Linear SVM with n-gram (1, 4), sentence + sentiment and 10-fold cross-validation is effective at filtering featured sentences on binary classification.

4.2.2. Multiclass Multi-label Classification (Categorization)

In this section, we examine the outputs for multiclass multi-label classification. The detailed results of multiclass multi-label categorization on both average performance and per-class performance by classifiers can be found in Appendix D, Tables D.11-D.20.

An overview of Figures 4.15, 4.16 and 4.17 illustrates that precision rates tend to be higher than the recall and F-scores by all classifiers. Figure 4.15 shows that SVMs are likely to perform better than other contending classifiers, compared by the precision, recall and F-scores. Linear SVM shows a slightly better performance on average, but Multinomial SVM can be comparative with its best tuning parameters, while the other categorization models (Naïve Bayes and Logistic Regression) perform relatively poor at categorizing our datasets.

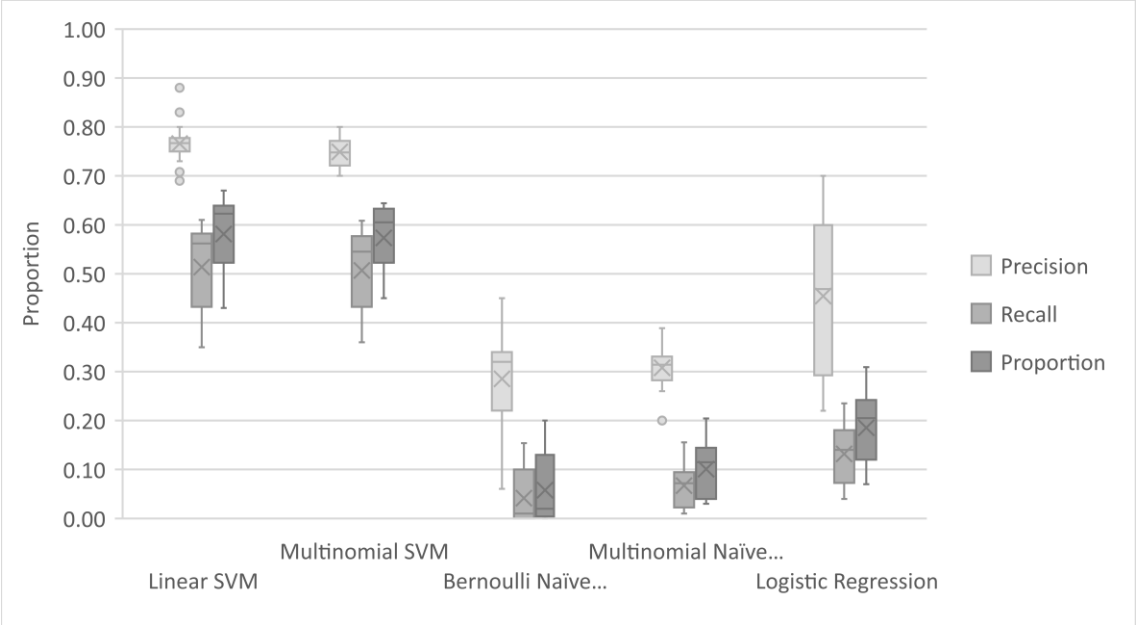


Figure 4.15 Average Performance per Classifier for Multiclass Multi-label Classification

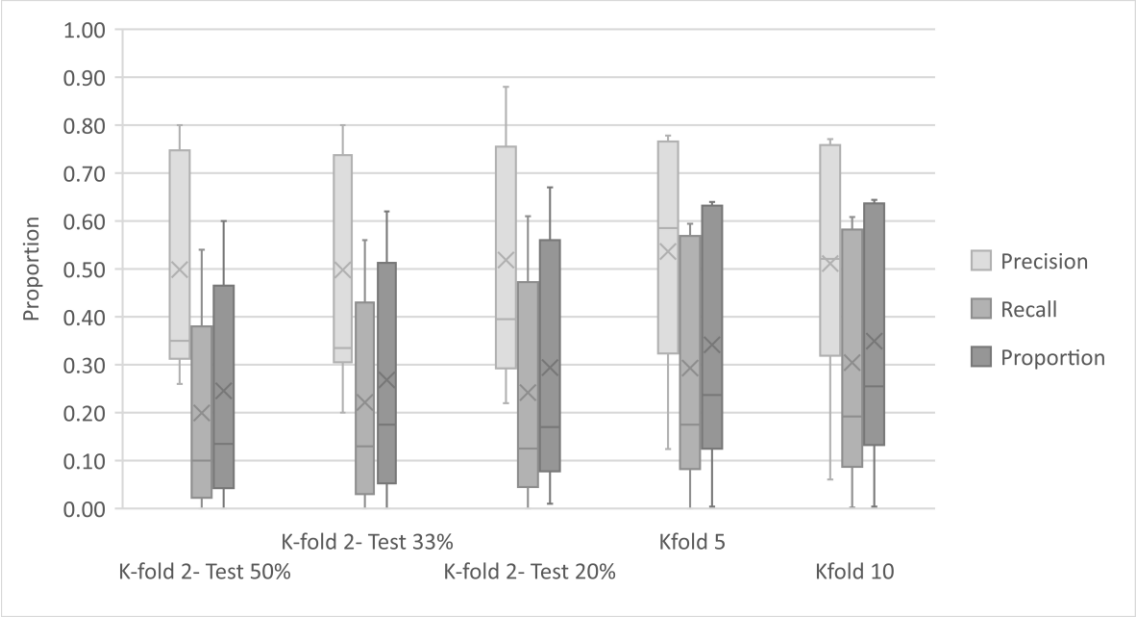


Figure 4.16 Average Performance per Cross-Validation Technique for Multiclass Multi-label Classification

Similar to the binary classification, an increase in the number of samples for the training of classifiers helps improve the overall performance (Figure 4.16). K-fold 5 tends to show promising rates of precision, recall and F-scores on average. Meanwhile, K-fold 2- Test 20% and K-fold 10 show relatively competitive results.

It can be seen in Figure 4.17 that when the number of tokens in n-grams is lower, average performance of categorization is likely to be better. Precision, recall, F-

measures of n-gram (1,1) are generally greater than those by n-gram (1,2), n-gram (1,3) and n-gram (1,4) respectively, which means that n-gram techniques of higher tokens are not very helpful in terms of multiclass multi-label categorization. Single or fewer words or tokens learned by classifiers might be more effective at representing the product features in each category.

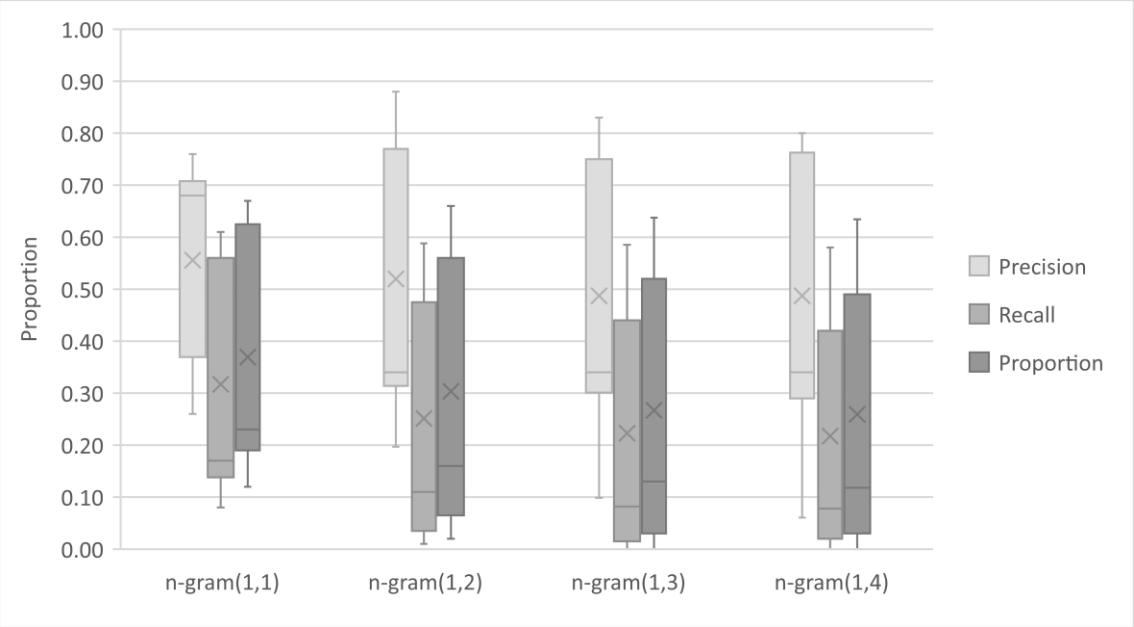


Figure 4.17 Average Performance per n-gram Technique for Multiclass Multi-label Classification

The overall trend of performance by our classification models is analysed above, which obtains an overview performance of multiclass multi-label classification. However, to determine the best model for the categorization, the performance at class level is required to be further explored, especially the performance scores by SVMs. Furthermore, by the fact that larger samples are better for training the classifiers with our small-scaled datasets (1,249 samples in 10 categories), the next discussion focuses on K-fold 2- Test 20%, K-fold 5 and K-fold 10 along with n-gram parameters.

Table 4.5 shows the average precision, recall and F-measures of linear SVM versus multinomial SVM with K-fold validations of 2, 5 and 10, and n-grams. It can be observed that while Linear SVM and Multinomial SVM are comparative in performance, Linear SVM has slightly better performance scores. Also, when n-gram techniques are applied, the precision scores tend to improve. Particularly, the precision values are likely to increase with an increase in the number of n-grams. On the contrary, n-gram techniques seem to distract the recall rates in categorization as

well as the F-scores. On average, the highest precision score (88%) can be seen by Linear SVM with K-fold 2- Test 20% and n-gram (1, 2), whereas Linear SVM with K-fold 2- Test 20% and n-gram (1, 1) outnumbers the other models in terms of recall (61%) and F-measures (67%).

Table 4.5 Average Performance Measures of Linear and Multinomial SVMs for Multiclass Multi-label Classification

Classifier	Validation	N-gram	Precision	Recall	F1 Scores
Linear SVM	K-fold 2 Test 20%	n-gram (1,1)	0.76	**0.61**	**0.67**
		n-gram (1,2)	**0.88**	0.58	0.66
		n-gram (1,3)	0.83	0.48	0.57
		n-gram (1,4)	0.80	0.43	0.51
	K-fold 5	n-gram (1,1)	0.70	0.59	0.63
		n-gram (1,2)	0.78	0.57	0.64
		n-gram (1,3)	0.77	0.57	0.63
		n-gram (1,4)	0.78	0.56	0.63
	K-fold 10	n-gram (1,1)	0.71	0.60	0.64
		n-gram (1,2)	0.76	0.58	0.64
		n-gram (1,3)	0.77	0.59	0.64
		n-gram (1,4)	0.77	0.58	0.63
Multinomial SVM	K-fold 2 Test 20%	n-gram (1,1)	0.74	0.56	0.63
		n-gram (1,2)	0.80	0.50	0.59
		n-gram (1,3)	0.72	0.45	0.52
		n-gram (1,4)	0.72	0.43	0.53
	K-fold 5	n-gram (1,1)	0.72	0.58	0.63
		n-gram (1,2)	0.78	0.58	0.64
		n-gram (1,3)	0.77	0.56	0.62
		n-gram (1,4)	0.76	0.56	0.62
	K-fold 10	n-gram (1,1)	0.71	0.61	0.64
		n-gram (1,2)	0.76	0.59	0.64
		n-gram (1,3)	0.76	0.57	0.63
		n-gram (1,4)	0.77	0.58	0.63

Tables 4.6, 4.7 and 4.8 show the precision, recall and F-scores at class level for Linear SVM and Multinomial SVM respectively. Again, Linear SVM with K-fold 2- Test 20% and n-gram (1, 2) has distinctively high precision rates in all classes, but it is poor at categorizing the samples at Classes 1, 2 ,3 ,4 and 9 with the recall rates ranging merely from 13% to 45%. Meanwhile, even though Linear SVM with K-fold 2- Test 20% and n-gram (1, 1) is weak at categorizing the Class 1 and Class 2 samples, the performance of the model is better at the other classes with an average recall of over 60% and average F-scores above 70%. The classifiers with the

potential to differentiate Class 1 samples are those models of Linear and Multinomial SVMs with K-folds 5 and 10 and n-gram (1,1**).**

Table 4.6 Precision per Class of Linear and Multinomial SVMs for Multiclass Multi-label Classification

Classifier	Validation	Tuning Parameter	Precision/ Class									
			Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Linear SVM	K-fold 2-Test 20%	n-gram (1,1)	0.70	0.00	0.80	0.54	0.88	0.89	**1.00**	0.93	0.90	0.79
		n-gram (1,2)	**0.84**	**1.00**	**1.00**	**1.00**	0.80	0.89	**1.00**	**1.00**	0.83	0.83
		n-gram (1,3)	0.75	**1.00**	**1.00**	0.80	**1.00**	0.93	**1.00**	**1.00**	0.86	0.68
		n-gram (1,4)	0.76	**1.00**	0.00	0.67	**1.00**	**1.00**	**1.00**	**1.00**	**0.93**	0.81
	K-fold 5	n-gram (1,1)	0.67	0.52	0.81	0.49	0.68	0.83	**1.00**	0.94	0.80	0.62
		n-gram (1,2)	0.71	0.89	0.95	0.57	0.78	0.92	**1.00**	0.96	0.83	0.72
		n-gram (1,3)	0.70	0.66	0.96	0.56	0.78	0.92	**1.00**	0.94	0.83	0.74
		n-gram (1,4)	0.71	0.88	0.97	0.57	0.81	0.93	**1.00**	0.95	0.80	0.74
	K-fold 10	n-gram (1,1)	0.68	0.61	0.83	0.45	0.82	0.78	**1.00**	0.97	0.79	0.63
		n-gram (1,2)	0.71	0.63	0.89	0.56	0.86	0.91	0.94	0.96	0.85	0.69
		n-gram (1,3)	0.71	0.77	0.84	0.60	0.79	0.92	**1.00**	0.97	0.83	0.73
		n-gram (1,4)	0.71	0.65	0.90	0.63	0.80	0.95	**1.00**	0.95	0.79	0.69
Multinomial SVM	K-fold 2-Test 20%	n-gram (1,1)	0.66	0.56	0.82	0.46	0.83	0.84	**1.00**	0.96	0.78	0.61
		n-gram (1,2)	0.66	0.50	**1.00**	0.62	0.77	0.95	**1.00**	0.96	0.84	**0.85**
		n-gram (1,3)	0.69	0.00	**1.00**	0.75	**1.00**	0.81	**1.00**	**1.00**	0.80	0.68
		n-gram (1,4)	0.67	0.00	**1.00**	**1.00**	**1.00**	0.95	**1.00**	0.95	0.81	0.68
	K-fold 5	n-gram (1,1)	0.70	0.55	0.87	0.50	0.67	0.84	0.96	0.94	0.81	0.65
		n-gram (1,2)	0.70	0.88	0.94	0.65	0.79	0.93	**1.00**	0.97	0.79	0.71
		n-gram (1,3)	0.71	0.86	**1.00**	0.55	0.76	0.94	**1.00**	0.97	0.79	0.70
		n-gram (1,4)	0.71	0.60	1.00	0.55	0.77	0.88	1.00	0.97	0.80	0.72
	K-fold 10	n-gram (1,1)	0.67	0.62	0.79	0.42	0.81	0.83	1.00	0.94	0.83	0.65
		n-gram (1,2)	0.70	0.82	0.91	0.59	0.80	0.91	1.00	0.95	0.83	0.70
		n-gram (1,3)	0.70	0.63	0.91	0.53	0.82	0.94	1.00	0.96	0.83	0.69
		n-gram (1,4)	0.72	0.66	0.91	0.62	0.67	0.91	1.00	0.96	0.84	0.73

Table 4.7 Recall per Class of Linear and Multinomial SVMs for Multiclass Multi-label Classification

Classifier	Validation	Tuning Parameter	Recall/ Class									
			Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Linear SVM	K-fold 2-Test 20%	n-gram (1,1)	0.63	0.00	0.29	**0.54**	**0.70**	0.62	0.86	0.82	0.64	**0.61**
		n-gram (1,2)	0.77	0.17	0.13	0.25	0.36	0.67	**1.00**	0.75	0.62	0.45
		n-gram (1,3)	0.65	0.08	0.17	0.13	0.50	0.44	0.50	0.64	0.63	0.45
		n-gram (1,4)	0.68	0.07	0.00	0.08	0.29	0.41	0.60	**0.86**	0.59	0.30
	K-fold 5	n-gram (1,1)	0.65	0.27	0.43	0.33	0.43	**0.72**	0.82	0.78	0.72	0.54
		n-gram (1,2)	0.71	0.27	0.37	0.26	0.41	0.63	0.76	0.75	0.63	0.49
		n-gram (1,3)	0.74	0.21	0.33	0.22	0.39	0.61	0.79	0.73	0.65	0.46

		n-gram (1,4)	0.75	0.19	0.32	0.21	0.36	0.59	0.73	0.76	0.62	0.44
	K-fold 10	n-gram (1,1)	0.69	**0.40**	0.44	0.35	0.51	0.65	0.85	0.77	0.71	0.50
		n-gram (1,2)	0.72	0.25	0.35	0.26	0.49	0.65	0.79	0.76	0.67	0.48
		n-gram (1,3)	0.76	0.23	0.30	0.26	0.47	0.62	0.79	0.77	0.65	0.47
		n-gram (1,4)	**0.77**	0.15	0.32	0.23	0.40	0.62	0.76	0.73	0.66	0.46
Multinomial SVM	K-fold 2-Test 20%	n-gram (1,1)	0.69	0.28	**0.48**	0.21	0.29	0.51	0.81	0.76	0.60	0.45
		n-gram (1,2)	0.63	0.05	0.31	0.08	0.40	0.32	0.64	0.63	0.56	0.33
		n-gram (1,3)	0.59	0.00	0.03	0.05	0.25	0.35	0.57	0.29	0.41	0.24
		n-gram (1,4)	0.66	0.00	0.07	0.03	0.11	0.28	0.56	0.47	0.35	0.22
	K-fold 5	n-gram (1,1)	0.68	0.32	0.41	0.31	0.43	0.64	0.79	0.77	0.69	0.50
		n-gram (1,2)	0.73	0.27	0.32	0.26	0.43	0.60	0.79	0.79	0.65	0.49
		n-gram (1,3)	0.74	0.19	0.35	0.23	0.30	0.60	0.73	0.75	0.62	0.44
		n-gram (1,4)	0.75	0.19	0.30	0.21	0.38	0.59	0.73	0.72	0.64	0.45
	K-fold 10	n-gram (1,1)	0.67	0.38	0.41	0.31	0.45	0.72	0.82	0.76	0.75	0.57
		n-gram (1,2)	0.72	0.27	0.35	0.28	0.40	0.65	0.76	0.78	0.68	0.50
		n-gram (1,3)	0.74	0.21	0.35	0.24	0.45	0.63	0.76	0.75	0.66	0.44
		n-gram (1,4)	0.77	0.21	0.32	0.22	0.38	0.60	0.73	0.75	0.67	0.47

Table 4.8 F1-score per Class of Linear and Multinomial SVMs for Multiclass Multi-label Classification

Classifier	Validation	Tuning Parameter	F1/ Class									
			Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Linear SVM	K-fold 2-Test 20%	n-gram (1,1)	0.66	0.00	0.42	**0.54**	**0.78**	0.73	0.92	0.87	0.75	**0.69**
		n-gram (1,2)	**0.80**	0.29	0.24	0.40	0.50	0.76	**1.00**	0.86	0.71	0.58
		n-gram (1,3)	0.69	0.15	0.29	0.23	0.67	0.60	0.67	0.78	0.73	0.55
		n-gram (1,4)	0.72	0.12	0.00	0.15	0.45	0.58	0.75	**0.92**	0.72	0.44
	K-fold 5	n-gram (1,1)	0.66	0.35	0.56	0.39	0.52	**0.76**	0.89	0.84	0.76	0.57
		n-gram (1,2)	0.71	0.39	0.51	0.35	0.52	0.74	0.86	0.83	0.71	0.58
		n-gram (1,3)	0.72	0.31	0.49	0.30	0.52	0.73	0.88	0.81	0.73	0.57
		n-gram (1,4)	0.73	0.31	0.46	0.31	0.48	0.72	0.82	0.84	0.70	0.55
	K-fold 10	n-gram (1,1)	0.69	**0.46**	0.56	0.38	0.60	0.70	0.91	0.85	0.74	0.55
		n-gram (1,2)	0.71	0.34	0.49	0.34	0.60	0.75	0.85	0.84	0.74	0.56
		n-gram (1,3)	0.73	0.34	0.43	0.35	0.57	0.74	0.87	0.86	0.72	0.56
		n-gram (1,4)	0.74	0.23	0.45	0.33	0.52	0.74	0.85	0.81	0.71	0.55
Multinomial SVM	K-fold 2-Test 20%	n-gram (1,1)	0.67	0.37	**0.61**	0.29	0.43	0.63	0.9	0.85	0.68	0.52
		n-gram (1,2)	0.64	0.09	0.47	0.14	0.53	0.48	0.78	0.76	0.67	0.47
		n-gram (1,3)	0.64	0.00	0.07	0.10	0.40	0.49	0.73	0.45	0.55	0.35
		n-gram (1,4)	0.66	0.00	0.14	0.06	0.19	0.44	0.71	0.63	0.49	0.34
	K-fold 5	n-gram (1,1)	0.69	0.38	0.54	0.37	0.52	0.73	0.86	0.84	0.75	0.56
		n-gram (1,2)	0.71	0.39	0.45	0.37	0.53	0.73	0.87	0.87	0.70	0.58
		n-gram (1,3)	0.72	0.30	0.51	0.32	0.43	0.73	0.84	0.84	0.69	0.54
		n-gram (1,4)	0.73	0.28	0.46	0.30	0.49	0.70	0.84	0.82	0.71	0.54
	K-fold 10	n-gram (1,1)	0.67	0.45	0.51	0.35	0.55	0.76	0.89	0.82	**0.78**	0.60

		n-gram (1,2)	0.71	0.40	0.49	0.38	0.50	0.75	0.84	0.84	0.74	0.58
		n-gram (1,3)	0.72	0.30	0.49	0.32	0.55	0.74	0.85	0.84	0.72	0.53
		n-gram (1,4)	0.74	0.30	0.45	0.32	0.47	0.72	0.82	0.83	0.74	0.56

The most satisfactory classifier among the group is **Linear SVM with K-fold 10 and n-gram (1, 1)**, where the precision rates on every class are over 60%, except for Class 3 which accounts for only 45%. The recall score on Class 3 is also the lowest (35%), whereas Classes 1 and 2 can be distinguished at 40% and 44% of recall accordingly. Meanwhile, the recall proportion of the other classes is more than 50%. Similar trends can be seen for F1-measures by this model. The model is especially good at classifying the data in Classes 6, 7 and 8 with the precisions, recalls and F-scores of more than 70%. This model is thus the best model in terms of categorization with the highest recall rates in all categories, though not with satisfactory precisions, recalls and F-scores at every class. The justification behind this might be because the number of samples of these specific classes for training classifiers is low. Plus, the samples themselves might belong to multiple classes or might contain a complicated set of tokens for generalizing the pattern of the category. They, therefore, are too ambiguous to be correctly classified into those categories.

