UNIVERSITY OF TARTU

Institute of Computer Science

Software Engineering Curriculum

Muhammad Bilal Shahid

Splitting User Stories Using Supervised Machine Learning

Master's Thesis (30 ECTS)

Supervisor: Ezequiel Scott

Tartu 2020

Splitting User Stories Using Supervised Machine Learning

Abstract:

User stories are a well-known tool for representing requirements. They define small fragments of the system and help in the developer's daily work. When we talk about user stories, then splitting them into tasks is common. Many approaches can be used to split a user story into tasks but these all approaches are based on manual working. In this era, where everything is now becoming digitalized. User stories should move to the next phase as well. In this paper, we will implement a novel idea to split a user story into tasks atomically using machine learning. We have used four machine learning algorithms random forest, SVM, KNN, and decision tree (ctree) on three open-source projects from Jira. The dataset we have used for this thesis was imbalanced, so we have used ROSE (randomly over sampling examples) and SMOTE (Synthetic Minority Oversampling Technique) to make a balanced dataset. We have applied machine learning algorithms separately on each project and also all projects combined into one dataset and then made comparisons on the results.

Keywords:

User Story, Splitting, Requirements, Random Forest, SVM, KNN, Decision Tree

CERCS: P170

Kasutajalugude jagamine Kasutades ülevaadatud masinaõpet

Abstraktne:

Kasutaja lood on tuntud vahend nõuete esindamiseks. Nad määratlevad süsteemi väikesed fragmentid ja aitavad arendaja igapäevatöös. Kui me räägime kasutaja lugusid, siis jagada need ülesanneteks on levinud. On palju lähenemisi, mida saab kasutada, et jagada kasutaja lugu ülesannetesse, kuid need kõik lähenemised põhinevad käsitsi töötamise. Praegusel ajastul, kus kõik muutub digitaalseks. Kasutaja lugusid peaks liikuma ka järgmisesse etappi. Selles raamatus rakendame uue idee jagada kasutaja lugu ülesanneteks aatomiliselt kasutades masinaõpet. Oleme kasutanud nelja masinaõppe algoritmi juhusliku metsa, SVM, KNN ja otsustuspuu (ctree) kolme avatud lähteprojekti Jira. Selle väitekirja jaoks kasutatud andmekogu oli tasakaalustamata, nii et me kasutasime tasakaalustatud andmekogumi tegemiseks roost (harvaliselt üle proovide näidete) ja smote (sünteetiline vähemus üleproovimise tehnika). Me oleme rakendanud masinaõppe algoritme eraldi iga projekti ja ka kõik projektid ühendatud ühe andmekogu ja seejärel tehtud võrdlusi tulemusi.

Märksõnad:

Kasutaja lugu, Jagamine, nõuded, Juhuslik metsa, SVM, KNN, otsustuspuu

CERCS: P170

Table of content

1. Introduction

Requirement gathering is an essential part of software development. The requirements phase of software development requires crucial communication and involves customers, stockholders, and users [1]. Deficient communication among the team members of a project can lead to misunderstanding, which results in undesired products and unsatisfied customers.

In Agile Software Development, requirements are written in the form of user stories, which saves a lot of time during the development process and assists the developer to understand the customer desires. User stories can help to break down difficult problems into small parts and develop them autonomously. It creates a precise picture of what to build and when in a prioritized manner and also it is a good way to inspire the non-technical team members to contribute.

Agile requirements engineering practices are based on continuous interaction and clients are a significant part of this process, over time they assure the development of requirements. They assure the prioritization of the requirements which helps to deliver the most valued features first. Additionally, in the requirements gathering course, direct communication of developers and the customer is recommended by agile companies. For user stories, there is no special process. Based on user stories, the followings are the most significant agile requirements engineering practices: gathering the user stories, user role, testing/acceptance criteria, planning the release, iterations, estimation, and communication.

While working with user stories, splitting a user story into tasks is common. But splitting can be very challenging. To decide whether a story needs to split or not and if yes then how. A badly written user story will mislead the developer or one user story dependent on others will consume too much time resulting in missing the deadlines. Which will result in an unsatisfactory customer and more time-consuming means more expense on the development end.

Several approaches can be used to determine whether a user story has to be split or not. For example, Davide et al. [6] collected nine user story splitting patterns workflow steps, defer performance, operations, business rule variations, variations in data, major effort, simple/complex, data entry methods, and break out a spike. They collected these patterns from various practitioners and put them together in one article. However, these patterns have limitations, for example how to identify which pattern will be a perfect fit for a user story. If we want to split based on estimated size (story points), then how we will measure the estimated effort for the story and what estimated size (story points), will be considered good to be split. Liskin et al. [11] explored the efficiency, benefits, and limitations of using requirements artifacts. They highlighted the scope granularity of the requirements artifacts. They suggested splitting of requirements artifacts where granularity varies one or more than one week but the evaluation of their research was mostly based on the individual opinion of team members. Therefore, their findings cannot be generalized. Most of the current approaches are based on manual identification of patterns and heavily rely on the developer or analysis experience.

This thesis aims to address the issue by exploring the use of machine learning to make predictions about whether user stories need to be split into tasks or not. For that, we extracted data from issue trackers (JIRA), which consists of three projects XD, DNN, and APSTUD. If an issue report has

children, then it was considered a parent and each parent needs to be split. A parent issue report was labeled as 1 and if it's not a parent then it's labeled as 0. By labeling as 0 and 1, we model the problem as a binary classification problem, where the prediction target is 1. We extracted data from the issue reports like summary, description, subtasks, and type for training the model. We evaluated the performance of different classification algorithms such as Random Forest, KNN, SVM, and Decision Trees. As for performance measures, we used accuracy, precision, recall, f1-score, and ROC/AUC.

1.1 Problem statement

In this technology era, when everything is becoming digitized, splitting user stories into tasks is still achieved by manual efforts. Performing a task manually takes more time and resources, and user stories are an important part of Agile development, so we can't avoid it. Splitting user stories manually requires good communication and expertise. For example, simple/complex is one of the user story splitting patterns reported by Lawrence [5]. Now, measuring the simplicity/complexity of a user story is dependent on its domain or goal and there is no optimal way to measure the simplicity/complexity of a user story. Due to inadequate communication if the simplicity/complexity of the story is measured wrongfully then it could cast during the development process.

Since the current approaches perform a manual approach to splitting the user stories into tasks. We are introducing a novel approach for splitting user stories into tasks automatically. This thesis aims to explore the use of machine learning to determine whether we need to split a user story or not. In particular, we model the problem as a binary classification task where we have 5 features on which we will train the model and then make predictions.

Primarily, we will extract projects from Jira and then clean the data. For cleaning, we will observe all the columns of the dataset and select only those which are relevant to our work, then we will select only those issue reports type which has the maximum number of parents (higher rate of splitting). After then we will deal with null values. Secondly, we will start choosing features on which we will train our model. Once the feature selection is done then we will construct a final table/ final dataset that will be used for this thesis. In the end, we will set a baseline classifier and then start training our models with multiple supervised machine learning algorithms. Once the models are ready then we will start making predictions on them. At this stage, we will make a comparison of all the algorithms with each other and with baseline classifiers.

