A Natural Language Processing Model to Improve the Software Testing Process Under an Agile Methodology

by Hamed Farahanchi

B.S. in Computer Science, May 2008, Saint Mary's University
M.S. in Software Engineering, May 2020, California State University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

January 6, 2023

Praxis directed by

Rebecca Yassan
Professorial Lecturer of Engineering Management and Systems Engineering

The School of Engineering and Applied Science of The George Washington University certifies that Hamed Farahanchi has passed the Final Examination for the degree of Doctor of Engineering as of January 6, 2023. This is the final and approved form of the Praxis.

A Natural Language Processing Model to Improve the Software Testing Process Under an Agile Methodology

Hamed Farahanchi

Praxis Research Committee:

Rebecca Yassan, Professorial Lecturer in Engineering and Applied Science, Praxis Director

Enrique Campos-Nanez, Professorial Lecturer in Engineering and Applied Science, Committee Member

Ari Dimitriou, Professorial Lecturer in Engineering and Applied Science, Committee Member

© Copyright 2022 by Hamed Farahanchi
All rights reserved

Dedication

The author wishes to dedicate this effort, and all of my work to my wife who has been very supportive of me in all of my endeavors throughout the years, especially during the past five years of my master's degree and doctoral program. You have stood by me and given me the time necessary to pursue this dream. I thank you for your love and support allowing me to concentrate on this program. I owe you a lot of time in return now that this journey is complete.

To my uncle, he has been a never-ending source of encouragement, inspiration, motivation, and suggestions, and for always being there for me.

To my children, for all their love and give me more time to work on this praxis.

Finally, to Woman, Life, Freedom movement which is happening in Iran. For their bravery and courage who have inspired the world. They are sacrificing their precious lives to stand up for their rights and freedoms. Their pain is everyone's pain, as Saadi Shirazi famous Persian poet says:

"Human beings are body parts of each other,

In creation they are indeed of one essence.

If a body part is afflicted with pain,

Other body parts uneasy will remain.

If you have no sympathy for human pain,

The name of human you shall not retain."

Acknowledgements

The author wishes to acknowledge the generous support, feedback, and mentorship of Dr. Rebecca Yassan. Her valuable recommendations and support made it possible for me to stay on track and write and complete this praxis on time.

I would also like to express my deep gratitude to Dr. Etemadi for his valuable instructions, and academic expertise which has been a great propeller to perfecting the content of this research.

Also, thanks to my two helpful colleagues, Sandeep Adimadhyam and Wiramen Chokry for their unconditional help and support.

Lastly, but certainly not least, thanks to Dr. Sarkani, Dr. Mazzuchi, and to all the professors at The George Washington University, and the students in my cohort.

Abstract of Praxis

A Natural Language Processing Model to Improve the Software Testing Process Under an Agile Methodology

When verifying the quality of a software product, software testing is the most important phase of the software development life cycle. In 2020, the cost of poor software quality was estimated to be $2.08 trillion in the United States. When using an agile methodology, the software development life cycle is short and fast. When there are any cuts in a project to reduce or maintain the budget, software testing usually is the first step to be impacted, which can cause issues in both the budget and the quality of the software.

This praxis presents a machine learning model by developing a natural processing algorithm to predict the classification of the required type of testing for each agile software requirement user story. The software project data set was obtained from a large American medical device company. Five hundred user stories were selected for this praxis from a software development team utilizing the agile methodology.

This praxis demonstrates how the term frequency by inverse document frequency algorithm can effectively and efficiently improve the software testing process. The algorithm achieves this by processing the number of times a word appears in a user story requirement divided by the total number of words in the user story and calculating the weight of rare words across all datasets. This research establishes a benchmark for the software industry to measure software testing time and cost and to improve methods in the early stages of the software development life cycle.

Table of Contents

Dedication ..... **iv**

Acknowledgements ..... **v**

Abstract of Praxis ..... **vi**

List of Figures ..... **xi**

List of Tables ..... **xii**

List of Equation ..... **xiii**

List of Acronyms ..... **xiv**

Glossary of Terms ..... **xvii**

Chapter 1 —Introduction 1

1.1 Background 1

1.2 Research Motivation 1

1.3 Problem Statement 2

1.4 Thesis Statement 3

1.5 Research Objectives 3

1.6 Research Questions and Hypotheses 4

1.7 Scope of Research 5

1.8 Research Limitations 5

1.9 Organization of Praxis 6

Chapter 2 —Literature Review 7

2.1 Introduction 7

2.2 Success Factors in Software Testing During Agile Development 8

2.2.1 Customer Collaboration 8

2.2.2 Agile Testing Mindset 9
2.2.3 Agile Automation Regression Testing 10
2.3 Agile Development and Scheduling Methods 10
2.4 Software Testing Time 11
2.4.1 Historical Data Method 12
2.4.2 Fuzzy Model 13
2.5 Machine Learning (ML) Forecasting Methods 14
2.5.1 Natural Language Processing (NLP) 15
2.5.2 Decision Tree (DT) and Random Forest (RF) Methods 17
2.5.3 K-Nearest Neighbor (KNN) Method 18
2.6 Summary and Conclusion 19
Chapter 3 —Methodology 21
3.1 Introduction 21
3.2 Data Source Collection 21
3.2.1 Data Analysis Tools 23
3.3. Data Preparation 24
3.3.1 Punctuation Removal 25
3.3.2 Stop Words Removal 25
3.3.3 Tokenization 26
3.3.4 Lemmatization 27
3.4 Vectorization and Ranking 28
3.4.1 Term Frequency (TF) 29
3.4.2 Inverse Document Frequency (IDF) 30

3.4.3 Term Frequency–Inverse Document Frequency (TD-IDF) 31
3.5 Model Classification 32
3.6 Model Evaluation 32
3.6.1 Confusion Matrix 33
3.7 Model Validation 40
Chapter 4 —Results 42
4.1 Introduction 42
4.2 Feature Engineering (FE) 42
4.3 Model Evaluation and Ranking 44
4.3.1 Corpus Vectorization and Ranking Evaluation 44
4.4 Hypothesis 1 and Results 45
4.5 Hypothesis 2 and Results 47
4.6 Hypothesis 3 and Results 54
4.6.1 Historical Agile User Story Classification Data Comparison 54
Chapter 5 — Discussion and Conclusions 56
5.1 Discussion 56
5.1.1 Discussion of Hypothesis 1 57
5.1.2 Discussion of Hypothesis 2 57
5.1.3 Discussion of Hypothesis 3 58
5.2 Conclusion 58
5.2.1 Lessons Learned 60
5.3 Contributions to the Body of Knowledge 61
5.4 Recommendations for Future Research 62

References 63

Appendix A 74

List of Figures

Figure 3.1. Text Processing for Machine Learning 24
Figure 3.2. Confusion Matrix with Misclassification Error Type (Explorium, n.d.) 33
Figure 4.1. Confusion Matrix Output From the Model 51

List of Tables

Table 2.1. ML Forecasting Frequency 15
Table 3.1. Stop words Removal 26
Table 3.2. Documents Vs. Tokens 27
Table 3.3. Lemmatizing the Word without Considering the Type (POS Tag) 28
Table 3.4. Lemmatizing the Word with Considering the Type (POS Tag) 28
Table 3.5. Model Level Example 34
Table 3.6. Class Level Evaluation Example 35
Table 4.1. Sample of Feature Engineering Results 43
Table 4.2. Unique Words Identified in Corpus 44
Table 4.3. Sample of Corpus Vectorization and Ranking Evaluation 45
Table 4.4. Sample of Source Vectorization & Ranking Evaluation 45
Table 4.5. Expected Vs. Identified Communication Testing Keywords 46
Table 4.6. Expected Vs. Identified Security Testing Keywords 46
Table 4.7. Expected Vs. Identified Surgical Testing Keywords 46
Table 4.8. Expected Vs. Identified System Testing Keywords 47
Table 4.9. Expected Type of Classification 47
Table 4.10. Sample Results of Predicted Vs. Actual Category Comparison 50
Table 4.11. Confusion Matrix Statistical Results 53
Table 4.12. Confusion Matrix Result with Error Type 54

List of Equation

Equation 3.1. Term Frequency Calculation 29
Equation 3.2. Inverse Document Frequency Concept 30
Equation 3.3. Inverse Document Frequency Calculation 30
Equation 3.4. (TF * IDF) Calculation 31
Equation 3.5. Misclassification Equation 36
Equation 3.6. Misclassification Error Rate Equation 36
Equation 3.7. Precision Equation 37
Equation 3.8. Recall Equation 38
Equation 3.9. F1-Score Equation 39
Equation 3.10. Accuracy Equation 40

List of Acronyms

AE	Actual Effort
ARIMA	Auto Regressive Integrated Moving Average
BoW	Bag of Words
CBR	Case Base Reasoning
CISQ	Consortium for Information & Software Quality
COCOMO	Constructive Cost Estimation Model
CPSQ	Cost of Poor Software Quality
CSV	Comma Separated Values
DT	Decision Trees
EE	Estimated Effort
EV	External Validation
EVO	Evolutionary Delivery
FE	Feature Engineering
FN	False Negative
FP	False Positive
ICOSST	International Conference on Open-Source Systems and Technologies
IDF	Inverse Document Frequency
IJCST	International Journal of Computer Science and Technology
ISBSG	International Software Benchmarking Standards Group
IT	Information Technology
KNN	K-Nearest Neighbor

ML	Machine Learning
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
NHPP	Nonhomogeneous Poisson Process
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PMF	Probability Mass Function
POS	Part Of Speech
RE	Requirements Engineering
RF	Random Forest
RMSE	Root Mean Square Error
RV	Relative Variance
SDEE	Software Development Effort Estimation
SDLC	Software Development Life Cycle
SME	Subject Matter Expert
SMT	Satisfiability Modulo Theories
SRGM	Software Reliability Growth Model
SRM	Statistical Regression Model
ST	Software Testing
STEP	Software Testing Effort Prediction
SV&V	Software Verification and Validation
SVR	Support Vector Regression
TF	Term Frequency

TN	True Negative
TP	True Positive
U.S.	United States
XP	Extreme Programming

Glossary of Terms

Agile Method	An iterative approach in software development for delivering the product value to the customer as fast as possible.
Comprehensive Testing	A testing approach that allows testers to verify the detailed implementation of the developed code.
Confusion Matrix	Performance of a machine learning classification model given a test data where the ground truth is known by reporting the true positives, true negatives, false positives, and false negatives.
Corpus	A large and structured set of machine-readable texts that have been produced in a natural communicative setting. It can be derived in different ways like text, transcripts, spoken language and optical character recognition.
Iterations	In the context of the Agile methodology, iteration is a timebox during which the software development takes place.
Machine Learning	Programming computers to optimize a performance criterion using example data or past experience.

Natural Language Process	A subfield of linguistics, computer science, and artificial intelligence in which a computer program to understand human language as it is spoken and written -- referred to as natural
Software Defect	A bug, error, or fault in a program or system that causes it to produce erroneous results that can lead to fatal system outcomes.
Software Development Lifecycle	A lifecycle process used to design, develop, and test high quality software in a systematic way.
Software Requirements	Also known as software requirements specification, which is a document that describes the intended use of the system. It is the basis of developing the code, testing the code and software.
Software Verification and Validation	Process of checking the software of the system to verify it meets the software requirements and specifications.
Sprint	In the context of an Agile methodology, a sprint is a period of time in which particular work can be completed and ready for review and going to the next phase.

Chapter 1 —Introduction

1.1 Background

Agile software development teams use the agile software development lifecycle (SDLC). Testing the software is the important phase; during this step, the quality of the product is validated (Rajasekhar & Yadav, 2013). Companies and organizations spend a large amount of time and money on software testing, which makes up more than 50% of the software development cost (Ekwoge et al., 2017). In an agile methodology, software requirements change frequently in response to stakeholder requests, particularly during early iterations. As a result, the code is constantly changing, and bugs are common (Highsmith, 2009). When there are any cuts in the project schedule or delays due to changes in requirements and code development, software testing is usually the first step to be impacted (Liou, 2010). In the case of the medical field, testing should completely follow the U.S. Food and Drug Administration (FDA) and other regulations. This praxis develops a natural language processing model to facilitate software testing. This praxis also analyzes and compares the result from natural language processing (NLP) model with historical data for solving a real-world problem in the software industry.

1.2 Research Motivation

Although many published studies highlight the importance of testing and the long-term costs of incomplete or less-than-comprehensive software testing, it is challenging to dedicate sufficient time to testing, particularly in fast methodologies such

as agile development. This praxis develops a model and suggests a solution to this challenge.

1.3 Problem Statement

Reduced comprehensive testing time utilizing the agile rapid software development process contributes to the accumulation of $2.08 trillion cost of poor software quality in the U.S.

During agile SDLC, software requirements may change frequently at the request of stakeholders. Constant changes to the code can lead to increased bugs, contributing to the accumulation of $2.08 trillion costs due to poor software quality in the United States (Consortium for Information & Software Quality (CISQ), 2021). Since agile SDLC must be done quickly, finding adequate time for complete, satisfactory testing to ensure the code is developed correctly is challenging. Incomplete and selective testing increases bugs and issues in the released software. Testing effort accounts for 40% of the total software development cost of the project (Liou, 2010). Depending on the scope and complexity of the project, this can increase to more than 50% (Burnstein, 2003). Performing additional testing after the software has been released to the market is one of the most expensive factors in SDLC apart from a product recall, which can have severe impacts up to and including the company's bankruptcy (U.S. Financial Crisis Inquiry Report, 2011).

1.4 Thesis Statement

A predictive model is required for classification of Agile user stories and requirements to improve testing time and reduce project cost.

During software testing, the Software Verification and Validation (SVV) team carefully matches the software requirements to the required test cases (Liou, 2010). The proposed predictive model is implemented after the agile user stories have been determined. Then the model classifies what type of testing is required for each user story. The time for identifying and classifying the required type of testing is reduced by omitting the manual classification. In this case, the SVV team has more time to focus on performing enough ad-hoc comprehensive testing. This can result in increasing the quality of software before releasing it to the market, which means reducing the rework, redesign, retest, and overall reducing project cost. Because low-quality released software to the market has a higher chance of a recall, the proposed model would also reduce costs associated with fixing the issues.

1.5 Research Objectives

The core objective of this praxis is to develop a predictive model using machine learning to accurately identify and predict the type of software testing required. It also provides evidence for the best forecasting method by:

1. Developing an NLP method;
2. Identifying the errors associated with the developed method;

3. Identifying the accuracy, speed and cost of the developed method; and
4. Comparing the result from the candidate method with the result from a manually developed method.

1.6 Research Questions and Hypotheses

The following research questions and hypotheses were developed to achieve the research objectives identified above:

RQ1: How can the proposed model identify the keywords for the required type of testing?

RQ2: Does the proposed model predict the required type of testing?

RQ3: Is there a significant improvement in software testing time using the proposed model as compared to manual classification?

H1: The proposed model can learn the format of agile user story to identify the keywords.

H2: A Natural Language Processing model can accurately predict the required type of testing.

H3: The proposed model can significantly improve the software testing time as compared to manual classification.

1.7 Scope of Research

This praxis includes the presentation of the natural language processing technique in order to identify the performance of the method for extracting the keywords and requirements from agile user stories to predict the type of testing required. The final result from the predictive model is compared to the result of manual testing. The software testing time is also included in the scope of this project. Testing time is evaluated based on the methods of collecting historical data. The time saved on software testing is also measured in U.S. dollars.

The predictive model developed and evaluated in this praxis can later be adjusted and applied to a broader methodology, such as waterfall, in different industries. This can impact software testing procedures in different areas of the software industry.

1.8 Research Limitations

1- Normalization: The model was developed in a way that can understand a specific format of the agile user story requirements.

2- Numerical dataset: If the agile user story requirement has numerical values that may play an important role for choosing the required type of testing, the model will not be able to recognize them. The dataset that is used in this praxis does not have numerical values or they do not play a role for the required type of testing.