For this reason, supplementary experiments are conducted, which aim to test some of the best performed classifiers as mentioned above under the different criteria as of the limitation in testing datasets. Because our classifiers rely on probability in prediction, the reduction in the number of classifying categories might boost their performance. The setting of the additional experiments is that the number of categories regarding the multiclass multi-label classification is reduced from ten to five classes. The deducted categories are virtually treated as outliers and are eliminated. The samples from Classes 0, 5, 6, 7 and 8 are chosen to retain in the experiments. The reason for the reduction of samples from the other categories (Classes 1, 2, 3 and 4) is that the number of samples in these classes is quite low, as well as their weak performance scores from the initial experiments. For Class 9 which is the class for the unknown-category features (Others), the categorization on this class can be very ambiguous; the data in this class are thereby rejected. Hence, new class labels for the classification experiments are re-defined as: Class 0 = Board management, Class 1 = Data management, visualization and integration, Class 2 =

Mobile services, Class 3 = Notification and Class 4 = Navigation and search. For the classification models, those tending to have average performance scores in the original experiments are selected, such as linear SVM with K-fold 2- Test 20%, K-fold 5 and K-fold 10, n-gram (1,1) and n-gram (1,2) (6 models overall).

The results of class-reduction experiments can be seen in Table 4.9. It is clear that the classification in five categories is better than that of ten categories. By average, the performance scores under this experiment setting are above three quarters, relatively better than the ten-category classification (roughly 65%). Using 5-fold and 10-fold cross-validation methods gain higher measures of precision, recall and F_1 and applying n-gram (1,1) generally offers improved results from n-gram (1,2). To compare, precision scores of every testing model here are not much different both on per-class and average performances, where the values are moderately high, all above 80%. When considering recall rates, **Linear SVM with K-fold 5 and n-gram (1,1)** is the best-performed classifier, having high recall scores in every classifying category. The same pattern can also be seen in F1-measures by the same model. (See Appendix D, Table D.21 for more detail on the class-reduction experiments)

Table 4.9 Performance of Linear SVM for Multiclass Multi-label Classification on Reduced-Class Dataset

Performance Measure /Support	Validation	N-gram	Linear SVM					
			Class 0	Class 1	Class 2	Class 3	Class 4	Average
Precision	K-fold 2-Test 20%	n-gram (1,1)	0.81	0.88	**1.00**	**1.00**	**0.91**	0.86
		n-gram (1,2)	0.80	**0.95**	**1.00**	**1.00**	0.80	0.85
	K-fold 5	n-gram (1,1)	0.85	0.88	**1.00**	0.95	0.83	0.86
		n-gram (1,2)	0.84	0.94	**1.00**	0.98	0.81	0.88
	K-fold 10	n-gram (1,1)	**0.86**	0.90	0.94	0.95	0.85	0.88
		n-gram (1,2)	0.85	0.95	0.94	0.98	0.89	**0.89**
Recall	K-fold 2-Test 20%	n-gram (1,1)	0.84	0.70	0.33	0.82	**0.77**	**0.79**
		n-gram (1,2)	**0.88**	0.59	0.67	**0.84**	0.59	0.77
	K-fold 5	n-gram (1,1)	0.81	**0.74**	**0.79**	0.76	0.74	0.78
		n-gram (1,2)	0.85	0.67	**0.79**	0.75	0.64	0.77
	K-fold 10	n-gram (1,1)	0.83	0.70	0.79	0.76	0.76	0.79
		n-gram (1,2)	0.84	0.70	0.79	0.76	0.72	0.79
F1 Scores	K-fold 2-Test 20%	n-gram (1,1)	0.83	0.78	0.50	0.90	**0.83**	0.82
		n-gram (1,2)	0.84	0.73	0.80	**0.91**	0.68	0.80
	K-fold 5	n-gram (1,1)	0.83	**0.80**	**0.87**	0.85	0.78	0.82
		n-gram (1,2)	**0.85**	0.78	**0.87**	0.85	0.71	0.81
	K-fold 10	n-gram (1,1)	0.84	0.77	0.85	0.84	0.80	0.82
		n-gram (1,2)	0.84	0.80	0.85	0.85	0.79	**0.83**

#Samples Support	K-fold 2-Test 20%	n-gram (1,1)	90	33	3	17	26	169
		n-gram (1,2)	85	32	3	19	27	166
	K-fold 5	n-gram (1,1)	440	148	33	80	129	830
		n-gram (1,2)	440	148	33	80	129	830
	K-fold 10	n-gram (1,1)	440	148	33	80	129	830
		n-gram (1,2)	440	148	33	80	129	830

In addition, our well-trained model from the previous five-class classification is tested on an external dataset as a simulation process. The simulation of input data by the Trello app as in binary classification cannot be conducted for this multiclass multi-label classification, because the size of samples per class is too small. Instead, the simulation of input relies on the same external dataset used in the binary simulation experiment, but only documents with featured requirements are used for the categorization.

The result of this data simulation is shown in Table 4.10. The selected classifier for multiclass categorization performed relatively well with the simulated data with an average precision of 89%, recall of 81% and F1 of 83%, but the performance in Class 1 is lower than the preceding reduced-class experiments. The experiments show that **Linear SVM with K-fold 5 and n-gram (1,1)** has high capacity in categorizing our experimental datasets, especially when the number of classes is small. (See Appendix D, Table D.22 for more detail on the class-reduction experiments on external data simulation.)

Table 4.10 Performance of Linear SVM for Multiclass Multi-label Classification on External Data Simulation

Classifier	Class	Precision	Recall	F1 Scores
Linear SVM, n-gram (1,1), K-fold 5	Class 0	0.86	0.90	0.88
	Class 1	1.00	0.55	0.71
	Class 2	1.00	1.00	1.00
	Class 3	0.57	1.00	0.73
	Class 4	0.91	0.91	0.91
	Average	0.89	0.81	0.83

4.2.3. Summary for the Evaluation of Classification Models

Table 4.11 sums up the best alternative techniques in the evaluation for the binary and multiclass multi-label classifications in the current research.

Table 4.11 Summary of Classification Model Evaluation

Experiment	Top-performed alternative techniques
Binary filtering of feature/non-feature classification	- Machine learning classifier: SVMs with linear variant - Cross-validation approach: 10-fold - Attribute extraction: n-gram (1,4) - Sentiment analysis: sentence + sentiment
Multiclass multi-label classification for feature categorisation by ten categories	- Machine learning classifier: SVMs with linear variant - Cross-validation approach: 10-fold - Attribute extraction: n-gram (1,1)
Multiclass multi-label classification for feature categorisation by five categories	- Machine learning classifier: SVMs with linear variant - Cross-validation approach: 5-fold - Attribute extraction: n-gram (1,1)

4.3. Ranking of Extracted Features

In order to decide which extracted features may be put into the next release, it is useful for the release manager to be able to order the extracted features in priority order, by applying some prioritization models to the features. A simple model orders the extracted the features according to the frequency with which reviewers mention them, with a view that implementing these can satisfy the maximum number of reviewers. The purpose of this process is to sort those featured sentences after the categorization to obtain an overview of app features which reviewers are mostly interested in. In this experiment, we extract key phrases from review sentences and compute scores by counting word frequency, as proposed in Section 3.7. This process extracts key phrases from each category of reviewing text along with their scores sorted in descending order, which offers the ranking of requested features by word frequency. Based on the obtained key-phrase scores, top sentences with feature requirements are identified and ranked. SQL scripts are used to offer the flexibility in information manipulation, such as date range, app name and numbers of featured requirement sentences. Examples of the top-three featured sentences of all experimented data are generated per each category of feature requirements and per app can be found in Appendix D, Tables D.23 and D.24.

The outputs from this process can give developers a brief understanding of features frequently requested by users for the purpose of comparison and prioritization, that can lead to the improvement of certain features and bringing in enthusiasm for the software application over a new release. The feature ranking process is not evaluated in this thesis. It is a pilot study for future feature comparison. The ranked features by category are aimed to link to the actual release of each app, in order that new 'required features' can be indicated and assessed at the end of the stage. More details on this are discussed in Future Work (Section 5.3).

4.4. Related Work

As previously explained in Chapter 2, not many studies have focused on extracting information from general product reviews for software evolution. Our work can, however, be compared to the studies of mobile and desktop apps by similar approaches, as is asserted by the following aspects.

The results of our experiments show that binary classification performs better than multiclass classification. This is perfectly matched with the work by Maalej and Nabil on the classification of mobile app reviews into four categories of bug reports, feature requests, user experience and user rating (Maalej & Nabil, 2015). From our experiments, it can also be said that the classification of data into smaller categories is generally more achievable than into larger categories. It is found that binary classification obtained higher precision, recall and F1-scores, followed by five-class classification and ten-class classification.

Compared to similar studies on multiclass classification, our experiments provide a different result in terms of classification algorithms. In our work, SVMs achieve statistically higher performance (Qi, Zhang, Jeon, & Zhou, 2015; Xiao et al., 2015) over other testing classification algorithms such as Naïve Bayes and Logistic Regression. While Naïve Bayes worked very well in some literatures (Maalej et al., 2016; Maalej & Nabil, 2015; Panichella et al., 2015), they perform very ineffectively on our dataset. This might be because either the proportion of our experimental dataset is quite low, or the quality of our dataset is not sufficient, especially for those data classes with low performance scores.

The size of training and testing samples is directly related to the performance of our classifiers (Xiao et al., 2015). In this thesis, we have tested on different scales of cross-validation, 2-fold with different testing percentages, 5-fold and 10-fold. Our experiments reveal that using all samples for training (in 5-fold and 10-fold cross-validation) is prone to better performance of classification. Even with the small size of dataset, our results confirm the argument very well.

In addition, similar improvement in the trend of performance can be seen when sentiment analysis (as the document metadata) is used as an input in combination with actual text for classification (Maalej et al., 2016; Maalej & Nabil, 2015; Panichella et al., 2015). In our current study, this technique helps improve the performance scores of classifications by roughly 0.1% to 2%. However, the other document metadata like rating might also be used in future experiments of classification to further boost the performance.

There are a number of studies that applied n-gram techniques in the classification process to fine-tune the classifiers (Keertipati et al., 2016; Lee et al., 2016; S. A. Licorish, Lee, et al., 2015; S. Licorish, Tahir, Bosu, & MacDonell, 2015; Maalej et al., 2016). Our experiments show that while the use of n-grams ($1 < n \leq 4$) gains better performance scores in binary classification, multiclass multi-label classifications are more achievable only with unigram ($n = 1$). Applying n-grams where $1 < n \leq 4$ really improves precision scores, but it deters the recall rates in multiclass multi-label classification.

In this thesis, we only experiment on ranking the extracted app features based on word frequency (a term-weighted approach), which is one of prioritization measures proposed in a study by Keertipati et al. (2016). This offers us some basic overview of what reviewers have discussed about the features of the reviewed app over a specified period. Our work in terms of feature prioritization can be extended by using other prioritization measures, for example using a graph-based model like SNA (Lee et al., 2016) to gain more insight to our datasets. Moreover, it would be more beneficial to app developers or any relevant software development entities if our current work can be furthered by comparing the extracted feature requirements to the actual software release notes in an automated way, as exampled in S. Licorish, Tahir, et al. (2015) and Palomba et al. (2015). By this, the solved and new features

can be identified and listed, which thereby would further facilitate the prioritization of the required features for software improvement and evolution in the future release.

Chapter 5

Conclusion

5.1. Conclusion

The goal of this thesis is to support the evolution of software product applications by the identification of potential new product features from online user reviews. These can then be prioritised for inclusion (or not) in the next product release. The online reviews are processed using text processing techniques to extract, classify and rank potential new features. Relevant state-of-the-art studies on text processing techniques and methods for classification, clustering, ranking, comparison and deduplication of software features for the enhancement of mobile and desktop applications are studied. By synthesizing those techniques and models, a prototype is developed and experimented with different settings to select the most suited setup for a classification system to filter, categorize and rank the extracted software elements. Our models are tested on the experimental datasets of two software applications' reviews and an external dataset. The results are evaluated, discussed and compared to the relevant studies.

Regarding our research questions, the following answers (A; A1 to A3) are provided.

RQ: How can modern text processing and machine learning techniques be used to support the extraction of a set of candidate requirements from large volumes of online user product reviews?

A: Based on a review of related literature and relevant text processing techniques, in this research, we have proposed a sequence of text processing and machine learning methods as a process to extract candidate release requirements from online reviews. The feasibility of the process is confirmed by implementing, testing and evaluating it over multiple iterations with different tuning parameters to optimise the performance results. New datasets of user reviews are scraped specifically for the purpose of the thesis experiments. Our experiments are focused on sentence-based granularity. Pre-processing techniques such as sentence tokenization, lowercasing, word tokenization, non-standard word and punctuation removal, stop word removal, short sentence removal, word stemming and lemmatization are applied to our datasets. TF-IDF is used in document vectorization. Different n-gram ranges (1 to 4 n-grams) are tested. Sentence sentiment is also experimented as a co-input for binary classification. Semantic similarity measures are used for sentence de-duplication

before categorization of feature requests. Supervised learning algorithms such as Naïve Bayes, Support Vector Machines and Logistic Regressions are selected for the classification process. Two stages of classification are applied: first, binary classification is used as a de-noise process by filtering sentences with feature requests; and second, multiclass multi-label classification is used for the categorization of extracted features. Both classification stages are evaluated by precision, recall and F-scores, where SVMs showed distinctive results at both stages of classification on average and by class. A term-weighted approach is used to arrange the extracted features by categories, dates and apps which offer the prioritization of features by term-weighted frequency score. The outputs of our proposed approach are a set of extracted product requirements which can be sorted and prioritized by the pre-defined categories over the specified period. These outcomes can provide corresponding product developers or decision makers with information and ideas to improve the software products by each release.

SQ1: What is a suitable text analysis process to achieve the extraction of new features from such a large corpus of noisy reviews?

A1: Our research proposed a semi-supervised approach in sequence as shown in Figure 3.2. The approach starts with research design, data collection, data pre-processing, binary classification as the filtering of feature requests, reduction of duplicate features, multiclass multi-label classification as feature categorisation and feature prioritisation by term-weighted frequency. Two stages of classification are evaluated by precision, recall and F-measures and compared by testing on tuning parameters, n-grams, sentence sentiments and the division of training dataset by cross-validation techniques.

SQ2: What specific text-processing and machine learning techniques are candidates for the proposed feature extraction process?

A2: In this thesis, we rely on the use of different text processing techniques such as word and sentence tokenization, lowercasing, stop word removals, short sentence removal, stemming and lemmatization for pre-processing of raw reviews. Also, sentiment analysis and n-grams are experimented to boost the performances of the proposed models. Sentence semantic similarity measures are used to reduce the redundancy in the dataset. Binary classification is used for the filtering of feature

requests from online comments and multiclass multi-label classification for the process of feature categorization. Machine learning algorithms like Naïve Bayes, SVMs and Logistic Regress are experimented in both classification stages, where SVMs show the outstanding performances at both stages of classification by comparing the rates of precision, recall and F1.

SQ3: Which of the candidate techniques perform the best in terms of Precision, Recall and F-Measure in this context?

A3: In binary filtering, linear SVM with n-gram (1, 4), sentence + sentiment and 10-fold cross-validation is the top-performed model for culling sentences with feature requirements out of those with no feature requirements with a distinctive average precision of 90.9%, recall of 91.1% and F1 of 91.0%. In the data simulation, the model also performs well with the Trello samples (Precision = 90.9%, Recall = 89.7%, F1 = 91.0%) and the external data (Precision = 88.2%, Recall = 87.1%, F1 = 87.0%). For multiclass multi-label classification, Linear SVM with 5-fold and n-gram (1,1) is the exceptionally effective classifier. The precision, recall and F-scores for 5-class classification are 86%, 78% and 82% respectively. On the external data experiment, the performance is also promising (precision = 89%, recall = 81% and F1 = 83%). The proposed semi-automated approach can help extract and categorise the requirements reflected in the product reviews with over 80% effectiveness.

5.2. Limitations and Threats to Validity

Despite high performance measures achieved, our proposed approach has some limitations. The limitations and threats to the validity of our approach are discussed next.

Our first limitation is the quality of the written product reviews can have a significant effect on the learning and classification process. Poorly written reviews can be accidently ignored during the pre-processing, de-noising or classification stages, which resulted in the loss of valuable information on feature requirements for product evolution. Misspelt comments can lead to misclassification of information. Though there exist techniques to auto-correct the misspelled words, machines cannot be well-trained to predict what the writer wanted to convey or to effectively deal with severe misspelling. In addition, in sentence-based processing, punctuations are

very important. For example, the use of full stops should necessarily be precise, not to be confusing in sentence segmentation as well as in attribute extraction.

The application of machine learning classification is the process of prediction. It is difficult to replace human inspection by a wholly automated process. In our approach, the semi-automated classifications can reduce workloads for humans by 80-90%, which can save a lot of time and cost for such operations, but still human labour is required to recheck the work for 100% accurate information.

In our experiments, effectiveness – not efficiency – in both classifications, is our focus. This is because the size of our experimental dataset is relatively small when compared to other studies of text processing. Our classification stages consume a very small operating time using a medium-to-low processing computer. As a result, larger datasets are suggested to further test on our best classifiers, SVMs, for efficiency, which might also require more advanced operating systems and comparisons.

The use of a 10-fold cross-validation technique might obtain overly optimistic results in terms of classification. This is because all the data are used both in the training and testing of the classifier. To address this, the unseen dataset is applied as the simulation of input data on both binary and multiclass classifications and the performance of our classifiers on the dataset is reported.

The last limitation of this current study is the use of a semi-supervised learning approach. Because of this, the classification algorithms require a qualitative training set for better performance results. The generation of a truth dataset for training can be time-consuming and it can be difficult to locate a qualitative truth set for training the classifiers. Additionally, in the learning process, pre-defined labels are necessary. Using fixed labels in classification is quite domain-specific, as the trained classifiers cannot be applied with new labels without additional training sets for those new labelled classes. Furthermore, the generation of a truth set for both training and testing of our classifiers heavily relied on manual work, which makes it difficult to completely avoid bias. Our trial to reduce bias in the creation of truth set data is to provide a guide with definitions and the use of two-pass verification. On the other hand, it might be worthy to further work on clustering algorithms which are more automated. However, this would rather depend on the actual purpose of the

application. Clustering might not offer the expected outputs as does the classification, because it is mostly reliant on natural patterns of data.

5.3. Future work

Beyond the improvement of this current thesis over its limitations, the extension of this work to fulfil the objective of our main project (as briefed in Chapter 1) is ideal.

This thesis is only a small part of this project, which utilizes online reviews of software products to support the enhancement of such apps in terms of providing software features as requested by actual users. The scope of data sources can be expanded to include expert reviews, peer reviews, competitor reviews and internal reviews. Using feedback comments from different perspectives might offer more interesting information in terms of targeted features for software enhancement over the future release. These data sources might be approached by other state-of-the-art methods and more automated techniques to benefit the project implementation and contribute to the field of text processing and analysis.

Furthermore, it is suggested that the extracted information regarding software feature requests from these multiple sources should be integrated and compared to the actual features on the software release notes. By the way, “new” features can be identified to facilitate app developers or software managers in prioritizing certain features worthily required for an enhancement.

To facilitate the reporting of extracted features, it can be useful to visualize the information from the above operations in a graphical platform, for example by using a sliding window. On this sliding window, the information of solved and unsolved features might be illustrated over a scalable period (by week, month, quarter or release), where a slider would be the determinant of viewing time. Some statistics regarding the top features on demand might be a good option illustrated in the window.

By achieving the goal of the project, a complete system to exploit the information from online resources can be developed, which would fully benefit the development and evolution of software products in each new release.

References

Auria, L., & Moro, R. A. (2008). Support vector machines (SVM) as a technique for solvency analysis. Retrieved from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1424949

Bakar, N. H., Kasirun, Z. M., Salleh, N., & Jalab, H. A. (2016). Extracting features from online software reviews to aid requirements reuse. *Applied Soft Computing*, *49*, 1297–1315.

Bilenko, M., & Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 39–48). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=956759

Bongirwar, V. K. (2015). A Survey on Sentence Level Sentiment Analysis. *International Journal of Computer Science Trends and Technology (IJCST)*, *3*, 110–113.

Chen, N., Lin, J., Hoi, S. C. H., Xiao, X., & Zhang, B. (2014). Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *International Conference on Software Engineering*.

Christoph, L. (2015). Measuring semantic similarity and relatedness with distributional and knowledge-based approaches. *Information and Media Technologies*, *10*(3), 493–501.

Ektefa, M., Jabar, M. A., Sidi, F., Memar, S., Ibrahim, H., & Ramli, A. (2011). A threshold-based similarity measure for duplicate detection (pp. 37–41). IEEE. https://doi.org/10.1109/ICOS.2011.6079233

Elmagarmid, A. K., Ipeirotis, P. G., & Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *19*(1). Retrieved from http://ieeexplore.ieee.org/abstract/document/4016511/

Ferro, A., Giugno, R., Puglisi, P. L., & Pulvirenti, A. (2010). An efficient duplicate record detection using q-grams array inverted index. In *International Conference on Data Warehousing and Knowledge Discovery* (pp. 309–323). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-15105-7_25

Freitas, A. A. (2000). Understanding the crucial differences between classification and discovery of association rules: a position paper. *AcM sIGKDD Explorations Newsletter*, *2*(1), 65–69.

Galvis Carreño, L. V., & Winbladh, K. (2013). Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering* (pp. 582–591). IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=2486865

Genc-Nayebi, N., & Abran, A. (2017). A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software*, *125*, 207–219.

