MARE: an Active Learning Approach for Requirements Classification

Cláudia Magalhães
NOVA LINCS
Universidade NOVA de Lisboa
Lisbon, Portugal
cm.magalhaes@campus.fct.unl.pt

João Araujo
NOVA LINCS
Universidade NOVA de Lisboa
Lisbon, Portugal
joao.araujo@fct.unl.pt

Alberto Sardinha
INESC-ID and Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
jose.alberto.sardinha@tecnico.ulisboa.pt

***Abstract*—Several studies indicate that poor requirements practices, that result in incomplete or inaccurate requirements, poorly managed requirement changes, and missed requirements, are the most common factors in project failure. Possible solutions for better requirements definition include better requirements documentation, and requirements reuse. In this paper, we present a novel application of machine learning and active learning to classify the requirements of a given dataset. This approach can accelerate project development. By organizing the requirements into categories, developers can easily see what requirements were already implemented, and where they need to focus on the next step of development.**

***Index Terms*—Requirements, Agile Development, Machine Learning, Natural Language Processing, Active Learning**

I. Introduction

Poor requirement practices are the main issue related to Project Failure. Incomplete or inaccurate requirements, poorly managed requirement changes, and missing requirements are common problems found in projects developed with agile methodologies [1]. Solutions include increasing the quality of existing requirements documentation, reusing requirements from previous projects, and "creating" requirements where there are none [2].

Another issue is the abundance of unorganized, unclassified requirement data. It would be ideal for the evolution of a company if it could restructure and polish existing requirements. Through Machine Learning (ML) and Natural Language Processing (NLP), we can process, categorize and organize the existing requirements so that they can be reused.

ML practices are all about understanding data patterns on a larger scale. With millions of projects every year, ML can make a real contribution in recognizing different types of requirements and connections between them. NLP is an invaluable field of linguistics and computer science, that has driven forward communication between human and machine. The contributions of this field are what makes it possible for computers to process and analyze large amounts of natural language data. With the help of NLP techniques, we can correctly extract and transform textual requirements into formats that ML algorithms can utilize to the fullest. This will encourage companies to adopt proper requirements principles and techniques and thus improving software evolution. However, manually classifying the existing data is a very costly, time-consuming process [3]. Active Learning (AL) is a way to significantly reduce this cost and do it in a more efficient and less laborious way. AL is founded on the notion that actively requesting information to label the training set can be more time and cost efficient than labelling the entire set by hand. Our goal is to take a list of unorganized, unclassified requirements and turn it into a usable list of requirements that can be easily understood and enhanced.

II. Background

Here we introduce the main topics of this work. Machine Learning is a branch of Artificial Intelligence (AI) that focuses on pattern identification and data analysis. It builds on the idea that systems can use existing data to improve their performance. In this work, we will focus on Supervised Learning approaches, where data being studied are already labelled. **Classification** is a Supervised Leaning process that approximates a mapping function (f) from input variables (x) to discrete output variables (y). The discrete output values are often probabilities that are used in the classification process to separate a dataset into discrete categories. The **Naïve Bayes** classifier is the ML method used in this paper.

Active learning [3] is a subfield of AI and machine learning. AL is founded on the notion that actively requesting information to label the training set can be more time and cost efficient than labelling the entire set by hand. A key driving factor for using AL is that there is often an abundance of unlabeled real-world data that we can feed into the classification process. Active Learning systems can use uncertainty sampling methods (or strategies) to choose which data to present to the "oracle" for labelling. We employ three AL strategies in this work: Least Confident (LC), Margin Sampling (MS), and Entropy Measure (EM). LC is the simplest of the strategies. In this case, the algorithm chooses the instances for which the learner is least confident in its most likely label. The most uncertain their classification is, the more likely they will get chosen to be incorporated into the training set. MS is a strategy where the learner picks the instances with the smallest difference between most probable and second most probable classification. It hones on uncertainty between classifications. EM is the case where the learner calculates the entropy

measure of the probabilities associated with each instance and selects the instances with the highest entropy.