We formulated the following research questions:

RQ1: What is the performance of predicting the splitting of user stories using information extracted from issue reports?

RQ2: What is the best strategy to address the data imbalance when splitting user stories?

1.2 Outline

The thesis is organized as follows. Section 2 describes the background. Section 3 describes the related work. Section 4 explains the method used in this thesis along with the dataset details and features used for training the model. Section 5 shows all the tables with machine learning algorithm results and their comparisons, the section 6 is conclusion and future work.

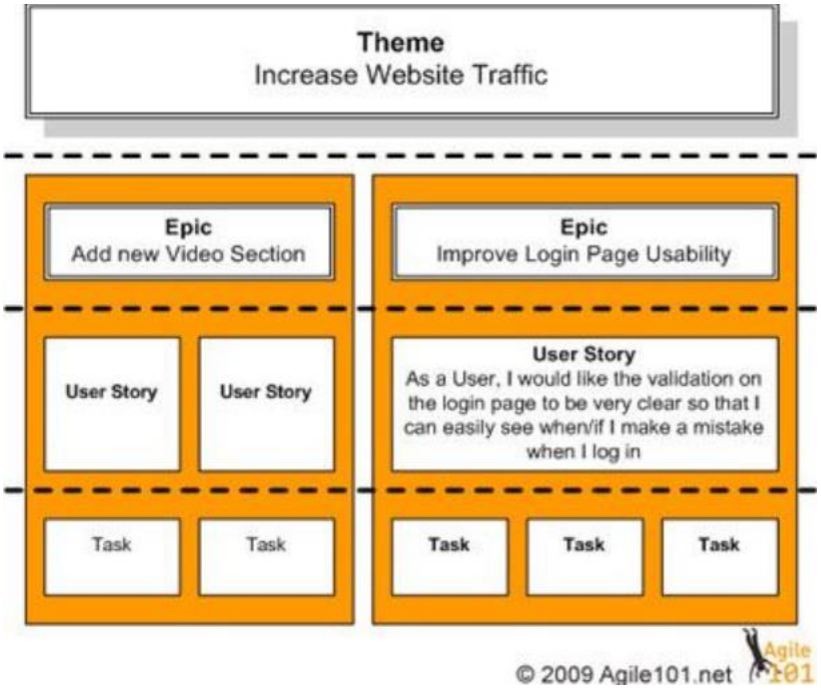
2. Background

Agile is an iterative approach to project management. Commonly used in software development. Agile software development methods do not contain long-tern planning and break the tasks into minor iterations. An important aspect of Agile Development is that the projects can start without whole requirements upfront. In the software development process, requirements are the description of features of a system that needs to be developed. In Agile software development, the requirements are written in the form of user stories. A user story is a description of a feature, valuable for customers and developers of software. It mainly consists of three parts.

1. Role: As a (someone), As a (someone)
2. Mean: I want/would to, I need to, I should be able, I am able to
3. Result: This part is not necessary. Points 1 and 2 are enough to make a Well-formed user story. It is the benefit/result, what we will get from the story (so that I can).

Splitting is when requirements/user stories are divided into smaller parts and each part is assigned to a team member. So the development process moves faster. When a user story becomes too complex and can take a lot of time of a developer then splitting it into smaller tasks is favorable because there might be some task or story which is dependent on this story. So until this story is implemented, the development is going to be delayed. Splitting a complex story into smaller tasks will improve the development process. Figure 1 shows the hierarchy of a requirement from Epic to one or more than one user story and then to one or more than one task.

Figure 1. The hierarchy of Epic->User Story-> Tasks[1]



[1] https://i.pinimg.com/originals/a3/f5/d9/a3f5d990be3295cf4dd69a6068b1ee00.jpg

Davide et al. [6] and Lawrence [5] has described nine splitting patterns to split large user stories into smaller. Lawrence [5] suggests that a good user story should follow the INVEST criteria described by Wake [11] and after splitting the user stories, the model still needs to be followed. The followings are the techniques to split a user story into smaller tasks developed by various practitioners.

1. **Workflow steps**: Workflow steps: Some user stories seem small but actually they require a complete workflow step. Dividing a big user story into small ones based on their workflow so that the development process can be done in a step by step stories.
2. **Defer Performance**: If the user story gets its most of the difficulty from its non-functional requirements, then it needs to split. So, in this pattern, once the functional requirements are done then move towards the non-functional requirements.
3. **Operations**: If the user story contains multiple operations, like CURD or manage account/system, all are possible operations for a user story which can be split into more than one task, and then each operation/task can be developed separately.
4. **Business Rule Variations**: If the user story contains business rules, then this pattern can be applied. Business rules are the constraints in which software needs to be developed. Each business rule can consist of multiple variations.
5. **Variations in Data**: This pattern can be implemented if there is an occurrence of variation in data. This is a complex pattern because data variations can depend on various factors like geographic areas, language, etc.
6. **Major Effort**: In this pattern, a user story can be divided into major efforts. For example, if one paying method is already implemented then adding more into the system is not tricky but adding the first one can take time.
7. **Simple/Complex**: When defining a user story, it seems that it's getting complex then stop and check what is the simplest variation for this story and consider acceptance criteria. First, implement the simplest variation and then more towards the complex part step by step.
8. **Data Entry Methods**: Sometimes the complexity is not in the system's functionality but in the user interface. So, in this pattern, a user story can be split into the simplest user interfaces and after implementing the simplest user interfaces then more fancy user interfaces can be implemented.
9. **Break Out a Spike**: It is not necessary that complexity makes a user story large. The implementation process is not adequately understood and could lead to the same results. The complexity of this pattern, it's been considered to use as a last resort. It is recommended to try to work out with other patterns before moving to this one as it required a lot of research.

Wake [10] has stated good user story criteria as INVEST. When creating stories and tasks, using the INVEST acronym will increase the performance. He has created INVEST strategies for writing good user stories where six quality principles are

- **I – Independent**: A user story should be independent. The dependence of user stories on each other will affect the development process. If there is no overlap of stories, then we

can achieve true prioritization which means user stories can be implemented in any order and time. If there is an overlap of stories, then there is a chance that an important story remains without developing due to dependency.

- **N – Negotiable**: A good user story should be negotiable. A user story should express the core of the requirement, not the details. The aim is to develop what the customer desires, not to flow the user story letters and develop something which is not required.
- **V – Valuable**: A good user story should be valuable. It should be valuable to the customer. If it does not have distinct importance, then we should not develop it. This issue may come when splitting a user story. We might split it into two parts, one of them consists of the valuable fragment and the other not. Which might make a developer or customer think it's not essential.
- **E – Estimable**: A user story should be estimated. Making an exact estimation is hard but it should be enough to have a prioritized implementation. If a user story is big then instead of making an estimation, it is more appropriate to split it.
- **S – Small**: A good user story should be small. A story often represents a few days or weeks of work of a developer but it is not easy to know the scope of a user story. If a user story is small, then probably we would not need to split, and also it would be easy to make an estimation.
- **T – Testable**: A user story should be Testable. The testability of a user story needs to be its acceptance criteria. It does not mean that all the tests should be written before starting the development process but it should be enough to know what we are developing.