Gilbert, C. H. E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. In *Eighth International Conference on*

Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf. Retrieved from http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf

Gong, C., Huang, Y., Cheng, X., & Bai, S. (2008). Detecting near-duplicates in large-scale short text databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 877–883). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-68125-0_87

Gregory, R. W. (2011). Design science research and the grounded theory method: characteristics, differences, and complementary uses. *Theory-Guided Modeling and Empiricism in Information Systems Research*, 111–127.

Guzman, E., & Maalej, W. (2014). How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)* (pp. 153–162). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6912257

Hadžić, \DJulaga, Sarajlić, N., & Malkić, J. (2016). Different similarity measures to identify duplicate records in relational databases. In *Telecommunications Forum (TELFOR), 2016 24th* (pp. 1–4). IEEE. Retrieved from http://ieeexplore.ieee.org/abstract/document/7818899/

Harman, M., Jia, Y., & Zhang, Y. (2012). App store mining and analysis: MSR for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories* (pp. 108–111). IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=2664461

Jiang, W., Ruan, H., & Zhang, L. (2014). Analysis of economic impact of online reviews: an approach for market-driven requirements evolution. In *Requirements Engineering* (pp. 45–59). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-662-43610-3_4

Jiang, W., Ruan, H., Zhang, L., Lew, P., & Jiang, J. (2014). For User-Driven Software Evolution: Requirements Elicitation Derived from Mining Online Reviews. In V. S. Tseng, T. B. Ho, Z.-H. Zhou, A. L. P. Chen, & H.-Y. Kao (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 584–595). Springer International Publishing. Retrieved from http://link.springer.com/chapter/10.1007/978-3-319-06605-9_48

Karamizadeh, S., Abdullah, S. M., Halimi, M., Shayan, J., & Rajabi, M. Javad. (2014). Advantage and drawback of support vector machine functionality (pp. 63–65). IEEE. https://doi.org/10.1109/I4CT.2014.6914146

Keertipati, S., Savarimuthu, B. T. R., & Licorish, S. A. (2016). Approaches for Prioritizing Feature Improvements Extracted from App Reviews. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (p. 33:1–33:6). New York, NY, USA: ACM. https://doi.org/10.1145/2915970.2916003

Khan, A., Baharudin, B., Lee, L. H., & Khan, K. (2010). A review of machine learning algorithms for text-documents classification. *Journal of Advances in Information Technology*, *1*(1), 4–20.

Lee, C. W., Licorish, S. A., Savarimuthu, B. T. R., & MacDonell, S. G. (2016). Augmenting Text Mining Approaches with Social Network Analysis to Understand the Complex Relationships among Users' Requests: A Case Study of the Android Operating System. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (pp. 1144–1153). https://doi.org/10.1109/HICSS.2016.145

Li, M., Wang, H., Li, J., & Gao, H. (2010). Efficient Duplicate Record Detection Based on Similarity Estimation. In L. Chen, C. Tang, J. Yang, & Y. Gao (Eds.), *Web-Age Information Management* (pp. 595–607). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-14246-8_58

Li, Y., McLean, D., Bandar, Z. A., O'Shea, J. D., & Crockett, K. (2006). Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, *18*(8), 1138–1150. https://doi.org/10.1109/TKDE.2006.130

Li, Y., McLean, D., Bandar, Z. A., O'shea, J. D., & Crockett, K. (2006). Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, *18*(8), 1138–1150.

Licorish, S. A., Lee, C. W., Savarimuthu, B. T. R., Patel, P., & MacDonell, S. G. (2015). They'll Know It When They See It: Analyzing Post-Release Feedback from the Android Community. In *Proc 21st Amer Conf. Info. Sys.-AMCIS. Puerto Rico: AISeL. pp. 1* (Vol. 11).

Licorish, S. A., Tahir, A., Bosu, M. F., & MacDonell, S. G. (2015). On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios (pp. 78–87). IEEE. https://doi.org/10.1109/ASWEC.2015.19

Licorish, S., Tahir, A., Bosu, M. F., & MacDonell, S. G. (2015). On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios [Paper presented at a conference, workshop or other event, and published in the proceedings]. Retrieved January 10, 2017, from http://ieeexplore.ieee.org/document/7365796/

Maalej, W., Kurtanović, Z., Nabil, H., & Stanik, C. (2016). On the automatic classification of app reviews. *Requirements Engineering*, *21*(3), 311–331. https://doi.org/10.1007/s00766-016-0251-9

Maalej, W., & Nabil, H. (2015). Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)* (pp. 116–125). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7320414

Martin, W., Sarro, F., Jia, Y., Zhang, Y., & Harman, M. (2016). A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*. Retrieved from http://ieeexplore.ieee.org/abstract/document/7765038/

Pagano, D., & Maalej, W. (2013). User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)* (pp. 125–134). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6636712

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 (pp. 291–300). IEEE. https://doi.org/10.1109/ICSM.2015.7332475

Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C. A., Canfora, G., & Gall, H. C. (2015). How can i improve my app? classifying user reviews for software maintenance and evolution. In *Software maintenance and evolution (ICSME), 2015 IEEE international conference on* (pp. 281–290). IEEE. Retrieved from http://ieeexplore.ieee.org/abstract/document/7332474/

Qi, J., Zhang, Z., Jeon, S., & Zhou, Y. (2015). Mining Customer Requirement from Online Reviews: A Product Improvement Perspective. *Available at SSRN 2637145*. Retrieved from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2637145

Rose, S., Engel, D., Cramer, N., & Cowley, W. (2010). Automatic keyword extraction from individual documents. *Text Mining*, 1–20.

Shah, F., & Pfahl, D. (2016). Evaluating and Improving Software Quality Using Text Analysis Techniques-A Mapping Study. In *REFSQ Workshops*. Retrieved from http://ceur-ws.org/Vol-1564/paper21.pdf

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), 427–437.

Stoltzfus, J. C. (2011). Logistic Regression: A Brief Primer. *Academic Emergency Medicine*, *18*(10), 1099–1104. https://doi.org/10.1111/j.1553-2712.2011.01185.x

Xiao, M., Yin, G., Wang, T., Yang, C., & Chen, M. (2015). Requirement Acquisition from Social Q&A Sites. In L. Liu & M. Aoyama (Eds.), *Requirements Engineering in the Big Data Era* (pp. 64–74). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-662-48634-4_5

Yu, Y., Wang, H., Yin, G., & Liu, B. (2013). Mining and recommending software features across multiple web repositories. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware* (p. 9). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=2532453

Appendices

Appendix A Annotation Guideline

This guideline presents a description of this research implementation as per the provided source files. The source files of this research experiments have been made available at https://github.com/Noi-Phonephasouk/masters-c.The scripts are developed in Python 2.7 with the requirements of the following Python modules (as in requirements.txt).

Table A.1 Required Python Modules for this Research

Modules	Version	Purpose
beautifulsoup	4 4.5.1	Data scraping
cfscrape	1.6.8	
requests	2.12.4	
selenium	3.0.2	
Nltk	3.2.1	Text processing and analysis
pandas	0.18.1	
scikit-learn	0.17.1	
numpy	1.11.1	
vaderSentiment	0.5	
textblob	0.11.1	
rake-nltk	1.0.0	

For the implementation of this research, the following steps has been practised.

1. Operate manual scraping of user reviews (of Jira and Trello) from of the targeted web site by the scripts ‘web_scraper.py’ (Figure A.1).

```
import re
import requests
from selenium import webdriver
from datetime import datetime
import time


#input the url (1)
browser.get("https://www.g2crowd.com/products/jira/reviews")
link = browser.find_elements_by_partial_link_text("Last")
for item in link:
    print item.get_attribute("href")
    total_pages = int(re.findall("\d+", item.get_attribute("href"))[1])
print "Retrieving reviews from ", total_pages, " pages..."

scraper = cfscrape.CloudflareScraper()


conn = sqlite3.connect('ureviewdata.sqlite')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS Comments
    (cid INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, uid INTEGER NOT NULL, time TEX
     title TEXT, review TEXT, UNIQUE (uid, date, title, review))''')
urunning = 1

for i in range(total_pages):
    #input the url (2)
    url = "https://www.g2crowd.com/products/jira/reviews?page={}".format(i+1)
    r = scraper.get(url).content
    data = r
    soup = BeautifulSoup(data, "lxml")
    cur.execute('''INSERT OR IGNORE INTO Webs (url) VALUES ( ? )''', ( url, ))
    cur.execute('SELECT wid FROM Webs WHERE url = ? ', (url, ))
    web_id = cur.fetchone()[0]
```

Figure A.1 Sample Scripts for Data Scraping in ‘web_scraper.py’

The scraped data are stored in the SQLite database, ‘ureviewdata.sqlite’. (See Appendix B for more detail on the experimental dataset.)

2. The retrieved raw reviews are then tokenized into sentences by ‘sent_tokenize.py’. The processed data are stored in the database.

```
import sqlite3
from nltk.tokenize import sent_tokenize, word_tokenize, RegexpTokenizer
from nltk.tokenize.punkt import PunktParameters
import nltk
import csv
import re
conn = sqlite3.connect('ureviewdata.sqlite')
curs = conn.cursor()
curs.execute('''CREATE TABLE IF NOT EXISTS Sent_breakdown
    (sentid INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, cid INTEGER NOT NULL,
    review_sent TEXT NOT NULL, dup_status INTEGER, truebiclassid INTEGER,
    predict_biclassid INTEGER, UNIQUE (cid, review_sent) )''')

cur = curs.execute('''SELECT review, cid FROM Comments ORDER BY cid''')
rows = cur.fetchall()

count =0
countsent =0
punkt_param = PunktParameters()
abbr = ['e.g', 'i.e','1','2','3','4','5','6','7', 'inc', 'smth']
punkt_param.abbrev_types=set(abbr)
ureview_txt_ls=[]

tokenizer=nltk.tokenize.punkt.PunktSentenceTokenizer(punkt_param)

for row in rows:
    ureview_txt=row[0]
    ureview_txt = re.sub(r"(etc.(?=\W+[a-z]))",r"etc",ureview_txt)

    all_sentence_lst = tokenizer.tokenize(ureview_txt) # sentence tokenize
    for sent in all_sentence_lst:
        sentn= re.sub(r"^\d+\W\s" , "", sent)
        sentn = remove_puncz(sentn)
        sentn = sentn.lower()
        sentn = contractions_replace(sentn)
```

Figure A.2 Sample Scripts for Sentence Tokenization

3. From the tokenized feedback sentences, truth sets for binary and multiclass multi-label classifications are generated by hand (‘truthset_filter.csv’ and ‘truthset_categorize.csv’ respectively). Truth set data are recorded in the database by ‘rec_trueset.py’ and ‘rec_trueset2.py’ for both classifications accordingly for evaluation purpose.
4. Next, it is the stages of sentence pre-processing and binary classification to filter the sentences with feature requirements by the script files: ‘binary_filter.py’ and binary_filter_kfold.py’. Because K-fold cross validation ($k > 2$) requires multiple iterations of operations and the overall performance is from an average of each loop, the different script file has been programmed. The simulation of input data has been conducted using the same files.

```
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1,4))), # ngram_range=(1,4)
    ('tfidf_transformer',  TfidfTransformer()),
    ('classifier', LinearSVC(C=1000000.0, class_weight=None, dual=True, fit_intercept=True,
    random_state=None, tol=0.0001, verbose=1))
    ])
k = 10

k_fold = KFold(n = len(data), n_folds = k)

for train_indices, test_indices in k_fold:
    print datetime.datetime.now()
    train_text = data.iloc[train_indices]['can_sent'].values
    train_y = data.iloc[train_indices]['class'].values

    test_text = data.iloc[test_indices]['can_sent'].values
    test_y = data.iloc[test_indices]['class'].values

    pipeline.fit(train_text, train_y)
    predictions = pipeline.predict(test_text)

    nloop += 1
    print "Loop: ", (nloop)
    for i, j in enumerate(test_indices):
        curs.execute('''UPDATE Sent_breakdown SET predict_biclassid = ? WHERE sentid = ? '

    confusion += confusion_matrix(test_y, predictions)
    precision = precision_score(test_y, predictions)
    recall = recall_score(test_y, predictions)
    score = f1_score(test_y, predictions)

    print classification_report(test_y, predictions)
    scores.append(score)
    precisions.append(precision)
    recalls.append(recall)
```

Figure A.3 Sample Scripts for Binary Classification using 10-fold Validation, Linear SVM, n-gram (1,4)

5. Then the filtered sentences are grouped by user based on semantics between sentences (sent_dedup.py) to reduce the number of duplicate feature requirements.

```
def semantic_similarity(sentence_1, sentence_2, info_content_norm):
    """
    Computes the semantic similarity between two sentences as the cosine
    similarity between the semantic vectors computed for each sentence.
    """
    words_1 = nltk.word_tokenize(sentence_1)
    words_2 = nltk.word_tokenize(sentence_2)
    joint_words = set(words_1).union(set(words_2))
    vec_1 = semantic_vector(words_1, joint_words, info_content_norm)
    vec_2 = semantic_vector(words_2, joint_words, info_content_norm)
    return np.dot(vec_1, vec_2.T) / (np.linalg.norm(vec_1) * np.linalg.norm(vec_2))
def similarity(sentence_1, sentence_2, info_content_norm):
    """
    Calculate the semantic similarity between two sentences. The last
    parameter is True or False depending on whether information content
    normalization is desired or not.
    """
    return DELTA * semantic_similarity(sentence_1, sentence_2, info_content_norm) + \
        (1.0 - DELTA) * word_order_similarity(sentence_1, sentence_2)
for i,j in enumerate(userid_ulst):

    cur_g = curs.execute('''SELECT comments.uid, sent_breakdown.sentid, sent_breakdown.rev
    truebiclassid = 1 AND sent_breakdown.cid = comments.cid AND sent_breakdown.cid = ?''',
    rows_g = cur_g.fetchall()
    for row in rows_g:
        minelist.append({'sentid':row[1], 'review_sent':row[2]})

    for k in range(len(minelist)):
        for l in range(k + 1, len(minelist)):
            sim_score = similarity(str(feature_cull(minelist[k]['review_sent'])), str(feat

            if sim_score >= 0.5:
                if minelist[k]['sentid'] not in sent_group_id:
                    sent_group_id.append(int(minelist[k]['sentid']))
                if minelist[l]['sentid'] not in sent_group_id:
```

Figure A.4 Sample Scripts for Duplication Reduction by Sentence Semantic Similarity

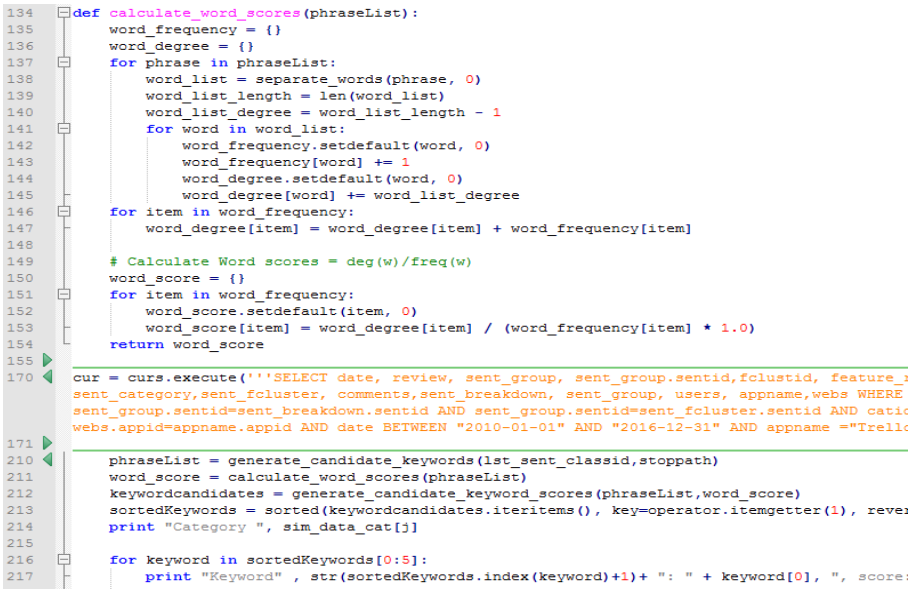
6. After the sentence redundancy has been reduced, multiclass multi-label classification is performed by ‘class_categorize.py’ and ‘class_categorize_kfold.py’. Here, separated files are used for 2-fold and k-

fold validation of $k > 2$. The experiments on the reduction of class number are conducted on 'class_categorize_reduce.py' and 'class_categorize_reduce_kfold.py'; and the file 'reduce_class_categorize.csv' is used for evaluation instead of database. The simulation of external input (categorize_simulation.csv and reduce_class_simulation.csv) has been conducted using the same files.

```
    ('vectorizer', CountVectorizer(ngram_range=(1,1))), # ngram_range=(1,4)
    ('tfidf_transformer',  TfidfTransformer()),
    ('classifier', OneVsRestClassifier(LinearSVC(C=1000000.0, class_weight=None, dual=True,
    'ovr', penalty='l2', random_state=None, tol=0.0001, verbose=1)))
    ])

Y = MultiLabelBinarizer().fit_transform(data['classid'].values)
pipeline.fit(data['review_sent'].values, Y)

testsze = 0.20
print "Using", testsze*100.0,"% data for testing...\n"

train_text, test_text, train_y, test_y, idx_train, idx_test = train_test_split(data['reviev

pipeline.fit(train_text, train_y)
predictions = pipeline.predict(test_text)

train_test_precision = precision_score(test_y, predictions)
train_test_recall = recall_score(test_y, predictions)
train_test_F1 = f1_score(test_y, predictions)

print
print "Classification Report "
print classification_report(test_y, predictions)
print
print "train_test_precision:", train_test_precision
print "train_test_recall:", train_test_recall
print "train_test_F1:", train_test_F1

curs.execute('''UPDATE Sent_categorize SET predict_classid = Null ''')

for i, j in enumerate(idx_test):
    for k,l in enumerate(predictions[i]):
        if predictions[i][k] == 1:
            curs.execute('''UPDATE Sent_categorize SET predict_classid = ? WHERE sentid =?
```

Figure A.5 Sample Scripts for Multiclass Multi-label Classification using 2-fold Validation, 20% Testing Samples, Linear SVM, n-gram (1,1)

7. Finally, the ranking of feature requirements by user based on term weighted frequency has been practiced in 'rake_ranking.py'.



```
def calculate_word_scores(phraseList):
    word_frequency = {}
    word_degree = {}
    for phrase in phraseList:
        word_list = separate_words(phrase, 0)
        word_list_length = len(word_list)
        word_list_degree = word_list_length - 1
        for word in word_list:
            word_frequency.setdefault(word, 0)
            word_frequency[word] += 1
            word_degree.setdefault(word, 0)
            word_degree[word] += word_list_degree
    for item in word_frequency:
        word_degree[item] = word_degree[item] + word_frequency[item]

    # Calculate Word scores = deg(w)/freq(w)
    word_score = {}
    for item in word_frequency:
        word_score.setdefault(item, 0)
        word_score[item] = word_degree[item] / (word_frequency[item] * 1.0)
    return word_score

cur = curs.execute('''SELECT date, review, sent_group, sent_group.sentid,fclustid, feature_
sent_category,sent_fcluster, comments,sent_breakdown, sent_group, users, appname,webs WHERE
sent_group.sentid=sent_breakdown.sentid AND sent_group.sentid=sent_fcluster.sentid AND catic
webs.appid=appname.appid AND date BETWEEN "2010-01-01" AND "2016-12-31" AND appname ="Trell

    phraseList = generate_candidate_keywords(lst_sent_classid,stoppath)
    word_score = calculate_word_scores(phraseList)
    keywordcandidates = generate_candidate_keyword_scores(phraseList,word_score)
    sortedKeywords = sorted(keywordcandidates.iteritems(), key=operator.itemgetter(1), reve
    print "Category ", sim_data_cat[j]

    for keyword in sortedKeywords[0:5]:
        print "Keyword" , str(sortedKeywords.index(keyword)+1)+ ": " + keyword[0], ", score
```

Figure A.6 Sample Scripts for Ranking Feature Requirements by Term-weighted Frequency using RAKE

Appendix B Sample Data

Table B.1 Summary of Review Sentence and Truth Set for Binary Classification

App	Total Review	Total Review Sentence	Sentence with Non-Feature Requirements (Class = 0)	Sentence with Feature Requirements (Class = 1)	Sentence Set with Feature Requirements (Class = 0)	Sentence Set with Feature Requirements (Class = 1)
Jira	603	7,202	6,500	702	3,778	702
Trello	404	5,292	4,823	469	2,930	467
Total	**1,007**	**12,494**	**11,323**	**1,171**	**6,708**	**1,169**

Table B.2 Summary of Truth Set Data for Multiclass Multi-label Classification

Class	Category	Sentence			
		Jira	Trello	Total	Proportion
0	Board management	221	191	412	33.0%
1	User management and permission	32	16	48	3.8%
2	Customization	51	13	64	5.1%
3	Infrastructural design	67	54	121	9.7%
4	Configuration and maintenance	53	0	53	4.3%
5	Data management, visualization and integration	74	60	134	10.7%
6	Mobile services	23	10	33	2.6%
7	Notifications	47	28	75	6.0%
8	Navigation and search	88	28	116	9.3%
9	Others	118	75	193	15.5%
	Total	**774**	**475**	**1,249**	**100%**

	cid	uid	time	date	title	review
	Filter	Filter	Filter	Filter	Filter	Filter
1	295	295	2016-12-19	2016-12-19	Good Ticket Management tool	What do you like best? Custom workflows and dashboards. Bugs and defect management. Requirements management. It is easy to use and manage as well. What do you dislike? For one thing, a lot of the functions some teams might consider essential to their workflow are not included in the basic JIRA product; they require add-ons. Knowledge base management requires the Confluence add-on; code repositories require Stash; agile develop...
2	336	336	2016-12-19	2016-12-19	Jira Review	What do you like best? Very easy release organization and tracking. Love the custom fields that allow for flexibility with team organization and structure. Also great bulk editing options. What do you dislike? Emails are overwhelming. Somewhat hard to effectively utilize all of the features. What business problems are you solving? What benefits have you realized? Using it for Product Management and and release coordination.
3	369	369	2016-12-19	2016-12-19	Fully Customizable, for better or worse...	What do you like best? There is endless configuration possibilietes, from the fields on an issue(ticket) to the workflow status and settings. The JIRA Rest API, is well features, and very well documented. What do you dislike? It can take a while to get all of that set up, though it is not overly complicated. There can be setup paralsys at times. As a developer, the development environment setup ca...
4	400	400	2016-12-19	2016-12-19	Great Platform Onced Configured	What do you like best? Rich Feature Set and Customization and Plug-ins. What do you dislike? Can be complex to configure and hard to find where and how to configure. Recommendations to others considering the product. Spend the time to understand what you want out

