Using Natural Language Processing and Neural Network Models to Classify, Cluster, and Organize Software Requirements

1st Christian Crosser
Computer Science Dept.
Florida Polytechnic University
Lakeland, United States
ccrosser6344@floridapoly.edu

***Abstract*—In order to improve and optimize the requirements gathering phase Machine Learning can be used. By collecting the user input into a meaningful and useful dataset, the power of machine learning, natural language processing, and AI can automate and improve the process of determining the direction for a software development project and the most desired features to focus on.**

***Index Terms*—Software Requirements, Machine Learning, Natural Language Processing, Text Classification, Data-Augmentation**

I. Introduction

As modern software development methods have become standard practice, the adoption of Agile methodologies and fast past development has grown drastically. However, the drawbacks of this approach have become clear, more emphasis is placed upon speedy development rather than methodical project planning and execution. As these drawbacks have become more apparent and as Agile methodologies mature, the needs for a hybrid software development approach are clear to individuals in the industry. The value of proper planning and design has gone to the wayside due to the cost and time it can often take to properly develop and plan software. This time and cost can be greatly decreased and mitigated with the incorporation of Machine Learning and Artificial Intelligence applications, starting at the beginning of the software development process.

Proposed is a methodology for using Natural Language Processing and Machine Learning to reduce the time and cost of software requirements definition, seeking to create an automated process starting at the very beginning of software development. In the beginning stages of a software project user stories and requirements are collected from all stakeholders involved in the project. This can vary from technical to non-technical users and can often grow drastically. These requirements typically are collected in a haphazard fashion with no organization or classification. From here the tedious task of organizing, classifying, and compiling user stories and requirements often falls upon technical team members. This can become a costly process and often leaves requirements undefined or missed due to the large sets of requirements collected.

A solution to this tedious process is the automatic classification and sorting of user stories and requirements using machine learning tools and natural language processing to train classification models to recognize and correctly classify a given requirement. This can serve as the stepping-stone for future software design automation and is the most important initial step. Outlined in this proposal is the methodology to train classification models to label software requirements, cluster related requirements, and prioritize requirements based on classification.

II. Related Work

A. Classifying Functional and Non-Functional Software Requirements

Prior to classifying software requirements by more specific catagories past work has been done to label requirements as functional and non-functional. These works utilize simple word patterns to define if a requirement is a functional or non-functional requirement. More specific and specialized classification methods are needed to build upon in future and more advanced work. The binary nature of these software requirement definition methods do not allow for clustering and advanced model building.

B. NLP approaches to RE practice

Past work has been done to improve requirements engineering processes using NLP. This work was broken down into six major categories, quality checks, extraction, classification, modeling, traceability, and search and retrieval. Quality checks are done by simply incorporating existing methods of NLP analysis and determining the relative quality of a given software requirement or specification. Extraction involves utilizing natural language processing to extract domain or project specific information based on predetermined patterns. Classification is utilized to label software requirements, giving the ability for clustering. Requirements modeling can help to build models and concepts from the extracted requirements and specifications. Traceability refers to linking or relationships

between software requirements that are determined with NLP methodologies.

C. Past Project Work

The results of this work have shown promising results in using different models to classify and label software requirements, with significant improvements made when the dataset was increased in size and augmented. Highlighted in the following section are the results of using a single Support Vector Classifier with the non-augmented dataset and then comparing with the augmented dataset results.

D. Support Vector Classifier without Augmented Data

	precision	recall	f1-score
Availability	0.70	0.88	0.78
Fault Tolerance	1.00	0.78	0.88
Functional	0.83	1.00	0.91
Legal	1.00	1.00	1.00
Look & Feel	0.91	1.00	0.95
Maintainability	0.91	0.91	0.91
Operational	1.00	0.80	0.89
Performance	0.89	1.00	0.94
Portability	1.00	1.00	1.00
Scalability	1.00	0.88	0.93
Security	1.00	1.00	1.00
Usability	1.00	0.91	0.95

Fig. 1. Results of SVC without Augmented Data

The results of running a support vector classifier model with non-augmented dataset showed promising results for the properly represented classes but showed poor performance when attempting to classify the minority classes. This can be greatly improved using augmented datasets.

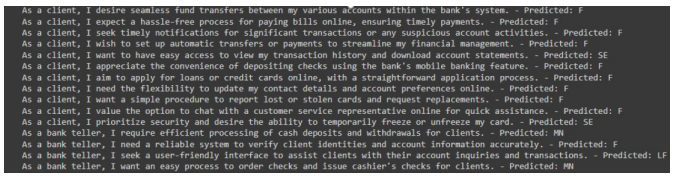

Fig. 2. SVC Model Tested on Unclassified Requirements

E. Support Vector Classifier with Augmented Data

As shown in the figure above, after data augmentation and boosting the minority classes the model was able to classify all labels and achieve note worthy F-1 scores for each. This shows that by using even basic data-augmentation methods it is possible to retain all non-functional requirements labels. This is essential for future work and development in leveraging text classification specific to software engineering.