3. Related work

In this section, an overview is given on user stories and what has been done in terms of requirements granularity and eliciting. The purpose is to relate the most recent studies.

a. Requirements Granularity

Olga et al. [2] Granularity is a way to break down a huge task into smaller ones. The granularity of a story can be understood in three terms:

- **Celerity**: A user story will be considered as vaguely written if there is a lot of information missing from it.
- **Concreteness**: An abstract user story will be able to define the desired functionality.
- **Scope**: It characterizes the scope of system functionality. With the given volume of functionality, the scope of a story can be determined.

These three aspects are significant to determine the quality of a story. They focused on the expected implementation time/duration of a user story by splitting them and duration relevance factor with a story. They stated that a story with a duration of 4 to 5 days is more likely to be split based on the questionnaire of 39 developers.

Lucassen et al. [8] showed a novel method by using clustering and semantic relatedness for visualizing user story requirements at several granularity levels. They contributed with a proof of concept. The idea is to create K clusters and then zooming them. They used 4 real-world cases of user stories to check the feasibility and to evaluate. They stated 3 possible applications for their method: 1st is to determine missing relationships between clusters which could affect further user stories. 2nd, training system functionality by discovering basic, controllable chunks. 3rd is to examine estimated system changes after adding new user stories.

Davide et al. [6] explained that eliciting requirements from clients is difficult. When working with Agile processes, the client discusses with the developer directly and reports the requirements in an unstructured manner. They did requirements decomposition processes using three Agile processes where 1st is XP, 2nd is Scrum and the 3rd is Scrum with Kanban and an unstructured process. They conducted their research on several case studies and compared the requirements decomposition process with using processes the Extreme Programming, the Scrum, the ad-hoc process namely called the Banana process, and the Scrum with Kanban. They stated that the requirements decomposition just not rely on the team but it's also process-dependent.

b. Manually splitting of user stories

Coelho and Basu [3] has described the story point approached in different steps

- **Customer Expectation**: It's essential to know the success/failure standards of the project. Which can be determined by schedule, scope, and resources.
- **Estimation of User Stories**: It's not essential to estimate every story at the same time but to target those which are part of forthcoming releases.
- **Select an Iteration Length**: Iteration time should not be too short in case of a setback and also should not be too long in case of working relaxed at the beginning and rushing at the end. To estimate correctly several factors, need to be considered including the duration of the whole release plan, priorities, feedback, and iteration overhead uncertainty.
- **Estimate Velocity**: Velocity can be measured with the team's progress rate.
- **Prioritize User Stories**: Prioritize a story based on financial value, risk mitigation, and cost involved for implementation.
- **Estimation of Delivery Date**: Delivery date estimation is based on iteration duration.

Liskin et al. [11] have also characterized the granularity of requirements into three terms. 1st is clarity/vagueness, if a user story is written vaguely. 2nd is concreteness/abstraction, if a user story has an abstract concept of the desired functionality. 3rd is scope, if a user story entails a lot of system functionality would have a big scope. They conducted a Kanban project in which a questionnaire was taken after each sprint from students who worked on the requirements artifacts. In total they had five students, four were the developers and one was the customer. In the end, they took a 30-minute interview from each participant. The requirements artifacts were based on user stories and tasks, often the stories were split into subtasks. Splitting accord when a developer was planning to work on a story and considered splitting important. They stated that requirements artifacts had a good impact on the project but the misunderstandings ensued where requirements artifacts were not observable to all the members.

Cohn [4] the user stories are often fairly small, so typically there is not much disaggregation needed, and also it is not essential to set a specific standard about the preferred size of a story. They have given three strategies for disaggregating user stories into tasks.

- If a specific task of a user story is hard to estimate. For instance, if we need approval on data format or font and the manager is too slow to respond. So this is a particular reason to split that task and make a new story.
- If a specific task of a user story can be done more efficiently by discrete developers, then we should split those tasks. For instance, at the end of an iteration when we are running out of time then it is useful to split a user story into multiple tasks.
- If it helps the development process and the developers know that the part of a user story is complete then we should split it as a task. For instance, if we want to develop a search system. We can split this story into tasks as in a simple

search system and an advanced search system. Now, these two tasks need a database connection along with SQL but if we keep these tasks as one then both database and search system development will be delayed. Whereas, after splitting these tasks we can implement the simple search screen and its database together and we will have a working prototype in lesser time.

c. Conclusion

We have conducted our research on requirements granularity, disaggregation, eliciting, and splitting user stories both manually and automatically. We were able to find papers on requirements granularity where Olga et al. [2] describes granularity as a way to divide a task into smaller parts and Cohn [4] explaining the guidelines for disaggregating user stories into tasks. This related work section is consisting of 5 papers and one book. All of them are illustrating a breakdown structure for user stories but they are using a manual approach. None of them has suggested an automated approach for splitting. According to our research, there are no research papers on automatic user story splitting.

4. Method

In this section, we present the approach used to determine whether a user story must be split or not.

4.1 Data Collection

We use data collected from three open-source projects that use Jira. To collect the data, we used an open-source data extractor "jiraextractor" [7]. Using this extractor, we were able to get the project data as CSV files. The dataset consists of issue report data along with a changelog of events. Out of more than 100 fields, we use only the following ones: project_name, project_id, summary, description, subtasks, and type. The changelog file contains multiple information in it but we use only three fields: the number of comments, the number of attachments, and the number of story points.

The three projects collected from the issue trackers are:

1. XD[2]: this is a unified, distributed for data ingestion project. It also consists of real-time analytics and data export.
2. DNN[3]: this is a web content management system.
3. APSTUD[4]: this is a web development IDE.

4.2 Data pre-processing

Since the prediction target is whether a user story has to be split or not, we first identify the issue reports that were split. To do this, we inspect the field "subtasks" of the issue report data. This field can either have an object or be empty. When the field has an object, the object contains a list of issue reports (children) along with their details like summary, description, etc. Based on this field, we labeled each issue report as "parent" or "other" using the following criteria:

- Parent: An issue report is labeled as a parent if an issue report's subtasks field is not empty, which means the issue report is a parent issue report. Parent issues are labeled as "1".
- Other: If an issue report has no subtasks (empty field), then the issue report is neither a parent nor a child. We characterize these issue reports as 'other' category, which are labeled as "0".

In the collected dataset, there are seven types of issue reports: Improvement, Technical task, Bug, Story, Epic, New Feature, Task. We selected issue reports of type Story, Epic, Improvement, and Task (Table 1) since these types of issue reports are more likely to have children.