Figure B.1 Sample Data in 'Sent_breakdown' Attribute

wid	url
Filter	Filter
1	https://www.g2crowd.com/products/jira/reviews?page=1
2	https://www.g2crowd.com/products/jira/reviews?page=2
3	https://www.g2crowd.com/products/jira/reviews?page=3
4	https://www.g2crowd.com/products/jira/reviews?page=4
5	https://www.g2crowd.com/products/jira/reviews?page=5
6	https://www.g2crowd.com/products/jira/reviews?page=6
7	https://www.g2crowd.com/products/jira/reviews?page=7
8	https://www.g2crowd.com/products/jira/reviews?page=8
9	https://www.g2crowd.com/products/jira/reviews?page=9

Figure B.2 Sample Data in 'Webs' Attribute

uid	wid	username	rating
Filter	Filter	Filter	Filter
1	1	sai jayanth p.	9
2	1	Sebastian P.	10
3	1	Jyotirmoy K.	9
4	1	Paul B.	8
5	1	Jason B.	9
6	1	Deborah D.	7
7	2	Benjamin G.	9
8	2	Chad C.	10
9	2	Chrissy B.	3
10	2	Stanislav B.	9

Figure B.3 Sample Data in 'Users' Attribute

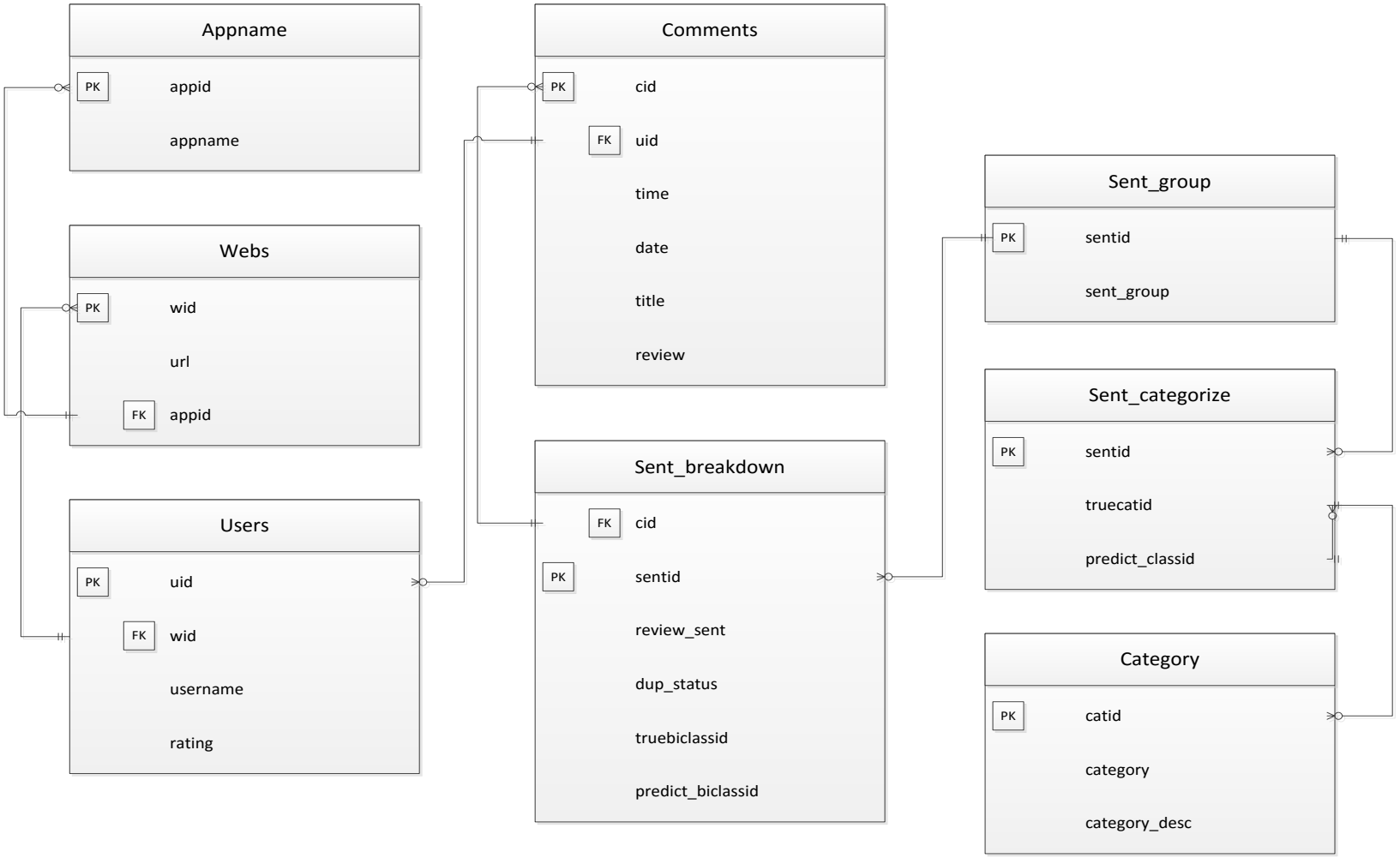


Figure B.4 Database Attributes and Relationship

Appendix C Stop Word List

The following is the list of stop words used for binary classification:

‘a’, ‘about’, ‘above’, ‘across’, ‘after’, ‘afterwards’, ‘again’, ‘against’, ‘all’, ‘almost’, ‘alone’, ‘along’, ‘already’, ‘also’, ‘although’, ‘am’, ‘among’, ‘amongst’, ‘amount’, ‘an’, ‘and’, ‘another’, ‘anyhow’, ‘anyone’, ‘anything’, ‘anyway’, ‘anywhere’, ‘are’, ‘around’, ‘as’, ‘at’, ‘back’, ‘be’, ‘because’, ‘been’, ‘before’, ‘beforehand’, ‘behind’, ‘being’, ‘below’, ‘beside’, ‘besides’, ‘between’, ‘beyond’, ‘bill’, ‘both’, ‘bottom’, ‘but’, ‘by’, ‘call’, ‘co’, ‘con’, ‘cry’, ‘de’, ‘describe’, ‘detail’, ‘due’, ‘during’, ‘each’, ‘eg’, ‘eight’, ‘either’, ‘eleven’, ‘else’, ‘elsewhere’, ‘empty’, ‘etc’, ‘ever’, ‘every’, ‘everyone’, ‘everything’, ‘everywhere’, ‘few’, ‘fifteen’, ‘fifty’, ‘fill’, ‘fire’, ‘first’, ‘five’, ‘for’, ‘former’, ‘formerly’, ‘forty’, ‘four’, ‘from’, ‘front’, ‘full’, ‘further’, ‘go’, ‘he’, ‘hence’, ‘her’, ‘here’, ‘hereafter’, ‘hereby’, ‘herein’, ‘hereupon’, ‘hers’, ‘herself’, ‘him’, ‘himself’, ‘his’, ‘how’, ‘however’, ‘hundred’, ‘i’, ‘ie’, ‘if’, ‘in’, ‘inc’, ‘indeed’, ‘interest’, ‘into’, ‘is’, ‘it’, ‘its’, ‘itself’, ‘jira’, ‘keep’, ‘last’, ‘latter’, ‘latterly’, ‘least’, ‘ltd’, ‘me’, ‘meanwhile’, ‘mill’, ‘mine’, ‘moreover’, ‘most’, ‘mostly’, ‘my’, ‘myself’, ‘name’, ‘namely’, ‘neither’, ‘nevertheless’, ‘next’, ‘nine’, ‘nobody’, ‘noon’, ‘nor’, ‘nothing’, ‘now’, ‘nowhere’, ‘of’, ‘often’, ‘on’, ‘once’, ‘one’, ‘only’, ‘onto’, ‘or’, ‘other’, ‘others’, ‘otherwise’, ‘our’, ‘ours’, ‘ourselves’, ‘out’, ‘over’, ‘own’, ‘part’, ‘per’, ‘perhaps’, ‘put’, ‘rather’, ‘re’, ‘same’, ‘several’, ‘she’, ‘side’, ‘since’, ‘sincere’, ‘six’, ‘sixty’, ‘some’, ‘somehow’, ‘someone’, ‘something’, ‘sometime’, ‘sometimes’, ‘somewhere’, ‘such’, ‘system’, ‘ten’, ‘that’, ‘the’, ‘their’, ‘them’, ‘themselves’, ‘then’, ‘thence’, ‘there’, ‘thereafter’, ‘thereby’, ‘therefore’, ‘therein’, ‘thereupon’, ‘these’, ‘they’, ‘thick’, ‘thin’, ‘third’, ‘this’, ‘those’, ‘though’, ‘three’, ‘through’, ‘throughout’, ‘thru’, ‘thus’, ‘to’, ‘together’, ‘top’, ‘toward’, ‘towards’, ‘trello’, ‘twelve’, ‘twenty’, ‘two’, ‘under’, ‘until’, ‘up’, ‘upon’, ‘us’, ‘via’, ‘was’, ‘we’, ‘were’, ‘what’, ‘whatever’, ‘when’, ‘whence’, ‘whenever’, ‘where’, ‘whereafter’, ‘whereas’, ‘whereby’, ‘wherein’, ‘whereupon’, ‘wherever’, ‘whether’, ‘which’, ‘while’, ‘whither’, ‘who’, ‘whoever’, ‘whole’, ‘whom’, ‘whose’, ‘why’, ‘within’, ‘you’, ‘your’, ‘yours’, ‘yourself’, ‘yourselves’.

The following is the stop words for multiclass multi-label classification:

‘a’, ‘about’, ‘above’, ‘across’, ‘after’, ‘afterwards’, ‘again’, ‘against’, ‘all’, ‘almost’, ‘alone’, ‘along’, ‘already’, ‘also’, ‘although’, ‘always’, ‘am’, ‘among’, ‘amongst’, ‘amoungst’, ‘amount’, ‘an’, ‘and’, ‘another’, ‘any’, ‘anyhow’, ‘anyone’, ‘anything’, ‘anyway’, ‘anywhere’, ‘are’, ‘around’, ‘as’, ‘at’, ‘back’, ‘be’, ‘became’, ‘because’, ‘become’, ‘becomes’, ‘becoming’, ‘been’, ‘before’, ‘beforehand’, ‘behind’, ‘being’, ‘below’, ‘beside’, ‘besides’, ‘between’, ‘beyond’, ‘bill’, ‘both’, ‘bottom’, ‘but’, ‘by’, ‘call’, ‘can’, ‘cannot’, ‘cant’, ‘co’, ‘con’, ‘could’, ‘couldnt’, ‘cry’, ‘de’, ‘describe’, ‘detail’, ‘do’, ‘done’, ‘down’, ‘due’, ‘during’, ‘each’, ‘eg’, ‘eight’, ‘either’, ‘eleven’, ‘else’, ‘elsewhere’, ‘empty’, ‘enough’, ‘etc’, ‘even’, ‘ever’, ‘every’, ‘everyone’, ‘everything’, ‘everywhere’, ‘except’, ‘few’, ‘fifteen’, ‘fify’, ‘fill’, ‘find’, ‘fire’, ‘first’, ‘five’, ‘for’, ‘former’, ‘formerly’, ‘forty’, ‘found’, ‘four’, ‘from’, ‘front’, ‘full’, ‘further’, ‘get’, ‘give’, ‘go’, ‘had’, ‘has’, ‘hasnt’, ‘have’, ‘he’, ‘hence’, ‘her’, ‘here’, ‘hereafter’, ‘hereby’, ‘herein’, ‘hereupon’, ‘hers’, ‘herself’, ‘him’, ‘himself’, ‘his’, ‘how’, ‘however’, ‘hundred’, ‘i’, ‘ie’, ‘if’, ‘in’, ‘inc’, ‘indeed’, ‘interest’, ‘into’, ‘is’, ‘it’, ‘its’, ‘itself’, ‘jira’, ‘keep’, ‘last’, ‘latter’, ‘latterly’, ‘least’, ‘less’, ‘ltd’, ‘made’, ‘many’, ‘may’, ‘me’, ‘meanwhile’, ‘might’, ‘mill’, ‘mine’, ‘more’, ‘moreover’, ‘most’, ‘mostly’, ‘move’, ‘much’, ‘must’, ‘my’, ‘myself’, ‘name’, ‘namely’, ‘neither’, ‘never’, ‘nevertheless’, ‘next’, ‘nine’, ‘no’, ‘nobody’, ‘none’, ‘noone’, ‘nor’, ‘not’, ‘nothing’, ‘now’, ‘nowhere’, ‘of’, ‘off’, ‘often’, ‘on’, ‘once’, ‘one’, ‘only’, ‘onto’, ‘or’, ‘other’, ‘others’, ‘otherwise’, ‘our’, ‘ours’, ‘ourselves’, ‘out’, ‘over’, ‘own’, ‘part’, ‘per’, ‘perhaps’, ‘please’, ‘put’, ‘rather’, ‘re’, ‘same’, ‘see’, ‘seem’, ‘seemed’, ‘seeming’, ‘seems’, ‘serious’, ‘several’, ‘she’, ‘should’, ‘show’, ‘side’, ‘since’, ‘sincere’, ‘six’, ‘sixty’, ‘so’, ‘some’, ‘somehow’, ‘someone’, ‘something’, ‘sometime’, ‘sometimes’, ‘somewhere’, ‘still’, ‘such’, ‘system’, ‘take’, ‘ten’, ‘than’, ‘that’, ‘the’, ‘their’, ‘them’, ‘themselves’, ‘then’, ‘thence’, ‘there’, ‘thereafter’, ‘thereby’, ‘therefore’, ‘therein’, ‘thereupon’, ‘these’, ‘they’, ‘thick’, ‘thin’, ‘third’, ‘this’, ‘those’, ‘though’, ‘three’, ‘through’, ‘throughout’, ‘thru’, ‘thus’, ‘to’, ‘together’, ‘too’, ‘top’, ‘toward’, ‘towards’, ‘trello’, ‘twelve’, ‘twenty’, ‘two’, ‘un’, ‘under’, ‘until’, ‘up’, ‘upon’, ‘us’, ‘very’, ‘via’, ‘was’, ‘we’, ‘well’, ‘were’, ‘what’, ‘whatever’, ‘when’, ‘whence’, ‘whenever’, ‘where’, ‘whereafter’, ‘whereas’, ‘whereby’, ‘wherein’, ‘whereupon’, ‘wherever’, ‘whether’, ‘which’, ‘while’, ‘whither’, ‘who’, ‘whoever’, ‘whole’, ‘whom’, ‘whose’, ‘why’, ‘will’, ‘with’, ‘within’, ‘without’, ‘would’, ‘yet’, ‘you’, ‘your’,

‘yours’, ‘yourself’, ‘yourselves’.

Stop word list used in the experiments were put in the Sk-learn library through the file path ‘.\Python27\Lib\site-packages\sklearn\feature_extraction’ and in the file name ‘stop_words.py’.

The following is the stop word list that comes with RAKE for raking required features:

'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', 'afterwards', 'again', 'against', ‘ain't’, 'all', 'allow', 'allows', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst', 'an', 'and', 'another', 'any', 'anybody', 'anyhow', 'anyone', 'anything', 'anyway', 'anyways', 'anywhere', 'apart', 'appear', 'appreciate', 'appropriate', 'are', ‘aren't’, 'around', 'as', ‘a's’, 'aside', 'ask', 'asking', 'associated', 'at', 'available', 'away', 'awfully', 'b', 'be', 'became', 'because', 'become', 'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being', 'believe', 'below', 'beside', 'besides', 'best', 'better', 'between', 'beyond', 'bit', 'both', 'brief', 'but', 'by', 'c', 'came', 'can', 'cannot', 'cant', ‘can't’, 'cause', 'causes', 'certain', 'certainly', 'changes', 'clearly', ‘c'mon’, 'co', 'com', 'come', 'comes', 'concerning', 'consequently', 'consider', 'considering', 'contain', 'containing', 'contains', 'corresponding', 'could', ‘couldn't’, 'course', ‘c's’, 'currently', 'd', 'definitely', 'described', 'despite', 'did', ‘didn't’, 'different', 'do', 'does', ‘doesn't’, 'doing', 'done', ‘don't’, 'down', 'downwards', 'during', 'e', 'each', 'edu', 'eg', 'eight', 'either', 'else', 'elsewhere', 'enough', 'entirely', 'especially', 'et', 'etc', 'even', 'ever', 'every', 'everybody', 'everyone', 'everything', 'everywhere', 'ex', 'exactly', 'example', 'except', 'f', 'far', 'few', 'fifth', 'first', 'five', 'followed', 'following', 'follows', 'for', 'former', 'formerly', 'forth', 'four', 'from', 'further', 'furthermore', 'g', 'general', 'get', 'gets', 'getting', 'given', 'gives', 'go', 'goes', 'going', 'gone', 'got', 'gotten', 'greetings', 'h', 'had', ‘hadn't’, 'happens', 'hardly', 'has', ‘hasn't’, 'have', ‘haven't’, 'having', 'he', 'hello', 'help', 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein', ‘here's’, 'hereupon', 'hers', 'herself', ‘he's’, 'hi', 'him', 'himself', 'his', 'hither', 'hopefully', 'how', 'howbeit', 'however', 'i', ‘i'd’, 'ie', 'if', 'ignored', ‘i'll’, ‘i'm’, 'immediate', 'in', 'inasmuch', 'inc', 'include', 'indeed', 'indicate', 'indicated', 'indicates', 'inner', 'insofar', 'instead', 'into', 'inward', 'is', ‘isn't’, 'it', ‘it'd’, ‘it'll’, 'its', ‘it's’, 'itself', ‘i've’, 'j', 'jira', 'just', 'k', 'keep', 'keeps', 'kept', 'know', 'known', 'knows', 'l', 'last', 'lately', 'later', 'latter', 'latterly', 'least', 'less', 'lest', 'let', ‘let's’, 'like', 'liked', 'likely',

'little', 'look', 'looking', 'looks', 'ltd', 'm', 'mainly', 'many', 'may', 'maybe', 'me', 'mean', 'meanwhile', 'merely', 'might', 'more', 'moreover', 'most', 'mostly', 'much', 'must', 'my', 'myself', 'n', 'name', 'namely', 'nd', 'near', 'nearly', 'necessary', 'need', 'needs', 'neither', 'never', 'nevertheless', 'new', 'next', 'nine', 'no', 'nobody', 'non', 'none', 'noone', 'nor', 'normally', 'not', 'nothing', 'novel', 'now', 'nowhere', 'o', 'obviously', 'of', 'off', 'often', 'oh', 'ok', 'okay', 'old', 'on', 'once', 'one', 'ones', 'only', 'onto', 'or', 'other', 'others', 'otherwise', 'ought', 'our', 'ours', 'ourselves', 'out', 'outside', 'over', 'overall', 'own', 'p', 'particular', 'particularly', 'people', 'per', 'perhaps', 'placed', 'please', 'plus', 'possible', 'presumably', 'probably', 'provides', 'q', 'que', 'quite', 'qv', 'r', 'rather', 'rd', 're', 'really', 'reasonably', 'regarding', 'regardless', 'regards', 'relatively', 'respectively', 'right', 's', 'said', 'same', 'saw', 'say', 'saying', 'says', 'second', 'secondly', 'see', 'seeing', 'seem', 'seemed', 'seeming', 'seems', 'seen', 'self', 'selves', 'sensible', 'sent', 'serious', 'seriously', 'seven', 'several', 'shall', 'she', 'should', ‘shouldn't’, 'since', 'six', 'so', 'some', 'somebody', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhat', 'somewhere', 'soon', 'sorry', 'specified', 'specify', 'specifying', 'still', 'sub', 'such', 'sup', 'sure', 't', 'take', 'taken', 'tell', 'tends', 'th', 'than', 'thank', 'thanks', 'thanx', 'that', 'thats', ‘that's’, 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby', 'therefore', 'therein', 'theres', ‘there's’, 'thereupon', 'these', 'they', ‘they'd’, ‘they'll’, ‘they're’, ‘they've’, 'think', 'third', 'this', 'thorough', 'thoroughly', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'took', 'toward', 'towards', 'trello', 'tried', 'tries', 'truly', 'try', 'trying', ‘t's’, 'twice', 'two', 'u', 'un', 'under', 'unfortunately', 'unless', 'unlikely', 'until', 'unto', 'up', 'upon', 'us', 'use', 'used', 'useful', 'uses', 'using', 'usually', 'uucp', 'v', 'value', 'various', 'very', 'via', 'viz', 'vs', 'w', 'want', 'wants', 'was', ‘wasn't’, 'way', 'we', ‘we'd’, 'welcome', 'well', ‘we'll’, 'went', 'were', ‘we're’, ‘weren't’, ‘we've’, 'what', 'whatever', ‘what's’, 'when', 'whence', 'whenever', 'where', 'whereafter', 'whereas', 'whereby', 'wherein', ‘where's’, 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', ‘who's’, 'whose', 'why', 'will', 'willing', 'wish', 'with', 'within', 'without', 'wonder', ‘won't’, 'would', ‘wouldn't’, 'x', 'y', 'yes', 'yet', 'you', ‘you'd’, ‘you'll’, 'your', ‘you're’, 'yours', 'yourself', 'yourselves', ‘you've’, 'z', 'zero'.