3- Type of input: Based on the limitation of the python library and the model developed, the input format should be MS Excel .csv and each agile user story

should be written separately in each cell of the excel so the model can read the data.

4- Dataset: It is limited to 500 user stories from the same software development team and software project.

1.9 Organization of Praxis

The remainder of this praxis research organized as follows:

- Chapter 2 - Literature Review: This chapter provides an in-depth review of peer-reviewed literature on success factors in agile software testing and project scheduling, different methods for estimation of software testing time, and discusses different machine learning models.
- Chapter 3 - Methodology: This chapter covers the statistical and machine learning methodologies which have been applied during this research.
- Chapter 4 - Result: This chapter summarizes and evaluates the results of the data analysis associated with each research question and hypothesis and proposed methodology.
- Chapter 5 – Discussion and Conclusion: This chapter conclude the praxis with a discussion of the results and summarizes the contributions to the body of knowledge with the look at the recommendations and opportunities for future research.

Chapter 2 —Literature Review

2.1 Introduction

This literature review provides some background on software testing and demonstrates the key role of fast, short iterations in agile methodology. It aims to identify and analyze relevant academic research on the use of different machine-learning (ML) and non-ML methods to reduce the time and cost of software testing and to establish the most accurate scheduling in agile software development. This chapter also summarizes the factors that impact the success of testing in agile development.

Section 2.2 addresses the factors that impact the success of agile software testing. This section reviews the literature on non-ML methods for improving the software development life cycle (SDLC). This can reduce the time and cost of testing in agile development. Section 2.3 considers the agile scrum method, focusing on scheduling. As will be discussed further in that section, proper scheduling can allow sufficient time for testing without causing a project to go over budget. Section 2.4 explains the different methods used to estimate software testing time. In this praxis, the historical data method is used to compare the results of the model with those of manual classification. Section 2.5 focuses on different ML methods that can enhance time, cost, and scheduling during SDLC. Section 2.6 summarizes the chapter and draws some conclusions from the literature review.

2.2 Success Factors in Software Testing During Agile Development

Agile development was introduced in the early 1990s to address problems with other software development methodologies, particularly the waterfall method. Agile development enables quick management of changes (Highsmith, 2009). These changes play an important role in the success of software development. The following sections describe some factors that impact the success of agile development.

2.2.1 Customer Collaboration

Customers frequently request changes to the requirements of a software program (Highsmith, 2009). Frequent changes to software requirements are the main reason why software developers adopt agile approaches (Rehman et al., 2020). This method can enable them to quickly deliver high-quality products that fulfill the customer's requirements (Khan, 2020). The quality of a software product is directly related to comprehensive testing (Burnstein, 2003). One factor in successful development is collaboration with the customer to ensure that all requested changes are reflected in the tests (Power & Heavin, 2021). Customer collaboration during contract negotiation is one of the four core values of agile development (Power & Heavin, 2021).

Although "changing requirements, even late in the project, are welcomed" (Power & Heavin, 2021, p. 4), such changes can impact the project schedule. Furthermore, whenever there is a change to the overall software project schedule or a delay in code development due to the actions of the software development team or other stakeholders, software testing is usually the first step to be impacted (Liou, 2010).

While customer collaboration and quick responses to customer requests are considered strengths, the analysis performed in this study shows that frequent customer change requests can negatively impact the software development process and software testing time.

2.2.2 Agile Testing Mindset

An agile testing mindset is another factor in successful software testing with an agile methodology. One pillar of this mindset is that team members must proactively test the requirements as early as possible (Highsmith, 2009). This means that the entire agile team is responsible for software quality (Highsmith, 2009). During software development, everyone in the team should consider testing (Gregory & Crispin, 2014). This implies a need to change and improve development culture as well. This mindset impacts every aspect of software development and the software testing workflow in some way (Gregory & Crispin, 2014). It is not easy to transition to an agile testing mindset. Once this transition has been made, projects developed using agile methodology are three times more successful than those developed using other methodologies (Standish Group, 2012).

An agile testing mindset is one of the core values applied during the development of the model in this praxis. Although this model was mainly designed to assist software verification and validation (SV&V) team members to perform comprehensive testing, the ultimate result is the release of a high-quality software, which is the pillar of the agile testing mindset.

2.2.3 Agile Automation Regression Testing

Automation testing is another factor that impacts the success of agile testing. Automation testing requires minimal or no human intervention or involvement (Rajasekhar & Yadav, 2013). The goal of the present study is to develop a model that can be used to support automation testing in an agile development approach. There are many reasons to automate testing (Rajasekhar & Yadav, 2013):

1. To save time. For projects that involve hundreds of requirements and thousands of lines of code, automation testing saves time.
2. To meet frequently changing customer requirements.
3. To reduce the cost of long-term or repeated testing.

Companies and organizations may use automation testing for different reasons; the decision to use automation testing may be influenced by the type of project or by management goals and strategies. The three points listed above are the reasons that are most frequently mentioned in information technology (IT) journals (Rajasekhar & Yadav, 2013).

2.3 Agile Development and Scheduling Methods

An agile methodology may use different development methods. Each method has its own SDLC, and each SDLC has a different testing phase and schedule. This section examines the scrum method. The testing time for scrum is explored, and previous literature is reviewed.

2.3.1 Agile Scrum Method

A scrum iteration usually lasts a maximum of four weeks; both software development and software testing should be completed within that time (Larman, 2004). The code developed during each iteration is tested and debugged during that iteration (Martinez et al., 2016). This method is recommended for small projects that do not involve frequently changing software requirements and that are developed by a relatively small team (including the SV&V team) (Abrahamsson et al., 2017). Everyone involved in the agile scrum method should clearly understand the expectations, including the requirements for testing during code development (Martinez et al., 2016).

Different ML methods are used in agile software development. In the agile scrum method, ML techniques are used to estimate the scrum schedule (Arora et al., 2020). One estimation approach used in the scrum method is the ontology-based multiagent technique (Adnan & Afzal, 2017). This method can estimate the schedule with 91% accuracy (Adnan & Afzal, 2017).

2.4 Software Testing Time

One significant challenge of the SDLC is finding the correct balance between meeting the software release date and ensuring the quality of the software at the time of release (Burnstein, 2003). The time needed to run all possible test cases, ensure the software meets all the customer requirements, and address related risks could make it necessary to postpone the software release (Liou, 2010). Testing resource allocation, such as the number of testers, can also significantly impact testing time and cost and software

reliability (Okamura et al., 2018). In practice, many companies and organizations face time and resource constraints that significantly limit their ability to effectively complete comprehensive testing in a timely manner (Yang et al., 2012).

Okamura et al. (2018) developed a model to optimize testing resource allocation. The model is based on the architecture of the targeted software. It focuses on the purpose of the software testing environment, which is to identify bugs and issues in the software. Thus, the testing environment might not be a time-accelerated version of the operational environment. As a result, this model is an architectural software reliability model. Moreover, this model assumes that there are finite numbers of faults in each component before testing. It is also assumed that the fault detection rate for a given component is proportional to the number of testing efforts for that component.

Okamura et al.'s (2018) model was developed based on the number of bugs and issues that were expected before testing. This is controversial approach, since the expected number could vary significantly. This variance is also impacted by many other components, such as the software developer's experience, the maturity of the software requirements, and the complexity of the project. Okamura et al.'s (2018) model is based on unreliable data, which can produce inaccurate results. This is the model's main weakness.

2.4.1 Historical Data Method

Different organizations, including software companies, are interested in determining the time and cost of software development and testing as early as possible so

they can plan ahead (Najadat et al., 2012). One method for doing this is using historical data. This method uses previous performance data as a guideline for estimating the time and cost of future development. Software testers frequently use this method (Burnstein, 2003).

Halkidi et al. (2011) developed a data mining process for identifying data that can be used to find software bugs. The first step is to identify the types of bugs that are being already fixed in the software. This is done by reviewing historical data on the software project; these data can then be used to estimate the time and cost of software testing.

Connor and MacDonell (2006) used historical data for stochastic estimation of the duration of software development projects. They developed a tool that captures and analyzes historical data using Microsoft Excel. Historical data are captured as a list of the actual efforts required for each project and broken down by the phase of the project.

The model developed by Connor and MacDonell (2006) has a better structure since the data are categorized by the phase of the project. This can help indicate which phases of development require more time and effort.

2.4.2 Fuzzy Model

The fuzzy method is another approach for estimating the cost and time of software testing. The core structure of this model aims to solve uncertainty problems. Khalid and Yeoh (2017) developed this model, which focuses on reworking software during the testing process. The model aims to provide an early estimation of the costs of reworking the software during development due to a lack of proper and comprehensive

testing. The size of the software project is determined by the software requirements. As team experience increases, the quality of deliverables is expected to increase, resulting in fewer reworks. This model considers software size, software team experience (including the development and testing teams), and the expected number of reworks during the testing process. The actual cost at completion is calculated by adding the cost of development to the cost of reworking.

The fuzzy model has some pros and cons. Reworking is part of the SDLC, and it is important to consider it as a distinct phase and to measure its impact on overall project costs. This can allow the management team to focus on this phase, which can significantly impact software project costs. In quick methodologies such as agile development, the software is usually not completely rejected or accepted. This model is more useful in methodologies such as waterfall, which defines separate, sequential phases of development.

2.5 Machine Learning (ML) Forecasting Methods

To develop an ML method, first, outliers are removed, and the data are cleaned. Then the data are normalized to ensure that the features have a balanced influence on the labels. If there are large features, feature selection is performed (Vabalas et al., 2019). Software development effort estimation (SDEE) is used to predict the effort that will be required to develop a software product. Effort includes the cost and time needed for software development and testing. ML forecasting methods based on SDEE, and other processes can be used to improve the accuracy of effort estimations (Wen et al., 2012).

ML forecasting models have been demonstrated to predict SDLC, including the cost and time of software testing, more accurately than non-ML models (Martin, 2021). The SDLC ML model also results in more accurate estimates than statistical regression models (SRMs) (Gautam & Singh, 2018). Wen et al. (2012) and Martin (2021) examined the frequency at which various machine learning methods were utilized to predict the estimated time and cost of software testing. The frequencies of each ML methods are shown in Table 2.5 (Martin, 2021).

ML Forecasting Method	Frequency (%)
Case-based reasoning (CBR)	37%
Neural networks	26%
Decision trees (DT)	17%
Support vector regression (SVR)	5%

Table 2.1. ML Forecasting Frequency

Studies report that neural networks and SVRs are the most accurate ML methods (Martin, 2021).

2.5.1 Natural Language Processing (NLP)

Natural language processing (NLP) enables machines and systems to interpret and break down human languages. NLP is also used to identify similarities in software requirements (Holbrook et al., 2012; Falessi et al., 2010). In large software projects and

in those that must be completed very quickly, it may not always be feasible to meet all stakeholders' requirements due to timelines, resource limitations, and costs (Sarkani et al., 2015). One potential solution to this is using linguistic tools such as NLP to understand and prioritize stakeholder requirements using a satisfiability modulo theories (SMT) solver (Sarkani et al., 2015). NLP is also used to translate software requirements that are provided in human natural language into formal specifications by using the control model to structure the requirements, extract the characteristics of each requirement, and translate the requirements for automation testing, such as agile automation testing (Rui & Deming, 2013). Sharma and Kumar (2019) developed an ML tool that uses an NLP algorithm to break down and understand agile user stories. The tool then uses story mapping to create an agile estimate for the software release plan.

The strengths of Sharma and Kumar's (2019) NLP model lie in its improvements on previous models. Prior to the development of their model, two working weeks (~84 hours) were needed to plan 28 releases of 340 agile user stories. This model reduces this to 18 hours for the same number of software releases and agile user stories, representing a savings of 78.57% in working hours. The model's overall accuracy is 82.38%. For every release, the product owner must provide a set of keywords for that specific release. This can impact the quality of the project and the purpose of the ML method. In addition, similar keywords are likely to appear in multiple user stories, which can negatively impact the accuracy of the results.

2.5.1.1 NLP Vectorization Methods

Different methods are used to vectorize data for NLP algorithms. Bag of Words (BoW), Word2Vec, and TF-IDF are the most common methods, depending on the requirements of the ML model. This vectorizer calculates and analyzes the presence of specific words in a text.

Word2Vec is a vectorizing method that uses BoW as the data source. In this method, the context of each word is taken as the input for the model. In this approach, most of the applicable information about the text is preserved to perform the prediction.

Neither BoW nor Word2Vec was used as the core vectorization method in the NLP model developed in this praxis. BoW was considered inappropriate for two reasons. First, the length of the feature vector can significantly increase the number of words in our dictionary. Second, whether a given word is present or absent does not really help the model classify agile user stories by the type of testing required. Word2Vec was also determined to be inappropriate as predicting the following word does not help classify agile user stories.

2.5.2 Decision Tree (DT) and Random Forest (RF) Methods

A decision tree (DT) is an ML method for binary classification. This technique is easier to implement than other ML methods, and it can be used for dataset classification and regression (Manna et al., 2019). DTs are also widely used to support language comprehension, mainly for NLP tasks such as tagging (Schmid, 2013). Larivière and Poel

(2005) find that DTs are a less robust tool for predictions and do not perform as well as other ML algorithms.

Random forest (RF) is a common ensemble learning method that combines several randomized decision trees for classification or regression (Wang et al., 2018; Breiman, 2001). Ensemble learning models such as the RF method are a robust approach for dealing with noise and outliers and can prevent overfitting (Pospieszny et al., 2018).

One advantage of the RF method is better variance performance than the simple DT method (Wang et al., 2018) and Breiman (2001). The RF method also offers significantly more accurate predictions than a single DT and prevents overfitting by decorrelating the individual trees from one another. The RF method does this during tree construction; instead of using the entire universe of attributes, the RF algorithm is only sensitive to the selected features. This decreases the model's dependency on strong predictors, which can cause homogeneity across trees. This also protects the ensembled tree models from unnecessary individual error risks.

2.5.3 K-Nearest Neighbor (KNN) Method

The K-nearest neighbor (KNN) method is another ML forecasting algorithm that predicts new observations by pooling and averaging the value of the nearest K data points (Imandoust & Bolandraftar, 2013). KNN is based on the similarity of features, so it is sensitive to inaccurate or irrelevant features (Imandoust & Bolandraftar, 2013). The core algorithm of this method maps data from the training set and then predicts operating expenditures for the test set based on the local proximity of the nearest neighbor.

Different K values are examined to determine the optimal value, which will generate the best possible prediction for the test set.

KNN, a non-parametric method, uses a tuning parameter, K, to control how the model should be trained, in contrast to parameters that are learned via training (Dalpiaz, 2020). This method can be used to develop a highly accurate model for classifying the text in user stories and requirements (Li et al., 2018). The KNN algorithm proposed by Li et al. (2018) improves the accuracy of document classification by improving the measures of similarity between the text documents and software requirements.

2.6 Summary and Conclusion

The short time frame of an agile SDLC allows insufficient time for comprehensive software testing (Rehman et al., 2020). When software development is behind schedule or there is a delay in code development, then software testing is the phase that is most severely impacted (Liou, 2010). To identify appropriate solutions for this issue, this literature review has examined journal articles, conference papers, government publications, and books on the most important factors impacting the success of software testing, focusing on agile methodology. Members of the software team, including the SV&V and software teams, must work closely with each other, customers, and other stakeholders to ensure that the final product incorporates all requested changes. Frequent change requests could also negatively impact the process and severely impact software testing time. There should be a balance between the number of requested changes and the number of approved and implemented changes.

This literature review of ML methodologies has discussed the most popular forecasting algorithms and the methods they use to recognize and classify text and requirements. The NLP method was identified as the best fit for the present work. The use of the NLP method to develop the model in this praxis was inspired by Sarkani et al. (2015). They also use NLP to understand, prioritize, and meet all stakeholders' requirements. In this praxis, their idea is further developed to classify user story requirements based on the required type of testing.

The present study aims to identify the best way to classify agile user stories to shorten software testing time. This literature review has highlighted the need for an NLP model that is more reliable and accurate than any existing models.

