Using Natural Language Processing and Data Augmentation to Classify Software Requirements

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

2nd Divyesh Desai
Computer Science Dept.
Florida Polytechnic University
Lakeland, United States
ddesai7656@floridapoly.edu

3rd Halle Fortune
Computer Science Dept.
Florida Polytechnic University
Lakeland, United States
hfortune6785@floridapoly.edu

4th Philip Wilson
Computer Science Dept.
Florida Polytechnic University
Lakeland, United States
pwilson8655@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 [6]. 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.

Laid out in the following research paper is the proposed methodology for using Natural Language Processing and Machine Learning [2] 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. They are often collected in a haphazard function 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 paper is the methodology used to train classification models to label software requirements, the results of the trained models, and the suggestions for future work and development to follow this work.

II. Methodology and Dataset

A. Initial PROMISE Dataset

The initial dataset is an open-source repository [5] 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:

- 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

While it will be discussed in the later sections, it is important to highlight that with this initial dataset there is a large over-representation of functional requirements and as such the minority classes are much harder to train the models on. This has led other similar projects to drop most of the labels and only train models on the most commonly occurring labels. This work attempts to augment this dataset and improve the ability of the models to classify the minority classes and improve the accuracy of the model. With consideration for future work and automation it is important that all labels are retained, and the models properly trained.

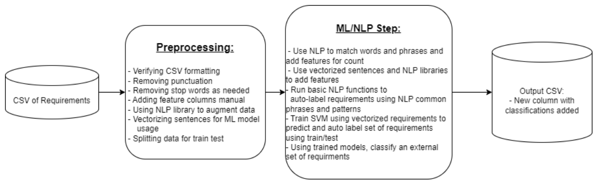

CSV of Requirements
Preprocessing:
- Verifying CSV formatting
- Removing punctuation
- Removing stop words as needed
- Adding feature columns manual
- Using NLP library to augment data
- Vectorizing sentences for ML model usage
- Splitting data for train test
ML/NLP Step:
- Use NLP to match words and phrases and add features for count
- Use vectorized sentences and NLP libraries to add features
- Run basic NLP functions to auto-label requirements using NLP common phrases and patterns
- Train SVM using vectorized requirements to predict and auto label set of requirements using train/test
- Using trained models, classify an external set of requirments
Output CSV:
- New column with classifications added

Fig. 1. Methodology Flow Chart

B. 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 [3]. 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, first the 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.

C. Model Selection and Training

After completing the pre-processing steps the model selection and training can begin. For the sake of this implementation four models were chosen:

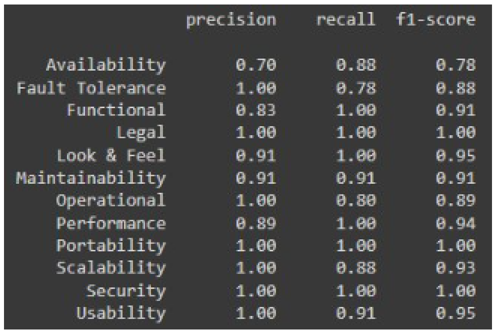
- Support Vector Classifier
- Multinomial Naive Bayes
- Random Forest Classifier
- Multinomial Logistic Regression

To begin Support Vector Classifier was used as the initial model for training and testing. This involved splitting the dataset into 80 percent for the training dataset and 20 percent for the testing set. Each of the models is initially trained using the non-augmented dataset and the initial 925 requirements to train, test, and evaluate the model's performance using accuracy, recall, and F-1 score [1]. After determining the proper methodology using Support Vector Classifier as the initial model, the process is then applied to the remaining models and the results compared. With the results of the non-augmented models collected and compared, the same process is applied to all four models using the augmented dataset with an increased dataset size from 925 to 1330. The comparison of these results is presented in the following section for analysis.

III. Results and Model Comparison

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.