Appendix D Results

Table D.1 Average Precision, Recall, F1-measures by Linear SVM for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision	Recall	F1	# Sample
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.858	0.856	0.857	5199
		n-gram (1,2)	0.891	0.896	0.893	5199
		n-gram (1,3)	0.890	0.896	0.892	5199
		n-gram (1,4)	0.895	0.902	0.897	5199
	Sentence + Sentiment	n-gram (1,1)	0.861	0.859	0.860	5199
		n-gram (1,2)	0.891	0.899	0.892	5199
		n-gram (1,3)	0.894	0.901	0.896	5199
		n-gram (1,4)	0.891	0.899	0.893	5199
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.857	0.858	0.858	3939
		n-gram (1,2)	0.891	0.896	0.893	3939
		n-gram (1,3)	0.901	0.906	0.902	3939
		n-gram (1,4)	0.900	0.905	0.902	3939
	Sentence + Sentiment	n-gram (1,1)	0.860	0.863	0.861	3939
		n-gram (1,2)	0.898	0.903	0.900	3939
		n-gram (1,3)	0.902	0.906	0.903	3939
		n-gram (1,4)	0.901	0.904	0.902	3939
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.868	0.865	0.866	2600
		n-gram (1,2)	0.902	0.907	0.903	2600
		n-gram (1,3)	0.891	0.898	0.892	2600
		n-gram (1,4)	0.895	0.898	0.896	2600
	Sentence + Sentiment	n-gram (1,1)	0.868	0.870	0.869	2600
		n-gram (1,2)	0.908	0.913	0.910	2600
		n-gram (1,3)	**0.911**	**0.915**	**0.912**	2600
		n-gram (1,4)	0.903	0.906	0.904	2600
K-fold 5	Sentence only	n-gram (1,1)	0.857	0.851	0.854	7877
		n-gram (1,2)	0.897	0.901	0.898	7877
		n-gram (1,3)	0.903	0.906	0.904	7877
		n-gram (1,4)	0.903	0.905	0.904	7877
	Sentence + Sentiment	n-gram (1,1)	0.864	0.857	0.860	7877
		n-gram (1,2)	0.901	0.905	0.902	7877
		n-gram (1,3)	0.903	0.907	0.904	7877
		n-gram (1,4)	0.907	0.910	0.908	7877
K-fold 10	Sentence only	n-gram (1,1)	0.856	0.850	0.853	7877
		n-gram (1,2)	0.899	0.903	0.900	7877
		n-gram (1,3)	0.904	0.907	0.906	7877
		n-gram (1,4)	0.900	0.902	0.901	7877
	Sentence + Sentiment	n-gram (1,1)	0.864	0.856	0.860	7877
		n-gram (1,2)	0.905	0.910	0.907	7877
		n-gram (1,3)	0.908	0.912	0.909	7877
		n-gram (1,4)	0.909	0.911	0.910	7877

Table D.2 Precision, Recall, F1-measures per class by Linear SVM for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision		Recall		F1		#Samples	
			Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.919	0.499	0.912	0.521	0.915	0.510	4449	750
		n-gram (1,2)	0.931	0.663	0.948	0.590	0.939	0.624	4434	765
		n-gram (1,3)	0.925	0.690	0.956	0.558	0.940	0.617	4420	779
		n-gram (1,4)	0.927	0.709	0.960	0.566	0.943	0.629	4432	767
	Sentence + Sentiment	n-gram (1,1)	0.919	0.526	0.915	0.539	0.917	0.533	4426	773
		n-gram (1,2)	0.921	0.722	**0.965**	0.525	0.942	0.608	4425	774
		n-gram (1,3)	0.925	0.715	0.961	0.561	0.943	0.629	4420	779
		n-gram (1,4)	0.923	0.715	0.961	0.546	0.942	0.619	4414	785
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.915	0.531	0.918	0.521	0.917	0.526	3344	595
		n-gram (1,2)	0.927	0.684	0.952	0.582	0.940	0.629	3344	595
		n-gram (1,3)	0.931	0.727	0.962	0.588	0.946	0.650	3356	583
		n-gram (1,4)	0.933	0.714	0.958	0.603	0.945	0.654	3355	584
	Sentence + Sentiment	n-gram (1,1)	0.916	0.549	0.924	0.521	0.920	0.534	3344	595
		n-gram (1,2)	0.932	0.689	0.957	0.580	0.944	0.630	3380	559
		n-gram (1,3)	0.936	0.697	0.955	0.615	0.945	0.654	3370	569
		n-gram (1,4)	0.937	0.683	0.951	0.623	0.944	0.652	3371	568
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.924	0.554	0.917	0.576	0.920	0.565	2206	394
		n-gram (1,2)	0.934	0.713	0.960	0.596	0.946	0.649	2224	376
		n-gram (1,3)	0.920	0.738	0.963	0.555	0.941	0.633	2189	411
		n-gram (1,4)	0.935	0.669	0.946	0.625	0.940	0.646	2211	389
	Sentence + Sentiment	n-gram (1,1)	0.921	0.577	0.925	0.561	0.923	0.569	2201	399
		n-gram (1,2)	0.937	**0.743**	0.964	0.617	0.950	0.674	2221	379
		n-gram (1,3)	0.940	0.741	0.963	0.630	**0.951**	0.681	2227	373
		n-gram (1,4)	0.939	0.702	0.951	0.650	0.945	0.675	2209	391
K-fold 5	Sentence only	n-gram (1,1)	0.920	0.499	0.904	0.547	0.912	0.522	6708	1169
		n-gram (1,2)	0.932	0.693	0.953	0.601	0.943	0.644	6708	1169
		n-gram (1,3)	0.937	0.705	0.954	0.635	0.946	0.668	6708	1169
		n-gram (1,4)	0.940	0.690	0.949	0.651	0.944	0.670	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.925	0.515	0.905	0.576	0.915	0.544	6708	1169
		n-gram (1,2)	0.934	0.711	0.957	0.611	0.945	0.657	6708	1169
		n-gram (1,3)	0.935	0.718	0.958	0.618	0.946	0.665	6708	1169
		n-gram (1,4)	0.940	0.715	0.955	0.653	0.947	0.682	6708	1169
K-fold 10	Sentence only	n-gram (1,1)	0.919	0.494	0.902	0.547	0.911	0.519	6708	1169
		n-gram (1,2)	0.934	0.698	0.954	0.612	0.944	0.652	6708	1169
		n-gram (1,3)	0.940	0.701	0.952	0.651	0.946	0.675	6708	1169
		n-gram (1,4)	0.938	0.679	0.947	0.640	0.943	0.659	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.925	0.513	0.904	0.582	0.914	0.545	6708	1169
		n-gram (1,2)	0.936	0.731	0.960	0.620	0.948	0.671	6708	1169
		n-gram (1,3)	0.939	0.729	0.958	0.644	0.949	0.684	6708	1169
		n-gram (1,4)	**0.943**	0.713	0.953	**0.668**	0.948	**0.690**	6708	1169

Table D.3 Average Precision, Recall, F1-measures by Multinomial SVM for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision	Recall	F1	# Sample
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.858	0.861	0.859	5199
		n-gram (1,2)	0.885	0.893	0.887	5199
		n-gram (1,3)	0.893	0.899	0.895	5199
		n-gram (1,4)	0.893	0.900	0.895	5199
	Sentence + Sentiment	n-gram (1,1)	0.869	0.867	0.868	5199
		n-gram (1,2)	0.889	0.896	0.891	5199
		n-gram (1,3)	0.891	0.898	0.892	5199
		n-gram (1,4)	0.888	0.896	0.890	5199
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.863	0.854	0.858	3939
		n-gram (1,2)	0.892	0.898	0.894	3939
		n-gram (1,3)	0.885	0.892	0.887	3939
		n-gram (1,4)	0.894	0.900	0.896	3939
	Sentence + Sentiment	n-gram (1,1)	0.864	0.865	0.864	3939
		n-gram (1,2)	0.894	0.900	0.896	3939
		n-gram (1,3)	0.897	0.903	0.899	3939
		n-gram (1,4)	0.902	0.906	0.904	3939
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.860	0.855	0.858	2600
		n-gram (1,2)	0.896	0.900	0.898	2600
		n-gram (1,3)	0.902	0.907	0.904	2600
		n-gram (1,4)	0.905	0.907	0.906	2600
	Sentence + Sentiment	n-gram (1,1)	0.871	0.868	0.870	2600
		n-gram (1,2)	0.898	0.903	0.900	2600
		n-gram (1,3)	**0.910**	0.912	**0.911**	2600
		n-gram (1,4)	0.906	0.908	0.907	2600
K-fold 5	Sentence only	n-gram (1,1)	0.859	0.856	0.858	7877
		n-gram (1,2)	0.896	0.900	0.897	7877
		n-gram (1,3)	0.902	0.904	0.902	7877
		n-gram (1,4)	0.902	0.901	0.902	7877
	Sentence + Sentiment	n-gram (1,1)	0.861	0.858	0.859	7877
		n-gram (1,2)	0.905	0.909	0.906	7877
		n-gram (1,3)	0.909	**0.913**	0.911	7877
		n-gram (1,4)	0.906	0.908	0.907	7877
K-fold 10	Sentence only	n-gram (1,1)	0.856	0.850	0.853	7877
		n-gram (1,2)	0.897	0.902	0.899	7877
		n-gram (1,3)	0.904	0.907	0.905	7877
		n-gram (1,4)	0.903	0.902	0.903	7877
	Sentence + Sentiment	n-gram (1,1)	0.865	0.857	0.861	7877
		n-gram (1,2)	0.901	0.906	0.902	7877
		n-gram (1,3)	0.907	0.911	0.909	7877
		n-gram (1,4)	0.905	0.906	0.905	7877

Table D.4 Precision, Recall, F1-measures per class by Multinomial SVM for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision		Recall		F1		#Samples	
			Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.914	0.531	0.923	0.502	0.919	0.516	4430	769
		n-gram (1,2)	0.920	0.687	0.957	0.533	0.938	0.600	4416	783
		n-gram (1,3)	0.927	0.689	0.957	0.558	0.942	0.616	4444	755
		n-gram (1,4)	0.927	0.694	0.957	0.562	0.942	0.621	4438	761
	Sentence + Sentiment	n-gram (1,1)	0.925	0.544	0.918	0.566	0.922	0.554	4437	762
		n-gram (1,2)	0.923	0.696	0.957	0.554	0.940	0.617	4414	785
		n-gram (1,3)	0.922	0.713	**0.962**	0.537	0.942	0.613	4421	778
		n-gram (1,4)	0.922	0.692	0.958	0.542	0.940	0.608	4422	777
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.925	0.507	0.902	0.581	0.913	0.542	3356	583
		n-gram (1,2)	0.929	0.681	0.953	0.582	0.941	0.628	3355	584
		n-gram (1,3)	0.923	0.677	0.952	0.559	0.937	0.612	3336	603
		n-gram (1,4)	0.928	0.701	0.957	0.575	0.942	0.632	3353	586
	Sentence + Sentiment	n-gram (1,1)	0.919	0.547	0.923	0.534	0.921	0.541	3353	586
		n-gram (1,2)	0.929	0.692	0.956	0.578	0.942	0.630	3361	578
		n-gram (1,3)	0.929	0.714	0.959	0.579	0.944	0.640	3354	585
		n-gram (1,4)	0.936	0.702	0.955	0.619	0.946	0.658	3364	575
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.921	0.518	0.907	0.564	0.914	0.540	2208	392
		n-gram (1,2)	0.933	0.685	0.952	0.605	0.942	0.643	2215	385
		n-gram (1,3)	0.935	0.712	0.959	0.604	0.947	0.653	2224	376
		n-gram (1,4)	0.940	0.703	0.952	0.654	0.946	0.677	2213	387
	Sentence + Sentiment	n-gram (1,1)	0.927	0.544	0.919	0.570	0.923	0.557	2223	377
		n-gram (1,2)	0.931	0.719	0.956	0.614	0.943	0.662	2196	404
		n-gram (1,3)	0.943	0.711	0.956	0.653	0.949	0.681	2228	372
		n-gram (1,4)	0.943	0.681	0.951	0.646	0.947	0.663	2236	364
K-fold 5	Sentence only	n-gram (1,1)	0.919	0.515	0.911	0.542	0.915	0.528	6708	1169
		n-gram (1,2)	0.933	0.683	0.951	0.606	0.942	0.642	6708	1169
		n-gram (1,3)	0.939	0.684	0.948	0.649	0.944	0.666	6708	1169
		n-gram (1,4)	0.944	0.664	0.940	**0.678**	0.942	0.671	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.920	0.524	0.912	0.551	0.916	0.537	6708	1169
		n-gram (1,2)	0.935	0.731	0.960	0.617	0.948	0.669	6708	1169
		n-gram (1,3)	0.940	**0.731**	0.958	0.650	**0.949**	**0.688**	6708	1169
		n-gram (1,4)	0.942	0.701	0.951	0.663	0.946	0.682	6708	1169
K-fold 10	Sentence only	n-gram (1,1)	0.919	0.494	0.903	0.543	0.911	0.518	6708	1169
		n-gram (1,2)	0.934	0.691	0.952	0.611	0.943	0.648	6708	1169
		n-gram (1,3)	0.941	0.697	0.950	0.655	0.945	0.675	6708	1169
		n-gram (1,4)	**0.944**	0.669	0.941	**0.678**	0.943	0.673	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.925	0.517	0.905	0.581	0.915	0.547	6708	1169
		n-gram (1,2)	0.932	0.721	0.960	0.599	0.946	0.654	6708	1169
		n-gram (1,3)	0.939	0.728	0.958	0.640	0.948	0.681	6708	1169
		n-gram (1,4)	0.944	0.685	0.946	0.676	0.945	0.680	6708	1169

Table D.5 Average Precision, Recall, F1-measures by Bernoulli Naïve Bayes for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision	Recall	F1	# Sample
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.855	0.873	0.851	5199
		n-gram (1,2)	0.874	0.852	0.784	5199
		n-gram (1,3)	0.731	0.855	0.788	5199
		n-gram (1,4)	0.729	0.854	0.786	5199
	Sentence + Sentiment	n-gram (1,1)	0.865	0.879	0.858	5199
		n-gram (1,2)	0.873	0.851	0.783	5199
		n-gram (1,3)	0.721	0.849	0.780	5199
		n-gram (1,4)	0.722	0.850	0.781	5199
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.851	0.870	0.854	3939
		n-gram (1,2)	0.836	0.851	0.783	3939
		n-gram (1,3)	0.721	0.849	0.779	3939
		n-gram (1,4)	0.722	0.850	0.781	3939
	Sentence + Sentiment	n-gram (1,1)	**0.875**	0.886	0.877	3939
		n-gram (1,2)	0.813	0.850	0.783	3939
		n-gram (1,3)	0.727	0.853	0.785	3939
		n-gram (1,4)	0.728	0.853	0.786	3939
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.874	**0.887**	0.877	2600
		n-gram (1,2)	0.842	0.850	0.782	2600
		n-gram (1,3)	0.723	0.850	0.781	2600
		n-gram (1,4)	0.720	0.848	0.779	2600
	Sentence + Sentiment	n-gram (1,1)	0.869	0.880	0.873	2600
		n-gram (1,2)	0.831	0.852	0.786	2600
		n-gram (1,3)	0.713	0.844	0.773	2600
		n-gram (1,4)	0.726	0.852	0.784	2600
K-fold 5	Sentence only	n-gram (1,1)	0.864	0.877	0.868	7877
		n-gram (1,2)	0.854	0.854	0.790	7877
		n-gram (1,3)	0.799	0.852	0.784	7877
		n-gram (1,4)	0.725	0.851	0.783	7877
	Sentence + Sentiment	n-gram (1,1)	0.875	0.885	**0.878**	7877
		n-gram (1,2)	0.857	0.854	0.790	7877
		n-gram (1,3)	0.874	0.852	0.784	7877
		n-gram (1,4)	0.725	0.851	0.783	7877
K-fold 10	Sentence only	n-gram (1,1)	0.870	0.882	0.873	7877
		n-gram (1,2)	0.848	0.854	0.791	7877
		n-gram (1,3)	0.799	0.852	0.784	7877
		n-gram (1,4)	0.725	0.851	0.783	7877
	Sentence + Sentiment	n-gram (1,1)	0.873	0.883	0.876	7877
		n-gram (1,2)	0.855	0.854	0.790	7877
		n-gram (1,3)	0.725	0.851	0.783	7877
		n-gram (1,4)	0.725	0.851	0.783	7877

Table D.6 Precision, Recall, F1-measures per class by Bernoulli Naïve Bayes for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision		Recall		F1		#Samples	
			Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.887	0.676	0.974	0.301	0.929	0.417	4415	784
		n-gram (1,2)	0.851	**1.000**	**1.000**	0.003	0.920	0.005	4425	774
		n-gram (1,3)	0.855	0.000	**1.000**	0.000	0.922	0.000	4444	755
		n-gram (1,4)	0.854	0.000	**1.000**	0.000	0.921	0.000	4439	760
	Sentence + Sentiment	n-gram (1,1)	0.889	0.726	0.979	0.312	0.932	0.436	4417	782
		n-gram (1,2)	0.851	**1.000**	**1.000**	0.004	0.919	0.008	4421	778
		n-gram (1,3)	0.849	0.000	**1.000**	0.000	0.918	0.000	4414	785
		n-gram (1,4)	0.850	0.000	**1.000**	0.000	0.919	0.000	4419	780
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.895	0.590	0.960	0.341	0.926	0.433	3365	574
		n-gram (1,2)	0.851	0.750	**1.000**	0.005	0.919	0.010	3349	590
		n-gram (1,3)	0.849	0.000	**1.000**	0.000	0.918	0.000	3344	595
		n-gram (1,4)	0.850	0.000	**1.000**	0.000	0.919	0.000	3348	591
	Sentence + Sentiment	n-gram (1,1)	0.911	0.665	0.960	0.461	0.935	0.545	3358	581
		n-gram (1,2)	0.851	0.600	0.999	0.005	0.919	0.010	3349	590
		n-gram (1,3)	0.853	0.000	**1.000**	0.000	0.920	0.000	3358	581
		n-gram (1,4)	0.853	0.000	**1.000**	0.000	0.921	0.000	3361	578
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.912	0.648	0.961	0.435	**0.936**	0.520	2232	368
		n-gram (1,2)	0.850	0.800	**1.000**	0.010	0.919	0.020	2206	394
		n-gram (1,3)	0.850	0.000	**1.000**	0.000	0.919	0.000	2210	390
		n-gram (1,4)	0.848	0.000	**1.000**	0.000	0.918	0.000	2206	394
	Sentence + Sentiment	n-gram (1,1)	0.913	0.605	0.950	0.458	0.931	0.521	2229	371
		n-gram (1,2)	0.852	0.714	0.999	0.013	0.920	0.025	2211	389
		n-gram (1,3)	0.844	0.000	**1.000**	0.000	0.916	0.000	2195	405
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	2216	384
K-fold 5	Sentence only	n-gram (1,1)	0.906	0.624	0.954	0.434	0.930	0.512	6708	1169
		n-gram (1,2)	0.854	0.852	0.999	0.020	0.921	0.038	6708	1169
		n-gram (1,3)	0.852	0.500	**1.000**	0.001	0.920	0.002	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.914	0.650	0.955	0.484	0.934	**0.555**	6708	1169
		n-gram (1,2)	0.854	0.875	**1.000**	0.018	0.921	0.035	6708	1169
		n-gram (1,3)	0.852	**1.000**	**1.000**	0.001	0.920	0.002	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
K-fold 10	Sentence only	n-gram (1,1)	0.910	0.641	0.955	0.459	0.932	0.535	6708	1169
		n-gram (1,2)	0.854	0.813	0.999	0.022	0.921	0.043	6708	1169
		n-gram (1,3)	0.852	0.500	**1.000**	0.001	0.920	0.002	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
	Sentence + Sentiment	n-gram (1,1)	**0.915**	0.636	0.951	**0.491**	0.932	0.554	6708	1169
		n-gram (1,2)	0.854	0.857	0.999	0.021	0.921	0.040	6708	1169
		n-gram (1,3)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169

Table D.7 Average Precision, Recall, F1-measures by Multinomial Naïve Bayes for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision	Recall	F1	# Sample
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.835	0.853	0.788	5199
		n-gram (1,2)	0.715	0.846	0.775	5199
		n-gram (1,3)	0.726	0.852	0.784	5199
		n-gram (1,4)	0.736	0.858	0.792	5199
	Sentence + Sentiment	n-gram (1,1)	0.873	0.851	0.783	5199
		n-gram (1,2)	0.722	0.850	0.780	5199
		n-gram (1,3)	0.735	0.857	0.791	5199
		n-gram (1,4)	0.727	0.853	0.785	5199
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.859	0.856	0.791	3939
		n-gram (1,2)	0.724	0.851	0.782	3939
		n-gram (1,3)	0.728	0.853	0.786	3939
		n-gram (1,4)	0.733	0.856	0.790	3939
	Sentence + Sentiment	n-gram (1,1)	0.858	**0.859**	**0.797**	3939
		n-gram (1,2)	0.732	0.856	0.789	3939
		n-gram (1,3)	0.732	0.856	0.789	3939
		n-gram (1,4)	0.724	0.851	0.782	3939
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.835	0.850	0.784	2600
		n-gram (1,2)	0.719	0.848	0.778	2600
		n-gram (1,3)	0.727	0.853	0.785	2600
		n-gram (1,4)	0.717	0.847	0.776	2600
	Sentence + Sentiment	n-gram (1,1)	**0.876**	0.855	0.791	2600
		n-gram (1,2)	0.716	0.846	0.776	2600
		n-gram (1,3)	0.721	0.849	0.779	2600
		n-gram (1,4)	0.720	0.848	0.779	2600
K-fold 5	Sentence only	n-gram (1,1)	0.848	0.854	0.791	7877
		n-gram (1,2)	0.725	0.852	0.783	7877
		n-gram (1,3)	0.725	0.852	0.783	7877
		n-gram (1,4)	0.725	0.852	0.783	7877
	Sentence + Sentiment	n-gram (1,1)	0.855	0.855	0.792	7877
		n-gram (1,2)	0.725	0.852	0.783	7877
		n-gram (1,3)	0.725	0.852	0.783	7877
		n-gram (1,4)	0.725	0.852	0.783	7877
K-fold 10	Sentence only	n-gram (1,1)	0.842	0.854	0.791	7877
		n-gram (1,2)	0.725	0.852	0.783	7877
		n-gram (1,3)	0.725	0.852	0.783	7877
		n-gram (1,4)	0.725	0.852	0.783	7877
	Sentence + Sentiment	n-gram (1,1)	0.860	0.855	0.793	7877
		n-gram (1,2)	0.725	0.852	0.783	7877
		n-gram (1,3)	0.725	0.852	0.783	7877
		n-gram (1,4)	0.725	0.852	0.783	7877