[2] http://projects.spring.io/spring-xd
[3] http://www.dnnsoftware.com/platform
[4] http://www.aptana.com/

Table 1. Story, Epic, Improvement and Task with respect to target

Project	**Prediction Target**	**Issue type**				**Total**	
		Story	Epic	Improvement	Task		
DNN	0	367	61	0	271	699	737
	1	34	2	0	2	38	
XD	0	2548	106	0	0	2654	2,702
	1	45	3	0	0	48	
APSTUD	0	517	6	1312	0	1835	1,856
	1	12	4	5	0	21	
ALL	0	3432	173	1312	271	5188	5,295
	1	91	9	5	2	107	

We also removed issue reports of type Improvement and Task (Table 3) since the type Improvement only existed in project APSTUD and Task only existed in project DNN. Also, the number of parents in both types of issue reports were very low. In total we are using two types of issue reports Epic and Story. Moreover, these two types exist in all three selected projects.

4.3 Data cleaning

For data cleansing, first the duplicated keys were removed. In the issue report file, there were no duplicated keys. In the changelog file, there was a duplication of keys. For example, an issue report can have one or more than one attachment. So, to extract attachments from the changelog file, only identical attachments by name were selected and if an attachment had the same name then the last updated were selected. The same approach was used for extracting the comments field. For story points, the last updated record was selected. There were issues reports with hyperlinks and tags like and {html}. So these issue reports were removed from the dataset. If the summary and description both were missing for an issue report, then that was also removed. There were no

missing values for field Type and no nans in field subtasks. For the story point, there were some missing values. So, instead of removing those records. We add the median value of the existed recodes.

Table 2. Final dataset used for making predictions

Project	**Target**	**Issue type**		**Total**	
		Story	Epic		
DNN	0	367	61	428	464
	1	34	2	36	
XD	0	2548	106	2654	2,702
	1	45	3	48	
APSTUD	0	517	6	523	539
	1	12	4	16	
ALL	0	3432	173	3,605	3,705
	1	91	9	100	

4.4 Features

Table 2 summarizes the features used to train the models. We use the following features calculated from the original dataset:

- **Type of user story**: Each issue report belongs to a certain category/type. An issue report can be a Bug, an Epic or a Story, etc. Based on this, we can train a model with which type of issue report is most likely to be split.

- **Story points**: Story points determine the size of an issue report. According to Lawrence [5], the size is one of the main reasons to split a user story into one or more than one user story.
- **Attachments**: Since the user story size is one of the main reasons that determine the splitting [5], the number of attachments can be used as an indicator of a big story that needs to be split.
- **Comments**: Issue report complexity is another significant reason to split it [5]. In this context, the number of comments can be one way to determine the complexity of an issue report. The number of comments/heavy discussion to an issue report shows that it might be a complex issue that needs more attention.

- **Wordcount**: The number of words was counted from the description field. Similar to the number of comments, a high number of words can be an indicator of a big issue that needs to split.

  For example: As a developer, I'd like to move 'serialization codec' from Spring XD repo into SI, so I can update Spring XD to inherit the features/functionalities via maven dependency.

  The above user story is extracted from project XD. In total it has 28 words and it's a parent with 3 children.

Figure 2 illustrates an issue report extracted from the XD project and used in the dataset. From this issue report, we have extracted one comment with story points 5 and the number of words is 20. This issue report is a parent with two children.

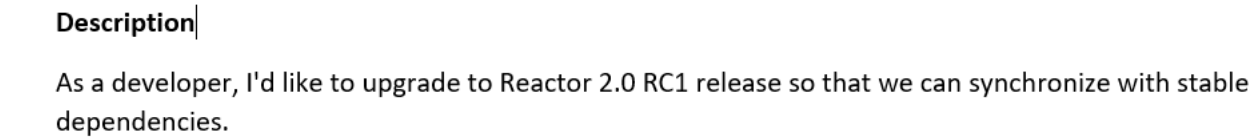


Figure 2. An issue report extracted from XD project

4.5 Predictive Models

Writing user stories are an important part of Agile software development and splitting a user story into smaller tasks often happens during development. Splitting a user story requires good communication and miscommunication can lead to confusion among the developers, which could cause problems in the development process.

Our prediction target would be 1 and 0, where 1 means the issue report is a parent task (i.e., the issue report must be split) and 0 means it's not a parent task (i.e., the issue report must not be split). In total, we are using 5 features for predicting the target: wordcount, storypoint, number of attachments, number of comments, and type.

The supervised machine learning algorithms used are SVM, KNN, Decision Tree (ctree), and Random Forest. In the case of Random Forest, the parameters used are: ntree = 300 to 500, mtry = 4, importance = TRUE, proximity = TRUE. For KNN, the parameter used are: tuneLength = 20, trControl = trainControl (method = "repeatedcv", number = 10, repeats = 3), preProc = c("center", "scale"). The SVM algorithm provides four types of kernels while training a model polynomial, linear, sigmoid, and radial (default). While finalizing Table 9, all these kernels were used in each scenario and only the best results with respect to the kernel have been illustrated in Table. For decision tree (ctree), controls were used to improve the performance where controls = ctree_control (mincriterion = 0.9, minsplit = 100). The mincriterion value and minsplit values were changed on each algorithm.

In the cleaned dataset, we have a total of 3705 issue reports, in which 0 class consists of 3605 issue reports, and 1 class consists of 100 issue reports. which means the dataset was very imbalanced. To cope with this problem, we used two data imbalance techniques to make a balanced dataset. The followings are the data imbalance techniques:

- SMOTE (Synthetic Minority Oversampling Technique): It increases the balanced number of cases in a dataset. It does not duplicate the minority class rather it creates new cases based on feature space and nearest neighbors.
- ROSE (randomly over sampling examples): it's used for binary classification problems of data imbalancing. It creates synthetic balanced samples for an imbalanced dataset.

We applied all three categories (over-sampling, under-sampling, and both-sampling) from ROSE to train the labeled dataset. For both-sampling, the total number of issue reports in the training data was used as a resampling number. For under-sampling, the total number of 1 (issue report is a parent) in the training data multiply by 2 was used as a resampling number because we are using under-sampling, the ROSE technique will remove the class 0 (issue report is not a parent) instances and create a balanced dataset. For over-sampling, the total number of 0 in the training data multiply by 2 was used as a resampling number because we are using over-sampling, the ROSE technique will duplicate the class 1 instances and create a balanced dataset. Extra parameters p = 0.5 and seed = 222 was set for both-sampling. In the case of SMOTE, K = 3 and dup_size = 0 was used. The resampling size for all projects combined was 5047. For DNN it was 612, for XD it was 3732, and APSTUD it was 748.

Primarily, we trained each project separately applying each machine learning algorithm with ROSE, and SMOTE then made comparisons of the predictions. Secondly, we combined all projects as one dataset and then made comparisons of the predictions.

4.6. Evaluation

To evaluate the predictive models, we split the dataset into two portions of 70% and 30%. We use the 70% portion as the training dataset and 30% portion as the testing dataset. Then, we train four supervised machine learning algorithms for binary classification using the training dataset. Once

the models were trained, we made predictions on the testing dataset. To compare the performance of the algorithms, we calculated the following metrics: Confusion matrix, Accuracy, precision, recall, F-measure, and ROC/AUC.