F. Initial PROMISE Dataset

The initial dataset is an open-source repository of 925 labeled natural language software requirements, broken down into 444 functional requirements and 481 non-functional requirements. The non-functional requirements are also broken down into 11 sub-classifications split as follows:

	precision	recall	f1-score
A	0.50	0.50	0.50
F	0.67	0.94	0.78
FT	0.61	0.70	0.65
L	0.87	0.68	0.76
LF	0.88	0.82	0.85
MN	0.63	0.50	0.56
O	0.50	0.86	0.63
PE	0.78	0.64	0.70
PO	0.83	0.91	0.87
SC	1.00	0.33	0.50
SE	0.71	0.38	0.50
US	0.73	0.57	0.64

Fig. 3. Results of SVC with Augmented Data

- Availability (A) 31
- Legal (L) 15
- Look-and-Feel (LF) 49
- Maintainability (MN) 24
- Operability (O) 77
- Performance (PE) 67
- Scalability (SC) 22
- Security (SE) 125
- Usability (US) 85
- Fault Tolerance (FT) 18
- Portability (PO) 12

In the first iteration of this project the PROMISE dataset was used as the primary dataset and data augmentation used to help classify the minority classes. While this served the initial project, this dataset will be expanded using additional similar datasets and more advanced methods of data augmentation. All of these improvements will be made using experience gained from the first project iteration.

III. Proposed Approaches

There are many approaches to solve the problem of software requirements classification, clustering, and organizing. Outlined below are several proposed approaches and a brief description of the methodology this approach would utilize.

A. Classifying Requirements Using Classification Algorithms

In this approach classification algorithms are used to train on a test set of software requirements and unclassified requirements are label using the trained models. While this methodology yields good results on the majority classes, the weaknesses of simpler classification algorithms becomes further highlighted at this point and the need for more advanced machine learning models apparent. The following are examples of the classification algorithms used in past project iterations:

- Support Vector Classifier

- Multinomial Naive Bayes
- Random Forest Classifier
- Multinomial Logistic Regression

B. Classifying Requirements Using Neural Networks

Improving upon the usage of classification algorithms, given a dataset of sufficient size neural network methods can be implemented in order to improve the capabilities of the classification models. This approach would require an increased data size from past project iterations and enhanced data specifically for the minority classes. Given the capabilities of neural networks this would enable the next steps of artificial intelligence aided software development. By utilizing several different models and training them for more specialized tasks more dynamic results can be achieved. The following are neural network types considered for requirments engineering:

- Convolutional neural network (CNN)
- Recurrent neural network (RNN)
- Multilayer perceptron (MLP)
- Long short-term memory (LSTM)

IV. Planned Experiment

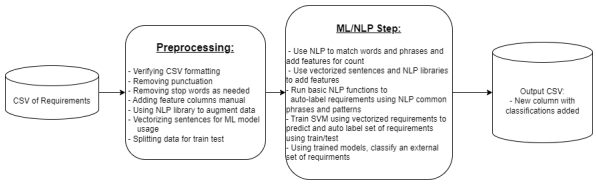

Fig. 4. Methodology Flow Chart

A. Pre-processing

In order to begin training machine learning models to correctly classify non-functional requirements there are necessary steps of data pre-processing that must be completed. These initial steps are standard to natural language processing implementations and allow for proper use of text requirements in machine learning applications. These pre-processing steps can be broken down as follows:

- Data Augmentation/Synonym replacement step
- Normalization step
- Natural language manual feature selection
- TFIDF text vectorization Step
- TFIDF feature extraction

Before beginning the remainder of the preprocessing steps, minority classes are determined and synonym replacement used to add requirements and labels to help boost these minority classes. This step enabled drastic improvements in the ML model training and in classifying the minority classes properly. After these extra requirements are added, the dataset is ready for normalization and vectorization. Once the text sentences have been normalized for vectorization and manual features added using features determined to be important, the natural language is processed into numeric vector format using the TFIDF text vectorization library from SkLearn. With the initial pre-processing steps completed the dataset is ready for model training and optimization.

B. Improvements for Current Experiment

Building on the foundation of the initial project iteration improvements will be made and following steps will be taken in the planned experiment. First, more and better representative data is needed. Due to the confidential nature of software requirements, it can be difficult to find usable datasets of requirements, and even more difficult to find labeled or manually label them. While the data augmentation process greatly improved the results further work is needed in this area. Secondly additional and more robust natural language processing is needed. Many of these tools are in relative infancy when applied to requirements engineering and could serve to greatly improve the results of preprocessing and the model training process. Additionally there will need to be the incorporation of domain knowledge by working with university professors and experts in the field.

Next incorporating expert judgment could serve to improve the application of machine learning and natural language processing by further defining the requirements engineering process, refining determined labels for Non-Functional requirements, and laying out the next steps for improving software design automation. Next, with a framework of natural language processing of software requirements developed future work can be done to improve the process, such as cost estimation, code structure prototyping, and automatic document creation (SRS, Design Documents, etc). Finally the usage of Neural Networks and Deep Learning techniques can be utilized, as these were not an option with the unbalanced small dataset initially used.