NLP is a sub-field of Computer Engineering, Data Science, AI and Linguistics that focuses on bridging the gap between computers and natural languages. Text preprocessing is an important step in NLP, because it transforms text into formats that are better processed by ML algorithms [4]. Without it, ML algorithms have a hard time understanding human speech patterns and very high error rates. Text preprocessing is comprised of three main components: Tokenization; Normalization; and Noise Removal. Tokenization is the task of dividing character sequences (like phrases or paragraphs) into tokens. Text Normalization aims to regularize the text, by transforming it to a single canonical. Examples of text normalization include converting characters to lowercase or transforming abbreviations to their normal form. Finally, Noise Removal consists of cleaning up text, for example, by removing extra white spaces.

III. Related Work

In [5], the authors present an automated tool for identifying conflicts in aspect-oriented requirements specified in natural-language text. Similarly to our work, they use the Naïve Bayes classifier to classify textual data. However, they do not use this tool to categorize requirements into classes such as "Functional" and "Non-Functional" (NF), but instead into "Conflict" and "Harmony". Two or more requirements are labeled as "Conflict" when they present a conflicting dependency, or "Harmony" when they interact harmoniously. Also, they do not use an AL as we do. It is worth noting that they mention that they needed a human annotator to label each example, and that was a very time-consuming process.

The work in [6] is focused on a data-driven approach to risk management in Requirements Engineering (RE). They aimed at predicting RE problems, their causes and their effects. For this purpose, they trained a series of Bayesian Networks to model cause-effect relationships that characterize the dependency between the three. They used data from the NaPiRE project, which offers survey data on RE practices. They focus on the risk management angle of RE whilst we focus on the documentation aspect of RE. We share the same goal, which is to decrease the rate of project failure. In their case, they accomplish that by finding the causes behind RE problems, while in our case we wish to possibly increase the quality of documentation by classifying existing requirements.

In [7], the authors propose the use of interpretative ML classifiers for RE. They used 8 datasets, of which they manually classified 1,500+ requirements. Some of these datasets were already classified, and were used to train and test other requirement classification approaches. In contrast, our approach to requirements classification is more flexible. as it is tailored for situations where there is no systematic collection and classification of requirement, and we have to work with very small datasets. Additionally, the author's approach for manually classifying requirements was time-consuming.

In [8], the authors reviewed 24 ML approaches for the identification and classification of NF requirements. In these approaches they studied 16 different ML algorithms. A key difference from our work is their focus on classes of NF requirements. Our approach is capable of classifying requirements into Functional, NF and other, user defined classes. In their review, the only ML algorithm that used AL techniques was S21. The work in [9] presents a process for automatically classifying requirements gathered through crowd-sourcing. Here, the authors were faced with the same issue we had to contend with, an unorganized and unclassified dataset. As with the other ML algorithms analyzed in [8], they classify user requirements into NF types. This is a much more limited usage of the classification process than the one we employ. In their case, they apply AL to reduce the number of labeled data required to train a classifier. They did not use AL to its full potential, as that was not the focus of their work.

The work in [10] approaches the concept of incorporating Active Learning in app review analysis. In this context, app reviews are descriptions users send in about their interaction and experience with an app. The authors intend to use AL to reduce the human effort involved in the process of analysing these reviews. They concluded that active learners are more effective than passive learners for their classification tasks. Their application of AL is similar to our own, an iterative process, with a small, random sample of data for initialization. Additionally, the AL strategies they employ are also the same, Least Confidence, Entropy and Margin Sampling. The biggest difference between this work and ours is its purpose. They intended to create an ML algorithm capable of classifying app reviews. For this, authors created four classes: "feature request", "bug", "user experience", and "rating".

IV. Approach

In different levels of abstraction (e.g., business, user, system) we can find Functional and Non-Functional requirements (NFRs). As these are common types of requirements, we used them as our main classes. Figure 1 shows an example of a requirements classification. If we map these requirement types into ML classes, we can categorize requirements into those classifications. We use text classification, with a Bayesian classifier, to calculate the probability of each requirement belonging to each class. The class with the highest probability is chosen for each requirement.