Table D.8 Precision, Recall, F1-measures per class by Multinomial Naïve Bayes for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision		Recall		F1		#Samples	
			Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.854	0.727	0.999	0.010	0.921	0.021	4431	768
		n-gram (1,2)	0.846	0.000	**1.000**	0.000	0.916	0.000	4397	802
		n-gram (1,3)	0.852	0.000	**1.000**	0.000	0.920	0.000	4431	768
		n-gram (1,4)	0.858	0.000	**1.000**	0.000	**0.924**	0.000	4461	738
	Sentence + Sentiment	n-gram (1,1)	0.851	**1.000**	**1.000**	0.003	0.919	0.005	4421	778
		n-gram (1,2)	0.850	0.000	**1.000**	0.000	0.919	0.000	4417	782
		n-gram (1,3)	0.857	0.000	**1.000**	0.000	0.923	0.000	4457	742
		n-gram (1,4)	0.853	0.000	**1.000**	0.000	0.921	0.000	4434	765
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.856	0.875	**1.000**	0.012	0.922	0.024	3365	574
		n-gram (1,2)	0.851	0.000	**1.000**	0.000	0.919	0.000	3352	587
		n-gram (1,3)	0.853	0.000	**1.000**	0.000	0.921	0.000	3361	578
		n-gram (1,4)	0.856	0.000	**1.000**	0.000	0.922	0.000	3372	567
	Sentence + Sentiment	n-gram (1,1)	**0.859**	0.857	0.999	0.021	**0.924**	0.041	3372	567
		n-gram (1,2)	0.856	0.000	**1.000**	0.000	0.922	0.000	3371	568
		n-gram (1,3)	0.856	0.000	**1.000**	0.000	0.922	0.000	3370	569
		n-gram (1,4)	0.851	0.000	**1.000**	0.000	0.919	0.000	3351	588
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.850	0.750	0.999	0.015	0.919	0.030	2206	394
		n-gram (1,2)	0.848	0.000	**1.000**	0.000	0.918	0.000	2205	395
		n-gram (1,3)	0.853	0.000	**1.000**	0.000	0.920	0.000	2217	383
		n-gram (1,4)	0.847	0.000	**1.000**	0.000	0.917	0.000	2201	399
	Sentence + Sentiment	n-gram (1,1)	0.855	**1.000**	**1.000**	0.018	0.922	0.036	2217	383
		n-gram (1,2)	0.846	0.000	**1.000**	0.000	0.917	0.000	2200	400
		n-gram (1,3)	0.849	0.000	**1.000**	0.000	0.918	0.000	2207	393
		n-gram (1,4)	0.848	0.000	**1.000**	0.000	0.918	0.000	2206	394
K-fold 5	Sentence only	n-gram (1,1)	0.854	0.813	0.999	0.022	0.921	0.043	6708	1169
		n-gram (1,2)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,3)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.855	0.857	0.999	0.026	0.921	0.050	6708	1169
		n-gram (1,2)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,3)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
K-fold 10	Sentence only	n-gram (1,1)	0.854	0.771	0.999	0.023	0.921	0.045	6708	1169
		n-gram (1,2)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,3)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.855	0.889	0.999	**0.027**	0.922	**0.053**	6708	1169
		n-gram (1,2)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,3)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169
		n-gram (1,4)	0.852	0.000	**1.000**	0.000	0.920	0.000	6708	1169

Table D.9 Average Precision, Recall, F1-measures by Logistic Regression for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision	Recall	F1	# Sample
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.866	0.875	0.841	5199
		n-gram (1,2)	0.864	0.865	0.817	5199
		n-gram (1,3)	0.866	0.869	0.820	5199
		n-gram (1,4)	0.853	0.850	0.785	5199
	Sentence + Sentiment	n-gram (1,1)	0.863	0.874	0.843	5199
		n-gram (1,2)	0.868	0.863	0.812	5199
		n-gram (1,3)	0.861	0.859	0.800	5199
		n-gram (1,4)	0.864	0.860	0.802	5199
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.869	0.879	0.851	3939
		n-gram (1,2)	0.874	0.873	0.834	3939
		n-gram (1,3)	0.861	0.865	0.817	3939
		n-gram (1,4)	0.863	0.857	0.802	3939
	Sentence + Sentiment	n-gram (1,1)	0.884	0.890	0.867	3939
		n-gram (1,2)	0.871	0.874	0.838	3939
		n-gram (1,3)	0.866	0.869	0.829	3939
		n-gram (1,4)	0.865	0.868	0.823	3939
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.874	0.885	0.862	2600
		n-gram (1,2)	0.872	0.873	0.839	2600
		n-gram (1,3)	0.873	0.870	0.830	2600
		n-gram (1,4)	0.876	0.872	0.830	2600
	Sentence + Sentiment	n-gram (1,1)	0.880	0.887	0.865	2600
		n-gram (1,2)	0.882	0.888	0.863	2600
		n-gram (1,3)	0.871	0.876	0.845	2600
		n-gram (1,4)	0.856	0.861	0.818	2600
K-fold 5	Sentence only	n-gram (1,1)	0.875	0.885	0.864	7877
		n-gram (1,2)	0.877	0.882	0.854	7877
		n-gram (1,3)	0.875	0.878	0.845	7877
		n-gram (1,4)	0.873	0.875	0.839	7877
	Sentence + Sentiment	n-gram (1,1)	**0.886**	**0.894**	0.877	7877
		n-gram (1,2)	0.883	0.887	0.862	7877
		n-gram (1,3)	0.879	0.883	0.856	7877
		n-gram (1,4)	0.876	0.880	0.849	7877
K-fold 10	Sentence only	n-gram (1,1)	0.878	0.888	0.869	7877
		n-gram (1,2)	0.880	0.885	0.859	7877
		n-gram (1,3)	0.876	0.880	0.849	7877
		n-gram (1,4)	0.874	0.878	0.845	7877
	Sentence + Sentiment	n-gram (1,1)	0.885	0.893	**0.877**	7877
		n-gram (1,2)	0.886	0.889	0.866	7877
		n-gram (1,3)	0.881	0.885	0.859	7877
		n-gram (1,4)	0.878	0.882	0.854	7877

Table D.10 Precision, Recall, F1-measures per class by Logistic Regression for Binary Filtering of Feature Requirements

Validation	Tuning Parameter		Precision		Recall		F1		#Samples	
			Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
K-fold 2 Test 66%	Sentence only	n-gram (1,1)	0.877	0.800	0.991	0.198	0.931	0.318	4433	766
		n-gram (1,2)	0.865	0.854	0.997	0.099	0.927	0.178	4435	764
		n-gram (1,3)	0.870	0.847	0.998	0.084	0.929	0.152	4469	730
		n-gram (1,4)	0.850	0.870	**0.999**	0.025	0.919	0.049	4403	796
	Sentence + Sentiment	n-gram (1,1)	0.878	0.774	0.988	0.224	0.930	0.348	4418	781
		n-gram (1,2)	0.862	0.899	0.998	0.091	0.925	0.166	4422	777
		n-gram (1,3)	0.858	0.878	**0.999**	0.047	0.923	0.089	4433	766
		n-gram (1,4)	0.860	0.892	**0.999**	0.044	0.924	0.083	4442	757
K-fold 2 Test 50%	Sentence only	n-gram (1,1)	0.883	0.785	0.988	0.245	0.933	0.373	3359	580
		n-gram (1,2)	0.872	0.881	0.996	0.176	0.930	0.293	3347	592
		n-gram (1,3)	0.865	0.840	0.996	0.108	0.926	0.191	3355	584
		n-gram (1,4)	0.857	0.896	**0.999**	0.072	0.922	0.133	3339	600
	Sentence + Sentiment	n-gram (1,1)	0.893	0.829	0.989	0.305	0.939	0.446	3365	574
		n-gram (1,2)	0.875	0.851	0.994	0.193	0.931	0.315	3349	590
		n-gram (1,3)	0.870	0.843	0.995	0.154	0.928	0.261	3350	589
		n-gram (1,4)	0.869	0.846	0.996	0.115	0.928	0.203	3366	573
K-fold 2 Test 33%	Sentence only	n-gram (1,1)	0.891	0.778	0.986	0.295	0.936	0.427	2220	380
		n-gram (1,2)	0.874	0.863	0.994	0.206	0.930	0.333	2202	398
		n-gram (1,3)	0.869	0.897	0.996	0.175	0.928	0.292	2199	401
		n-gram (1,4)	0.871	**0.902**	0.997	0.144	0.930	0.248	2218	382
	Sentence + Sentiment	n-gram (1,1)	0.891	0.820	0.987	0.332	0.936	0.472	2202	398
		n-gram (1,2)	0.891	0.831	0.990	0.286	0.938	0.425	2222	378
		n-gram (1,3)	0.878	0.833	0.992	0.228	0.931	0.359	2206	394
		n-gram (1,4)	0.862	0.819	0.994	0.145	0.924	0.246	2193	407
K-fold 5	Sentence only	n-gram (1,1)	0.893	0.776	0.984	0.321	0.936	0.454	6708	1169
		n-gram (1,2)	0.884	0.831	0.991	0.257	0.935	0.393	6708	1169
		n-gram (1,3)	0.879	0.850	0.993	0.214	0.933	0.342	6708	1169
		n-gram (1,4)	0.875	0.856	0.994	0.188	0.931	0.309	6708	1169
	Sentence + Sentiment	n-gram (1,1)	0.900	0.803	0.984	0.376	**0.940**	0.512	6708	1169
		n-gram (1,2)	0.889	0.852	0.991	0.290	0.937	0.433	6708	1169
		n-gram (1,3)	0.885	0.844	0.992	0.263	0.935	0.402	6708	1169
		n-gram (1,4)	0.881	0.847	0.993	0.233	0.934	0.365	6708	1169
K-fold 10	Sentence only	n-gram (1,1)	0.896	0.778	0.983	0.344	0.937	0.477	6708	1169
		n-gram (1,2)	0.887	0.840	0.991	0.275	0.936	0.414	6708	1169
		n-gram (1,3)	0.881	0.845	0.993	0.233	0.934	0.365	6708	1169
		n-gram (1,4)	0.879	0.850	0.993	0.213	0.933	0.341	6708	1169
	Sentence + Sentiment	n-gram (1,1)	**0.901**	0.790	0.982	**0.384**	**0.940**	**0.517**	6708	1169
		n-gram (1,2)	0.892	0.851	0.991	0.309	0.938	0.453	6708	1169
		n-gram (1,3)	0.887	0.849	0.992	0.274	0.936	0.414	6708	1169
		n-gram (1,4)	0.884	0.844	0.992	0.254	0.935	0.391	6708	1169

Table D.11 Average Precision, Recall, F1-measures by Linear SVM for Multiclass Multi-label Classification

Validation	N-gram	Precision	Recall	F1
K-fold 2- Test 50%	n-gram (1,1)	0.69	0.52	0.57
	n-gram (1,2)	0.77	0.45	0.53
	n-gram (1,3)	0.75	0.35	0.43
	n-gram (1,4)	0.80	0.38	0.47
K-fold 2- Test 33%	n-gram (1,1)	0.73	0.56	0.62
	n-gram (1,2)	0.77	0.44	0.53
	n-gram (1,3)	0.75	0.43	0.52
	n-gram (1,4)	0.76	0.39	0.46
K-fold 2- Test 20%	n-gram (1,1)	0.76	**0.61**	**0.67**
	n-gram (1,2)	**0.88**	0.58	0.66
	n-gram (1,3)	0.83	0.48	0.57
	n-gram (1,4)	0.80	0.43	0.51
K-fold 5	n-gram (1,1)	0.70	0.59	0.63
	n-gram (1,2)	0.78	0.57	0.64
	n-gram (1,3)	0.77	0.57	0.63
	n-gram (1,4)	0.78	0.56	0.63
K-fold 10	n-gram (1,1)	0.71	0.60	0.64
	n-gram (1,2)	0.76	0.58	0.64
	n-gram (1,3)	0.77	0.59	0.64
	n-gram (1,4)	0.77	0.58	0.63

Table D.12 Precision, Recall, F1-measures per class by Linear SVM for Multiclass Multi-label Classification

Validation	Tuning Parameter	Precision									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.68	0.33	**1.00**	0.40	0.64	0.88	**1.00**	0.96	0.81	0.56
	n-gram (1,2)	0.67	0.50	**1.00**	0.64	**1.00**	0.84	**1.00**	**1.00**	0.89	0.71
	n-gram (1,3)	0.71	0.00	**1.00**	0.50	0.67	0.95	**1.00**	0.93	0.90	0.74
	n-gram (1,4)	0.72	**1.00**	**1.00**	0.43	**1.00**	**1.00**	**1.00**	0.94	0.85	0.81
K-fold 2-Test 33%	n-gram (1,1)	0.67	0.67	0.78	0.38	0.82	0.87	**1.00**	0.94	0.79	0.76
	n-gram (1,2)	0.67	0.75	**1.00**	0.62	0.75	0.83	**1.00**	**1.00**	0.87	0.76
	n-gram (1,3)	0.65	0.00	**1.00**	0.56	0.75	0.91	**1.00**	**1.00**	0.91	**0.84**
	n-gram (1,4)	0.62	**1.00**	**1.00**	0.50	0.67	**1.00**	**1.00**	**1.00**	0.89	0.67
K-fold 2-Test 20%	n-gram (1,1)	0.70	0.00	0.80	0.54	0.88	0.89	**1.00**	0.93	0.90	0.79
	n-gram (1,2)	**0.84**	**1.00**	**1.00**	**1.00**	0.80	0.89	**1.00**	**1.00**	0.83	0.83
	n-gram (1,3)	0.75	**1.00**	**1.00**	0.80	**1.00**	0.93	**1.00**	**1.00**	0.86	0.68
	n-gram (1,4)	0.76	**1.00**	0.00	0.67	**1.00**	**1.00**	**1.00**	**1.00**	**0.93**	0.81
K-fold 5	n-gram (1,1)	0.67	0.52	0.81	0.49	0.68	0.83	**1.00**	0.94	0.80	0.62
	n-gram (1,2)	0.71	0.89	0.95	0.57	0.78	0.92	**1.00**	0.96	0.83	0.72
	n-gram (1,3)	0.70	0.66	0.96	0.56	0.78	0.92	**1.00**	0.94	0.83	0.74
	n-gram (1,4)	0.71	0.88	0.97	0.57	0.81	0.93	**1.00**	0.95	0.80	0.74
K-fold 10	n-gram (1,1)	0.68	0.61	0.83	0.45	0.82	0.78	**1.00**	0.97	0.79	0.63
	n-gram (1,2)	0.71	0.63	0.89	0.56	0.86	0.91	0.94	0.96	0.85	0.69
	n-gram (1,3)	0.71	0.77	0.84	0.60	0.79	0.92	**1.00**	0.97	0.83	0.73
	n-gram (1,4)	0.71	0.65	0.90	0.63	0.80	0.95	**1.00**	0.95	0.79	0.69

Validation	Tuning Parameter	Recall									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.74	0.04	0.22	0.21	0.31	0.49	0.77	0.68	0.65	0.40
	n-gram (1,2)	0.66	0.04	0.16	0.11	0.36	0.39	0.67	0.54	0.45	0.38
	n-gram (1,3)	0.63	0.00	0.07	0.05	0.20	0.27	0.42	0.31	0.32	0.17
	n-gram (1,4)	0.64	0.05	0.15	0.05	0.13	0.27	0.50	0.41	0.35	0.22
K-fold 2-Test 33%	n-gram (1,1)	0.69	0.13	0.41	0.27	0.50	0.64	0.75	0.67	0.63	0.46
	n-gram (1,2)	0.61	0.25	0.08	0.14	0.18	0.47	0.57	0.68	0.47	0.33
	n-gram (1,3)	0.62	0.00	0.12	0.14	0.16	0.49	0.85	0.70	0.47	0.32
	n-gram (1,4)	0.69	0.06	0.12	0.04	0.12	0.28	0.56	0.58	0.37	0.33
K-fold 2-Test 20%	n-gram (1,1)	0.63	0.00	0.29	**0.54**	**0.70**	0.62	0.86	0.82	0.64	**0.61**
	n-gram (1,2)	**0.77**	0.17	0.13	0.25	0.36	0.67	**1.00**	0.75	0.62	0.45
	n-gram (1,3)	0.65	0.08	0.17	0.13	0.50	0.44	0.50	0.64	0.63	0.45
	n-gram (1,4)	0.68	0.07	0.00	0.08	0.29	0.41	0.60	**0.86**	0.59	0.30
K-fold 5	n-gram (1,1)	0.65	0.27	0.43	0.33	0.43	**0.72**	0.82	0.78	**0.72**	0.54
	n-gram (1,2)	0.71	0.27	0.37	0.26	0.41	0.63	0.76	0.75	0.63	0.49
	n-gram (1,3)	0.74	0.21	0.33	0.22	0.39	0.61	0.79	0.73	0.65	0.46
	n-gram (1,4)	0.75	0.19	0.32	0.21	0.36	0.59	0.73	0.76	0.62	0.44
K-fold 10	n-gram (1,1)	0.69	**0.40**	**0.44**	0.35	0.51	0.65	0.85	0.77	0.71	0.50
	n-gram (1,2)	0.72	0.25	0.35	0.26	0.49	0.65	0.79	0.76	0.67	0.48
	n-gram (1,3)	0.76	0.23	0.30	0.26	0.47	0.62	0.79	0.77	0.65	0.47
	n-gram (1,4)	**0.77**	0.15	0.32	0.23	0.40	0.62	0.76	0.73	0.66	0.46

Validation	Tuning Parameter	F1 /Class									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.70	0.07	0.36	0.28	0.42	0.62	0.87	0.79	0.72	0.46
	n-gram (1,2)	0.67	0.07	0.28	0.19	0.53	0.53	0.80	0.70	0.60	0.50
	n-gram (1,3)	0.67	0.00	0.13	0.09	0.31	0.42	0.59	0.47	0.48	0.28
	n-gram (1,4)	0.68	0.09	0.27	0.09	0.23	0.43	0.67	0.57	0.50	0.35
K-fold 2-Test 33%	n-gram (1,1)	0.68	0.22	0.54	0.31	0.62	0.74	0.86	0.78	0.70	0.57
	n-gram (1,2)	0.64	0.38	0.15	0.22	0.29	0.60	0.73	0.81	0.61	0.46
	n-gram (1,3)	0.64	0.00	0.21	0.22	0.26	0.64	0.92	0.82	0.62	0.47
	n-gram (1,4)	0.65	0.11	0.21	0.08	0.21	0.44	0.71	0.73	0.52	0.44
K-fold 2-Test 20%	n-gram (1,1)	0.66	0.00	0.42	**0.54**	**0.78**	0.73	0.92	0.87	0.75	**0.69**
	n-gram (1,2)	**0.80**	0.29	0.24	0.40	0.50	**0.76**	**1.00**	0.86	0.71	0.58
	n-gram (1,3)	0.69	0.15	0.29	0.23	0.67	0.60	0.67	0.78	0.73	0.55
	n-gram (1,4)	0.72	0.12	0.00	0.15	0.45	0.58	0.75	**0.92**	0.72	0.44
K-fold 5	n-gram (1,1)	0.66	0.35	**0.56**	0.39	0.52	**0.76**	0.89	0.84	**0.76**	0.57
	n-gram (1,2)	0.71	0.39	0.51	0.35	0.52	0.74	0.86	0.83	0.71	0.58
	n-gram (1,3)	0.72	0.31	0.49	0.30	0.52	0.73	0.88	0.81	0.73	0.57
	n-gram (1,4)	0.73	0.31	0.46	0.31	0.48	0.72	0.82	0.84	0.70	0.55
K-fold 10	n-gram (1,1)	0.69	**0.46**	**0.56**	0.38	0.60	0.70	0.91	0.85	0.74	0.55
	n-gram (1,2)	0.71	0.34	0.49	0.34	0.60	0.75	0.85	0.84	0.74	0.56
	n-gram (1,3)	0.73	0.34	0.43	0.35	0.57	0.74	0.87	0.86	0.72	0.56
	n-gram (1,4)	0.74	0.23	0.45	0.33	0.52	0.74	0.85	0.81	0.71	0.55

Validation	Tuning Parameter	Number of classified samples										
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Total
K-fold 2-Test 50%	n-gram (1,1)	204	25	36	56	29	72	13	37	54	98	624
	n-gram (1,2)	207	27	31	63	25	69	18	39	55	91	625
	n-gram (1,3)	214	24	28	60	20	67	19	45	59	98	634
	n-gram (1,4)	211	21	26	59	23	66	20	37	65	95	623
K-fold 2-Test 33%	n-gram (1,1)	129	15	17	41	18	53	16	24	35	67	415
	n-gram (1,2)	147	12	25	37	17	40	7	28	43	58	414
	n-gram (1,3)	131	21	25	36	19	43	13	23	45	65	421
	n-gram (1,4)	126	17	26	48	16	46	9	26	43	54	411
K-fold 2-Test 20%	n-gram (1,1)	86	5	14	26	10	26	7	17	28	36	255
	n-gram (1,2)	91	6	15	24	11	24	3	16	16	42	248
	n-gram (1,3)	82	12	6	30	8	32	10	14	19	33	246
	n-gram (1,4)	79	15	15	24	17	27	5	7	22	43	254
K-fold 5	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248
K-fold 10	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248

Table D.13 Average Precision, Recall, F1-measures by Multinomial SVM for Multiclass Multi-label Classification

Validation	N-gram	Precision	Recall	F1
K-fold 2- Test 50%	n-gram (1,1)	0.71	0.54	0.60
	n-gram (1,2)	0.77	0.44	0.53
	n-gram (1,3)	0.74	0.36	0.45
	n-gram (1,4)	0.78	0.38	0.45
K-fold 2- Test 33%	n-gram (1,1)	0.70	0.55	0.61
	n-gram (1,2)	**0.80**	0.45	0.53
	n-gram (1,3)	0.74	0.43	0.49
	n-gram (1,4)	0.73	0.41	0.47
K-fold 2- Test 20%	n-gram (1,1)	0.74	0.56	0.63
	n-gram (1,2)	**0.80**	0.50	0.59
	n-gram (1,3)	0.72	0.45	0.52
	n-gram (1,4)	0.72	0.43	0.53
K-fold 5	n-gram (1,1)	0.72	0.58	0.63
	n-gram (1,2)	0.78	0.58	**0.64**
	n-gram (1,3)	0.77	0.56	0.62
	n-gram (1,4)	0.76	0.56	0.62
K-fold 10	n-gram (1,1)	0.71	**0.61**	**0.64**
	n-gram (1,2)	0.76	0.59	**0.64**
	n-gram (1,3)	0.76	0.57	0.63
	n-gram (1,4)	0.77	0.58	0.63

Table D.14 Precision, Recall, F1 measures per class by Multinomial SVM for Multiclass Multi-label Classification