Chapter 3 —Methodology

3.1 Introduction

This chapter presents in detail the methodology used to classify the agile user story requirements using a subfield of machine learning (ML) called natural language processing (NLP). Section 3.2 describes the data source selection and explains how to read and explore the datasets used in the model. Section 3.3 describes the data preprocessing. Section 3.4 presents the ranking method and vectorization. Section 3.5 describes how the model was evaluated. Finally, section 3.6 describes the model validation.

3.2 Data Source Collection

The 500 records for this praxis were obtained from the largest American ophthalmic medical device company that specializes in both surgical and vision care products. In this paper, the organization's name, resource names, project names, and all other references to the company have been masked. The data collected from the Azure DevOps repositories consisted of software requirements, or statements of what a software system must be able to do.

The raw data used in this model has an agile structure and is stored in comma-separated values (CSV) file formats. These data are in the form of agile user stories; most are one sentence long. A few agile user stories may include a second or third sentence defining some terms used in the main sentence. The following example shows a typical agile user story in the same format as the data used in the present praxis.

- As a <**who** – a person who wants to accomplish something)>
- I want <**what** function/type of testing is needed>
- So that <**why** this function/type of testing is needed>

For example:

- As a Field Service Engineer (FSE),
- I want to verify the kernel memory event,
- So that I can schedule remote service.

The model developed in this praxis uses this standard format for agile user stories. In this model, the “who” part of the requirement is introduced with the phrase “as a.” The “what” part of the requirement is introduced with the phrase “I want.” The “why” part of the requirement is introduced with the phrase “so that.” This format is applied to all 500 agile user stories in this praxis.

The keywords for each type of required testing are identified; the testing types are communication, security, surgical, and system. The results of this classification are also listed in “src_assign.csv,” In this file, the classification is identified with the user story number to simplify tracking. The source file is used to identify the required type of testing for the corpus file.

3.2.1 Data Analysis Tools

In this praxis, the following required Python tools and libraries were used to build and develop the NLP model:

- Scipy (Jones et al., 2001) - an open-source Python library containing tools and modules for technical computing. This includes methods for interpolation, linear algebra, optimization, integration, ordinary differential equation solvers, and a variety of statistical tests.
- Numpy (Harris et al., 2020) - an open-source scientific Python library that supports work with linear algebra and matrices.
- Imbalanced-learn (Lemaître et al., 2017) - an open-source Python library containing various resampling methods used in datasets with strong class imbalance.
- Scikit-learn (Pedregosa et al., 2011) - an open-source Python library containing tools for classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.
- Pandas (Reback et al., 2020) - an open-source Python library containing functions for analyzing, exploring, and manipulating data.

The raw data is uploaded *as is*, and a different algorithm cleanses, vectorizes and classifies the information. The first step was to enable the model to read CSV files; using the *os.path* and *string* Python libraries (Reback et al., 2020). Appendix A contains the actual code.

3.3. Data Preparation

Data preparation and cleansing is a critical phase in NLP and can significantly impact the accuracy of the final result (Ridzuan et.al., 2019). During NLP preparation, it is necessary to highlight the attributes that the ML system should notice. This involved a number of steps that needed to be performed in sequence. First, all words in the agile user story requirements were converted to lowercase. “Surgery” and “surgery” are equivalent and should not be counted as two separate words (Denny & Spirling, 2018). The Python internal library *lower_string* was used to perform this action (Reback et al., 2020). Figure 3.1 shows the overview of data preparation used for ML text processing.

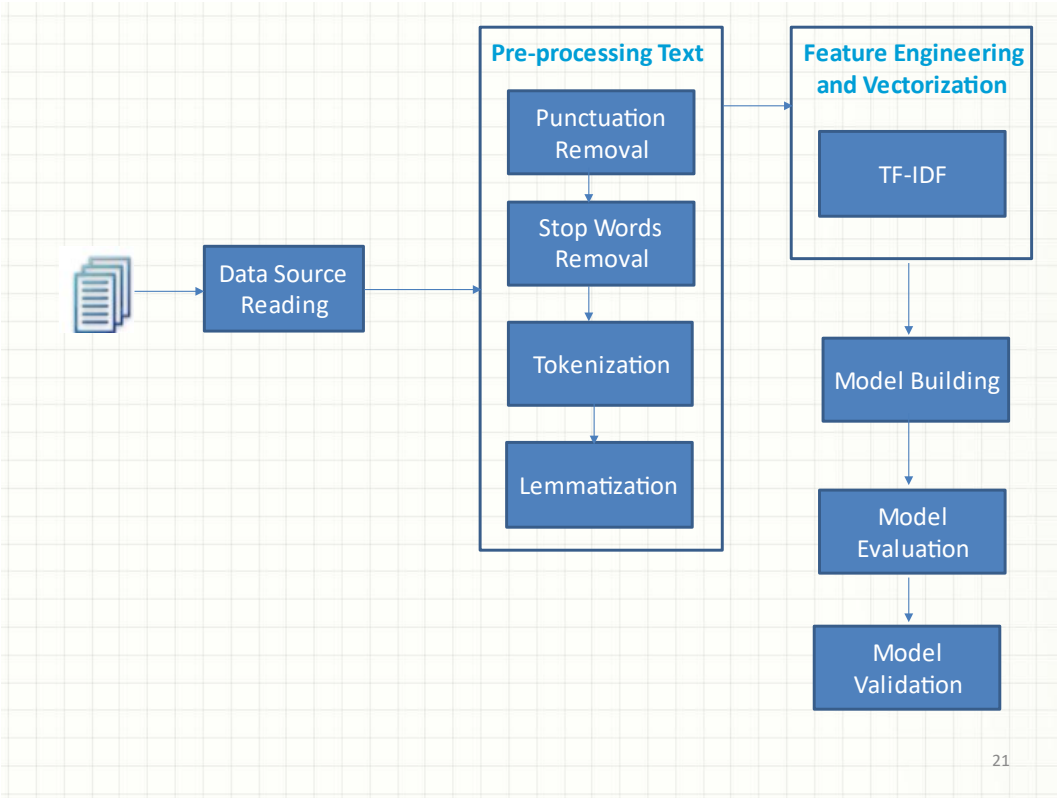


Figure 3.1. Text Processing for Machine Learning

3.3.1 Punctuation Removal

Punctuation provides a sentence with a grammatical framework that helps humans correctly read and understand the written text (Etaiwi & Naymat, 2017). For the NLP vectorizer, which counts the number of words and not their contextual meaning, punctuation does not add value. During the data preparation, punctuation marks were removed to prevent any negative effects during text processing and vectorization, especially impacts related to the frequency of the occurrence of words and phrases calculated using inverse document frequency (TF-IDF). The Python *string* library and *string.punctuation* were used to remove punctuation in the model (Reback et al., 2020). Appendix A contains the actual code.

3.3.2 Stop Words Removal

Stop words are a group of words that occur frequently and that help humans understand written texts, including agile user stories. Stop words needs to be removed in order to put weight only on the words that play a role in the TF-IDF calculation. This helps the model remove outliers and focus on the key features for user story classification (Silva & Ribeiro, 2003). Removing stop words before NLP begins enables the model to focus on the important words in the corpus. Table 3.1 shows a basic example of stop words removal. The Python *nltk* and *stopwords* libraries and *string.stopwords* were used to perform this task.

With Stop Words	Without Stop Words
/As-a-developer-I-want-change-the-implementation/	/developer-want-change-implementation/
/As-a-clinical-technician-I-want-a-convenient-and-sleek-way-to-create-new-patients/	/clinical-technician-want-convenient-sleek-way-create-new-patient/
/Software-shall-display-fixation-target-current-position/	/Software-shall-display-fixation-target-current-position/

Table 3.1. Stop words Removal

3.3.3 Tokenization

Identifying the words that constitute a string of characters during NLP text preprocessing is required (Thomas, 2020). In this process, a string of characters is broken down into words, called a “token”; this restructures the text (Table 3.2). This process is used to separate the text into single elements, or words, which are separated by spaces and will be used as input for vectorization (Nayak et al., 2016). Tokenization transforms a text document into numerical data that can be used directly as a vector to represent the document and for the TF-IDF calculation. Python has many different tokenization tools, including the natural language toolkit (nltk), Treebank and spaCy, all of which perform tokenization for different types of ML models. For the model developed in this praxis, the Python library *nltk* was used (Reback et al., 2020). Appendix A contains the actual code.

	docs	tokens
0	the cat see the mouse	[the, cat, see, the, mouse]
1	the house has a tiny little mouse	[the, house, has, a, tiny, little, mouse]
2	the mouse ran away from the house	[the, mouse, ran, away, from, the, house]
3	the cat finally ate the mouse	[the, cat, finally, ate, the, mouse]
4	the end of the mouse story	[the, end, of, the, mouse, story]

Table 3.2. Documents Vs. Tokens

3.3.4 Lemmatization

Lemmatization is defined as the text normalization process of reducing words to their root forms, or “lemmas” (Kang et al., 2020).” It is mainly applied in NLP and is used to increase the accuracy of text analysis. Lemmatization enables the ML model to extract relevant information from a text. In order to perform this task accurately, it is important that the Python code includes the part of speech (POS) tag. This is key to lemmatization, which eliminates word variations based on factors such as gender, case, number, and syntax (Vijayarani et al., 2015). POS tagging is the procedure of assigning the appropriate part of speech to each word in a corpus; a POS tagger assigns the appropriate POS tag to each word. Assigning the correct tag requires special attention because the POS can be different based on the meaning of the sentence. The process of POS tagging supports the ML model to handle each word correctly based on the position of POS in the sentence. Tables 3.3 and 3.4 show how a word can be lemmatized differently with and without a POS tag. In our model, the Python libraries *nltk*

and *WordNetLemmatizer* and pos_*tag* were used for POS tagging and lemmatization (Reback et al., 2020). Appendix A contains the actual code.

Word	Lemmatized Word
Driving	Driving
Dogs	Dog
Was	Was
Feet	Foot

Table 3.3. Lemmatizing the Word without Considering the Type (POS Tag)

Word	Type (POS tag)	Lemmatized Word
Driving	Verb ‘v’	Drive
Dogs	Noun ‘n’	Dog
Was	Verb ‘v’	Be
Feet	Noun ‘n’	Foot

Table 3.4. Lemmatizing the Word with Considering the Type (POS Tag)

3.4 Vectorization and Ranking

Vectorization is based on identifying the positional connections between the most relevant words to predict the classification of the text. This enables NLP algorithms to use the interactions of positional syntax as well as keywords.

The ML model first inspects the text to identify its relationship to some target class. Each agile user story in the model was compared to classes defined by the type of testing required. Next, the model identifies the most common words in the target class and then scores each word in the text based on its relationship to the most common terms in the target class. In our model, the Python libraries *TfidfVectorizer* was used for vectorization (Reback et al., 2020). Appendix A contains actual code.

3.4.1 Term Frequency (TF)

The term frequency (TF), $tf_{t,d}$, is the number of times that a term *t* occurs in story *d*. Equation 3.1 is used to calculate TF. Depending on the model, there are several ways to calculate this frequency. For developing the model in this praxis, method number (2) from the below list is used since it is simpler and easier to perform the calculation.

1. Total number of times a term occurs in a story, or the raw count
2. TF adjusted for the length of the story, or the raw count of occurrences divided by the number of words in the story
3. Logarithmic scale (or log scale) frequency, or log(1 + raw count)
4. Boolean frequency, which results in a value of 1 if a term occurs and 0 if the term does not occur in the story

$$TF = \frac{Number\ of\ Times\ the\ Term\ Appears\ in\ the\ Story}{Total\ Number\ of\ Terms\ in\ the\ Story}$$

Equation 3.1. Term Frequency Calculation

3.4.2 Inverse Document Frequency (IDF)

IDF is the measurement of how common or uncommon a word is in a given corpus. In this measurement, words that appear in a small percentage of stories are ranked higher than words that are common in all stories (Yahav et al., 2019). IDF is calculated by the logarithm of the term *t*, the word to be measured, and *N*, the total number of stories *(d)* in the corpus *(D)*, divided by the number of stories in which the term *t* appears (Equations 3.2 and 3.3). A value closer to 0 indicates that a word is more common in the corpus. The value for less common words will approach 1 (Silva & Ribeiro, 2003).

$$IDF = log \frac{Number\ of\ the\ Stories\ in\ the\ Corpus}{Number\ of\ Stories\ in\ the\ Corpus\ Contain\ the\ Term}$$

Equation 3.2. Inverse Document Frequency Concept

$$IDF(t, D) = log \frac{N}{Count\ (d \in D : t \in d)}$$

Equation 3.3. Inverse Document Frequency Calculation

Where:

N is the number of stories in the corpus,

t is the term occurs in story,

d is the story, and

D is the corpus.

3.4.3 Term Frequency–Inverse Document Frequency (TD-IDF)

The importance of a term *t* is inversely related to its frequency across stories (Yahav et al., 2019). TF indicates how frequently a term *t* occurs in a story, and IDF indicates the comparative rarity of a term *t* in a collection of stories *D.* A higher TF-IDF score indicates that the term *t* is more important or relevant. If TF-IDF approaches 0, the term *t* is less relevant. TF-IDF is calculated using Equation 3.4.

$$tf * idf\ (t, d, D) = tf(t, d)\ .\ idf\ (t, D)$$

Equation 3.4. (TF * IDF) Calculation

Where:

t is the term occurs in story,

d is the story, and

D is the corpus.

TF-IDF is used in many ML algorithms with NLP applications because algorithms deal with numbers, and natural language is text. It is necessary to transform the text into numbers which can be accomplished with TF-IDF. This equation is used in applications such as search engines, text summarization, and text classification to rank the relevance of a document to a query. In the present praxis, TF-IDF is used to extract keywords from agile user stories. The words with the highest TD-IDF are the most relevant words in a given story and can be considered keywords for that document (Zhuohao et al., 2021). Several Python libraries calculate TF-IDF. In this praxis, the

Python libraries *sklearn*, *pandas*, and *numpy* were used with the *TfidfVectorizer()* function to calculate TF-IDF (Reback et al., 2020).

3.5 Model Classification

The dataset is divided into two main categories. A set that has keywords, is called *source* stories, and another set that does not have the keyword is called *corpus* stories. The required type of testing for the *source* stories is determined by the TF-IDF calculation of both *source* and *corpus* stories with the highest rank for each keyword. Table A-4 in appendix A, shows the complete result of assigning a *corpus* story to the *source* story type of testing classification. Table A-5 in appendix A, shows the complete result of the classification of the entire dataset.

3.6 Model Evaluation

The goal of studying and developing the NLP model is to accurately classify agile user story requirements. Stories were classified using a form of supervised ML with the aim of using a training dataset to develop an ML model for the keyword distribution in each class (Kotsiantis et al., 2006). Text classification enables automatic labeling of software requirements by extracting patterns (Baharudin et al., 2010). However, an NLP model may misclassify agile user stories for a number of reasons, such as identifying the wrong keyword for the type of testing required. The trained ML model needed to be

evaluated carefully to ensure that it was adequately generalizable to the corpus (Kang et al., 2020). This was done using a confusion and performance matrix (Kang et al., 2020).

3.6.1 Confusion Matrix

In this praxis, to evaluate the ML model's classification, the test data was used to predict the required type of testing for each agile user story. The confusion matrix shows the visualization of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), as shown in Figure 3.2.