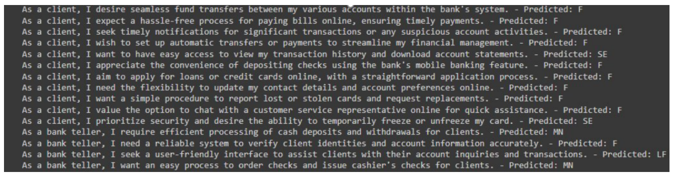
A. Support Vector Classifier without Augmented Data



	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. 2. 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.



```
As a client, I desire seamless fund transfers between my various accounts within the bank's system. - Predicted: F
As a client, I expect a hassle-free process for paying bills online, ensuring timely payments. - Predicted: F
As a client, I seek timely notifications for significant transactions or any suspicious account activities. - Predicted: F
As a client, I wish to set up automatic transfers or payments to streamline my financial management. - Predicted: F
As a client, I want to have easy access to view my transaction history and download account statements. - Predicted: SE
As a client, I appreciate the convenience of depositing checks using the bank's mobile banking feature. - Predicted: F
As a client, I aim to apply for loans or credit cards online, with a straightforward application process. - Predicted: F
As a client, I need the flexibility to update my contact details and account preferences online. - Predicted: F
As a client, I want a simple procedure to report lost or stolen cards and request replacements. - Predicted: F
As a client, I value the option to chat with a customer service representative online for quick assistance. - Predicted: F
As a client, I prioritize security and desire the ability to temporarily freeze or unfreeze my card. - Predicted: SE
As a bank teller, I require efficient processing of cash deposits and withdrawals for clients. - Predicted: MN
As a bank teller, I need a reliable system to verify client identities and account information accurately. - Predicted: F
As a bank teller, I seek a user-friendly interface to assist clients with their account inquiries and transactions. - Predicted: LF
As a bank teller, I want an easy process to order checks and issue cashier's checks for clients. - Predicted: MN
```

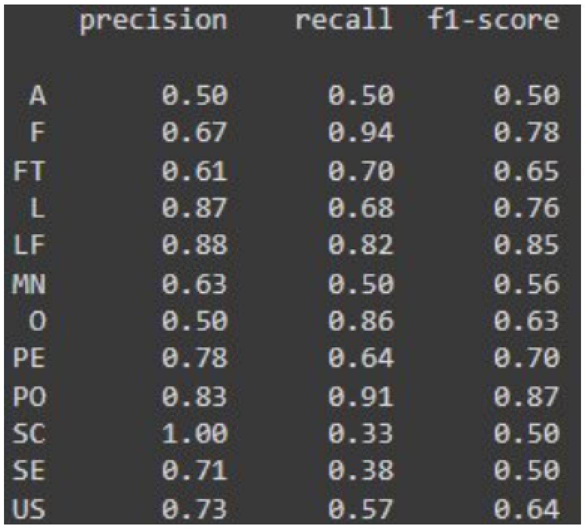
Fig. 3. SVC Model Tested on Unclassified Requirements

B. 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.

C. Four Model Comparison without Augmented Data

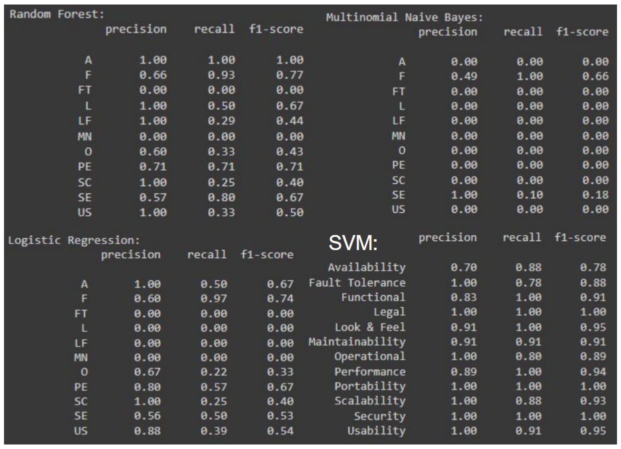
When running the non-augmented data set on four chosen models it was shown that Support Vector Classifier out performs the other models drastically. Multinomial Naive Bayes, however, was not given enough data to begin the classification



```
     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. 4. Results of SVC with Augmented Data