Validation	Tuning Parameter	Precision									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.64	0.40	0.75	0.50	0.75	0.94	**1.00**	0.90	0.83	0.80
	n-gram (1,2)	0.65	**1.00**	**1.00**	0.78	0.40	0.94	**1.00**	**1.00**	0.83	**0.88**
	n-gram (1,3)	0.67	0.00	0.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.83	0.68
	n-gram (1,4)	**0.77**	0.00	0.00	0.00	0.50	0.93	**1.00**	**1.00**	**1.00**	0.87
K-fold 2-Test 33%	n-gram (1,1)	0.69	0.55	0.75	0.43	0.70	0.85	0.86	**1.00**	0.81	0.60
	n-gram (1,2)	0.69	**1.00**	**1.00**	0.57	**1.00**	0.90	**1.00**	0.94	0.95	0.79
	n-gram (1,3)	0.64	0.00	**1.00**	0.67	**1.00**	0.95	**1.00**	**1.00**	0.84	0.71
	n-gram (1,4)	0.65	0.00	**1.00**	0.38	0.67	0.93	**1.00**	0.93	0.83	0.77
K-fold 2-Test 20%	n-gram (1,1)	0.66	0.56	0.82	0.46	0.83	0.84	**1.00**	0.96	0.78	0.61
	n-gram (1,2)	0.66	0.50	**1.00**	0.62	0.77	0.95	**1.00**	0.96	0.84	0.85
	n-gram (1,3)	0.69	0.00	**1.00**	0.75	**1.00**	0.81	**1.00**	**1.00**	0.80	0.68
	n-gram (1,4)	0.67	0.00	**1.00**	**1.00**	**1.00**	0.95	**1.00**	0.95	0.81	0.68
K-fold 5	n-gram (1,1)	0.70	0.55	0.87	0.50	0.67	0.84	0.96	0.94	0.81	0.65
	n-gram (1,2)	0.70	0.88	0.94	0.65	0.79	0.93	**1.00**	0.97	0.79	0.71
	n-gram (1,3)	0.71	0.86	**1.00**	0.55	0.76	0.94	**1.00**	0.97	0.79	0.70
	n-gram (1,4)	0.71	0.60	**1.00**	0.55	0.77	0.88	**1.00**	0.97	0.80	0.72
K-fold 10	n-gram (1,1)	0.67	0.62	0.79	0.42	0.81	0.83	**1.00**	0.94	0.83	0.65
	n-gram (1,2)	0.70	0.82	0.91	0.59	0.80	0.91	**1.00**	0.95	0.83	0.70
	n-gram (1,3)	0.70	0.63	0.91	0.53	0.82	0.94	**1.00**	0.96	0.83	0.69
	n-gram (1,4)	0.72	0.66	0.91	0.62	0.67	0.91	**1.00**	0.96	0.84	0.73

Validation	Tuning Parameter	Recall									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.65	0.22	0.25	0.38	**0.50**	0.50	0.67	0.75	0.68	**0.57**
	n-gram (1,2)	0.58	0.14	0.31	**0.39**	0.22	0.59	**1.00**	0.67	0.60	0.38
	n-gram (1,3)	0.63	0.00	0.00	0.09	0.40	0.43	**1.00**	0.70	0.43	0.39
	n-gram (1,4)	0.63	0.00	0.00	0.00	0.20	0.42	**1.00**	0.56	0.36	0.36
K-fold 2-Test 33%	n-gram (1,1)	0.62	**0.38**	0.29	0.31	0.39	0.73	0.67	0.60	0.62	0.52
	n-gram (1,2)	0.69	0.11	0.19	0.08	0.31	0.42	0.86	0.70	0.55	0.30
	n-gram (1,3)	0.71	0.00	0.09	0.05	0.31	0.39	0.69	0.57	0.54	0.22
	n-gram (1,4)	0.73	0.00	0.05	0.11	0.08	0.29	0.45	0.43	0.39	0.26
K-fold 2-Test 20%	n-gram (1,1)	0.69	0.28	**0.48**	0.21	0.29	0.51	0.81	0.76	0.60	0.45
	n-gram (1,2)	0.63	0.05	0.31	0.08	0.40	0.32	0.64	0.63	0.56	0.33
	n-gram (1,3)	0.59	0.00	0.03	0.05	0.25	0.35	0.57	0.29	0.41	0.24
	n-gram (1,4)	0.66	0.00	0.07	0.03	0.11	0.28	0.56	0.47	0.35	0.22
K-fold 5	n-gram (1,1)	0.68	0.32	0.41	0.31	0.43	0.64	0.79	0.77	0.69	0.50
	n-gram (1,2)	0.73	0.27	0.32	0.26	0.43	0.60	0.79	**0.79**	0.65	0.49
	n-gram (1,3)	0.74	0.19	0.35	0.23	0.30	0.60	0.73	0.75	0.62	0.44
	n-gram (1,4)	0.75	0.19	0.30	0.21	0.38	0.59	0.73	0.72	0.64	0.45
K-fold 10	n-gram (1,1)	0.67	**0.38**	0.41	0.31	0.45	0.72	0.82	0.76	**0.75**	**0.57**
	n-gram (1,2)	0.72	0.27	0.35	0.28	0.40	0.65	0.76	0.78	0.68	0.50
	n-gram (1,3)	0.74	0.21	0.35	0.24	0.45	0.63	0.76	0.75	0.66	0.44
	n-gram (1,4)	**0.77**	0.21	0.32	0.22	0.38	0.60	0.73	0.75	0.67	0.47

Validation	Tuning Parameter	F1 /Class									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.65	0.29	0.38	0.43	**0.6**	0.65	0.8	0.82	0.75	**0.67**
	n-gram (1,2)	0.61	0.25	0.48	**0.52**	0.29	0.73	**1.00**	0.80	0.70	0.53
	n-gram (1,3)	0.65	0.00	0.00	0.16	0.57	0.61	**1.00**	0.82	0.57	0.50
	n-gram (1,4)	0.69	0.00	0.00	0.00	0.29	0.58	**1.00**	0.72	0.53	0.51
K-fold 2-Test 33%	n-gram (1,1)	0.65	0.44	0.41	0.36	0.5	**0.79**	0.75	0.75	0.7	0.56
	n-gram (1,2)	0.69	0.20	0.31	0.15	0.47	0.57	0.92	0.80	0.70	0.43
	n-gram (1,3)	0.67	0.00	0.17	0.09	0.47	0.55	0.82	0.73	0.66	0.34
	n-gram (1,4)	0.69	0.00	0.09	0.17	0.15	0.44	0.62	0.59	0.54	0.39
K-fold 2-Test 20%	n-gram (1,1)	0.67	0.37	**0.61**	0.29	0.43	0.63	0.9	0.85	0.68	0.52
	n-gram (1,2)	0.64	0.09	0.47	0.14	0.53	0.48	0.78	0.76	0.67	0.47
	n-gram (1,3)	0.64	0.00	0.07	0.10	0.40	0.49	0.73	0.45	0.55	0.35
	n-gram (1,4)	0.66	0.00	0.14	0.06	0.19	0.44	0.71	0.63	0.49	0.34
K-fold 5	n-gram (1,1)	0.69	0.38	0.54	0.37	0.52	0.73	0.86	0.84	0.75	0.56
	n-gram (1,2)	0.71	0.39	0.45	0.37	0.53	0.73	0.87	**0.87**	0.70	0.58
	n-gram (1,3)	0.72	0.30	0.51	0.32	0.43	0.73	0.84	0.84	0.69	0.54
	n-gram (1,4)	0.73	0.28	0.46	0.30	0.49	0.70	0.84	0.82	0.71	0.54
K-fold 10	n-gram (1,1)	0.67	**0.45**	0.51	0.35	0.55	0.76	0.89	0.82	**0.78**	0.60
	n-gram (1,2)	0.71	0.40	0.49	0.38	0.50	0.75	0.84	0.84	0.74	0.58
	n-gram (1,3)	0.72	0.30	0.49	0.32	0.55	0.74	0.85	0.84	0.72	0.53
	n-gram (1,4)	**0.74**	0.30	0.45	0.32	0.47	0.72	0.82	0.83	0.74	0.56

Validation	Tuning Parameter	Number of classified samples										
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Total
K-fold 2-Test 50%	n-gram (1,1)	201	18	29	58	34	73	16	33	60	95	617
	n-gram (1,2)	209	21	26	64	25	63	14	41	55	107	625
	n-gram (1,3)	211	25	29	55	24	63	14	41	58	106	626
	n-gram (1,4)	212	22	27	64	28	67	18	38	60	85	621
K-fold 2-Test 33%	n-gram (1,1)	142	16	21	42	18	48	9	20	42	61	419
	n-gram (1,2)	134	18	27	48	13	43	7	23	38	64	415
	n-gram (1,3)	131	18	22	40	13	46	13	21	39	67	410
	n-gram (1,4)	135	13	21	27	24	48	11	30	38	65	412
K-fold 2-Test 20%	n-gram (1,1)	81	9	12	21	12	32	6	12	22	49	256
	n-gram (1,2)	90	14	16	18	9	27	3	15	25	37	254
	n-gram (1,3)	92	11	13	23	10	23	7	10	23	33	245
	n-gram (1,4)	98	8	13	20	5	33	4	16	25	36	258
K-fold 5	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248
K-fold 10	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248

Table D.15 Average Precision, Recall, F1-measures by Bernoulli Naïve Bayes for Multiclass Multi-label Classification

Validation	N-gram	Precision	Recall	F1
K-fold 2- Test 50%	n-gram (1,1)	0.36	0.12	0.15
	n-gram (1,2)	0.32	0.02	0.04
	n-gram (1,3)	0.34	0.00	0.00
	n-gram (1,4)	0.33	0.00	0.00
K-fold 2- Test 33%	n-gram (1,1)	0.32	**0.15**	**0.20**
	n-gram (1,2)	0.32	0.03	0.05
	n-gram (1,3)	0.33	0.00	0.00
	n-gram (1,4)	0.34	0.00	0.00
K-fold 2- Test 20%	n-gram (1,1)	**0.45**	0.13	0.17
	n-gram (1,2)	0.29	0.04	0.07
	n-gram (1,3)	0.32	0.01	0.02
	n-gram (1,4)	0.34	0.00	0.01
K-fold 5	n-gram (1,1)	0.36	**0.15**	0.19
	n-gram (1,2)	0.30	0.01	0.02
	n-gram (1,3)	0.12	0.00	0.00
	n-gram (1,4)	0.13	0.00	0.00
K-fold 10	n-gram (1,1)	0.38	**0.15**	**0.20**
	n-gram (1,2)	0.20	0.01	0.02
	n-gram (1,3)	0.10	0.00	0.01
	n-gram (1,4)	**0.45**	0.13	0.17

Table D.16 Precision, Recall, F1-measures per class by Bernoulli Naïve Bayes for Multiclass Multi-label Classification

Validation	Tuning Parameter	Precision									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.67	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	1.00
	n-gram (1,2)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 2-Test 33%	n-gram (1,1)	0.69	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.71
	n-gram (1,2)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 2-Test 20%	n-gram (1,1)	0.61	**0.00**	**0.00**	0.00	**0.00**	**1.00**	**0.00**	**0.00**	0.00	**1.00**
	n-gram (1,2)	0.91	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 5	n-gram (1,1)	0.74	**0.00**	**0.00**	0.00	**0.00**	0.22	**0.00**	**0.00**	0.00	0.55
	n-gram (1,2)	0.89	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.38	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.40	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 10	n-gram (1,1)	0.74	**0.00**	**0.00**	**0.10**	**0.00**	0.30	**0.00**	**0.00**	**0.10**	0.56
	n-gram (1,2)	0.60	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.30	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.18	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00

Validation	Tuning Parameter	Recall									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.37	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.01
	n-gram (1,2)	0.06	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.00	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.00	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 2-Test 33%	n-gram (1,1)	**0.43**	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	**0.09**
	n-gram (1,2)	0.09	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 2-Test 20%	n-gram (1,1)	0.34	**0.00**	**0.00**	0.00	**0.00**	**0.05**	**0.00**	**0.00**	0.00	0.03
	n-gram (1,2)	0.12	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.02	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 5	n-gram (1,1)	0.42	**0.00**	**0.00**	0.00	**0.00**	0.01	**0.00**	**0.00**	0.00	0.04
	n-gram (1,2)	0.03	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.00	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.00	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 10	n-gram (1,1)	**0.43**	**0.00**	**0.00**	**0.01**	**0.00**	0.02	**0.00**	**0.00**	**0.01**	0.05
	n-gram (1,2)	0.03	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00

Validation	Tuning Parameter	F1 /Class									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.48	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.02
	n-gram (1,2)	0.11	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 2-Test 33%	n-gram (1,1)	0.53	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	**0.16**
	n-gram (1,2)	0.17	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 2-Test 20%	n-gram (1,1)	0.44	**0.00**	**0.00**	0.00	**0.00**	**0.09**	**0.00**	**0.00**	0.00	0.05
	n-gram (1,2)	0.22	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.05	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.02	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 5	n-gram (1,1)	**0.54**	**0.00**	**0.00**	0.00	**0.00**	0.01	**0.00**	**0.00**	0.00	0.08
	n-gram (1,2)	0.06	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
K-fold 10	n-gram (1,1)	**0.54**	**0.00**	**0.00**	**0.01**	**0.00**	0.04	**0.00**	**0.00**	**0.02**	0.09
	n-gram (1,2)	0.06	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,3)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00
	n-gram (1,4)	0.01	**0.00**	**0.00**	0.00	**0.00**	0.00	**0.00**	**0.00**	0.00	0.00

Validation	Tuning Parameter	Number of classified samples										
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Total
K-fold 2-Test 50%	n-gram (1,1)	197	26	41	66	22	66	17	36	63	93	627
	n-gram (1,2)	201	23	42	56	25	64	16	38	57	104	626
	n-gram (1,3)	209	28	30	60	26	61	18	37	64	88	621
	n-gram (1,4)	204	21	37	60	21	63	20	40	63	96	625
K-fold 2-Test 33%	n-gram (1,1)	137	14	18	50	14	44	9	25	51	55	417
	n-gram (1,2)	132	14	21	40	20	40	12	24	38	69	410
	n-gram (1,3)	134	17	21	43	18	47	11	25	44	49	409
	n-gram (1,4)	143	19	26	40	8	41	12	23	41	64	417
K-fold 2-Test 20%	n-gram (1,1)	88	10	11	19	17	22	7	15	23	37	249
	n-gram (1,2)	80	10	17	19	14	27	8	19	23	35	252
	n-gram (1,3)	81	8	14	29	7	33	6	15	16	42	251
	n-gram (1,4)	86	9	16	26	9	25	4	11	27	38	251
K-fold 5	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248
K-fold 10	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248

Table D.17 Average Precision, Recall, F1-measures by Multinomial Naïve Bayes for Multiclass Multi-label Classification

Validation	N-gram	Precision	Recall	F1
K-fold 2- Test 50%	n-gram (1,1)	0.26	0.08	0.12
	n-gram (1,2)	0.31	0.03	0.05
	n-gram (1,3)	0.32	0.01	0.03
	n-gram (1,4)	0.28	0.02	0.03
K-fold 2- Test 33%	n-gram (1,1)	0.29	0.09	0.14
	n-gram (1,2)	0.32	0.03	0.06
	n-gram (1,3)	0.27	0.02	0.04
	n-gram (1,4)	0.20	0.03	0.04
K-fold 2- Test 20%	n-gram (1,1)	0.29	0.12	0.17
	n-gram (1,2)	0.34	0.06	0.10
	n-gram (1,3)	0.34	0.02	0.03
	n-gram (1,4)	0.27	0.02	0.03
K-fold 5	n-gram (1,1)	**0.39**	0.15	**0.20**
	n-gram (1,2)	0.33	0.10	0.15
	n-gram (1,3)	0.30	0.07	0.11
	n-gram (1,4)	0.32	0.08	0.12
K-fold 10	n-gram (1,1)	0.38	**0.16**	**0.20**
	n-gram (1,2)	0.32	0.10	0.15
	n-gram (1,3)	0.31	0.08	0.13
	n-gram (1,4)	0.32	0.07	0.12

Table D.18 Precision, Recall, F1-measures per class by Multinomial Naïve Bayes for Multiclass Multi-label Classification

Validation	Tuning Parameter	Precision									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.80	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.94	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.83	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 2-Test 33%	n-gram (1,1)	0.93	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.91	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.67	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 2-Test 20%	n-gram (1,1)	0.88	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 5	n-gram (1,1)	0.83	**0.00**	**0.00**	**0.00**	**0.00**	**0.79**	**0.00**	0.49	0.00	**0.00**
	n-gram (1,2)	0.92	**0.00**	**0.00**	**0.00**	**0.00**	0.21	**0.00**	0.11	0.00	**0.00**
	n-gram (1,3)	0.86	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.28	0.00	**0.00**
	n-gram (1,4)	0.93	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.23	0.00	**0.00**
K-fold 10	n-gram (1,1)	0.82	**0.00**	**0.00**	**0.00**	**0.00**	0.59	**0.00**	**0.53**	**0.15**	**0.00**
	n-gram (1,2)	0.92	**0.00**	**0.00**	**0.00**	**0.00**	0.07	**0.00**	0.15	0.00	**0.00**
	n-gram (1,3)	0.91	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.09	0.00	**0.00**
	n-gram (1,4)	0.95	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.12	0.00	**0.00**

Validation	Tuning Parameter	Recall									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.24	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.08	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.04	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.05	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 2-Test 33%	n-gram (1,1)	0.30	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.11	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.08	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.08	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 2-Test 20%	n-gram (1,1)	0.35	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.16	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.05	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.06	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 5	n-gram (1,1)	**0.44**	**0.00**	**0.00**	**0.00**	**0.00**	0.03	**0.00**	**0.07**	0.00	**0.00**
	n-gram (1,2)	0.28	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.01	0.00	**0.00**
	n-gram (1,3)	0.20	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.01	0.00	**0.00**
	n-gram (1,4)	0.23	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.01	0.00	**0.00**
K-fold 10	n-gram (1,1)	**0.44**	**0.00**	**0.00**	**0.00**	**0.00**	**0.05**	**0.00**	0.05	**0.01**	**0.00**
	n-gram (1,2)	0.30	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.03	0.00	**0.00**
	n-gram (1,3)	0.24	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.01	0.00	**0.00**
	n-gram (1,4)	0.22	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.01	0.00	**0.00**

Validation	Tuning Parameter	F1 /Class									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.37	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.14	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.08	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.09	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 2-Test 33%	n-gram (1,1)	0.45	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.19	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.15	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.15	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 2-Test 20%	n-gram (1,1)	0.50	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,2)	0.28	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,3)	0.09	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
	n-gram (1,4)	0.11	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	**0.00**
K-fold 5	n-gram (1,1)	**0.57**	**0.00**	**0.00**	**0.00**	**0.00**	0.06	**0.00**	**0.11**	0.00	**0.00**
	n-gram (1,2)	0.43	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.02	0.00	**0.00**
	n-gram (1,3)	0.33	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.03	0.00	**0.00**
	n-gram (1,4)	0.36	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.02	0.00	**0.00**
K-fold 10	n-gram (1,1)	**0.57**	**0.00**	**0.00**	**0.00**	**0.00**	**0.08**	**0.00**	0.10	**0.02**	**0.00**
	n-gram (1,2)	0.45	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.05	0.00	**0.00**
	n-gram (1,3)	0.38	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.02	0.00	**0.00**
	n-gram (1,4)	0.35	**0.00**	**0.00**	**0.00**	**0.00**	0.00	**0.00**	0.02	0.00	**0.00**

Validation	Tuning Parameter	Number of classified samples										
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Total
K-fold 2-Test 50%	n-gram (1,1)	199	28	35	59	26	69	15	35	56	97	619
	n-gram (1,2)	208	30	30	59	31	60	16	32	58	106	630
	n-gram (1,3)	203	21	31	68	26	59	17	45	55	103	628
	n-gram (1,4)	208	18	31	59	20	72	18	40	56	95	617
K-fold 2-Test 33%	n-gram (1,1)	128	14	18	47	12	42	10	25	46	69	411
	n-gram (1,2)	132	15	18	35	22	45	9	25	45	68	414
	n-gram (1,3)	123	24	27	38	12	45	17	24	37	63	410
	n-gram (1,4)	119	15	21	39	26	47	12	21	38	62	400
K-fold 2-Test 20%	n-gram (1,1)	84	7	11	26	14	34	8	13	21	34	252
	n-gram (1,2)	85	12	6	20	10	27	10	17	25	37	249
	n-gram (1,3)	85	10	14	18	17	19	5	16	29	36	249
	n-gram (1,4)	68	9	17	23	13	24	5	18	27	46	250
K-fold 5	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248
K-fold 10	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248

Table D.19 Average Precision, Recall, F1-measures by Logistic Regression for Multiclass Multi-label Classification

Validation	N-gram	Precision	Recall	F1
K-fold 2- Test 50%	n-gram (1,1)	0.53	0.13	0.19
	n-gram (1,2)	0.29	0.07	0.12
	n-gram (1,3)	0.29	0.05	0.08
	n-gram (1,4)	0.33	0.04	0.07
K-fold 2- Test 33%	n-gram (1,1)	**0.70**	0.17	0.23
	n-gram (1,2)	0.33	0.11	0.15
	n-gram (1,3)	0.30	0.08	0.13
	n-gram (1,4)	0.26	0.06	0.10
K-fold 2- Test 20%	n-gram (1,1)	0.68	0.15	0.22
	n-gram (1,2)	0.28	0.11	0.16
	n-gram (1,3)	0.22	0.08	0.12
	n-gram (1,4)	0.30	0.06	0.10
K-fold 5	n-gram (1,1)	0.69	0.22	0.29
	n-gram (1,2)	0.61	0.18	0.24
	n-gram (1,3)	0.56	0.17	0.23
	n-gram (1,4)	0.56	0.16	0.22
K-fold 10	n-gram (1,1)	0.65	**0.24**	**0.31**
	n-gram (1,2)	0.57	0.20	0.27
	n-gram (1,3)	0.46	0.18	0.24
	n-gram (1,4)	0.47	0.18	0.24

Table D.20 Precision, Recall, F1-measures per class by Logistic Regression for Multiclass Multi-label Classification

Validation	Tuning Parameter	Precision									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.83	**0.00**	**0.00**	0.00	**0.00**	**1.00**	0.00	**1.00**	**1.00**	0.00
	n-gram (1,2)	0.90	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.85	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	**1.00**	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 2-Test 33%	n-gram (1,1)	0.85	**0.00**	**0.00**	0.00	**0.00**	**1.00**	0.00	**1.00**	**1.00**	**1.00**
	n-gram (1,2)	0.72	**0.00**	**0.00**	0.00	**0.00**	**1.00**	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.89	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.83	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 2-Test 20%	n-gram (1,1)	0.82	**0.00**	**0.00**	0.00	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.67
	n-gram (1,2)	0.85	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.68	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.88	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 5	n-gram (1,1)	0.83	**0.00**	**0.00**	0.00	**0.00**	**1.00**	0.76	**1.00**	**1.00**	0.86
	n-gram (1,2)	0.85	**0.00**	**0.00**	0.00	**0.00**	0.78	0.18	**1.00**	0.58	0.81
	n-gram (1,3)	0.81	**0.00**	**0.00**	0.00	**0.00**	0.62	0.21	**1.00**	0.40	0.79
	n-gram (1,4)	0.83	**0.00**	**0.00**	0.00	**0.00**	0.58	0.30	**1.00**	0.18	0.90
K-fold 10	n-gram (1,1)	0.84	**0.00**	**0.00**	**0.07**	**0.00**	0.84	0.67	0.95	0.87	0.82
	n-gram (1,2)	0.83	**0.00**	**0.00**	0.00	**0.00**	0.64	0.27	0.93	0.63	0.68
	n-gram (1,3)	0.82	**0.00**	**0.00**	0.00	**0.00**	0.51	0.42	0.67	0.34	0.35
	n-gram (1,4)	0.84	**0.00**	**0.00**	0.00	**0.00**	0.43	0.15	0.69	0.34	0.48