As illustrated in figure 3 the Confusion matrix:

True Positives (TP): True positive value is correctly predicted positive values. In this case, the value of actual and predicted class both are 1. True Negatives (TN): True Negative value is correctly predicted negative values. In this case, the value of actual and predicted class both are 0. False Positives (FP): In this case, the value of the actual class is 0 and the value of the predicted class is 1. Which means the predicted class made some wrong predictions. False Negatives (FN): In this case, the value of the actual class is 1 and the value of the predicted class is 0. Which means the predicted class was not able to predict some of the values.

Figure 3. Confusion matrix (a.k.a. contingency table)[5]

	Predicted = Yes	**Predicted = No**	
Actual = Yes	True positives (TP)	False negatives (FN) (Type II error)	Positives (Pos)
Actual = No	False positives (FP) (Type I error)	True negatives (TN)	Negatives (Neg)
	Predicted positives (PPos)	Predicted negatives (PNeg)	Total

As illustrated in figure 4 the formulas:

Accuracy: The accuracy is one of the intuitive performance measures. It is the true predicted values from the testing dataset. Precision: The precision is the true predicted vales from the total predicted values. Recall: The recall is the true predicted vales from the total actual class values. F-measure: F-measure is the subjective average of the Recall and the Precision. So, it takes false negatives and false positives both into account.

[5] https://courses.cs.ut.ee/MTAT.03.183/2017_fall/uploads/Main/DM2017_lecture_08_machine_learning_2.pdf

[6]Figure 4. Formulas used to calculate Accuracy, precision, recall, F-measure

$$Accuracy = (TP + TN)/Total$$

$$Precision = TP/PPos$$

$$Recall = TP/Pos$$

$$F - measure = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

In addition, we used two baseline classifiers. A basic technique to set a baseline or benchmark to see which of the algorithms in all cases have performed better than the baseline classifier. The following are baseline classifier

- Majority Class Classifier
- Random Guess

For random guess, we assigned the labeled dataset a random value of 1 and 0, and used RandomGuess from the lolR library to train the model then made predictions on it. For the majority class classifier, we used MajorityClassClassifier from library RSSL to train the model then made predictions on it. The majority class classifier always predicts the majority class. As in our original dataset, the majority class is 0, which means it will always predict 0 (the issue has not to be split).

[6] https://courses.cs.ut.ee/MTAT.03.183/2017_fall/uploads/Main/DM2017_lecture_08_machine_learning_2.pdf

5. Results

In this section we will present the results of applying the predictions models and a comparison of them.

5.1 Baseline classifiers

Table 4 shows the results of the baseline classifiers.

Table 3. Results of the baseline classification.[7]

Algorithm	**Sampling strategy**	**Project**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1 score**
Random Guess Classifier	--	DNN	0.5	0.4	0.0550	0.4917	0.0483
		XD	0.4856	0.3529	0.0147	0.5115	0.0141
		APSTUD	0.5054	0.3333	0.0036	0.4306	0.0036
		ALL	0.5306	0.3714	0.0177	0.5204	0.0169
Majority Class Classifier	--	DNN	0.933	0	NaN	0.5	NaN
		XD	0.9787	0	NaN	0.5	NaN
		APSTUD	0.9946	0	NaN	0.5	NaN
		ALL	0.9779	0	NaN	0.5	NaN

[7] In the original dataset, we have 0.9730% of class 0 and 0.0269% class 1. So our majority class is 0. That's why Majority Class Classifier didn't predicted any issue report of class 1. Which resulted in precision 0. Now using Recall and F-measure formulas, if anything divides by 0 then the answer is undefined (NaN).

5.2 Decision tree (ctree)

Comparison with majority class classifier: When all the projects are used (project ALL) with bassline classifier, the accuracy is 97% and the decision tree in all the scenarios was not able to get a higher accuracy then bassline classifier because the dataset is imbalanced. So, we can't only rely on accuracy. To solve this issue, we are using F-measure and ROC AUC. The algorithm has performed better than the baseline classifier in each scenario when calculating the F1 score because the F1 score for the bassline classifier is undefined and it has also performed better than the baseline classifier in each scenario when calculating ROC AUC, except the APSTUD project with the Undersampling technique. It has the ROC AUC value of 47% which is lower than the bassline classifier but in the rest of the scenarios, it's higher.

Comparison with random guess classifier: The decision tree (ctree) algorithm was able to predict more parents with far more accurately than the baseline classifier using SMOTE technique. Especially in the case of DNN and APSTUD projects, the number of predicting parents is far higher than projects XD and altogether.

Using ROSE technique with both-sampling, undersampling, and oversampling, the algorithm was able to predict more parents with far more accurately than the baseline classifier. Especially in the case of the APSTUD project, using ROSE with both-sampling and SMOTE, we were able to predict the same number of parents with the same accuracy in the DNN project. In the case of the DNN project, the predicted number of parents is two times higher than the baseline classifier with better accuracy.

Overall Comparison: In conclusion, the decision tree (ctree) has given a better performance than the random guess classifier. All projects combined, SMOTE were able to give better results with higher accuracy along with 5% of parents predicted with the rate of 18% of prediction. ROSE with both-sampling and ROSE with oversampling, we were able to predict the same and highest number of parents with the same and highest accuracy in the APSTUD project. For the DNN project, using ROSE with both-sampling and SMOTE, we were able to predict the same and highest number of parents with the same and highest accuracy. For the XD project, ROSE with oversampling has performed better than others. If we consider the F1 score and ROC AUC, then the decision tree has performed better then majority class classifiers in most of the cases.

Table 4. Decision tree results with data sampling techniques

Algorithm	**Sampling strategy**	**Project**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1 score**
Decision tree (ctree)	SMOTE	DNN	0.8106	0.1875	0.2	0.542	0.1935
		XD	0.8611	0.3529	0.0566	0.6125	0.0975

		APSTUD	0.9221	0.8	0.2666	0.8631	0.3999
		ALL	0.8843	0.1875	0.0555	0.5919	0.0856
Decision tree (ctree)	Both-sampling	DNN	0.8106	0.1875	0.2	0.542	0.1935
		XD	0.8473	0.4705	0.0661	0.663	0.1159
		APSTUD	0.9481	0.8	0.3636	0.8765	0.4999
		ALL	0.7667	0.4062	0.0515	0.5919	0.0914
Decision tree (ctree)	Undersampling	DNN	0.6136	0.5	0.1568	0.5647	0.2387
		XD	0.4919	0.7058	0.0290	0.5965	0.0557
		APSTUD	0.539	0.4	0.0285	0.4718	0.0532
		ALL	0.6935	0.3437	0.0334	0.5238	0.0608
Decision tree (ctree)	Oversampling	DNN	0.75	0.1875	0.1304	0.5075	0.1538
		XD	0.8611	0.4117	0.0648	0.6413	0.1119
		APSTUD	0.9481	0.8	0.3636	0.8765	0.4999
		ALL	0.7486	0.4062	0.0477	0.5825	0.0853

5.3 KNN

Comparison with majority class classifier: When Comparing with F-measure and ROC AUC, the algorithm has performed better than the baseline classifier in each scenario when Comparing the F1 score because the F1 score for the bassline classifier is undefined and in the case of ROC AUC, it has also performed better when Comparing ROC AUC with all projects separately except

XD and APSTUD project with SMOTE technique and DNN project with all techniques of ROSE. Also in the case of all projects combined, the SMOTE technique was not able to perform better than the bassline classifier when calculating ROC AUC.