```
Random Forest:
              precision    recall  f1-score

           A       1.00      1.00      1.00
           F       0.66      0.93      0.77
          FT       0.00      0.00      0.00
           L       1.00      0.50      0.67
          LF       1.00      0.29      0.44
          MN       0.00      0.00      0.00
           O       0.60      0.33      0.43
          PE       0.71      0.71      0.71
          SC       1.00      0.25      0.40
          SE       0.57      0.80      0.67
          US       1.00      0.33      0.50

Multinomial Naive Bayes:
              precision    recall  f1-score

           A       0.00      0.00      0.00
           F       0.49      1.00      0.66
          FT       0.00      0.00      0.00
           L       0.00      0.00      0.00
          LF       0.00      0.00      0.00
          MN       0.00      0.00      0.00
           O       0.00      0.00      0.00
          PE       0.00      0.00      0.00
          SC       0.00      0.00      0.00
          SE       1.00      0.10      0.18
          US       0.00      0.00      0.00

Logistic Regression:
              precision    recall  f1-score

           A       1.00      0.50      0.67
           F       0.60      0.97      0.74
          FT       0.00      0.00      0.00
           L       0.00      0.00      0.00
          LF       0.00      0.00      0.00
          MN       0.00      0.00      0.00
           O       0.67      0.22      0.33
          PE       0.80      0.57      0.67
          SC       1.00      0.25      0.40
          SE       0.56      0.50      0.53
          US       0.88      0.39      0.54

SVM:          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. 5. Results of Four Model Comparison without Augmented Data

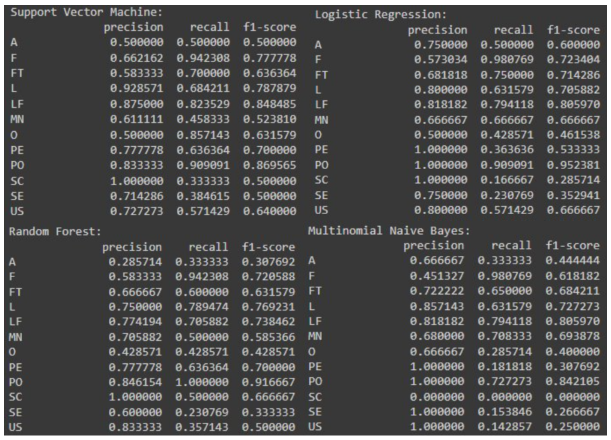
process and performed poorly. Random Forest Classification was able to begin the labeling steps but did not perform as well as Support Vector Classifier. Finally Multinomal Logistic Regression was able to perform some classification but was inferior to both Random Forest and SVC.

D. Four Model Comparison with Augmented Data

Running all four models with augmented data yielded surprising results when compared to the non-augmented data results. With the minority classes boosted, all models could perform classification on almost every label and achieve notable F-1 scores. Support Vector was no longer the top performing model, with Random Forest achieving comparable results. This shows that if further data augmentation and dataset improvements could prove drastic improvements to the classification models.