Validation	Tuning Parameter	Recall									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.39	**0.00**	**0.00**	0.00	**0.00**	0.03	0.00	0.05	0.02	0.00
	n-gram (1,2)	0.23	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.13	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.11	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 2-Test 33%	n-gram (1,1)	0.46	**0.00**	**0.00**	0.00	**0.00**	0.04	0.00	0.12	0.03	0.02
	n-gram (1,2)	0.37	**0.00**	**0.00**	0.00	**0.00**	0.02	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.24	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.19	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 2-Test 20%	n-gram (1,1)	0.34	**0.00**	**0.00**	0.00	**0.00**	0.07	**0.33**	0.09	0.04	**0.11**
	n-gram (1,2)	0.34	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.25	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.17	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 5	n-gram (1,1)	0.50	**0.00**	**0.00**	0.00	**0.00**	0.10	0.18	0.25	0.08	0.08
	n-gram (1,2)	0.46	**0.00**	**0.00**	0.00	**0.00**	0.05	0.03	0.20	0.04	0.05
	n-gram (1,3)	0.44	**0.00**	**0.00**	0.00	**0.00**	0.03	0.03	0.23	0.02	0.03
	n-gram (1,4)	0.43	**0.00**	**0.00**	0.00	**0.00**	0.04	0.03	0.16	0.01	0.03
K-fold 10	n-gram (1,1)	**0.54**	**0.00**	**0.00**	**0.01**	**0.00**	**0.11**	0.21	**0.28**	**0.11**	0.08
	n-gram (1,2)	0.49	**0.00**	**0.00**	0.00	**0.00**	0.07	0.12	0.20	0.06	0.06
	n-gram (1,3)	0.47	**0.00**	**0.00**	0.00	**0.00**	0.04	0.06	0.21	0.03	0.04
	n-gram (1,4)	0.47	**0.00**	**0.00**	0.00	**0.00**	0.04	0.03	0.19	0.03	0.03

Validation	**Tuning Parameter**	**F1 /Class**									
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
K-fold 2-Test 50%	n-gram (1,1)	0.53	**0.00**	**0.00**	0.00	**0.00**	0.06	0.00	0.10	0.03	0.00
	n-gram (1,2)	0.36	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.23	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.21	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 2-Test 33%	n-gram (1,1)	0.59	**0.00**	**0.00**	0.00	**0.00**	0.08	0.00	0.22	0.05	0.03
	n-gram (1,2)	0.49	**0.00**	**0.00**	0.00	**0.00**	0.04	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.38	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.31	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 2-Test 20%	n-gram (1,1)	0.48	**0.00**	**0.00**	0.00	**0.00**	0.12	**0.50**	0.17	0.07	**0.19**
	n-gram (1,2)	0.49	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,3)	0.37	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
	n-gram (1,4)	0.29	**0.00**	**0.00**	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00
K-fold 5	n-gram (1,1)	0.62	**0.00**	**0.00**	0.00	**0.00**	0.17	0.28	0.39	0.14	0.15
	n-gram (1,2)	0.58	**0.00**	**0.00**	0.00	**0.00**	0.10	0.05	0.32	0.08	0.10
	n-gram (1,3)	0.57	**0.00**	**0.00**	0.00	**0.00**	0.06	0.05	0.35	0.03	0.06
	n-gram (1,4)	0.56	**0.00**	**0.00**	0.00	**0.00**	0.07	0.05	0.26	0.02	0.05
K-fold 10	n-gram (1,1)	**0.65**	**0.00**	**0.00**	**0.01**	**0.00**	**0.19**	0.29	**0.42**	**0.19**	0.14
	n-gram (1,2)	0.61	**0.00**	**0.00**	0.00	**0.00**	0.13	0.17	0.32	0.11	0.11
	n-gram (1,3)	0.60	**0.00**	**0.00**	0.00	**0.00**	0.08	0.11	0.32	0.06	0.06
	n-gram (1,4)	0.60	**0.00**	**0.00**	0.00	**0.00**	0.07	0.05	0.27	0.05	0.06

Validation	Tuning Parameter	Number of classified samples										
		Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Total
K-fold 2-Test 50%	n-gram (1,1)	202	22	31	58	28	63	21	40	59	103	627
	n-gram (1,2)	198	21	29	57	27	66	19	37	58	104	616
	n-gram (1,3)	215	25	34	55	30	69	18	29	56	99	630
	n-gram (1,4)	209	24	36	50	30	62	17	41	58	99	626
K-fold 2-Test 33%	n-gram (1,1)	134	18	11	40	22	46	12	24	40	61	408
	n-gram (1,2)	125	24	16	37	17	46	11	23	44	66	409
	n-gram (1,3)	136	19	18	31	12	52	14	30	31	68	411
	n-gram (1,4)	132	24	17	44	14	52	11	21	42	62	419
K-fold 2-Test 20%	n-gram (1,1)	80	8	10	18	16	30	3	22	26	37	250
	n-gram (1,2)	82	11	11	24	8	34	5	19	17	35	246
	n-gram (1,3)	83	9	16	26	14	31	4	8	27	35	253
	n-gram (1,4)	86	9	14	24	13	22	8	16	26	32	250
K-fold 5	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248
K-fold 10	n-gram (1,1)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,2)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,3)	412	48	63	121	53	134	33	75	116	193	1248
	n-gram (1,4)	412	48	63	121	53	134	33	75	116	193	1248

Table D.21 Precision, Recall, F1-measures for Multiclass Multi-label Classification using Class-Reduced Approach

			SVM Linear				**SVM Multinomial**			
			Precision	Recall	F1 Scores	Support	Precision	Recall	F1 Scores	Support
K-fold 2- Test 20%	n-gram(1,1)	Class 0	0.81	0.84	0.83	90	0.84	**0.90**	**0.87**	92
		Class 1	0.88	0.70	0.78	33	**0.96**	0.71	**0.81**	31
		Class 2	**1.00**	0.33	0.5	3	**1.00**	**0.83**	**0.91**	6
		Class 3	**1.00**	0.82	0.9	17	**1.00**	0.64	0.78	11
		Class 4	**0.91**	**0.77**	**0.83**	26	0.86	**0.78**	**0.82**	23
		Average	0.86	**0.79**	0.82	169	0.88	0.83	**0.85**	163
	n-gram(1,2)	Class 0	0.80	**0.88**	0.84	85	0.75	0.86	0.80	80
		Class 1	**0.95**	0.59	0.73	32	0.95	0.57	0.71	37
		Class 2	**1.00**	0.67	0.80	3	**1.00**	0.43	0.60	7
		Class 3	**1.00**	**0.84**	**0.91**	19	0.91	0.67	0.77	15
		Class 4	0.80	0.59	0.68	27	0.88	0.71	0.79	31
		Average	0.85	0.77	0.80	166	0.84	0.74	0.77	170
K-fold 5	n-gram(1,1)	Class 0	0.85	0.81	0.83	440	0.85	0.81	0.83	440
		Class 1	0.88	**0.74**	**0.80**	148	0.88	**0.74**	0.80	148
		Class 2	**1.00**	**0.79**	**0.87**	33	**1.00**	0.79	0.87	33
		Class 3	0.95	0.76	0.85	80	0.95	0.75	0.84	80
		Class 4	0.83	0.74	0.78	129	0.83	0.74	0.78	129
		Average	0.86	0.78	0.82	830	0.86	0.78	0.82	830
	n-gram(1,2)	Class 0	0.84	0.85	**0.85**	440	0.84	0.85	0.85	440
		Class 1	0.94	0.67	0.78	148	0.94	0.67	0.78	148
		Class 2	**1.00**	**0.79**	**0.87**	33	**1.00**	0.79	0.87	33
		Class 3	0.98	0.75	0.85	80	0.98	0.75	**0.85**	80
		Class 4	0.81	0.64	0.71	129	0.81	0.64	0.72	129
		Average	0.88	0.77	0.81	830	0.88	0.77	0.82	830
K-fold 10	n-gram(1,1)	Class 0	0.86	0.83	0.84	440	**0.86**	0.83	0.84	440
		Class 1	0.90	0.70	0.77	148	0.90	0.70	0.77	148
		Class 2	0.94	0.79	0.85	33	0.94	0.79	0.85	33
		Class 3	0.95	0.76	0.84	80	0.95	**0.76**	0.84	80
		Class 4	0.85	0.76	0.80	129	0.85	0.76	0.80	129
		Average	0.88	0.79	0.82	830	0.88	0.79	0.82	830
	n-gram(1,2)	Class 0	0.85	0.84	0.84	440	0.85	0.84	0.84	440
		Class 1	0.95	0.70	0.80	148	0.95	0.70	0.80	148
		Class 2	0.94	0.79	0.85	33	0.94	0.79	0.85	33
		Class 3	0.98	0.76	0.85	80	0.98	**0.76**	0.85	80
		Class 4	0.89	0.72	0.79	129	**0.89**	0.72	0.79	129
		Average	**0.89**	0.79	**0.83**	830	**0.89**	0.79	0.83	830

Table D.22 Precision, Recall, F1-measures for Multiclass Multi-label Classification using Class-Reduced Approach on External Simulated Dataset

			SVM Linear				SVM Multinomial			
			Precision	Recall	F1 Scores	Support	Precision	Recall	F1 Scores	Support
K-fold 2- Test 20%	n-gram(1,1)	Class 0	**0.89**	0.78	0.83	41	0.83	0.73	0.78	41
		Class 1	**1.00**	0.45	0.62	22	0.93	**0.59**	**0.72**	22
		Class 2	**1.00**	**1.00**	**1.00**	1	**1.00**	**1.00**	**1.00**	1
		Class 3	0.80	**1.00**	0.89	4	**1.00**	**1.00**	**1.00**	4
		Class 4	0.88	0.64	0.74	11	0.78	0.64	0.70	11
		Average	**0.91**	0.68	0.77	79	0.86	0.70	0.77	79
	n-gram(1,2)	Class 0	0.79	0.80	0.80	41	0.83	0.83	0.83	41
		Class 1	**1.00**	0.27	0.43	22	**1.00**	0.45	0.62	22
		Class 2	**1.00**	**1.00**	**1.00**	1	**1.00**	**1.00**	**1.00**	1
		Class 3	**1.00**	0.75	0.86	4	**1.00**	**1.00**	**1.00**	4
		Class 4	**1.00**	0.55	0.71	11	**0.91**	**0.91**	**0.91**	11
		Average	0.89	0.62	0.69	79	**0.90**	0.75	0.79	79
K-fold 5	n-gram(1,1)	Class 0	0.86	0.90	**0.88**	41	**0.86**	**0.90**	**0.88**	41
		Class 1	**1.00**	**0.55**	**0.71**	22	**1.00**	0.55	0.71	22
		Class 2	**1.00**	**1.00**	**1.00**	1	**1.00**	**1.00**	**1.00**	1
		Class 3	0.57	**1.00**	0.73	4	0.57	**1.00**	0.73	4
		Class 4	0.91	**0.91**	**0.91**	11	**0.91**	**0.91**	**0.91**	11
		Average	0.89	**0.81**	**0.83**	79	0.89	**0.81**	**0.83**	79
	n-gram(1,2)	Class 0	0.84	0.93	**0.88**	41	0.83	0.85	0.84	41
		Class 1	**1.00**	0.50	0.67	22	0.92	0.55	0.69	22
		Class 2	**1.00**	**1.00**	**1.00**	1	**1.00**	**1.00**	**1.00**	1
		Class 3	**1.00**	**1.00**	**1.00**	4	0.67	**1.00**	0.80	4
		Class 4	0.82	0.82	0.82	11	0.88	0.64	0.74	11
		Average	0.89	0.80	0.82	79	0.86	0.75	0.78	79
K-fold 10	n-gram(1,1)	Class 0	0.83	0.85	0.84	41	0.83	0.85	0.84	41
		Class 1	0.92	**0.55**	0.69	22	0.92	0.55	0.69	22
		Class 2	**1.00**	**1.00**	**1.00**	1	**1.00**	**1.00**	**1.00**	1
		Class 3	0.67	**1.00**	0.80	4	0.67	**1.00**	0.80	4
		Class 4	0.88	0.64	0.74	11	0.88	0.64	0.74	11
		Average	0.86	0.75	0.78	79	0.86	0.75	0.78	79
	n-gram(1,2)	Class 0	0.82	0.90	0.86	41	0.82	**0.90**	0.86	41
		Class 1	**1.00**	0.50	0.67	22	**1.00**	0.50	0.67	22
		Class 2	**1.00**	**1.00**	**1.00**	1	**1.00**	**1.00**	**1.00**	1
		Class 3	0.80	**1.00**	0.89	4	0.80	**1.00**	0.89	4
		Class 4	0.80	0.73	0.76	11	0.80	0.73	0.76	11
		Average	0.87	0.77	0.80	79	0.87	0.77	0.80	79

Table D.23 Results of Feature Requirement Ranking per Category by Jira over the observed period

Category for Feature Requirements (Jira)	Extracted Feature Requirements[2]	Frequency Ranking Score
Board management	Sentence 1: Performance was not the best, especially when the backlog was large and there were lots of projects loaded into our instance.; It is easy to set up, a bit complicated on the administration but a bit troublesome on the performance side if you really start to create scores of projects on it with large backlogs throughout.	291.234
	Once you open a Service Ticket aka Fault Record you cannot change certain fields - if this has been created correctly, with the exception of the service object (in our case), you still have to close it and create and entirely new FR.	245.007
	Tasks take inherently longer to put in, don't update easily based on due date, it's challenging to 'snooze' tasks, and inevitably tasks muck up my sprint and product backlogs.	235.016
User management and permission	Following can be improved. 1. Assign a JIRA Issue/Enhancement to multiple people. 2. Track the Status with Timeline (like a graph indicating when a praticular issue is created, assigned, worked upon, resolved, Tested and Implemented in Stage and then Prod, Closed).	361.133
	We require multiple approvals based on change type, etc and we haven't found anything in JIRA yet that fully supports what we need.	80.800
	No dislikes - more like tweak requests - add a little bit more permission schemes around workflows.	78.000
Customization	Admin Pitfall #1: while JIRA does in fact allow administrators the ability to build customized issue types, workflows and artifacts, JIRA's core functionality isn't designed or delivered to support all of the customized artifacts that end up being created.	355.833
	It is the missing feature to custom the email templates for notifications.; It's also missing the feature to have a default description and steps to reproduce template.	150.133
	Some of that difficulty can come from making this more complicated than they need to be via customization, but there are a handful of situations where you will be looking for the ability to customize a behavior and be stopped cold in your tracks.	133.792
Infrastructural design	The overview page uses up wayyyyy too much space listed stories, and when you click on a story for an overview it uses a small vertical sliver to display the story.; So much screen space is used to show small	523.533

[2] Feature requirements here are extracted from pure online product reviews; the content might contain mistakes in English grammar, misspelling, etc.

	story titles when it would be better to have more space for the expanded description and overview.	
	The two minor pain points are that cases open in a right bar when using a sprint board (I'd rather they automatically open in new tabs); and I often have issues pasting in unordered lists -- I formats the line spacing wrong almost every time.	385.667
	Need to improve the interface at times for kan ban .. sprint interfaces tracking and also looking at time estimates overall to reflect those takss that team members have added.	222.500
Configuration and maintenance	Also, it has a huge learning and configuration curve, it takes a long time to understand/administrate the system and chances are that users end up experiencing traumatic heisenbugs.	199.893
	Installation of the on-premise version is very straightforward on a Windows install, but a Linux install had some really weird anomalies that took multiple support tickets to resolve.	169.833
	It's possible we don't have it configured properly to match our development process so this may be a human error sort of thing but an intuitive tool shouldn't take an expert to set up and figure out.	162.667
Data management, visualization and integration	Also we used to host JIRA/Confluence on the same server that we had for a few other apps, and the processes for these two software were so heavy that would render the server useless for other software, on a quad-core machine.	148.400
	This may be how we have JIRA set up but I find reporting on time difficult, we have to manually adjust the remaining time on tasks to be able to get the burndown chart working as we want it.	144.000
	It is not well integrated with a knowledge management system, and Confluene requires too many touch points to manage the knowledge produce during sprints.	131.000
Mobile services	I am also not a fan of the mobile version of the site as it has limited functionality compared to the full site, and issues show reduced information. Keyword scores:	147.850
	license fees can be reduced and it can be made avaliable in mobile and tab and other devices to be used so that more people can access across devices.	94.850
	The iPhone app seems limited compared to the web version, which is not uncommon for iPhone versions of various software.	89.850
Notifications	One huge downside that I have realized is that when you work on a large scale program, the ALM Messaging alerts become annoying and then irrelevant.	133.167
	Many users have so many messages on a daily basis that they have completely stopped using this feature, which is very unfortunate, because if these problems were addressed, it would be very valuable. Keyword scores:	116.667
	Unfortunately there were consistent bugs - most importantly emails that would be consumed by Jira	115.783

	with no ticket or notification, causing tickets to go completely unaddressed.	
Navigation and search	I'm not sure why, since the search tools are pretty robust, but I think this is caused by the fact that if you have dozens of users submitting buys/enhancements, they often use slightly different terms to describe projects, making keyword searches less helpful.	287.733
	Some of the navigation is a little clunky like when you have that preview mode to see a list of your tickets and then a column to the right that has more information about the ticket, but I kind of wish that when you clicked the ticket number, it opened in a new tab so you could still reference your list of tickets.	240.267
	Sentence 3: Finding pages and navigating page trees can be confusing if you are not exactly sure where things are housed, it would be nice for the flow to be a little more logical to someone who is not experienced with using this type of platform.	168.833
Others	Jira suffers a bit in comparison to VersionOne and other already well-established project tracking tools because of having limited capability focused on issue tracking and offering little in the way of managing projects overall.	178.763
	Versions cannot be confined to individual projects (as of release 7.1.x); one sees versions from several projects when selecting a Fix Version or Affected Version(s) for an issue.	169.583
	- Don't force me to use the agile terminology/constrict me to strict agile methods, every company/team does things a little differently and it hinders workflow to be stopped by JIRA.	154.000

Table D.24 Results of Feature Requirement Ranking per Category by Trello over the observed period

Category for Feature Requirements (Trello)	Extracted Feature Requirements[3]	Frequency Ranking Score
Board management	Trello needs the ability to track due dates on multiple stages of the task or 'card'; the current implementation has one 'due date' field that must either be reused over and over in multi-step tasks or set up as a fixed due date for the whole of the project, which can lead to lack of accountability until the due date is looming.	481.288
	Also would have been nice if there were a way to look at tasks differently, not just by lists but by status or some other dimension added to them.. the vertical lists work great most of the time, but other times there's a need to see stuff in a bit more of a "bird's eye" overview and Trello is pretty missing this out.	266.111
	Even though it's really easy to add a card that only has a title, adding additional information to that card (like labels, adding contributors, and a description) is clunky and doesn't match the simplicity of other aspects of the tool.	221.178
User management and permission	Sentence 1: It's a bit too simple and lacks various power user capabilities: labeling is nice but too simple and having labels "move" between boards makes them prone to "labels overload".	184.667
	I dislike how you cannot assign a person to a list item on the check list (you should be able to assign more than one person if you wish as this is true collaboration and task tracking).	119.500
	I did noticed some privacy issues, things I didn't want to share with entire team went out, but then again it might be my fault, I needed to check settings.	116.667
Customization	For example if I'm thinking through a new Sales flow, but I'm getting caught up on the best way to lay it out, it would be great to just use template to guide me through.	64.000
	It lacks filtering or customizations of what you see, when you have a lot of cards on your board you have to scroll a lot to see the cards.	64.000
	I only use Trello for the same purpose (whether it's personal or business) and would prefer for a way to easily utilize templates.	47.000
Infrastructural design	When creating lists, it would be nice sometimes to have the option to set up bullet points or some other list order within a list box.; It would also be nice to have some font size options to utilize within each list.	199.433
	We try to put structure in place, but the lack of system-level structure (E.G. required fields or default sort orders for cards) makes this challenging.	183.250

[3] Feature requirements here are extracted from pure online product reviews; the content might contain mistakes in English grammar, misspelling, etc.

	My guess is MOST users just use the standard checklist name, so why not eliminate the extra click to name the checklist and throw me directly into the checklist once I add it from the card?	141.000
Data management, visualization and integration	There are some key features that would be nice to have. Full gmail integration for multiple users and the ability to set more than 1 due date, more milestones than due dates.	173.604
	Bottom line was it lacked the adequate reporting tools/metrics we were looking for and needed to dive deeper into those features of a project management app.	155.917
	We found the Sunrise tool that mirrors the look and feel of Google Calendars, but it doesn't pull in any of the color tags or coding for the cards of each board.	145.300
Mobile services	One thing that could use a change is the inability to edit my comments on the Android mobile application side (suggestion: press and hold).	136.667
	On Trello's mobile app, I am unable to download attached photos directly from the site or customize which boards I receive notifications from.	135.467
	Don't have a complete offline client and sometimes on mobile the drag and drop doesn't work well on the move (probably due to data connection issue).	131.667
Notifications	Unfortunately, I have to check into the website (or my Android app) in order to see whether certain tasks have moved forward - would love an e-mail summary.	136.900
	I really do miss being able to have better use of due dates - I'd love to get reminders on tasks, or be able to assign things to set times.	108.733
	-Also, the email notifications can be hard to sort through, especially if you have multiple boards with multiple users and cards.	89.625
Navigation and search	I understand the sprint/scrum nature of the design but it'd be nice to be able to filter out irrelevant tasks until I'm ready to either be assigned to them or not at all.	94.000
	The web version on the pc has navigation issues as the cursor unexpectedly moves the entire page.	93.500
	While basic information is easy to find, too many clicks are required to find and input additional information.	87.929
Others	I find it very useful at the beginning of a project to organize high level ideas but then I switch to a more powerful story tracking tool once user stories are created and honed.	228.383
	The flexibility is great but also an issue at the same time as the tool right now is lacking basic "lock-down" features.; It is also lacking basic PM tools for larger companies and complex projects.	223.799
	As long you don't expect Trello to be a full program /project management system and realize that it is a great visual Kanban board system then there aren't really too many cons to the product.	175.014