A. The MARE Process

In this section, we present the MARE process, which is a novel application of ML and Active Learning to classify requirements. The process can be described as follows:

1) Retrieve and clean the data in order to create the dataset of unlabeled instances;
2) Separate data into **Training Set** and **Test Set**;
3) Create a **Training Batch** with N entries from the **Training Set**;
4) Label the unlabeled instances in the **Training Batch** manually;

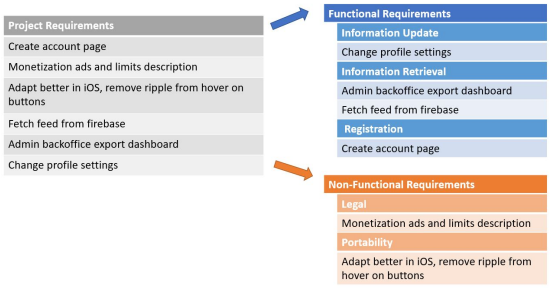

Fig. 1. Visual representation of categorizing requirements with ML

5) Train the Naïve Bayes Classifier with the **Training Batch**;
6) Classify the **Test Set** and the rest of the **Training Set** using the Naïve Bayes Classifier;
7) Use an **Active Learning strategy** to select the *N* unlabeled instances from the **Training Set** that will enter the **Training Batch**;
8) Repeat the process from 4 to 7 until stopping criteria is true or there are no more entries in the **Training Set**.

We will explore the first step in depth, to explain the data we are working with. Our dataset comes from a company that develops websites and apps. These requirements were extracted from the company's agile project development logs on Trello (a project management tool). We extracted each of the company's projects on Trello into a CSV file, that we then cleaned and manually classified. There were 1293 text entries in the final document, but only 436 were considered requirements. Whilst the MARE process includes a step where we ask a human to only label *N* instances at each iteration, for testing purposes we ended up labelling the entire set to evaluate the performance of the AL strategies. Then we separated our data into two datasets. The first dataset contained all 1293 text entries, and the second one contained only the 436 entries that were considered requirements. We used the first dataset to test how well our approach could **separate requirements and non-requirements**, and the second one to **sort requirements into classes and sub-classes**.

For our requirement sorting dataset, we came up with 4 functional requirements sub-classes, and 7 NFRs sub-classes; hence, 11 sub-classes in total. Unlike NFRs, there are no common sub-classifications for functional requirements. In this case, it made sense to create subcategories based on common traits in the company's projects. Taking into account that the company is involved with website and application development, the sub-classes we created are *Information Update*, *Information Retrieval*, *Registration* and *Site Feature*, described as follows:

- **Information Update** (IU) refers to requirements for changing information, e.g., "User edit profile name".
- **Information Retrieval** (IR) refers to requirements for downloads and auto-fill, e.g., "Fetch feed from Firebase".
- **Registration** (R) requirements are those that deal with registration and login functionalities, e.g., "Add login through Facebook".
- **Site Feature** (SF) requirements deal with site organization and functionality, e.g., "Button for editing a post".

The 7 NFRS subcategories selected from the dataset are:

- **Performance** (PE) refers to requirements for the quality of the application or website, e.g., "Reduce loading time".
- **Legal** (LE) refers to requirements that deal with laws, e.g., "Implement cookie policy".
- **Portability** (PO) requirements are those for devices changes, e.g., "Use accordion tabs on mobile devices".
- **Maintainability** (MA) refers to requirements that deal with documenting for ease of future development, e.g., "Add pagination to archives".
- **Security** (SE) refers to requirements for securing data, e.g., "Restrict profile role list based on user role".
- **Operational** (OP) refers to requirements that specify how a certain feature should be implemented, e.g., "Set maximum number of words on services content".
- **Usability** (US) refers to requirements for ease of use, e.g., "Increase font size when creating a new post".

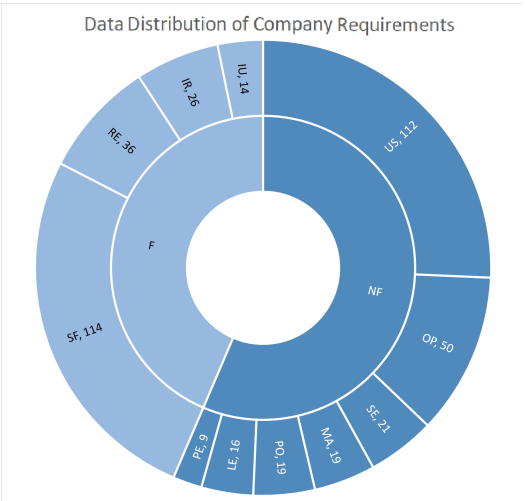

Fig. 2. Data Distribution of our dataset

Figure 2 shows how the requirements were sorted into each classification. As we can see in the figure, there are a total of 190 Functional requirements and 246 NFRs. Note that these classes were imbalanced, especially when we consider the sub-classes, which had a profound impact on the accuracy of our classification algorithm. With the dataset, the goals of our empirical evaluation was to: create a classifier that can classify requirements using a Bayesian classifier; find the amount of data points that are required to train the classifier; test different Active Learning strategies (explained in the next section). To accomplish these objectives, we created a step by step process.