IV. Future Work and Improvements

In order to improve on the results and pave the way for further machine learning applications in requirements engineering



```
Support Vector Machine:
         precision    recall  f1-score
A         0.500000  0.500000  0.500000
F         0.662162  0.942308  0.777778
FT        0.583333  0.700000  0.636364
L         0.928571  0.684211  0.787879
LF        0.875000  0.823529  0.848485
MN        0.611111  0.458333  0.523810
O         0.500000  0.857143  0.631579
PE        0.777778  0.636364  0.700000
PO        0.833333  0.909091  0.869565
SC        1.000000  0.333333  0.500000
SE        0.714286  0.384615  0.500000
US        0.727273  0.571429  0.640000

Logistic Regression:
         precision    recall  f1-score
A         0.750000  0.500000  0.600000
F         0.573034  0.980769  0.723404
FT        0.681818  0.750000  0.714286
L         0.800000  0.631579  0.705882
LF        0.818182  0.794118  0.805970
MN        0.666667  0.666667  0.666667
O         0.500000  0.428571  0.461538
PE        1.000000  0.363636  0.533333
PO        1.000000  0.909091  0.952381
SC        1.000000  0.166667  0.285714
SE        0.750000  0.230769  0.352941
US        0.800000  0.571429  0.666667

Random Forest:
         precision    recall  f1-score
A         0.285714  0.333333  0.307692
F         0.583333  0.942308  0.720588
FT        0.666667  0.600000  0.631579
L         0.750000  0.789474  0.769231
LF        0.774194  0.705882  0.738462
MN        0.705882  0.500000  0.585366
O         0.428571  0.428571  0.428571
PE        0.777778  0.636364  0.700000
PO        0.846154  1.000000  0.916667
SC        1.000000  0.500000  0.666667
SE        0.600000  0.230769  0.333333
US        0.833333  0.357143  0.500000

Multinomial Naive Bayes:
         precision    recall  f1-score
A         0.666667  0.333333  0.444444
F         0.451327  0.980769  0.618182
FT        0.722222  0.650000  0.684211
L         0.857143  0.631579  0.727273
LF        0.818182  0.794118  0.805970
MN        0.680000  0.708333  0.693878
O         0.666667  0.285714  0.400000
PE        1.000000  0.181818  0.307692
PO        1.000000  0.727273  0.842105
SC        0.000000  0.000000  0.000000
SE        1.000000  0.153846  0.266667
US        1.000000  0.142857  0.250000
```

Fig. 6. Results of Four Model Comparison with Augmented Data

there are several future works that could be implemented. 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.

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, by increasing the dataset size and quality 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.

Conclusion

While applying the models to a small dataset with under represented classes yielded only acceptable results when training a support vector classifier, applying even basic data augmentation has shown to greatly improve model performance and coverage. By boosting the minority classes and increasing the dataset size overall all of the non-functional labels were able to be retained and utilized in model training. This enables future work and improvements through a proper classification and labeling system, while also enhancing the ability of a model to learn domain specific terminology and apply the models to all areas of software development and design.

References

[1] Jane Cleland-Huang, Sepideh Mazrouee, Huang Liguo, Dan Port, ”NFR, OpenScience tera-PROMISE software requirements”, 2007. URL (last accessed: July 01, 2019)

[2] D. Khurana, A. Koli, K. Khatter, and S. Singh, “Natural language processing: state of the art, current trends and challenges,” Multimed Tools Appl, vol. 82, no. 3, pp. 3713–3744, Jan. 2023, doi: 10.1007/s11042-022-13428-4.

[3] R. Varaprasad and G. Mahalaxmi, “Applications and Techniques of Natural Language Processing: An Overview: IUP Journal of Computer Sciences,” IUP Journal of Computer Sciences, vol. 16, no. 3, pp. 7–21, Jul. 2022.

[4] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: an introduction,” J Am Med Inform Assoc, vol. 18, no. 5, pp. 544–551, Sep. 2011, doi: 10.1136/amiajnl-2011-000464.

[5] The Promise Repository of Empirical Software Engineering Data. 2015

[6] Tell, P., Klünder, J., Küpper, S., Raffo, D., MacDonell, S.G., Münch, J., Pfahl, D., Linssen, O., Kuhrmann, M. (2019) What are hybrid development methods made of? An evidence-based characterization, in Proceedings of the International Conference on Software and Systems Process (ICSSP2019). Montréal, Canada, IEEE Computer Society Press, pp.105-114. doi:10.1109/ICSSP.2019.00022