Comparison with random guess classifier: The KNN algorithm was able to predict more parents with more accurately than the baseline classifier using SMOTE technique. In the case of the DNN project, the baseline classifier was able to predict 5% of parents with a 40% prediction rate and the accuracy was 50%. Whereas, using the SMOTE we were able to predict 5% of parents with a 6% prediction rate and the accuracy was 75%. The SMOTE performance will be considered better because it has a 25% higher accuracy rate and the number of predicted parents is the same.

Using ROSE with both-sampling, undersampling, and oversampling, the algorithm was able to predict more parents with far more accurately than the baseline classifier. The XD and APSTUD projects were able to predict more number of parents with far more prediction rate and accuracy. Especially in the case of the APSTUD and XD projects using SMOTE, we were able to predict the highest number of parents with better accuracy.

Overall Comparison: In conclusion, KNN has given a better performance than the random guess classifier. All projects combined, SMOTE were able to give better results with higher accuracy along with 6% of parents predicted with the rate of 15% of prediction. For the DNN project, SMOTE performed better than others. For the XD project, SMOTE performed better than others. For the APSTUD project, SMOTE performed better than others. If we consider the F1 score and ROC AUC, then the KNN has performed better then majority class classifiers in most of the cases.

Table 5. KNN results with data sampling techniques

Algorithm	**Sampling strategy**	**Project**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1 score**
KNN	SMOTE	DNN	0.75	0.0625	0.0526	0.5463	0.0571
		XD	0.9011	0.3529	0.0810	0.367	0.1317
		APSTUD	0.8831	0.6	0.1578	0.2537	0.2498
		ALL	0.915	0.1562	0.0694	0.4531	0.0961
KNN	Both-sampling	DNN	0.5378	0.5	0.1311	0.5216	0.2077
		XD	0.8135	0.4705	0.0540	0.6458	0.0968

		APSTUD	0.7597	0.6	0.0789	0.6826	0.1394
		ALL	0.8544	0.1875	0.0425	0.5314	0.0692
KNN	Undersampling	DNN	0.6364	0.1875	0.0789	0.4429	0.111
		XD	0.7384	0.5294	0.0428	0.6362	0.0791
		APSTUD	0.7403	0.4	0.0512	0.5758	0.0907
		ALL	0.6438	0.5937	0.0475	0.6508	0.0879
KNN	Oversampling	DNN	0.6136	0.3125	0.1111	0.4838	0.1639
		XD	0.8623	0.3529	0.0571	0.6132	0.0982
		APSTUD	0.8701	0.6	0.1428	0.7396	0.2306
		ALL	0.887	0.1562	0.0485	0.5325	0.074

5.4 Random Forest

Comparison with majority class classifier: When Comparing with the F1 score, all projects separately and altogether were able to perform better than the bassline classifier. When comparing ROC AUC, in the case of all the projects, the algorithm has performed better than the bassline classifier. In the case of each project separately, only DNN with Both-sampling and Oversampling was not able to perform better than the bassline classifier. The rest all of the scenarios were able to perform better than the bassline classifier. Comparing based on accuracy, all projects separately and altogether were not able to perform better than the bassline classifier.

Comparison with random guess classifier: Dataset imbalancing technique SMOTE, the random forest algorithm was able to predict parents far more accurately than the baseline classifier. Especially in the case of the APSTUD project, the number of predicted parents is far higher with far more accuracy.

Dataset imbalancing technique ROSE with both-sampling, undersampling, and oversampling, the algorithm was able to predict more parents with far more accurately than the baseline classifier.

Especially in the case of the APSTUD project, the number of predicted parents is 30% and the prediction rate is 60% with 94% accuracy.

Overall Comparison: In conclusion, the random forest has given a better performance than the random guess classifier. All projects combined, SMOTE were able to give better results with higher accuracy along with 10% of parents predicted with the rate of 12% of prediction. For the DNN project, SMOTE performed better than others. For the XD project, SMOTE performed better than others. For the APSTUD project, ROSE with oversampling performed better than others. If we consider the F1 score and ROC AUC, then the random forest has performed better then majority class classifiers in most of the cases.

Table 6. Random Forest results with data sampling techniques

Algorithm	**Sampling strategy**	**Project**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1 score**
Random forest	SMOTE	DNN	0.8103	0.2222	0.1176	0.541	0.1538
		XD	0.9424	0.2352	0.1081	0.5965	0.1481
		APSTUD	0.9221	0.4	0.1818	0.6698	0.2499
		ALL	0.9421	0.1250	0.1000	0.5457	0.1111
Random forest	Both-sampling	DNN	0.7727	0.125	0.1111	0.4935	0.1176
		XD	0.8736	0.3529	0.0625	0.6189	0.1061
		APSTUD	0.9221	0.8	0.2666	0.8631	0.3999
		ALL	0.7595	0.3125	0.0392	0.5427	0.0696
Random forest	Undersampling	DNN	0.5152	0.5	0.125	0.5086	0.2
		XD	0.7534	0.4705	0.0408	0.6151	0.075
		APSTUD	0.6429	0.6	0.0535	0.6154	0.0982

		ALL	0.6944	0.5000	0.0428	0.5833	0.0788
Random forest	Oversampling	DNN	0.8106	0.0625	0.0909	0.4881	0.074
		XD	0.8723	0.3529	0.0618	0.6183	0.1051
		APSTUD	0.9416	0.6	0.3	0.7765	0.4
		ALL	0.8192	0.3125	0.0531	0.5655	0.0907

5.5 SVM

Comparison with majority class classifier: When comparing ROC AUC, in the case of all the projects, the SVM algorithm has performed better than the bassline classifier except in the case of the SMOTE technique. In the case of each project separately, only DNN with SMOTE was not able to perform better than the bassline classifier. When Comparing based on the F1 score, all projects separately and altogether were able to perform better than the bassline classifier. Comparing based on accuracy, all projects separately and altogether were not able to perform better than the bassline classifier.

Comparison with random guess classifier: Dataset imbalancing technique SMOTE, the SVM algorithm was able to predict parents more accurately than baseline classifiers. Especially in the case of APSTUD projects, the number of predicting parents is far higher with better accuracy.