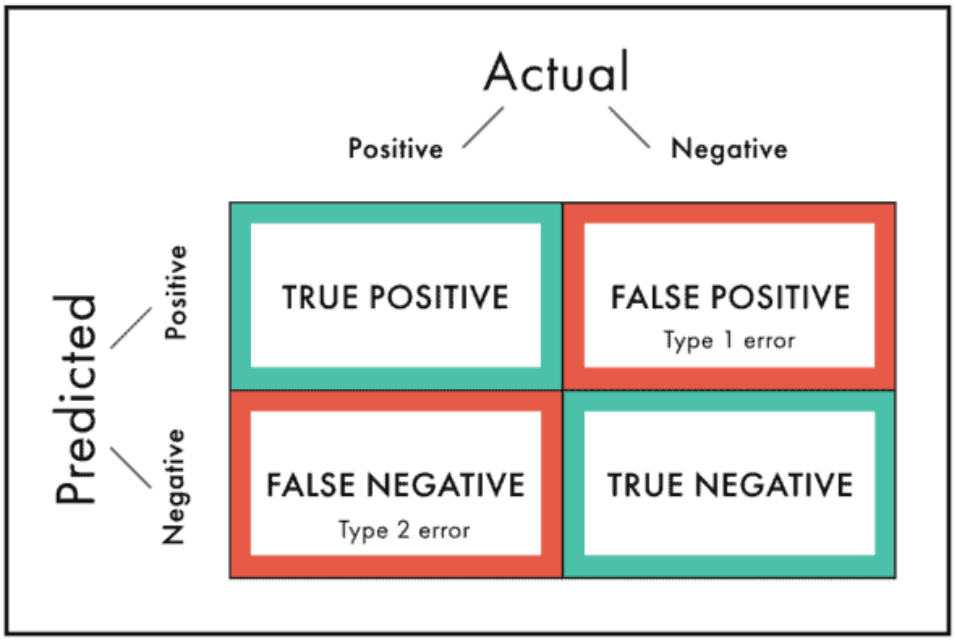


Figure 3.2. Confusion Matrix with Misclassification Error Type (Explorium, n.d.)

- True positives (TP): Cases when the classifier predicted the required type of testing, and it is correct (it needs that type of testing).

- True negative (TN): Cases when the classifier does not predict the required type of testing and it does not require that type of testing.
- False positives (FP): Cases when the classifier predicted the required type of testing, but it is wrong (does not need that type of testing).
- False negative (FN): Cases when the classifier is not predicated on the required type of testing, but it actually required that type of testing.

Tables 3.5 and 3.6 provide the Model-level and class-level evaluation metrics examples. These evaluations were used as input to the confusion matrix and also for the evaluation of the model.

User Story #	Actual Class	Predicted Class
1	Communication, Security	Security
2	Communication	Communication
3	Surgical, System	Surgical
4	Security	Communication

Table 3.5. Model Level Example

Key	Count	Explanation
True Positive (TP)	1	User story 2 was correctly classified
False Positive (FP) Type I error	1	User story 4 was incorrectly classified
False Negative (FN) Type II error	1	User Story 1 was not classified as communication though it should have
True Negative (NP)	0	N/A

Table 3.6. Class Level Evaluation Example

3.6.1.1 Misclassification and Error Rate

Two types of misclassification errors may occur when agile user stories (software requirement texts) are classified according to the type of testing required: type I errors and type II errors (Alpaydin, 2014). In this model, a type I error is less favorable than a type II error because, with a type I error, the software requirement is assigned to the wrong type of test. As a result, the wrong test is used. When this happens, bugs in the software may not be detected, and then the software could be released with issues. This would be costly and contradicts the primary focus of this research. In a type II error, the ML correctly identifies the keywords and the type of testing required, but it overlooks some keywords and so it is incomplete. This is problematic, but less serious than a type I error. In this praxis, misclassification was measured using the *misclassification_error* module from the imbalanced-learn Python library (Lemaître et al., 2017). In order to precisely measure the misclassification error rate, we divided the total number of misclassifications by the total number of instances evaluated (Hossin & Sulaiman, 2015).

The formulas for calculating the misclassification and misclassification error rate derived from the confusion matrix are shown in Equations 3.5 and 3.6, respectively.

$$Misclassification = (FP + FN)$$

Equation 3.5. Misclassification Equation

$$Misclassification\ Error\ Rate\ = \frac{(FP\ +\ FN)}{(TP +\ TN + FP + FN)}$$

Equation 3.6. Misclassification Error Rate Equation

where:

FP is the number of false positives,

FN is the number of false negatives,

TP is the number of true positives, and

TN is the number of true negatives.

3.6.1.2 Precision

The precision metric measures the ratio of correct positive predictions to the total number of positive predictions (Hossin & Sulaiman, 2015). This measures the precision of the ML model and indicates how many agile users story requirements were correctly

labeled and classified based on the type of testing required (Hossin & Sulaiman, 2015). The precision metric is the percentage of predicted documents on a given topic that are classified and labeled correctly (Silva & Ribeiro, 2003). The formula for calculating the precision score derived from the confusion matrix is shown in Equation 3.7. In this praxis, precision was calculated using the *precision_score* module from the imbalanced-learn Python library (Lemaître et al., 2017).

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

Equation 3.7. Precision Equation

where:

TP is the number of true positives, and

FP is the number of false positives.

3.6.1.3 Recall

Recall or sensitivity is the ratio of correct positive predictions to the total number of (incorrect and correct) positive predictions (Hossin & Sulaiman, 2015). This is a key evaluation metric because it compares the number of correctly predicted defective modules to the total number of defective modules (Caivano et al., 2011). It measures the ability of the ML model to predict the actual positive classes using the ratio of accurately predicted true positives and the one that was actually labeled. It reveals how many of the

predicted types of required testing are correct (Hossin & Sulaiman, 2015). This measure is the percentage of total documents on a given topic that are classified and labeled correctly (Silva & Ribeiro, 2003). The formula for calculating the sensitivity score derived from the confusion matrix is shown in Equation 3.8. This praxis uses the *sensitivity_score* module from the imbalanced-learn Python library (Lemaître et al., 2017).

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

Equation 3.8. Recall Equation

where:

TP is the number of true positives, and

FN is the number of false negatives

3.6.1.4 F1-score

The F1-score or F-measure is the harmonic mean between the precision and sensitivity values (Hossin & Sulaiman, 2015). This score is needed when it is necessary to balance precision and recall (Caivano et al., 2011). The F1-score is calculated separately at the class level and the model level. In this praxis, the ML model utilizes the *f1_score* module from the scikit-learn Python library (Pedregosa et al., 2011). The

formula for calculating the F1-score derived from the confusion matrix is shown in Equation 3.9.

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Equation 3.9. F1-Score Equation

where:

TP is the number of true positives,

FN is the number of false negatives, and

FP is the number of false positives.

3.6.1.5 Accuracy

Accuracy is the ratio of correct predictions to the total number of evaluated cases (Hossin & Sulaiman, 2015). It includes all positive and negative classifications and the numbers of correct and incorrect predictions (Caivano et al., 2011). In this praxis, it was necessary to calculate the accuracy of the model because the agile user stories had already been manually labeled and classified according to the type of testing required. The ML model's classifications were compared to the manual classifications to measure the model's accuracy. In this praxis, the ML model uses the *accuracy_score* module from the sklearn and numpy Python libraries. The formula for calculating the accuracy derived from the confusion matrix is shown in Equation 3.10.

$$Classification\ Accuracy = \frac{TP + TN}{(TP+TN+FP+FN)}$$

Equation 3.10. Accuracy Equation

where:

TP is the number of true positives,

TN is the number of true negatives,

FN is the number of false negatives, and

FP is the number of false positives.

3.7 Model Validation

Validating an ML classification model is one of the most important phases in the model development life cycle (Cabitza et al., 2021). Because this step is so sensitive, external validation (EV) is necessary. The accuracy of a model's causal relationships are measured by comparing the model's predictions to observations (Broniatowski & Tucker, 2017). Consistent, reliable observations are a required condition for this step. The first step to validate the NLP model is to run the model with only 10 datasets and check the results. Then tune the model and run the 10 datasets again until the integrity of the development is validated with manually ranking from the internal team of subject matter experts (SME). Then 30 datasets are run on the model and the results are checked. Further tunning and updates apply to the NLP model to check the integrity of the development with the manual rankings from the internal team of SME. This process is

continued once more with 50 datasets and 75 datasets. Once the model is thoroughly validated and no further tuning is required, the entire 500 datasets are run on the model and the results are compared with the historical data.

Chapter 4 —Results

4.1 Introduction

This chapter presents the results of the analysis. Data were collected from one of the largest medical device companies in the United States. One condition of the company's agreement to the use of these data for this research was that all identifying names and references to the company be masked. The dataset is collected from the company's most recent agile project, which was conducted in 2021. The results of the natural language processing (NLP) model can be compared to historical data to improve the software development life cycle (SDLC), focusing on agile testing.

Section 4.2 briefly describes the data preprocessing, or feature engineering. Section 4.3 describes the different types of possible classifications in the NLP model. Sections 4.4 to 4.6 discuss the findings in relation to Hypotheses 1 to 3, respectively.

4.2 Feature Engineering (FE)

A sample of the results using the algorithm to process the feature engineering is demonstrated in Table 4.1.

User Story 1	**Original:**	*As a doctor, I want to login to the system with pass level 2 and perform the cornea tunnel surgery so that I can verify the GUI parameters.*
	FE Algorithms Applied:	*doctor want login system passlevel2 perform cornea tunnel surgery verify gui parameter*
User Story 2	**Original:**	*As a field service engineer (FSE), I want to login to the system with pass level 1, verify the IP address and configuration of the system, and make sure that the DMM and ORClient sync and communicate with each other.*
	FE Algorithms Applied:	*fse field service engineer want login system passlevel1 verify ipaddress configuration system make sure dmm orclient sync communicate*
User Story 3	**Original:**	*As a developer, I want to verify system errors during login events and check the kernel memory dump default configuration.*
	FE Algorithms Applied:	*developer want verify system error event log check kernel memory dump default configuration*
User Story 4	**Original:**	*As a verification engineer, I want to be able to run automated tests on remote cloud-based test agents so that I can have an on-demand, scalable pool of reliable, easy-to-maintain test environments.*
	FE Algorithms Applied:	*verification engineer want able run automate test remote cloudbased test agent ondemand scalable pool reliable easy main test environment*
User Story 5	**Original:**	*As a software engineer, I want to move the log manager API to the logging API so that the same code can be used for all subsystem components.*
	FE Algorithms Applied:	*software engineer want move logmanager api log api code use subsystem component*

Table 4.1. Sample of Feature Engineering Results

As discussed, in section 3.4, identifying the unique words is a key to calculating the term frequency by inverse document frequency. The result is the core for the model

algorithm to perform the prediction for the required type of testing. The following Table 4.2 are the unique words identified in the corpus section of the dataset from sample result of feature engineering.

Unique Words Identified in Corpus							
run	error	code	api	log	surgery	event	component
sure	doctor	kernel	easy	gui	scalable	memory	passlevel2
fse	able	automate	sync	reliable	cloud	tunnel	developer
dmm	pool	main	use	default	agent	verify	configuration
engineer	test	remote	want	service	make	perform	On-demand
cornea	login	or-client	field	check	move	dump	communicate
system	verification		passlevel1		ip-address		parameter
environment		log-manager		subsystem			

Table 4.2. Unique Words Identified in Corpus

4.3 Model Evaluation and Ranking

The TF-IDF evaluations for the *corpus* and the *source* differ from each other. The NLP model evaluated each *corpus* and *source* as follows:

4.3.1 Corpus Vectorization and Ranking Evaluation

As discussed in section 3.4, this is the results of the corpus vectorization and ranking evaluation for two sample user story 1 and 2 displays in Table 4.3:

Words (User Story 1)	TF-IDF Rank (User Story 2)
Sync	0.0668
Component	0.0668
Check	0.0668
Software	0.0668
Error	0.0668
Parameter	0.0668
Engineer	0.0334

Table 4.3. Sample of Corpus Vectorization and Ranking Evaluation

Table 4.4 shows the TF-IDF vectorization ranking with respect to *corpus* evaluation to determine the required type of testing and assign that agile user story to that type. Appendix A contains more samples with different scenarios of the below table.

Final words (User Story 5)	TF-IDF rank (User Story 5)	Highest TF-IDF rank in source file	Words assigned to test user story #	Type of required testing for user story #
Sync	0.0668	0.0334	User Story 2	Communication, Security
Engineer	0.0334	0.0250	User Story 4	System
Component	0.0668	0.0501	User Story 4	System
Check	0.0668	0.0463	User Story 3	Communication
Software	0.0668	0.0501	User Story 4	System
Error	0.0668	0.0463	User Story 3	Communication
Parameter	0.0668	0.0501	User Story 1	Security, Surgical

Table 4.4. Sample of Source Vectorization & Ranking Evaluation with Comparison to Corpus

4.4 Hypothesis 1 and Results

H1: *The proposed model can learn the format of agile user stories to identify the keywords.*

The following tables presents the keywords that were identified by the NLP model for each type of testing. Discussion of these results is presented in Section 5.1.1.

Expected Communication Testing Keywords Type	Identified Communication Testing Keywords Type
Communication	Communication
Configuration	Configuration
Interact	Interact
IP-Address	IP-Address
Log	Log
Network	Network
Synchronization	Synchronization
Upload	Upload

Table 4.5. Expected Vs. Identified Communication Testing Keywords

Expected Security Testing Keywords Type	Identified Security Testing Keywords Type
Alert	Alert
Investigate	Investigate
Monitor	Monitor
Pass-level	Pass-level
Security	Security

Table 4.6. Expected Vs. Identified Security Testing Keywords

Expected Surgical Testing Keywords Type	Identified Surgical Testing Keywords Type
Cataract	Cataract
Doctor	Doctor
Exam	Exam
Eye	Eye
Measurement	Measurement
Operation	Operation
Patient	Patient
Surgeon	Surgeon
Surgery	Surgery
Surgical	Surgical

Table 4.7. Expected Vs. Identified Surgical Testing Keywords

Expected System Testing Keywords Type	Identified System Testing Keywords Type
Diagnose	Diagnose
Integrate	Integrate
Interact	Interact
Interface	Interface
Navigate	Navigate
Register	Register
Subsystem	Subsystem
System	System

Table 4.8. Expected Vs. Identified System Testing Keywords

4.5 Hypothesis 2 and Results

H2: *A Natural Language Processing model can accurately predict the required type of testing.*

The NLP model developed in this praxis classified the agile user stories into eleven different categories based on the type of software testing required by the story. The model assigned all 500 user stories to one of the following classifications as shown in Table 4.9. Discussion of these results is presented in Section 5.1.2.

Expected Classification Type
Communication
Communication, Security
Communication, Security, System
Communication, Surgical
Communication, System
Security
Security, Surgical
Security , System
Surgical
Surgical, System
System

Table 4.9. Expected Type of Classification

As discussed in section 4.4, the model uses one algorithm to categorize agile user stories with keywords. These are matched to the lists of keywords for each type of testing required. For stories without keywords TF-IDF, is used to identify the required type of testing. Table 4.10 shows part of the results extracted from “confusion_matrix.csv”.

Number of User Stories	Story Number Predicted	Predicted Category	Actual Category
1	1	System	System
2	4	System	System
3	13	System	System
4	18	System	System
5	23	Surgical	Surgical
6	26	Communication, System	Surgical
7	27	Communication	Communication
8	28	Communication	Communication
9	34	System	System
10	42	System	System
11	45	System	System
12	50	System	System
13	52	System	System
14	64	Communication	Communication
15	65	Communication	Communication
16	74	System	System
17	79	Communication, Security	System
18	81	System	System
19	97	Security	Communication, System
20	101	System	System

Table 4.10. Sample Results of Predicted Vs. Actual Category Comparison

Figure 4.1.is the printout result from the model which is used to calculate: misclassification rate, error rate, precision, sensitivity or recall, F1-score, and classification accuracy.