Figure 3 shows the classification algorithm. To begin with, we shuffle the dataset, then split it into a **Train Set** and a **Test Set**. We take the first *N* requirements from the **Training Set** to create a **Training Batch** and give these *N* requirements to a human for labeling. We use this labeled **Training Batch**

to train the Naïve Bayes Classifier. With this classifier, we classify the **Test Set**, calculate and store the accuracy of this classification. If the stopping criteria is not reached, the algorithm classifies the rest of the **Training Set**. The stopping criteria can be an accuracy threshold the algorithm achieved, or a fixed number of batches, or a lack of significant increase in the accuracy of X number of batches.

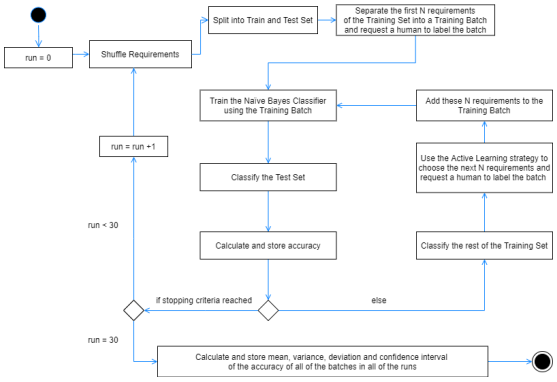

Fig. 3. General MARE Process for Classification

We then use an Active Learning strategy to pick out N more requirements to join our **Training Batch**. A key goal of adding more requirements is to see if the classifier improves its accuracy when tested with the test set. Then we start the process of training the Naïve Bayes Classifier again, and this loops until the stopping criteria is reached. When this occurs, the algorithm ends, and a new run begins. In the case of our first dataset (the one with 1293 entries), our Training Batches incremented in 80 entries at a time (N=80), and in the case of our second dataset (with 436 entries) it incremented in batches of 20. In both cases, our stopping criteria was to stop when the size of the **Training Batch** was equal to the size of the **Training Set**. We did this to test the Active Learning strategies (LC, MS, and EM) and compare them with the classical ML approach of training the classifier with the entire **Training Set**.

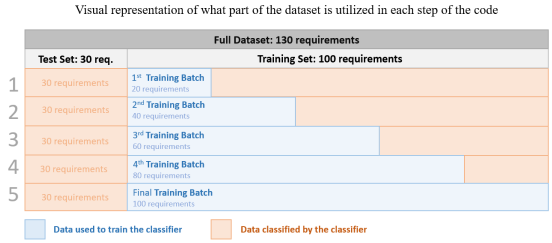

Fig. 4. Dataset segmentation and usage.

Figure 4 shows how the requirements dataset is divided and used throughout the algorithm. It follows a fictional example where there are 130 requirements in a dataset. That dataset is split into a **Test Set** with 30 requirements and a **Training Set** with 100 requirements. The stopping criteria for this example is that the size of the **Training Batch** is equal to the size of the **Training Set**. In the first line in the figure (marked with the number 1), we separated the first 20 requirements of the **Training Set** into our first **Training Batch**. The blue rectangle is our **Training Batch**, comprised of 20 requirements, and the orange rectangle to the right represents the rest of the **Training Set**, with 80 requirements. We train the classifier using the **Training Batch**, and use it to classify the rest of the **Training Set** and the **Test Set**.

In the second line, we add 20 more requirements into the **Training Batch**, meaning it now has 40 requirements, and the part of the **Training Set** we will classify drops down to 60 requirements. This pattern continues until the **Training Batch** is of the same size as the **Training Set**. Once the **Training Batch** has all 100 requirements, it classifies only the **Test Set** and ends that run of the code.

Active Learning is a process where the ML algorithm chooses how new data should be incorporated into the training set. From the classifier's and AL's perspective, the initial state of the training set has no labels and you only add the labels when you add a new batch (that is why it is easier to say that every time you add a new batch, a human adds the labels or classifies the requirements).