Dataset imbalancing technique ROSE with both-sampling, undersampling, and oversampling, the algorithm was able to predict more parents with far more accurately than the baseline classifier. Especially in the case of the APSTUD project, the number of predicted parents and accuracy is very high with SMOTE and ROES all the techniques. In the case of the DNN project with Both-sampling, the accuracy of the algorithm and baseline classifier is the same but the other metrics precision, recall, ROC AUC, and F1 score are quite high. So, we can say that the DNN project with Both-sampling has also performed better than the baseline classifier.

Overall Comparison: In conclusion, the SVM has given a better performance than the random guess classifier. All projects combined, ROSE with undersampling were able to give better results with higher accuracy along with 5% of parents predicted with the rate of 21% of prediction. For the DNN project, ROSE with undersampling performed better than others. For the XD project, SMOTE performed better than others. For the APSTUD project, ROSE with Both-sampling and ROSE with Undersampling both have the same results and performed better than others. If we

consider the F1 score and ROC AUC, then the SVM has performed better then majority class classifiers in most of the cases.

Table 7. SVM results with data sampling techniques

Algorithm	**Sampling strategy**	**Project**	**Method**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1 score**
SVM	SMOTE	DNN	Polynomial	0.7045	0.125	0.0741	0.4547	0.093
		XD	Radial	0.8248	0.5294	0.0638	0.6803	0.1138
		APSTUD	Polynomial	0.961	0.2	0.3333	0.5933	0.2499
		ALL	Radial	0.6971	0.4687	0.0450	0.4861	0.0821
SVM	Both-sampling	DNN	Radial	0.5	0.5625	0.1323	0.5269	0.2142
		XD	Radial	0.7797	0.5294	0.0508	0.6573	0.0927
		APSTUD	Radial	0.9805	0.4	1	0.7	0.5714
		ALL	Linear	0.8505	0.2187	0.0476	0.5442	0.0781
SVM	Undersampling	DNN	Polynomial	0.8182	0.125	0.1666	0.5194	0.1428
		XD	Radial	0.7059	0.6470	0.0458	0.6771	0.0855

		APSTUD	Linear	0.9805	0.4	1	0.7	0.5714
		ALL	Linear	0.8644	0.2187	0.0530	0.5512	0.085323
SVM	Oversampling	DNN	Polynomial	0.5606	0.5625	0.15	0.5614	0.2368
		XD	Radial	0.791	0.5294	0.0535	0.663	0.0971
		APSTUD	Linear	0.947	0.4	0.6666	0.6966	0.4999
		ALL	Radial	0.641	0.5625	0.0448	0.6029	0.0829

6. Discussion

In this chapter, we discuss the results in order to answer the research questions.

6.1 Answering the research questions

The first research question aims to identify which algorithm would give better prediction results. It also points out that whether it is better to make predictions on one project at a time or combined projects. (RQ1 – What is the performance of predicting the splitting of user stories using information extracted from issue reports?)

From the results, we can see that when we trained each project separately, the SVM has given the best results with the project APSTUD. Table 7 illustrates that the SVM has performed with the accuracy of 94% to 98% using the ROSE data imbalancing technique and 96% using the SMOTE data imbalancing technique with the APSTUD project. However, when all projects combined, we can see that the SVM has given the accuracy of 64% to 86% using ROSE and 69% using SMOTE. Whereas, the highest accuracy of 94% was achieved with Random Forest when all projects combined (Table 6).

Table 6 shows that the Random Forest has also performed well with the APSTUD project. Using ROSE with Oversampling the accuracy is 94% and with Both-sampling the accuracy is 92%, and with SMOTE it was also 92%. Also, XD projects have achieved the highest accuracy of 94% with the Random Forest using the SMOTE data imbalancing technique. Also with ROSE, the XD project accuracy is 75% to 87% which is higher than SVM.

In conclusion, when all projects were trained together then the Random Forest reached the highest accuracy with the SMOTE data imbalancing technique and when each project trained separately then the highest accuracy was achieved using SVM with the APSTUD project. Moreover, the XD project from Table 6 and APSTUD project from Table 7, has reached better accuracy when trained separately. So, it's safe to say that we can achieve better prediction results when training each project separately because 2 out of 3 projects have performed better when trained separately. The APSTUD project has performed well with SVM and Random Forest and it has also performed better than XD and DNN project.

The second research question aims to identify which data imbalance technique would give better results when splitting user stories based on the results presented in this thesis. (RQ2 – What is the best strategy to address the data imbalance when splitting user stories?)

From the results, we can see that the highest accuracy of 98% was achieved while using ROSE Both-sampling and Undersampling techniques. With data imbalancing technique SMOTE, the number of predicting parents is far higher with the accuracy of 96% in the APSTUD project. With data imbalancing technique ROSE with both-sampling, undersampling, and oversampling, the number of predicting parents is far higher with the accuracy of 98% in the APSTUD project. Using the SVM algorithm, we can see that the ROSE has performed better than the SMOTE technique in the APSTUD project.

From table 4, we can see that the ROSE and SMOTE was having the same accuracy in XD and DNN projects. The APSTUD project performed better with ROSE. The projects combined as one performed better with SMOTE.

From table 5, we can see that the SMOTE technique has performed better than ROSE in all scenarios. The XD, DNN, and APSTUD projects separately and altogether have better accuracy with SMOTE. Therefore, SMOTE is working better with the KNN algorithm.

From table 6, we can see that ROSE has performed better in the APSTUD project only but the SMOTE technique has performed better in XD, DNN, and all projects combined. So, in this case, the SMOTE technique was better than ROSE using the Random Forest algorithm because it was able to give better accuracy in more scenarios.

From table 7, we can see that ROSE has performed better in DNN and APSTUD projects and all the projects combined as one. In the case of the XD project, the SMOTE technique has performed better than ROSE. Overall the ROSE was able to give better accuracy in more scenarios.

In the case of all projects combined, the SMOTE has performed better with three of the machine learning algorithms Decision tree, KNN, and Random Forest. Whereas, the ROSE was able to perform better in the only SVM. Separately, the APSTUD projects have performed better with three algorithms using ROSE and the XD projects have performed better with three algorithms using SMOTE. Overall, we can say that the SMOTE technique is more likely to give better results. However, the highest accuracy was achieved by using the ROSE technique.

6.1 Limitations

In this thesis, there are threats to validity: finding open source projects, imbalanced datasets, and missing data. Which needs to be considered cautiously. We used the issue reports of open source projects to reduce these threats. To eliminate these threats, we used performance classifiers like random guess and majority class classifiers, normally used for calculating the performance of the predictive models.