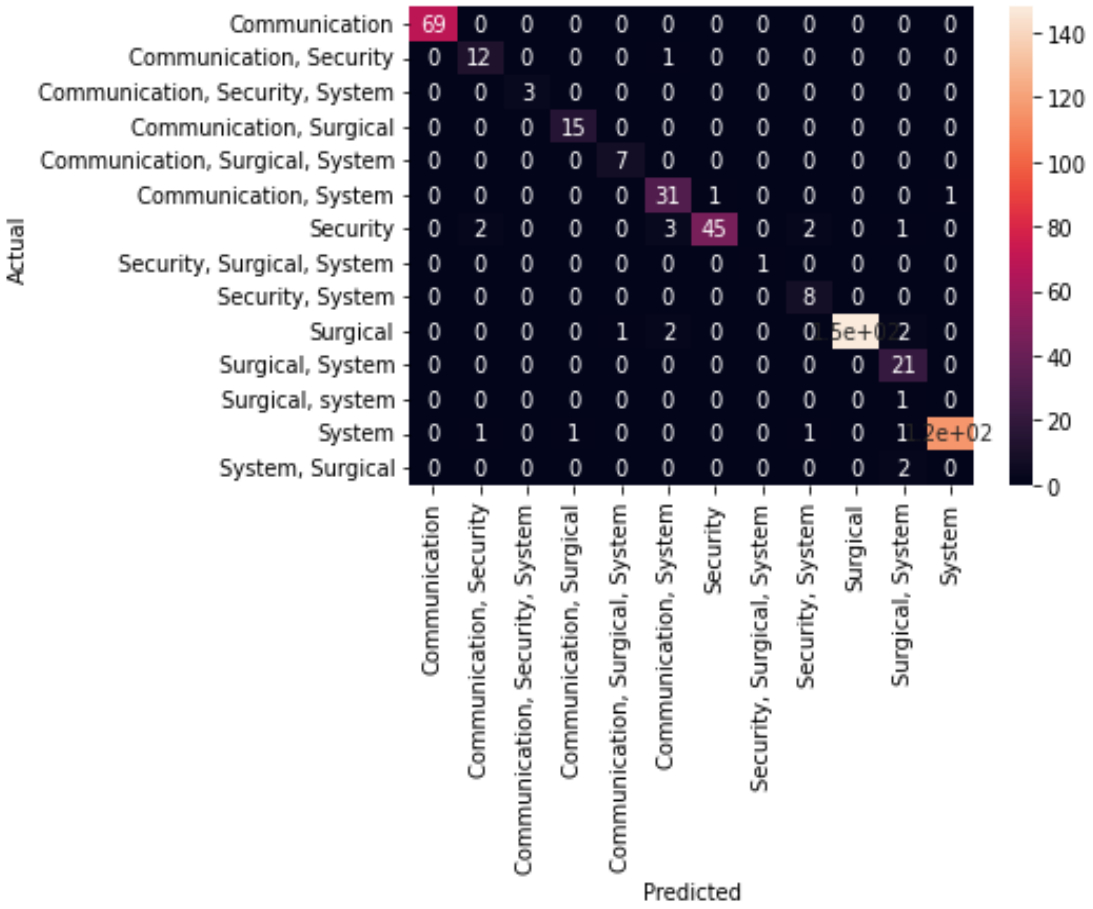


Figure 4.1. Confusion Matrix Output From the Model

As discussed in section 3.5.1 the following are the results from confusion matrix.

True Positives (TP): Out of 500 agile user stories, the model predicted 481 correctly. The NLP model's true positive rate is 481.

TP = 481.

False Positive (FP): Out of 500 agile user stories, the model predicted 8 users stories completely wrong. This is also considered as a Type (I) error. The NLP model false positive rate is 8.

FP = 8.

False Negative (FN): Out of 500 agile user stories, the model cannot predict all the required types of testing for 11 user stories. This is also considered as a Type (II) error. The NLP model's false negative rate is 11.

FN = 11.

True Negative (TN): All the 500 agile user story is predicted to be one or more required type of testing regardless of if it is correct or not. The NLP model's true negative rate is zero.

TN = 0.

In the context of the confusion matrix, the true negative is a case when the model predicted false, and the correct class was false. Zero true negative results in the confusion matrix demonstrate the low specificity. This means the number of observations the model predicted were negative that were actually negative is low. The NLP classification model developed in this praxis, as mentioned in section 3.6.1. is related to the cases when the classifier cannot predict the required type of testing regardless of right or wrong. This is not the case in prediction and classification in this praxis. All the user stories should have

a required type of testing and there are no instances that a user story from *source* or *corpus* does not assign a classification.

Table 4.11 provide the statistical results from the confusion matrix developed in this praxis. Table 4.12 provide the confusion matrix with emphasis on the error type.

Type	Equation Formula	Equation Result	Final Result	Result in %
Misclassification	$(FP + FN)$	$(8 + 11)$	**19**	**-**
Misclassification Error Rate	$\frac{(FP + FN)}{(TP + TN + FP + FN)}$	$\frac{(8 + 11)}{(481 + 0 + 8 + 11)}$	**0.038**	**3.8%**
Precision	$\frac{TP}{TP + FP}$	$\frac{481}{481 + 8}$	**0.9833**	**98.33%**
Recall	$\frac{TP}{TP + FN}$	$\frac{481}{481 + 11}$	**0.9776**	**97.66%**
F1-Score	$\frac{TP}{TP + \frac{1}{2}(FP + FN)}$	$\frac{481}{481 + \frac{1}{2}(8 + 11)}$	**0.9806**	**98.06%**
Accuracy	$\frac{TP + TN}{(TP + TN + FP + FN)}$	$\frac{481 + 0}{(481 + 0 + 8 + 11)}$	**0.962**	**96.2%**

Table 4.11. Confusion Matrix Statistical Results

Total Stories = 500	Actual		
Predicted	TP = 481	FP = 8 Type (I)	*489*
	FN = 11 Type (II)	TN = 0	*11*
	492	*8*	**500**

Table 4.12. Confusion Matrix Result with Error Type

4.6 Hypothesis 3 and Results

H3: *The proposed model can significantly improve software testing time compared to manual classification.*

This section compares the NLP model to the current methods. Discussion of these results is presented in Section 5.1.3.

4.6.1 Historical Agile User Story Classification Data Comparison

According to the company's historical data, the manual classification of 500 user stories requires the following steps. First, a software verification and validation (SV&V) engineer reads the stories to review and understand their requirements. The historical data

shows that this task takes five to seven working days; each day is eight hours long. This number could vary depending on the SVV's product knowledge, level of understanding, and fluency in English and medical terminology. It is also impacted by whether the SVV has to re-read or review the requirements more than once or ask a project lead to confirm their findings. The historical data indicates that the average time for this task is six eight-hour working days. The following equation calculates the total number of minutes needed to perform this task manually:

6 (days) * 8 (hours) * 60 (minutes) = 2,880 (minutes)

According to the U.S. Bureau of Labor Statistics (US BLS, 2021), the mean hourly wage rate for an engineer is $51.83. Therefore, this manual task would cost the organization $2,487.84:

6 (days) * 8 (hours) * 51.83 (USD) = 2,487.84 (USD)

To obtain the most reliable time measurement, the model analyzed the 500 user stories three times, and the average time for all three analyses is used here to calculate the cost. On average, it took the model five minutes to analyze the stories. Compared to manual classification, which takes an average of 2,880 minutes, this represents an improvement of 99.83%. For the overall 500 user stories dataset which is used in this praxis $2,487.84 is saved. If the organization needs to develop 10,000 user stories in a year, then the total saving would be at least $49,756.80 regardless of other related expenses.

Chapter 5 — Discussion and Conclusions

5.1 Discussion

This research praxis aims to identify ways to replace manual work with machine learning (ML) methods during the software development lifecycle (SDLC) in agile methodology. This reduction provides more time for the software verification and validation (SV&V) team to conduct comprehensive testing, resulting in higher quality software and fewer software patches, release versions, and product recalls due to software bugs and issues. This praxis focuses on agile methodology because this approach uses shorter iterations than any other SDLC methods, such as Waterfall. This means it is even more challenging to find sufficient time for software testing in agile method.

Previous literature describes many different efforts to reduce the time and cost of software testing during the SDLC using a variety of methods, approaches, and models. Releasing bug-free, high-quality software products is a constant challenge for various industries that use software in their products, however; earlier solutions are not easy or straightforward to implement, and they do not lead to simple, easy-to-understand improvements. Furthermore, many available models and methods seek to develop novel approaches instead of directly solving the issues with existing approaches.

The following sections apply the present study's results to the hypotheses and research questions. Sections 5.1.1 to 5.1.3 discuss hypotheses 1, 2, and 3. Section 5.2 describes the conclusions and suggests some ways that insights from this praxis can be applied to different methodologies and a variety of industries in the future; it also identifies the lessons learned during this study and the development of the model. Section

5.3 describes the contributions of this research to the current body of knowledge, and finally, section 5.4 concludes this praxis with recommendations for future research.

5.1.1 Discussion of Hypothesis 1

H1: *The proposed model can learn the format of agile user story to identify the keywords.*

Hypothesis 1 was tested using a dataset of 500 agile user stories; as shown in the results section 4.5, the model can correctly identify user stories with one type of testing and can learn the format to identify keywords correctly as shown in Tables 4.4 to 4.7.

Result: Hypothesis 1 confirmed.

5.1.2 Discussion of Hypothesis 2

H2: *A Natural Language Processing model can accurately predict the required type of testing.*

Hypothesis 2 was tested using the same dataset as hypotheses 1. The model accurately predicted 96.2% of all the required type of testing as shown in Table 4.10

Result: Hypothesis 2 confirmed.

5.1.3 Discussion of Hypothesis 3

H3: *The proposed model can significantly reduce software testing time compared to manual classification.*

Hypothesis 3 was also tested using the same dataset of 500 agile user stories. As discussed in section 4.7.1, the model improved testing time by 99.83% compared to manual classification.

Result: Hypothesis 3 confirmed.

5.2 Conclusion

During the development of the model used in this praxis, it was important to eliminate any manual work for the software engineering and software verification and validation (SV&V) teams. Any time saved during this process can be used during other phases of the software development life cycle (SDLC) particularly to focus on comprehensive testing. Testing impacts SDLC time and cost, particularly in processes with short iterations such as agile methodology (Rehman et al., 2020). A model that reduces cost and saves time to perform more comprehensive testing is the most effective approach to increase the quality of the released software to the market. The proposed Natural language processing (NLP) model effectively and efficiently reduce the time for classification of agile user stories as well as testing cost. The following list presents the key features of the model and results.

- The proposed model performed best at accurately identifying keywords in agile user stories and categorizing them based on the required type of testing.

- The result from the model shows when the user story does not have keywords, the algorithm developed in the model can still classify them with acceptable accuracy and identify the required type of testing.
- The proposed model improved software testing time by reducing the amount of manual work needed to classify items in the dataset.
- The proposed methodology suggests that this could reduce overall project costs, including rework, delay, redesign, retest, and investigation.

This research also extends the existing body of knowledge through the development of the most accurate NLP model for the classification of agile user stories by answering the research questions (RQ) 1 to 3.

RQ1: How can the proposed model identify the keywords for the required type of testing?

RQ1 Response: Agile user story has a standard format of as a person (who) wants to accomplish something; I want (what) function or type of testing is needed; so that (why) this function, or type of testing is needed. This format feed into the model and all the unnecessary words are omitted. The remaining words are compared to the list of words for the different required types of testing to identify the keywords.

RQ2: Does the proposed model predict the required type of testing?

RQ2 Response: The result from the model which is demonstrated in Table 4.15 shows the proposed model can predict the required type of testing. The accuracy of the

final result which is shown in detail in Table 4.16 confirms the model predicts the required type of testing with an acceptable result.

RQ3: Is there a significant improvement in software testing time using the proposed model as compared to manual classification?

RQ3 Response: By comparing the time for manual classification from the internal team of subject matter experts to the run time of the model, there is a significant improvement in software testing time using the proposed model. The cost is also measured between these two methods and the result shows there is a significant saving amount by using the NLP model.

5.2.1 Lessons Learned

The following are key lessons learned from this praxis and the development of the NLP model; they should be considered and incorporated into future research and analyses.

- The software requirements of agile user stories should be listed in the files, preferably using Microsoft Excel comma-separated value (CSV) file extensions. This is important since Python, the programming language used to develop the model, can understand this format.
- The software requirement for each agile user story should be written in a different cell of the Excel file; otherwise, the model cannot understand and learn the format of agile user stories to identify the keywords and accurately identify the required type of testing.

- Part-of-speech (POS) tagging is mandatory part of feature engineering during model development. Although the model was able to perform the lemmatization without POS tagging, the results were not correct, and the model then assigned user stories to the wrong types of testing. POS tagging should be done before lemmatization during model development.
- In order to get the best possible result from the model and reduce software testing time as much as possible, as discussed in section 5.1.2, the model should be run on a computer with at least 16GB RAM. Ideally, the computer should have as few other programs as possible installed on it, and no other program should be run while the model is running to obtain the ideal model running time of five minutes.
- Extra lines of code should be cleaned up and removed to further reduce the time the model needs to categorize the agile user stories and to prevent memory leaks.

5.3 Contributions to the Body of Knowledge

The model developed in this praxis uses a quantitative approach to reduce software testing time and software development project costs. The model can accurately identify keywords in an English text. The results also provide a reliable and accurate NLP model in terms of F-Measure, misclassification rate, precision, recall, and accuracy without using prior knowledge of the target project. This NLP model contributes to existing ML models as it can identify the required type of testing during agile iterations. The proposed model is distinguishable from any other models by using the term

frequency by inverse document frequency (TF-IDF) algorithm to classify the agile user stories even without any keywords with an acceptable accuracy rate.

5.4 Recommendations for Future Research

In the future, the model can be developed to perform automation testing. This would even further reduce the time and cost of the SDLC. The model developed in this praxis was based on agile methodology. In the future, this model could be applied to different methodologies, such as Waterfall. Additionally, the 500 agile user stories from one project and one sprint team for the model is from a medical device company but in the future could be expanded to address a heterogenous dataset from multiple projects and multiple industries. Finally, future researchers could use different libraries and train and tune the model manually to determine whether the model could then classify user stories with 100% accuracy.

References

Abrahamsson, Salo, O., Ronkainen, J., & Warsta, J. (2017). *Agile Software Development Methods: Review and Analysis*.

Alpaydin. (2014). *Introduction to machine learning* (Third edition.). MIT Press.

Amanda Casari, & Alice Zheng. (2018). *Feature Engineering for Machine Learning*. O'Reilly Media.

Arora, M., Verma, S., Kavita, Chopra, S. (2020). A Systematic Literature Review of Machine Learning Estimation Approaches in Scrum Projects. In: Mallick, P., Balas, V., Bhoi, A., Chae, GS. (eds) Cognitive Informatics and Soft Computing. *Advances in Intelligent Systems and Computing, vol 1040*. Springer, Singapore. https://doi.org/10.1007/978-981-15-1451-7_59

A. U. Rehman, A. Nawaz, M. T. Ali and M. Abbas, "A Comparative Study of Agile Methods, Testing Challenges, Solutions & Tool Support," *2020 14th International Conference on Open-Source Systems and Technologies (ICOSST)*, 2020, pp. 1-5, doi: 10.1109/ICOSST51357.2020.9332965.

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. https://doi.org/10.4304/jait.1.1.4-20

Bratsas, C., Koupidis, K., Salanova, J.M., Giannakopoulos, K., Kaloudis, A., & Aifadopoulou, G. (2019). A Comparison of Machine Learning Methods for the Prediction of Traffic Speed in Urban Places. *Sustainability*, *12*, 142, 1-15. https://doi.org/10.3390/su12010142

Breiman. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Broniatowski, D., & Tucker, C. (2017). Assessing Causal Claims about Complex Engineered Systems with Quantitative Data: Internal, External, and Construct Validity. *Systems Engineering*, *20*, 483-496. https://doi.org/10.1002/sys.21414

Burnstein, I. (2003). *Practical Software Testing*, Springer

Cabitza, Campagner, A., Soares, F., García de Guadiana-Romualdo, L., Challa, F., Sulejmani, A., Seghezzi, M., & Carobene, A. (2021). The importance of being external. methodological insights for the external validation of machine learning models in medicine. *Computer Methods and Programs in Biomedicine*, *208*, 106288–106288. https://doi.org/10.1016/j.cmpb.2021.106288

Caivano, D., Oivo, M., Baldassarre, M. T., & Visaggio, G. (2011). Product-Focused Software Process Improvement. In *12th International conference, PROFES 2011, Torre Canne, Italy, June 2011 Proceedings*.