One of our goals was to see how accurate the Naïve Bayes classifier was with different sizes of the **Training Set**. We divided it into **Training Batches**, using AL strategies, because it allowed us to grow the training set in a smarter way. AL strategies do this by automatically classifying and sorting the data that is neither in the training set nor in the test set according to different parameters. The AL strategies we used were **LC**, **MS** and **EM**.

To test how effective the Active Learning strategies are, we created a **Random** learning strategy that chooses the next 20 requirements to enter the training batch at random. This is to be used as a baseline, so that we can compare this random strategy against the Active Learning strategies.

Taking the example in Figure 4, we will describe how the same process looks like when we take into account the 3 different Active Learning strategies and the random selection. Figure 5 shows an example of how the first two Training Batches might be formed. The first batch is the same for all the Active Learning strategies and the random selector. It includes the first N requirements of the Training Set. The second Training Batch includes the previous batch, as well as the requirements that each AL strategy algorithm chose. We can see in the figure that all the different strategies chose different requirements to add into the 2nd Training Batch.

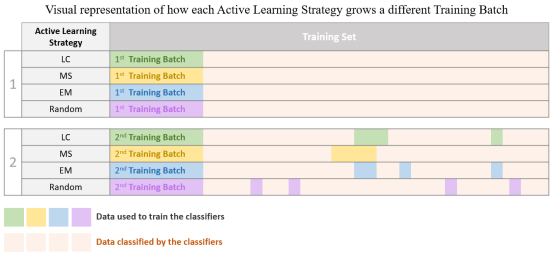

Fig. 5. How each Active Learning Strategy creates different Training Batches

V. Evaluation

This section is divided into two subsections, one for the separation of Requirements and Non-Requirements and the second for sorting Requirements into classes and sub-classes. We analyze the mean results for **30 runs of our code**. Our aim here is to see how Active Learning can improve the classification performance. We compare the results achieved via Active Learning with the ones achieved with with classical Machine Learning approach. To do this, we trained the classifier with the *entire* Training Set and using it to classify the Test Set. The mean accuracy results for this approach appear in the tables under the name "**Classic**". The results for each Active Learning strategy appear under their names (i.e., LC, MS, EM) with the addition of the **Random** selection process and the **Balanced** selection process. The Balanced selection process was applied only in the first dataset, and it is a strategy where every batch contains 40 entries of either class. Since there are 857 non-requirements and 436 requirements in this dataset, and we have a Test Set with 173 entries, there are *no more new "requirement" entries to the Training Batch* after it reaches a size of 720 requirements.

A. Separating Requirements and Non-Requirements

In this subsection, we will analyse the results achieved when separating the 1293 original text entries into requirements and non-requirements. We start with the mean accuracy.

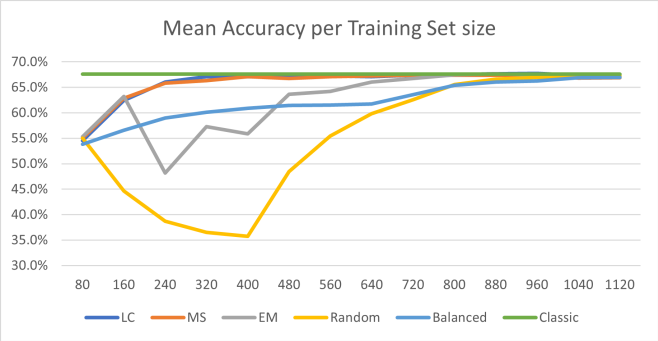

Fig. 6. Mean accuracy for separating requirements and non-requirements

Figure 6 shows the mean accuracy per training set size. We can see that the "Classic" Machine Learning approach achieves a mean accuracy of 67%. The LC and MS Active Learning strategies reached this threshold quicker than the Random, Balanced and EM. In fact, we can see that the Balanced strategy achieved surprisingly poor results.

For our classification problem it is much more important that the algorithm correctly selects the "requirements" than the "non-requirements". If it incorrectly classifies "non-requirements" as "requirements", they are easily classified as "non-requirements", by a human, later down the line. So we would like to see more True Positives (requirements classified correctly) and False Positives (non-requirements classified as requirements). We would also want to keep the number of False Negatives (requirements classified as non-requirements) low for this same reason.