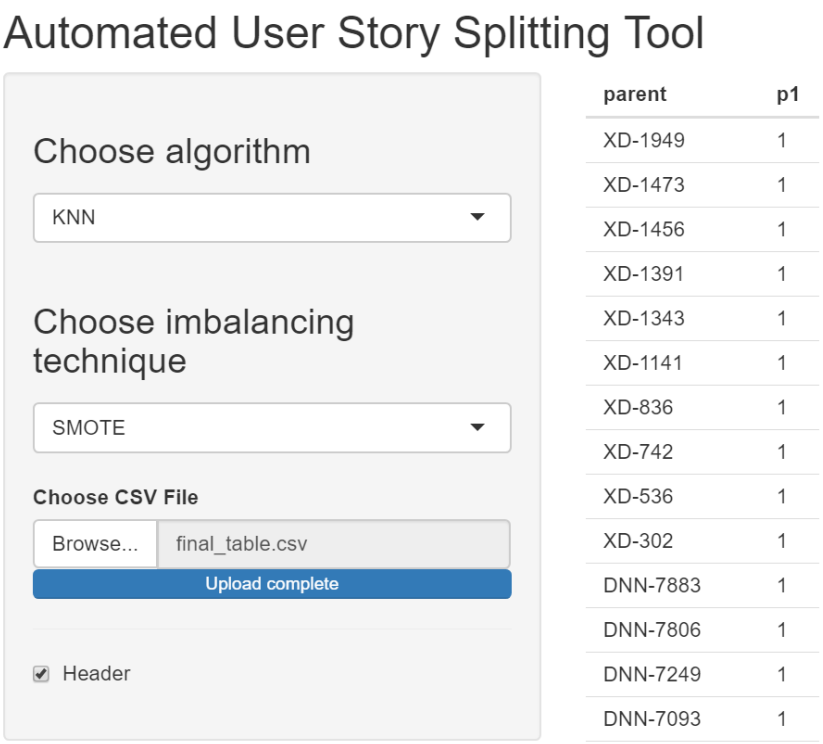
Find an open-source project with a good ratio of 0 and 1 (parent or not) and it was proven tough to find a good project. The dataset was big, which makes it hard to evaluate each issue report separately. It means there is a possibility of a problem with data cleaning, which could affect the results. Due to missing data, the records are removed from the dataset. Which could cause the loss of some important data. To manage this threat imputing strategy was used.

To reduce the data imbalancing, we used two different data imbalancing techniques SMOTE (Synthetic Minority Oversampling Technique) and ROSE (randomly over sampling examples). SMOTE uses feature space and nearest neighbors to create a balanced dataset and ROSE uses a synthetic balanced samples approach. ROSE all three resampling techniques were used to ensure the threat's elimination.

6.2 Tool Implementation

In this thesis, we are also presenting a prototype. This prototype illustrates the concept of an automated user story splitting tool. The tool has three input fields has shown in figure 5. The prototype was developed using Shiny. It is an open-source package of R. It is a web framework and it uses HTML, CSS, or JavaScript for developing web applications.
First, the user will select a supervised machine learning algorithm and then a data imbalancing technique. Secondly, the user will upload a CSV file with user stories and their features. Then the tool will apply the data cleaning process to ensure there are no NA's. Once the data cleaning is done then the tool will apply the data imbalancing technique which was selected by the user. Then the machine learning algorithm will train the model and start making predictions. Then the tool will recommend the list of user stories (predicted) for splitting. In this prototype, only SMOTE technique is implemented with all four algorithms as the SMOTE has performed better than the ROSE.

Figure 5. Automated user story splitting prototype



7. Conclusion and future work

In this thesis, we have used three different projects from Jira and have applied four machine learning algorithms separately on these projects, then we have combined all three projects into one dataset and applied four machine learning algorithms.

All three projects and all are combined into one dataset and have performed better than both baseline classifiers. The majority class classifier was not even able to predict one parent, so there is nothing to compare with it.

DNN project has performed best-using data imbalancing techniques ROSE with both-sampling and SMOTE with the decision tree (ctree). In this scenario, the accuracy is 0.8106% with a precision of 0.1875 and a recall rate of 0.2%. If we compare this result with other data imbalancing techniques and with other machine learning algorithms, then this result is the finest but overall the precision and recall rate is very low.

XD project has performed best-using data imbalancing technique SMOTE with random forest. In this scenario, the accuracy is 0.9424% with a precision of 0.2352 and a recall rate of 0.1081%. If we compare this result with other data imbalancing techniques and with other machine learning algorithms, then this result is the finest but overall the precision and recall rate is very low.

APSTUD project has performed best using data imbalancing techniques ROSE with Both-sampling and ROSE with Undersampling with SVM. In this scenario, the accuracy is 0.9805% with a precision of 0.4 and a recall rate of 100%. If we compare this result with other data imbalancing techniques and with other machine learning algorithms, then this result is the finest but overall the precision is very low but the accuracy and recall rate is really good.

When all three projects were combined into one dataset, then they performed best using the data imbalancing technique SMOTE with random forest. In this scenario, the accuracy is 0.9421% with a precision of 0.1250% and a recall rate of 0.1%. If we compare this result with other data imbalancing techniques and with other machine learning algorithms, then this result is the finest but overall the precision and recall rate is very low.

For future work, the tool can have interaction with a developer. For example, when the tool makes a recommendation that a story needs to be split then the developer can decide yes or no. Based on the developer's feedback the tool can improve its performance.

8. References:

1. Coughlan, J. and Macredie, R.D., 2002. Effective communication in requirements elicitation: a comparison of methodologies. Requirements Engineering, 7(2), pp.47-60.
2. Liskin, O., Pham, R., Kiesling, S. and Schneider, K., 2014, May. Why we need a granularity concept for user stories. In International Conference on Agile Software Development (pp. 110-125). Springer, Cham.
3. Coelho, E. and Basu, A., 2012. Effort estimation in agile software development using story points. International Journal of Applied Information Systems (IJAIS), 3(7).
4. Cohn, M., 2004. User stories applied: For agile software development. Addison-Wesley Professional.
5. Lawrence, R., 2009. Patterns for Splitting User Stories. URL: http://agileforall.com/patterns-for-splitting-user-stories.
6. Taibi, D., Lenarduzzi, V., Janes, A., Liukkunen, K. and Ahmad, M.O., 2017, May. Comparing requirements decomposition within the scrum, scrum with kanban, xp, and banana development processes. In International Conference on Agile Software Development (pp. 68-83). Springer, Cham.
7. Ezequiel Scott. Jira Extractor. from https://github.com/ezequielscott/jiraextractor
8. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E. and Brinkkemper, S., 2016, November. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In International Conference on Conceptual Modeling (pp. 463-478). Springer, Cham.
9. Gysel, M., Kölbener, L., Giersche, W. and Zimmermann, O., 2016, September. Service cutter: A systematic approach to service decomposition. In European Conference on Service-Oriented and Cloud Computing (pp. 185-200). Springer, Cham.
10. Wake, B., 2015. INVEST in good stories, and SMART tasks (2003). URL https://xp123.com/articles/invest-in-good-stories-and-smart-tasks
11. Liskin, O., Schneider, K., Fagerholm, F. and Münch, J., 2014, June. Understanding the role of requirements artifacts in kanban. In Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering (pp. 56-63).

9. Appendix

I. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Muhammad Bilal Shahid

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Splitting User Stories Using Supervised Machine Learning

supervised by Ezequiel Scott.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Muhammad Bilal Shahid

10/08/2020