CISQ Publishes the Cost of Poor Software Quality in the U.S.: A 2020 Report. (2021). *ICT Monitor Worldwide*

Cobb. (2015). *The project manager's guide to mastering agile: principles and practices for an adaptive approach* (1st edition). Wiley.

Connor & MacDonell, S. (2006). Using Historical Data in Stochastic Estimation of Software Project Duration. *19th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ'06). At: Wellington, New Zealand*

Dalpiaz, D. (2020). R for Statistical Learning. *Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License*. Retrieved on June 5, 2022, from: https://daviddalpiaz.github.io/r4sl/knn-class.html

Denny, M. J., & Spirling, A. (2018). Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, And What To Do About It. *Political Analysis*, *26*(2), 168–189. https://doi.org/10.1017/pan.2017.44

Ekwoge, A. Fontão and A. C. Dias-Neto, "Tester Experience: Concept, Issues and Definition," *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, 2017, pp. 208-213, doi: 10.1109/COMPSAC.2017.232.

Etaiwi, & Naymat, G. (2017). The Impact of applying Different Preprocessing Steps on Review Spam Detection. *Procedia Computer Science*, *113*, 273–279. https://doi.org/10.1016/j.procs.2017.08.368

Explorium. (n.d.). *Data Science, Confusion Matrix*. Retrieved November 6, 2022, from https://www.explorium.ai/wiki/confusion-matrix/

Falessi, Cantone, G., & Canfora, G. (2010). A comprehensive characterization of NLP techniques for identifying equivalent requirements. *ESEM 2010 - Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10. https://doi.org/10.1145/1852786.1852810

Gautam, Singh, V., & Darren, D. (2018). The state-of-the-art in software development effort estimation. *Journal of Software : Evolution and Process*, *30*(12), e1983–n/a. https://doi.org/10.1002/smr.1983

Gregory, & Crispin, L. (2014). *More Agile Testing: Learning Journeys for the Whole Team* (1st edition). Addison-Wesley Professional.

Halkidi, Spinellis, D., Tsatsaronis, G., & Vazirgiannis, M. (2011). Data mining in software engineering. *Intelligent Data Analysis, 15*(3), 413–441. https://doi.org/10.3233/IDA-2010-0475

Helmut Schmid. 2013. Probabilistic part-of-speech tagging using decision trees. *In New methods in language processing.* Routledge, page 154.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*(7825), 357-362.

Highsmith, J. (2009). *Agile Project Management: Creating Innovative Products.* Pearson Education

Holbrook, Hayes, J. H., Dekhtyar, A., & Li, W. (2012). A study of methods for textual satisfaction assessment. *Empirical Software Engineering : an International Journal*, *18*(1), 139–176. https://doi.org/10.1007/s10664-012-9198-8

Hossin, M., & Sulaiman, M. N. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11. https://doi.org/10.5121/ijdkp.2015.5201

Idrees, S.M., Alam, M.A., Agarwal, P., Ansari, L. (2019). Effective Predictive Analytics and Modeling Based on Historical Data. *ICACDS 2019. Communications in Computer and Information Science, vol 1046.* Springer, Singapore. https://doi.org/10.1007/978-981-13-9942-8_52

Imandoust, S., & Bolandraftar, M. (2013). Application of K-Nearest Neighbor (kNN) Approach for Predicting Economic Events: Theoretical Background. *International Journal of Engineering Research and Applications, 3*, 5, 605-610.

Jivani, A.G. (2011). A Comparative Study of Stemming Algorithms. *International Journal of Computer Technology Applications, 2*(6), 1930-1938.

Jones, E., Oliphant, T., & Peterson, P. (2001). SciPy: Open-source scientific tools for Python.

Kang, Cai, Z., Tan, C.-W., Huang, Q., & Liu, H. (2020). Natural language processing (NLP) in management research: A literature review. *Journal of Management Analytics*, *7*(2), 139–172. https://doi.org/10.1080/23270012.2020.1756939

Khalid, & Eng-Thiam Yeoh. (2017). Early cost estimation of software reworks using fuzzy requirement-based model. *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, 1–5. https://doi.org/10.1109/ICCCCEE.2017.7866082

Koehrsen, W. (2017). Random Forest Simple Explanation. Retrieved from https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

Kotsiantis, Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: A review of classification and combining techniques. *The Artificial Intelligence Review*, *26*(3), 159–190. https://doi.org/10.1007/s10462-007-9052-3

Larivière, B., & Poel, D. (2005) Predicting Customer Retention and Profitability by Using Random Forests and Regression Forests Techniques. *Expert Systems with Applications, 29,* 472-484, https://doi.org/10.1016/j.eswa.2005.04.043

Larman. (2004). *Agile and iterative development: a manager's guide* (1st edition). Addison-Wesley.

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research, 18*(1), 559-563.

Li, Jiang, H., Wang, D., & Han, B. (2018). An Improved KNN Algorithm for Text Classification. *2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, 1081–1085. https://doi.org/10.1109/IMCCC.2018.00225

Li, Yang, Y., Li, M., Wang, Q., Boehm, B. W., & Hu, C. (2012). Improving software testing process: feature prioritization to make winners of success-critical stakeholders. *Journal of Software: Evolution and Process*, *24*(7), 783–801. https://doi.org/10.1002/smr.512

Liou, J. (2010). On Software Test Estimate and Requirement Tracking. 19th International Conference on Software Engineering and Data Engineering, USA, pp. 57-62.

López-Martín. (2021). Machine learning techniques for software testing effort prediction. *Software Quality Journal*, *30*(1), 65–100. https://doi.org/10.1007/s11219-020-09545-8

Lopez-Martinez, Juarez-Ramirez, R., Huertas, C., Jimenez, S., & Guerra-Garcia, C. (2016). Problems in the Adoption of Agile-Scrum Methodologies: A Systematic Literature Review. *2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 141–148. https://doi.org/10.1109/CONISOFT.2016.30

M. Adnan and M. Afzal, "Ontology Based Multiagent Effort Estimation System for Scrum Agile Method," in *IEEE Access*, vol. 5, pp. 25993-26005, 2017, https://doi.org/10.1109/ACCESS.2017.2771257.

Manna, Swetapadma, A., & Abdar, M. (2019). Decision Tree Predictive Learner-Based Approach for False Alarm Detection in ICU. *Journal of Medical Systems*, *43*(7), 191–13. https://doi.org/10.1007/s10916-019-1337-y

Myers, Sandler, C., & Badgett, T. (2012). *The art of software testing* (3rd ed.). John Wiley & Sons.

Najadat, Alsmadi, I., & Shboul, Y. (2012). Predicting Software Projects Cost Estimation Based on Mining Historical Data. *ISRN Software Engineering*, *2012*. https://doi.org/10.5402/2012/823437

Nayak, A. S., Kanive, A. P., Chandavekar, N., & R, B. (2016). Survey on Pre-Processing Techniques for Text Mining. *International Journal Of Engineering And Computer Science*, *5*(6), 16875–16879. https://doi.org/10.18535/ijecs/v5i6.25

Okamura, H, Dohi, T & Bestoun, S. (2018). Optimizing Testing-Resource Allocation Using Architecture-Based Software Reliability Model. *Journal of Optimization*, *2018*, 1–7. https://doi.org/10.1155/2018/6948656

Oshiro, Perez, P. S., & Baranauskas, J. A. (2012). How Many Trees in a Random Forest? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7376, pp. 154–168). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31537-4_13

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research, 12*, 2825-2830.

Pospieszny, P., Czarnacka-Chrobot, B., & Kobylinski, A. (2018). An Effective Approach for Software Project Effort and Duration Estimation with Machine Learning Algorithms. *Journal of Systems and Software, 137*, 184-196. https://doi.org/10.1016/j.jss.2017.11.066

Power, & Heavin, C. (2021). *On Becoming Agile*. Business Expert Press.

Rajasekhar, P. R., & Yadav, A. Y. (2013). Critical Issues in Software Testing During Agile Development. *International Journal of Computer Science and Technology*, *4*(2), 1–5.

Reback, J., McKinney, W., Den Van Bossche, J., Augspurger, T., Cloud, P., Klein, A., ... & Seabold, S. (2020). pandas-dev/pandas: Pandas 1.0. 3. *Zenodo*.

Ridzuan, & Wan Zainon, W. M. N. (2019). A Review on Data Cleansing Methods for Big Data. *Procedia Computer Science*, *161*, 731–738. https://doi.org/10.1016/j.procs.2019.11.177

Sarkani, S., Holzer, T., Eveleigh, T., & McZara. (2015). Software requirements prioritization and selection using linguistic tools and constraint solvers—a controlled experiment. *Empirical Software Engineering : an International Journal*, *20*(6), 1721–1761. https://doi.org/10.1007/s10664-014-9334-8

Shah, U. S., Jinwala, D. C., & Patel, S. J. (2016). An excursion to software development life cycle models: an old to ever-growing models. ACM SIGSOFT Software Engineering Notes, 41(1), 1-6.

Shameem, Khan, A. A., Hasan, M. G., & Akbar, M. A. (2020). Analytic Hierarchy Process Based Prioritization and Taxonomy of Success Factors for Scaling Agile Methods in Global Software Development. *IET Software*, *14*(4), 389–401. https://doi.org/10.1049/iet-sen.2019.0196

Sharma, & Kumar, D. (2019). Agile Release Planning Using Natural Language Processing Algorithm. *2019 Amity International Conference on Artificial Intelligence (AICAI)*, 934–938. https://doi.org/10.1109/AICAI.2019.8701252

Shrivastava, Jaggi, I., Katoch, N., Gupta, D., & Gupta, S. (2021). A Systematic Review on Extreme Programming. *Journal of Physics: Conference Series*, *1969*(1), 12046–. https://doi.org/10.1088/1742-6596/1969/1/012046

Silva, & Ribeiro, B. (2003). The importance of stop word removal on recall values in text categorization. *Proceedings of the International Joint Conference on Neural Networks, 2003*, *3*, 1661–1666 vol.3. https://doi.org/10.1109/IJCNN.2003.1223656

Sun Rui and Zhong Deming, "Translating software requirement from natural language to automaton," *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, 2013, pp. 2456-2459, https://doi.org/10.1109/MEC.2013.6885449.

The financial crisis inquiry report final report of the National Commission on the Causes of the Financial and Economic Crisis in the United States (Authorized ed.). (2011). Public Affairs.

The Government of California. (2022). https://www.ca.gov/

The Standish Group. (2012). *CHAOS MANIFESTO 2012: The Year of the Executive Sponsor*. The Standish Group.

Thomas. (2020). *Natural language processing with spark NLP: learning to understand text at scale* (First edition.). O'Reilly.

Thompson, Hu, J., Mudaranthakam, D. P., Streeter, D., Neums, L., Park, M., Koestler, D. C., Gajewski, B., Jensen, R., & Mayo, M. S. (2019). Relevant Word Order Vectorization for Improved Natural Language Processing in Electronic Health Records. *Scientific Reports*, *9*(1), 9253–9259. https://doi.org/10.1038/s41598-019-45705-y

U.S. Bureau of Labor Statistics. (2021). *Occupational Employment and Wage Statistics*. https://www.bls.gov/oes/current/oes172199.htm

Vabalas A, Gowen E, Poliakoff E, Casson AJ (2019) Machine learning algorithm validation with a limited sample size. PLoS ONE 14(11): e0224365. https://doi.org/10.1371/journal. pone.0224365

Vijayarani, S., Ilamathi J., Nithya. (2015). Preprocessing Techniques for Text Mining – An Overview. *International Journal of Computer Science & Communication Networks*, *5*(1), 7-16.

Wang, Xia, S.-T., Tang, Q., Wu, J., & Zhu, X. (2018). A Novel Consistent Random Forest Framework: Bernoulli Random Forests. *IEEE Transaction on Neural Networks and Learning Systems*, *29*(8), 3510–3523. https://doi.org/10.1109/TNNLS.2017.2729778

Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information*

and Software Technology, Elsevier., 54, 41–59. https://doi.org/10.1016/j.infsof.2011.09.002.

Yahav, Shehory, O., & Schwartz, D. (2019). Comments Mining With TF-IDF: The Inherent Bias and Its Removal. *IEEE Transactions on Knowledge and Data Engineering*, *31*(3), 437–450. https://doi.org/10.1109/TKDE.2018.2840127

Zhang, & Tsai, J. J. . (2003). Machine learning and software engineering. *Software Quality Journal*, *11*(2), 87–119. https://doi.org/10.1023/A:1023760326768

Zhuohao, Dong, W., & Qing, L. (2021). Keyword Extraction from Scientific Research Projects Based on SRP-TF-IDF. *CHINESE JOURNAL OF ELECTRONICS*, *30*(4), 652–657. https://doi.org/10.1049/cje.2021.05.007

Appendix A

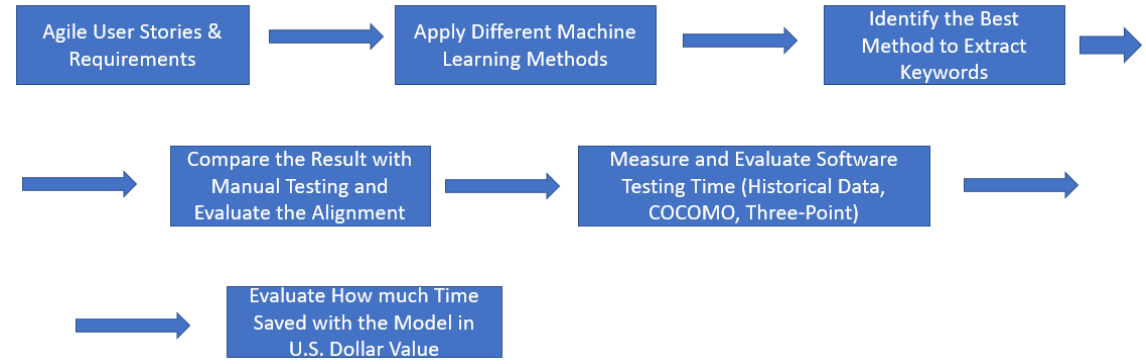


Figure A-1 Research Scope

Figure A-2 *os.patch* and *string* python libraries created in order to read .CSV raw data file from the data source.