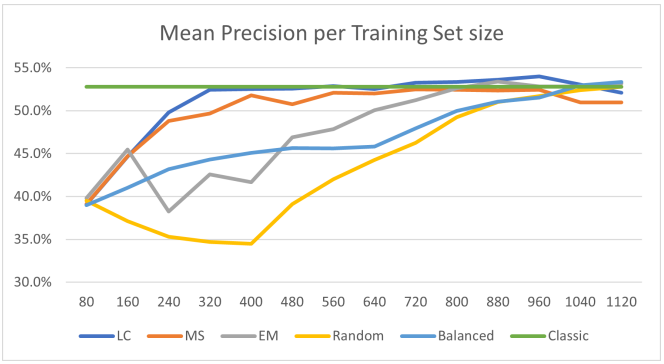

Fig. 7. Mean precision for separating requirements and non-requirements

Figure 7 shows the mean precision per training set size. Precision measures how many Positive class predictions actually belong to the Positive class. We considered the "requirement" class to be Positive and the "non-requirement" class as Negative. Here, precision quantifies how many instances that where **correctly** classified as "requirements". We can see that the classic approach reaches a precision level of 53% with a Training Set with 1120 entries. Once again, the LC and MS strategies reach this threshold quicker than the others, and we can see that the Balanced strategy performs very poorly. This indicates that the algorithm has difficulty in recognizing instances from the "requirement" class.

Figure 8 shows the mean recall per training set size. Recall measures how many "requirements" where classified as "non requirements". If there are many False Negatives, recall is low.

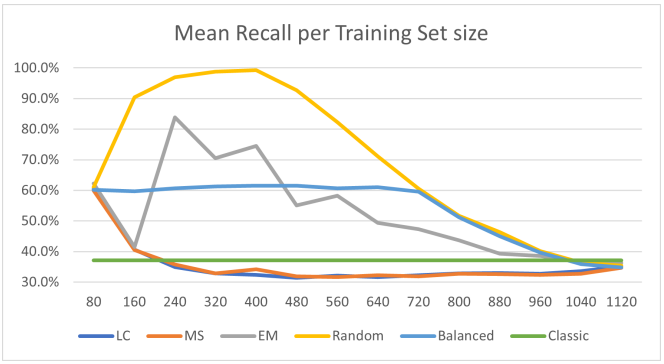

Fig. 8. Mean recall graph for separating requirements and non-requirements

B. Sorting Requirements into Classes and Sub-classes

Here we will analyse the results achieved when sorting the 436 requirements into classes and sub-classes. We will show two graphs, one with the accuracy results for the classes (Figure 9), and another for the sub-classes (Figure 10).

Figure 9 shows the mean accuracy results per training set size. The "Classic" Machine Learning approach achieves a mean accuracy of 73%. All the Active Learning strategies achieve similar results. They climb steadily towards the same accuracy as the "Classic" approach, but the "LC" and "EM"

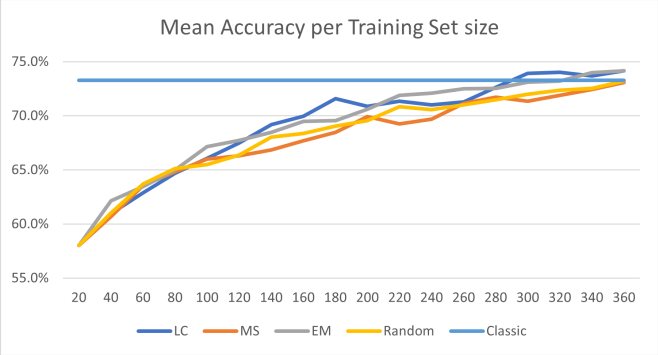

Fig. 9. Mean accuracy graph for sorting requirements into classes

strategies reach it in 280-300 requirements, and they even surpass these results. If we take into consideration that there are two classes, these are not excellent results. But we must also consider the inherent difficulties tied to this dataset. Firstly, the training set is very small (360 entries). Secondly, the dataset is comprised of entries in two languages, with some entries mixing the two in one sentence. Also, on average, each requirement is comprised of 8 words. For a text-based algorithm, all of these factors combined are a big challenge. Despite these difficulties, we can show that the LC reaches the 70% accuracy with only 200 requirements (i.e., 55.55% of the dataset), which shows that the Active Learning strategies are a promising approach for requirements classification.