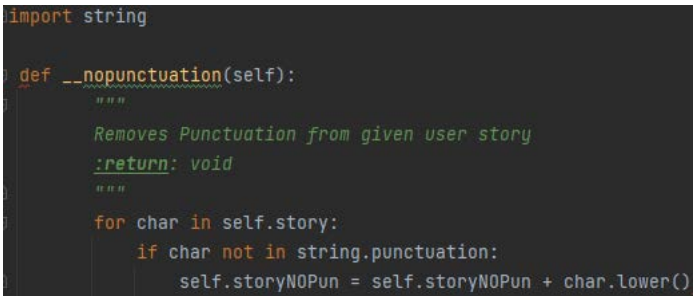
```
import os.path as ospath
import string

"""
Parse the main CSV file and get lines from the stories.

:parameter f: file name of the main CSV file
:returns a list of lines from the user story CSV file, removes the header
"""
def get_csv_stories(f : str):
    assert ospath.exists(f)
    with open(f, "r") as file:
        lines = file.readlines()
    return lines[0:]
```

Figure A-2 Read the .CSV Raw Data File

Figure A-3 Using Python *string* library and *string.punctuation* in the model remove punctuation from the data source.



```
import string

def __nopunctuation(self):
        """
        Removes Punctuation from given user story
        :return: void
        """
        for char in self.story:
            if char not in string.punctuation:
                self.storyNOPun = self.storyNOPun + char.lower()
```

Figure A-3 Remove Punctuation From Each Given Agile User Story

Figure A-4 Using Python *nltk* and *stopwords* library and *string.stopwrods* in the model perform this task.

```
import nltk
from nltk.corpus import stopwords
import string

def __nostopwords(self):
    """
    Removes all known stop words from given user story
    :return: void
    """
    for word in self.storyNOPun.split():
        if word not in stopwords.words('English'):
            self.storyNOStopWords += " " + word
```

Figure A-4 Remove Stopwords From Each Given Agile User Story

Figure A-5 Using Python *nltk* library and *WordNetLemmatizer* with considering the POS tag in the model perform this task.

```
import nltk

nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer


def lemmatizestory(self):
    """
    Lemmatizes given story
    :return: Lemmatized words with pos
    """
    self.__nopunctuation()
    self.__nostopwords()
    pos_word = {}
    lmtzr = WordNetLemmatizer()
    for word in self.__tokenizestory():
        if word not in self.pos_customization:
            pos_word[word] = nltk.pos_tag([word])[0][1][0]
        else:
            pos_word[word] = self.pos_customization[word]
        # print(word, pos_word[word])
        if pos_word[word] in ["N", "V"]:
            self.lemmatizedStory += " " + lmtzr.lemmatize(word, pos_word[word].lower())
        else:
            self.lemmatizedStory += " " + word
    self.__getwordlist()
    return pos_word
```

Figure A-5 lemmatizing the word with considering the type (POS tag)

Figure A-6 Python *sklearn, pandas and numpy* library and *TfidfVectorizer()* function perform the TD -IDF calculation.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Term Frequency (TF): Number of times a word appears in a document divided
by the total number of words in the
# document. The following code implements Term Frequency (TF).

def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict


tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)

print(tfA)
print("\n")
print(tfB)


# Inverse Data Frequency (IDF): Weight of rare words across all documents in
the corpus.
# The log of the number of documents divided by the number of documents that
contain the word w.
# The following code implements Inverse Data Frequency (IDF).
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict


def computeTFIDF(tfBow, idfs):
    tfidf = {}
    for word, val in tfBow.items():