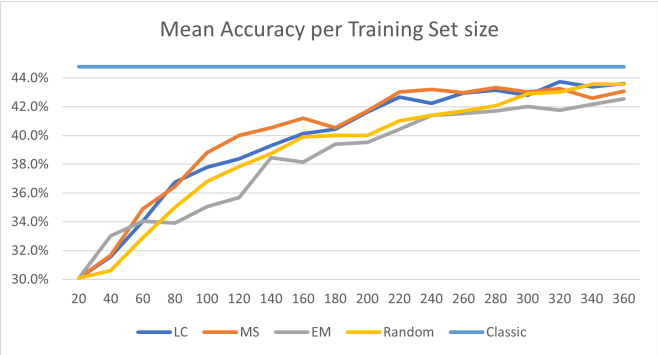

Fig. 10. Mean accuracy graph for sorting requirements into sub-classes

Finally, Figure 10 presents the mean accuracy results for sorting the requirements into sub-classes. The "Classic" approach achieves a mean accuracy slightly below 45% and none of the Active Learning strategies were able to reach the same value. Between the Active Learning strategies, "LC" and "MS" achieve the best results. Here, the dataset presents the same difficulties encountered in the previous classification problem, including an unbalanced dataset issue. As mentioned in the Approach section (IV), the number of requirements belonging to each sub-class vary wildly, from 9 to 112 requirements.

VI. Conclusions

This paper offers the groundwork for creating an ML approach capable of categorizing requirements with minor human involvement, with varying levels of abstraction (categories and subcategories) for system documentation purposes. We extracted requirements from agile development projects. We cleaned and labelled these requirements in order to create two datasets. We applied a classic ML text classification algorithm and tested how it performed on both datasets. We implemented an Active Learning framework, including 3 Active Learning algorithms (i.e., LC, MS, and EM). We tested how well the Active Learning system and algorithms performed for both of our datasets. The method of classification is useful for the identification and reuse of requirements for applications in a given domain, and particularly in the context of agile development like the one the company we worked with employs. It helps to organize and better document requirements, where before they where disorganized to the point where they were nestled in with non-requirement data.

There is a lot of work ahead in order to create a fully fledged system capable of categorizing requirements with minor human involvement, with varying levels of abstraction. Firstly, it is necessary to create and test new requirement datasets. Bigger requirements datasets are preferred, especially in order to properly test the Active Learning system.

VII. Acknowledgements

This work was partially supported by FCT, under project UIDB/04516/2020 (NOVA LINCS), UIDB/50021/2020 (INESC-ID), and the HOTSPOT project (PTDC/CCI-COM/7203/2020). It was also supported by the TAILOR project (EU Horizon 2020 research and innovation programme, GA No 952215).

References

[1] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," *Information and Software Technology*, vol. 45, no. 14, pp. 993–1009, Elsevier, 2003.
[2] I. Sommerville, "Software engineering," 10th Edition, Pearson, 2016.
[3] B. Settles, R. J. Brachman, W. A. Cohen, and T. G. Dietterich, *Active learning: Synthesis Lectures on Artificial Intelligence and Machine Learning 18*. Morgan and Claypool, 2012.
[4] J. Weng, "Nlp text preprocessing: A practical guide and template," *Towards Data Science*, vol. 26, 2019.
[5] A. Sardinha, R. Chitchyan, N. Weston, P. Greenwood, and A. Rashid, "Ea-analyzer: automating conflict detection in a large set of textual aspect-oriented requirements," *Automated Software Engineering*, vol. 20, no. 1, pp. 111–135, Springer, 2013.
[6] F. Wiesweg, A. Vogelsang, and D. Mendez, "Data-driven risk management for requirements engineering: An automated approach based on bayesian networks," in *RE'20*, pp. 125–135, IEEE, 2020.
[7] F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "Requirements classification with interpretable machine learning and dependency parsing," in *RE'19*, pp. 142–152, IEEE, 2019.
[8] M. Binkhonain and L. Zhao, "A review of machine learning algorithms for identification and classification of non-functional requirements," *Expert Systems with Applications: X*, vol. 1, Elsevier, 2019.
[9] C. Li, L. Huang, J. Ge, B. Luo, and V. Ng, "Automatically classifying user requests in crowdsourcing requirements engineering," *Journal of Systems and Software*, vol. 138, pp. 108–123, Elsevier, 2018.
[10] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah, "App review analysis via active learning: reducing supervision effort without compromising classification accuracy," in *Requirements Engineering Conference (RE) RE'18*, pp. 170–181, IEEE, 2018.