# The IDF is computed once for all documents.
idfs = computeIDF([numOfWordsA, numOfWordsB])
print("\n")
print(idfs)
```

Figure A-6 TF – IDF Calculation for Agile User Story

The calculation of TF vectorization is shown in Table A-1, the calculation of IDF vectorization is shown in Table A-2 and finally, the TF-IDF calculation is displayed in Table A-3.

TF of:

	run	error	code	api	log	surgery	event	engineer	default	cloudbased	sure	doctor	kernel	easy
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.055556	0.000000	0.000000	0.055556	0.000000	0.000000	0.000000
2	0.000000	0.076923	0.000000	0.000000	0.076923	0.000000	0.076923	0.000000	0.076923	0.000000	0.000000	0.000000	0.076923	0.000000
3	0.052632	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.052632	0.000000	0.052632	0.000000	0.000000	0.000000	0.052632
4	0.000000	0.000000	0.083333	0.166667	0.083333	0.000000	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

gui	software	scalable	configuration	memory	test	agent	tunnel	ondemand	able	fse	automate	reliable	sync
0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.055556	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.055556	0.000000	0.000000	0.055556
0.000000	0.000000	0.000000	0.076923	0.076923	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.052632	0.000000	0.000000	0.157895	0.052632	0.000000	0.052632	0.052632	0.000000	0.052632	0.052632	0.000000
0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

developer	component	verify	service	dmm	pool	orclient	logmanager	make	main	subsystem	perform	use	passlevel2
0.000000	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000	0.083333
0.000000	0.000000	0.055556	0.055556	0.055556	0.000000	0.055556	0.000000	0.055556	0.000000	0.000000	0.000000	0.000000	0.000000
0.076923	0.000000	0.076923	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.052632	0.000000	0.000000	0.000000	0.052632	0.000000	0.000000	0.000000	0.000000
0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000	0.000000	0.083333	0.000000	0.083333	0.000000

communicate	login	remote	want	check	environment	ipaddress	verification	system	field	parameter	dump	move	cornea
0.000000	0.083333	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000	0.083333	0.000000	0.000000	0.083333
0.055556	0.055556	0.000000	0.055556	0.000000	0.000000	0.055556	0.000000	0.111111	0.055556	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.076923	0.076923	0.000000	0.000000	0.000000	0.076923	0.000000	0.000000	0.076923	0.000000	0.000000
0.000000	0.000000	0.052632	0.052632	0.000000	0.052632	0.000000	0.052632	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.083333	0.000000

Table A-1. Term Frequency Vectorization Result

IDF of:

run:	0.6989700043360189	pool:	0.6989700043360189
error:	0.6989700043360189	orclient:	0.6989700043360189
code:	0.6989700043360189	logmanager:	0.6989700043360189
api:	0.6989700043360189	make:	0.6989700043360189
log:	0.3979400086720376	main:	0.6989700043360189
surgery:	0.6989700043360189	subsystem:	0.6989700043360189
event:	0.6989700043360189	perform:	0.6989700043360189
engineer:	0.2218487496163564	use:	0.6989700043360189
default:	0.6989700043360189	passlevel2:	0.6989700043360189
cloudbased:	0.6989700043360189	communicate:	0.6989700043360189
sure:	0.6989700043360189	login:	0.3979400086720376
doctor:	0.6989700043360189	remote:	0.6989700043360189
kernel:	0.6989700043360189	want:	0.0
easy:	0.6989700043360189	check:	0.6989700043360189
gui:	0.6989700043360189	environment:	0.6989700043360189
software:	0.6989700043360189	ipaddress:	0.6989700043360189
scalable:	0.6989700043360189	verification:	0.6989700043360189
configuration:	0.3979400086720376	system:	0.2218487496163564
memory:	0.6989700043360189	field:	0.6989700043360189
test:	0.6989700043360189	parameter:	0.6989700043360189
agent:	0.6989700043360189	dump:	0.6989700043360189
tunnel:	0.6989700043360189	move:	0.6989700043360189
ondemand:	0.6989700043360189	cornea:	0.6989700043360189
able:	0.6989700043360189		
fse:	0.6989700043360189		
automate:	0.6989700043360189		
reliable:	0.6989700043360189		
sync:	0.6989700043360189		
developer:	0.6989700043360189		
component:	0.6989700043360189		
verify:	0.2218487496163564		
service:	0.6989700043360189		
dmm:	0.6989700043360189		

Table A-2. Inverse Document Frequency Vectorization Result

TF * IDF of:

	run	error	code	api	log	surgery	event	engineer	default	cloudbased	sure	doctor	kernel	easy
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.058248	0.000000	0.000000	0.000000	0.000000	0.000000	0.058248	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.012325	0.000000	0.000000	0.038832	0.000000	0.000000	0.000000
2	0.000000	0.053767	0.000000	0.000000	0.030611	0.000000	0.053767	0.000000	0.053767	0.000000	0.000000	0.000000	0.053767	0.000000
3	0.036788	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.011676	0.000000	0.036788	0.000000	0.000000	0.000000	0.036788
4	0.000000	0.000000	0.058248	0.116495	0.033162	0.000000	0.000000	0.018487	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

gui	software	scalable	configuration	memory	test	agent	tunnel	ondemand	able	fse	automate	reliable	sync
0.058248	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.058248	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.022108	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.038832	0.000000	0.000000	0.038832
0.000000	0.000000	0.000000	0.030611	0.053767	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.036788	0.000000	0.000000	0.110364	0.036788	0.000000	0.036788	0.036788	0.000000	0.036788	0.036788	0.000000
0.000000	0.058248	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

developer	component	verify	service	dmm	pool	orclient	logmanager	make	main	subsystem	perform	use	passlevel2
0.000000	0.000000	0.018487	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.058248	0.000000	0.058248
0.000000	0.000000	0.012325	0.038832	0.038832	0.000000	0.038832	0.000000	0.038832	0.000000	0.000000	0.000000	0.000000	0.000000
0.053767	0.000000	0.017065	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.036788	0.000000	0.000000	0.000000	0.036788	0.000000	0.000000	0.000000	0.000000
0.000000	0.058248	0.000000	0.000000	0.000000	0.000000	0.000000	0.058248	0.000000	0.000000	0.058248	0.000000	0.058248	0.000000

communicate	login	remote	want	check	environment	ipaddress	verification	system	field	parameter	dump	move	cornea
0.000000	0.033162	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.018487	0.000000	0.058248	0.000000	0.000000	0.058248
0.038832	0.022108	0.000000	0.0	0.000000	0.000000	0.038832	0.000000	0.024650	0.038832	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.0	0.053767	0.000000	0.000000	0.000000	0.017065	0.000000	0.000000	0.053767	0.000000	0.000000
0.000000	0.000000	0.036788	0.0	0.000000	0.036788	0.000000	0.036788	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.058248	0.000000

Table A-3. Term Frequency-Inverse Document Frequency Vectorization Result

Table A-4 shows the complete result of assigning *corpus* story to *source* story based on TF-IDF calculation.

Number of User Stories	Corpus Stories	TF-IDF Calculation Result	Source Stories	Classification
0	1	0.24119301	91	System
1	4	0.23189276	170	System
2	13	0.11409884	62	System
3	18	0.19347447	100	System
4	23	0.27054155	17	Surgical
5	26	0.38485586	139	System, Communication
6	27	0.16079534	46	Communication
7	28	0.16079534	38	Communication
8	34	0.21380881	325	System

9	42	0.27054155	193	System
10	45	0.21380881	77	System
11	50	0.34617217	397	System
12	52	0.50930594	96	System
13	64	0.27054155	19	Communication
14	65	0.32071321	2	Communication
15	74	0.48106982	39	System
16	79	0.18888847	360	Security, Communication
17	81	0.21380881	22	System
18	97	0.1447158	221	Security
19	101	0.21643324	100	System
20	102	0.16035661	84	Surgical, Communication
21	103	0.66110965	234	Communication
22	126	0.21380881	422	Surgical
23	171	0.27054155	280	System, Security
24	173	0.21380881	10	System
25	177	0.27489704	305	System
26	185	0.26444386	125	Surgical
27	187	0.2226397	72	System, Surgical, Communication
28	188	0.24053491	147	Surgical
29	189	0.40843643	133	System, Communication
30	190	0.21380881	122	Surgical
31	191	0.24506186	227	Security
32	192	0.22439521	153	Surgical
33	195	0.24506186	243	Security
34	196	0.26444386	125	Surgical
35	206	0.27489704	332	Surgical
36	238	0.21380881	69	System
37	245	0.50930594	96	System
38	257	0.18036103	421	System
39	268	0.19347447	100	System
40	285	0.19242793	280	System, Security
41	292	0.0801783	20	System, Communication
42	294	0.17493448	446	Surgical
43	309	0.16035661	14	Communication
44	310	0.27489704	305	System
45	314	0.12828529	453	Surgical
46	323	0.24053491	116	Surgical
47	326	0.16232493	71	Communication
48	328	0.35475286	71	Communication
49	335	0.42761762	332	Surgical
50	343	0.32071321	303	Security
51	347	0.19242793	263	Surgical

52	350	0.20569561	153	Surgical
53	351	0.12486533	47	Surgical
54	352	0.31216791	349	Security
55	353	0.14415403	212	System, Surgical
56	354	0.16232493	47	Surgical
57	355	0.14802148	324	System
58	363	0.17493448	209	Security
59	368	0.39663968	394	Communication
60	369	0.24053491	8	System, Communication
61	371	0.24053491	197	System, Communication
62	373	0.21380881	374	System
63	375	0.21380881	296	Surgical
64	377	0.38380092	356	System, Communication
65	378	0.17493448	306	System
66	379	0.24053491	36	System, Security
67	380	0.22657451	390	Security, Communication
68	381	0.26776987	390	Security, Communication
69	383	0.27054155	320	Surgical
70	401	0.48806962	397	System
71	402	0.24973066	397	System
72	414	0.20290616	87	Surgical
73	416	0.12828529	38	Communication
74	440	0.25657057	441	Surgical
75	445	0.29284177	444	System
76	447	0.32071321	443	System, Surgical
77	448	0.12991302	446	Surgical
78	449	0.27054155	422	Surgical
79	451	0.32334559	318	System, Surgical
80	454	0.17493448	456	System, Surgical
81	484	0.14802148	150	Surgical
82	490	0.24053491	500	System

Table A-4. Complete Result of Assigning corpus to source based on TF-IDF Calculation

Table A-5 shows the complete result of the categorization of the entire datasets used in this praxis. The result also shows the type of error identified in the result. As it was discussed in section 4.5, FP considers a Type (I) error and FN a Type (II).

Number of User Stories	Story Number Predicted	Predicted Classification	Actual Classification	Type of Error
1	1	System	System	TP
2	4	System	System	TP
3	13	System	System	TP
4	18	System	System	TP
5	23	Surgical	Surgical	TP
6	26	Communication, System	Surgical	FP
7	27	Communication	Communication	TP
8	28	Communication	Communication	TP
9	34	System	System	TP
10	42	System	System	TP
11	45	System	System	TP
12	50	System	System	TP
13	52	System	System	TP
14	64	Communication	Communication	TP
15	65	Communication	Communication	TP
16	74	System	System	TP
17	79	Communication, Security	System	TP
18	81	System	System	TP
19	97	Security	Communication, System	FP
20	101	System	System	TP
21	102	Communication, Surgical	System	FP
22	103	Communication	Communication	TP
23	126	Surgical	Surgical	TP
24	171	Security, System	System	FN
25	173	System	System	TP
26	177	System	System	TP
27	185	Surgical	Surgical	TP
28	187	Communication, Surgical, System	Surgical	FN

29	188	Surgical	Surgical	TP
30	189	Communication, System	Surgical	FP
31	190	Surgical	Surgical	TP
32	191	Security	Security	TP
33	192	Surgical	Surgical	TP
34	195	Security	Security	TP
35	196	Surgical	Surgical	TP
36	206	Surgical	Surgical	TP
37	238	System	System	TP
38	245	System	System	TP
39	257	System	System	TP
40	268	System	System	TP
41	285	Security, System	Security	FN
42	292	Communication, System	Communication, Security	FN
43	294	Surgical	Surgical	TP
44	309	Communication	Communication	TP
45	310	System	System	TP
46	314	Surgical	Surgical	TP
47	323	Surgical	Surgical	TP
48	326	Communication	Communication	TP
49	328	Communication	Communication	TP
50	335	Surgical	Surgical	TP
51	343	Security	Security	TP
52	347	Surgical	Surgical	TP
53	350	Surgical	Surgical	TP
54	351	Surgical	Surgical	TP
55	352	Security	Security	TP
56	353	Surgical, System	Security	FP
57	354	Surgical	Surgical	TP
58	355	System	System	TP
59	363	Security	Security	TP
60	368	Communication	Communication	TP
61	369	Communication, System	Security	FP
62	371	Communication, System	Security	FP
63	373	System	System	TP

64	375	Surgical	Surgical	TP
65	377	Communication, System	Security	FP
66	378	System	System	TP
67	379	Security, System	Security	FN
68	380	Communication, Security	Security	FN
69	381	Communication, Security	Security	FN
70	383	Surgical	Surgical	TP
71	401	System	System	TP
72	402	System	Communication, System	FN
73	414	Surgical	Surgical	TP
74	416	Communication	Communication	TP
75	440	Surgical	Surgical	TP
76	445	System	System	TP
77	447	Surgical, System	Surgical	FN
78	448	Surgical	Surgical	TP
79	449	Surgical	Surgical	TP
80	451	Surgical, System	System	FN
81	454	Surgical, System	Surgical	FN
82	484	Surgical	Surgical	TP
83	490	System	System	TP
84	2	Communication	Communication	TP
85	3	Communication	Communication	TP
86	5	Communication	Communication	TP
87	6	System	System	TP
88	7	Communication, System	Communication, System	TP
89	8	Communication, System	Communication, System	TP
90	9	Surgical, System	Surgical, System	TP
91	10	System	System	TP
92	11	Communication, System	Communication, System	TP
93	12	System	System	TP

94	14	Communication	Communication	TP
95	15	Communication	Communication	TP
96	16	Communication, Surgical	Communication, Surgical	TP
97	17	Surgical	Surgical	TP
98	19	Communication	Communication	TP
99	20	Communication, System	Communication, System	TP
100	21	System	System	TP
101	22	System	System	TP
102	24	Surgical	Surgical	TP
103	25	Surgical	Surgical	TP
104	29	Communication, Surgical	Communication, Surgical	TP
105	30	System	System	TP
106	31	Communication, System	Communication, System	TP
107	32	Communication	Communication	TP
108	33	Communication, System	Communication, System	TP
109	35	Surgical	Surgical	TP
110	36	Security, System	Security, System	TP
111	37	Communication	Communication	TP
112	38	Communication	Communication	TP
113	39	System	System	TP
114	40	System	System	TP
115	41	Surgical, System	Surgical, System	TP
116	43	Communication	Communication	TP
117	44	Communication	Communication	TP
118	46	Communication	Communication	TP
119	47	Surgical	Surgical	TP
120	48	System	System	TP

121	49	Surgical, System	Surgical, System	TP
122	51	System	System	TP
123	53	Communication, System	Communication, System	TP
124	54	Communication, System	Communication, System	TP
125	55	System	System	TP
126	56	System	System	TP
127	57	Communication	Communication	TP
128	58	Communication, System	Communication, System	TP
129	59	Communication	Communication	TP
130	60	System	System	TP
131	61	Surgical, System	System, Surgical	TP
132	62	System	System	TP
133	63	Surgical	Surgical	TP
134	66	System	System	TP
135	67	Communication	Communication	TP
136	68	Communication, System	Communication, System	TP
137	69	System	System	TP
138	70	Communication	Communication	TP
139	71	Communication	Communication	TP
140	72	Communication, Surgical, System	Communication, Surgical, System	TP
141	73	System	System	TP
142	75	Security	Security	TP
143	76	System	System	TP
144	77	System	System	TP
145	78	Communication, System	Communication, System	TP
146	80	System	System	TP
147	82	Surgical	Surgical	TP

148	83	Communication, Security, System	Communication, Security, System	TP
149	84	Communication, Surgical	Communication, Surgical	TP
150	85	Surgical, System	Surgical, System	TP
151	86	Surgical, System	Surgical, System	TP
152	87	Surgical	Surgical	TP
153	88	Communication, System	Communication, System	TP
154	89	Communication, Surgical	Communication, Surgical	TP
155	90	Communication, Surgical, System	Communication, Surgical, System	TP
156	91	System	System	TP
157	92	Surgical, System	Surgical, System	TP
158	93	Surgical	Surgical	TP
159	94	Communication, System	Communication, System	TP
160	95	Surgical	Surgical	TP
161	96	System	System	TP
162	98	Surgical	Surgical	TP
163	99	Surgical, System	Surgical, System	TP
164	100	System	System	TP
165	104	Communication, System	Communication, System	TP
166	105	Surgical	Surgical	TP
167	106	Surgical	Surgical	TP
168	107	Surgical	Surgical	TP
169	108	System	System	TP
170	109	System	System	TP
171	110	System	System	TP
172	111	Communication	Communication	TP
173	112	Surgical	Surgical	TP
174	113	System	System	TP
175	114	System	System	TP
176	115	System	System	TP

177	116	Surgical	Surgical	TP
178	117	Surgical, System	Surgical, System	TP
179	118	System	System	TP
180	119	System	System	TP
181	120	Surgical	Surgical	TP
182	121	System	System	TP
183	122	Surgical	Surgical	TP
184	123	Communication, Surgical	Communication, Surgical	TP
185	124	Surgical	Surgical	TP
186	125	Surgical	Surgical	TP
187	127	Surgical	Surgical	TP
188	128	System	System	TP
189	129	System	System	TP
190	130	System	System	TP
191	131	Surgical, System	Surgical, System	TP
192	132	Communication, Surgical, System	Communication, Surgical, System	TP
193	133	Communication, System	Communication, System	TP
194	134	Communication, Surgical, System	Communication, Surgical, System	TP
195	135	Communication, System	Communication, System	TP
196	136	Communication, System	Communication, System	TP
197	137	Surgical	Surgical	TP
198	138	Surgical, System	Surgical, System	TP
199	139	Communication, System	Communication, System	TP
200	140	Surgical	Surgical	TP
201	141	Surgical	Surgical	TP
202	142	Surgical	Surgical	TP
203	143	Surgical	Surgical	TP
204	144	Surgical	Surgical	TP
205	145	Surgical	Surgical	TP
206	146	Surgical	Surgical	TP
207	147	Surgical	Surgical	TP

208	148	Surgical	Surgical	TP
209	149	Surgical	Surgical	TP
210	150	Surgical	Surgical	TP
211	151	Surgical	Surgical	TP
212	152	Surgical	Surgical	TP
213	153	Surgical	Surgical	TP
214	154	Surgical	Surgical	TP
215	155	Communication	Communication	TP
216	156	System	System	TP
217	157	Security	Security	TP
218	158	System	System	TP
219	159	System	System	TP
220	160	Surgical	Surgical	TP
221	161	Surgical	Surgical	TP
222	162	Surgical	Surgical	TP
223	163	Surgical	Surgical	TP
224	164	Surgical	Surgical	TP
225	165	Surgical	Surgical	TP
226	166	Surgical	Surgical	TP
227	167	Surgical	Surgical	TP
228	168	Surgical	Surgical	TP
229	169	Surgical	Surgical	TP
230	170	System	System	TP
231	172	System	System	TP
232	174	Surgical, System	Surgical, system	TP
233	175	Surgical	Surgical	TP
234	176	Surgical	Surgical	TP
235	178	Surgical	Surgical	TP
236	179	Communication, Surgical, System	Communication, Surgical, System	TP
237	180	Surgical	Surgical	TP
238	181	Surgical	Surgical	TP
239	182	Surgical	Surgical	TP
240	183	Surgical	Surgical	TP
241	184	Surgical	Surgical	TP
242	186	Surgical	Surgical	TP
243	193	System	System	TP
244	194	System	System	TP
245	197	Communication, System	Communication, System	TP

246	198	Communication, System	Communication, System	TP
247	199	Surgical	Surgical	TP
248	200	Surgical	Surgical	TP
249	201	Surgical	Surgical	TP
250	202	Surgical	Surgical	TP
251	203	Communication	Communication	TP
252	204	System	System	TP
253	205	System	System	TP
254	207	Security, System	Security, System	TP
255	208	Surgical	Surgical	TP
256	209	Security	Security	TP
257	210	Security	Security	TP
258	211	Security, System	Security, System	TP
259	212	Surgical, System	Surgical, System	TP
260	213	System	System	TP
261	214	Security	Security	TP
262	215	Surgical	Surgical	TP
263	216	Security	Security	TP
264	217	Surgical	Surgical	TP
265	218	Security	Security	TP
266	219	Surgical	Surgical	TP
267	220	Surgical	Surgical	TP
268	221	Security	Security	TP
269	222	Security	Security	TP
270	223	Security	Security	TP
271	224	Surgical, System	Surgical, System	TP
272	225	System	System	TP
273	226	System	System	TP
274	227	Security	Security	TP
275	228	Communication, System	Communication, System	TP
276	229	Communication, Surgical, System	Communication, Surgical, System	TP
277	230	Communication	Communication	TP

278	231	Communication	Communication	TP
279	232	Communication, System	Communication, System	TP
280	233	Communication	Communication	TP
281	234	Communication	Communication	TP
282	235	System	System	TP
283	236	Security	Security	TP
284	237	Security, System	Security, System	TP
285	239	System	System	TP
286	240	System	System	TP
287	241	System	System	TP
288	242	Communication	Communication	TP
289	243	Security	Security	TP
290	244	Security	Security	TP
291	246	Surgical	Surgical	TP
292	247	System	System	TP
293	248	Communication, System	Communication, System	TP
294	249	Communication, Surgical	Communication, Surgical	TP
295	250	Security	Security	TP
296	251	Security	Security	TP
297	252	System	System	TP
298	253	Security	Security	TP
299	254	Surgical	Surgical	TP
300	255	Communication	Communication	TP
301	256	Communication	Communication	TP
302	258	Surgical	Surgical	TP
303	259	Surgical	Surgical	TP
304	260	Surgical	Surgical	TP
305	261	Surgical	Surgical	TP
306	262	System	System	TP
307	263	Surgical	Surgical	TP
308	264	Security	Security	TP
309	265	System	System	TP
310	266	Surgical	Surgical	TP

311	267	Security	Security	TP
312	269	System	System	TP
313	270	Security	Security	TP
314	271	Surgical, System	Surgical, System	TP
315	272	Surgical	Surgical	TP
316	273	System	System	TP
317	274	System	System	TP
318	275	System	System	TP
319	276	System	System	TP
320	277	Communication	Communication	TP
321	278	Communication	Communication	TP
322	279	System	System	TP
323	280	Security, System	Security, System	TP
324	281	Security, System	Security, System	TP
325	282	Surgical	Surgical	TP
326	283	Security	Security	TP
327	284	Communication	Communication	TP
328	286	Security	Security	TP
329	287	Surgical	Surgical	TP
330	288	Surgical	Surgical	TP
331	289	Surgical	Surgical	TP
332	290	Security	Security	TP
333	291	Surgical	Surgical	TP
334	293	Communication, Surgical	Communication, Surgical	TP
335	295	Surgical	Surgical	TP
336	296	Surgical	Surgical	TP
337	297	Surgical	Surgical	TP
338	298	Surgical	Surgical	TP
339	299	Communication	Communication	TP
340	300	Communication	Communication	TP
341	301	Communication, Surgical	Communication, Surgical	TP
342	302	Communication, Surgical	Communication, Surgical	TP

343	303	Security	Security	TP
344	304	Security	Security	TP
345	305	System	System	TP
346	306	System	System	TP
347	307	Communication, System	Communication, System	TP
348	308	Surgical	Surgical	TP
349	311	Security	Security	TP
350	312	Surgical	Surgical	TP
351	313	Surgical	Surgical	TP
352	315	System	System	TP
353	316	Communication, Surgical, System	Communication, Surgical, System	TP
354	317	Security	Security	TP
355	318	Surgical, System	Surgical, System	TP
356	319	Surgical, System	Surgical, System	TP
357	320	Surgical	Surgical	TP
358	321	System	System	TP
359	322	Surgical	Surgical	TP
360	324	System	System	TP
361	325	System	System	TP
362	327	Communication	Communication	TP
363	329	Surgical, System	Surgical, System	TP
364	330	Surgical	Surgical	TP
365	331	Surgical, System	Surgical, System	TP
366	332	Surgical	Surgical	TP
367	333	System	System	TP
368	334	Security, System	Security, System	TP
369	336	Surgical	Surgical	TP
370	337	System	System	TP
371	338	System	System	TP
372	339	System	System	TP
373	340	Surgical	Surgical	TP
374	341	Surgical	Surgical	TP
375	342	Communication	Communication	TP

376	344	Security	Security	TP
377	345	Security	Security	TP
378	346	Communication	Communication	TP
379	348	Communication, System	Communication, System	TP
380	349	Security	Security	TP
381	356	Communication, System	Communication, System	TP
382	357	Security	Security	TP
383	358	Communication	Communication	TP
384	359	Communication	Communication	TP
385	360	Communication, Security	Communication, Security	TP
386	361	Communication, Security	Communication, Security	TP
387	362	Communication, Security	Communication, Security	TP
388	364	Communication, Security, System	Communication, Security, System	TP
389	365	Security	Security	TP
390	367	System	System	TP
391	370	System	System	TP
392	372	Surgical	Surgical	TP
393	374	System	System	TP
394	376	Security	Security	TP
395	382	Communication	Communication	TP
396	384	Communication, Security	Communication, Security	TP
397	385	Security	Security	TP
398	386	Communication, Security	Communication, Security	TP
399	387	Communication	Communication	TP
400	388	Security	Security	TP
401	389	Security	Security	TP
402	390	Communication, Security	Communication, Security	TP

403	391	Communication, Security	Communication, Security	TP
404	392	Communication, Security	Communication, Security	TP
405	393	System	System	TP
406	394	Communication	Communication	TP
407	395	Communication, System	Communication, System	TP
408	396	Communication	Communication	TP
409	397	System	System	TP
410	398	Surgical	Surgical	TP
411	399	Communication, Security	Communication, Security	TP
412	400	Communication	Communication	TP
413	403	Communication	Communication	TP
414	404	Communication, System	Communication, System	TP
415	405	Communication	Communication	TP
416	406	Communication	Communication	TP
417	407	Communication	Communication	TP
418	408	Communication, Security	Communication, Security	TP
419	409	Communication	Communication	TP
420	410	Communication	Communication	TP
421	411	Communication	Communication	TP
422	412	Surgical	Surgical	TP
423	413	Communication, System	Communication, System	TP
424	415	Surgical	Surgical	TP
425	417	System	System	TP
426	418	Communication	Communication	TP
427	419	Surgical	Surgical	TP

428	420	Surgical, System	Surgical, System	TP
429	421	System	System	TP
430	422	Surgical	Surgical	TP
431	423	System	System	TP
432	424	Surgical	Surgical	TP
433	425	Communication	Communication	TP
434	426	Surgical	Surgical	TP
435	427	Surgical	Surgical	TP
436	428	Surgical	Surgical	TP
437	429	System	System	TP
438	430	Surgical	Surgical	TP
439	431	Communication	Communication	TP
440	432	Communication	Communication	TP
441	433	System	System	TP
442	434	Surgical	Surgical	TP
443	435	Surgical	Surgical	TP
444	436	Surgical	Surgical	TP
445	437	Surgical	Surgical	TP
446	438	Surgical	Surgical	TP
447	439	Surgical	Surgical	TP
448	441	Surgical	Surgical	TP
449	442	Surgical	Surgical	TP
450	443	Surgical, System	Surgical, System	TP
451	444	System	System	TP
452	446	Surgical	Surgical	TP
453	450	System	System	TP
454	452	Security, Surgical, System	Security, Surgical, System	TP
455	453	Surgical	Surgical	TP
456	455	Communication, System	Communication, System	TP
457	456	Surgical, System	Surgical, System	TP
458	457	Surgical	Surgical	TP
459	458	Surgical	Surgical	TP
460	459	Communication, Surgical	Communication, Surgical	TP

461	460	Communication, Surgical	Communication, Surgical	TP
462	461	Communication, Surgical	Communication, Surgical	TP
463	462	Communication, Security, System	Communication, Security, System	TP
464	463	Surgical	Surgical	TP
465	464	System	System	TP
466	465	Communication	Communication	TP
467	466	Surgical	Surgical	TP
468	467	Surgical	Surgical	TP
469	468	Security	Security	TP
470	469	Communication, Surgical	Communication, Surgical	TP
471	470	Communication	Communication	TP
472	471	Surgical	Surgical	TP
473	472	Surgical	Surgical	TP
474	473	Surgical	Surgical	TP
475	474	System	System	TP
476	475	Surgical, System	System, Surgical	TP
477	476	System	System	TP
478	477	System	System	TP
479	478	Communication, Surgical	Communication, Surgical	TP
480	479	Communication, System	Communication, System	TP
481	480	Communication	Communication	TP
482	481	Security	Security	TP
483	482	Surgical	Surgical	TP
484	483	Communication	Communication	TP
485	485	Surgical	Surgical	TP
486	486	Security	Security	TP
487	487	Security, System	Security, System	TP
488	488	Communication, Security	Communication, Security	TP

489	489	Communication, Security	Communication, Security	TP
490	491	System	System	TP
491	492	System	System	TP
492	493	Surgical, System	Surgical, System	TP
493	494	Surgical	Surgical	TP
494	495	Communication	Communication	TP
495	496	System	System	TP
496	497	Communication	Communication	TP
497	498	Security	Security	TP
498	499	Communication, Surgical	Communication, Surgical	TP
499	500	System	System	TP

Table A-5 Complete Result of the Categorization the Entire Datasets