FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Assessing Risks in Software Projects Through Machine Learning Approaches

André Oliveira Sousa

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado em Engenharia de Software

Supervisor: João Carlos Pascoal Faria

Second Supervisor: João Pedro Carvalho Leal Mendes Moreira

July 28, 2021

Assessing Risks in Software Projects Through Machine Learning Approaches

André Oliveira Sousa

Mestrado em Engenharia de Software

Approved in oral examination by the committee:

July 28, 2021

Abstract

Risk management is one of the ten fields discussed in the Project Management Body of Knowledge (PMBOK), which serves as a guide that should be followed to increase the chances of successful project development and implementation. The popularity of research regarding the application of risk management in software projects has been consistently growing in recent years, particularly with the use of machine learning techniques to help identify risk levels or risk factors of a project before the project development begins, with the intent of improving the likelihood of successful development of software projects.

The main goal of the work that will be presented in this report was the development of a software module that can accurately predict the risks of a new software project before its development begins through the use of machine learning techniques, as part of a larger scale project. This project is the PROject ManagEment Intelligent aSSistAnt, or PROMESSA for short, with the goal of assisting in tasks such as project prioritization and optimization, recommendation of resources, and risk management. By using 140 risk items of 18 past projects of one of PROMESSA's partner companies (Strongstep), the models were trained to learn the relations between the characteristics of those projects and the impact and likelihood of occurrence of risks in various categories related to project development and management. After the models were trained, the software module developed was integrated with the main PROMESSA module, which is then used by Strongstep's project management software (SCRAIM) so that it can receive the various risk impact and likelihood predictions that are performed as a result of the creation of a new project in that project management tool.

The machine learning tasks were performed using the Scikit-learn, Numpy, and Pandas packages in Python. The Numpy and Pandas packages provided helpful utility functions for common tasks needed such as loading the data set and extracting the necessary variables, whereas Scikit-learn was used to create the machine learning models and to perform data preparation tasks prior to the models' training such as encoding and normalization. Additional machine learning-related packages were used, such as ELI5 and Yellowbrick, to analyse different aspects of the training process, such as which features the models were giving higher importance to, or the evolution of the models' accuracy as more data was used in training, among others.

Hyperparameterization was used to obtain the best set of parameters possible for various classification algorithms, which were then tested with the use of cross-validation, and whose results were compared and analysed in greater detail. Another approach taken to try to maximize the predictive performance of the algorithms tested was the conversion of the problem from a multi-class classification problem to a binary classification one, following a well-known approach in this research area.

The results obtained in the tests for various algorithms using both standard multi-class classification and the conversion to binary classification showed that multi-class classification using Support Vector Machine and Naive Bayes were the best options for this problem and were consequently selected for the risk impact and risk likelihood models, respectively.

The tests showed that the Support Vector Machine algorithm had an accuracy of 69%, AUC score of 0.69, F-Measure of 0.64, precision of 0.60, and recall of 0.69, the highest values found for all evaluation metrics for the risk impact target variable except for AUC, which was 0.03 lower than the best performing algorithm in this metric (Neural Network). Additionally, the Naive Bayes model trained with oversampling applied to the training data returned the highest scores for accuracy (63%), AUC (0.62, tied with Random Forest), F-Measure (0.59), precision (0.62), and recall (0.63) for the risk likelihood target variable.

Keywords: Risk Management, Risk Assessment, Software Projects, Machine Learning, Classification

Resumo

A gestão de riscos é um dos dez campos apresentados no *Project Management Body of Knowledge* (PMBOK), um guia que deve ser seguido de modo a aumentar a probabilidade de sucesso do desenvolvimento e implementação de projetos. A popularidade de investigação relacionada com a aplicação de conceitos de gestão de riscos em projetos de *software* tem crescido de forma consistente nos últimos anos, em particular quando associada à aplicação de técnicas de aprendizagem computacional para ajudar a identificar fatores ou níveis de risco de um projeto antes do início do seu desenvolvimento, com o objetivo de aumentar a probabilidade de sucesso de desenvolvimento de projetos de *software*.

O objetivo principal do trabalho que será apresentado neste relatório foi o desenvolvimento de um módulo de *software* que consiga prever com previsão os riscos de um novo projeto de *software* antes do início do seu desenvolvimento através do uso de técnicas de *machine learning*, como parte de um projeto de grande escala. Este projeto é o PROject ManagEment Intelligent aSSistAnt, ou PROMESSA, que tem como alguns dos seus objetivos a priorização e otimização de projetos, recomendação de alocação de recursos humanos a tarefas, e gestão de riscos num projeto. Ao utilizar 140 itens de risco de 18 projetos anteriores de uma das empresas pareceiras do projeto PROMESSA, a Strongstep, os modelos foram treinados de modo a conseguirem aprender as relações entre as características desses projetos e o impacto e a probabilidade de ocorrência de riscos em várias categorias relacionadas com o desenvolvimento e gestão de projetos. Após o treino do modelo, a aplicação desenvolvida foi integrada com o módulo principal do projeto PROMESSA, que é utilizado pelo *software* de gestão de projetos da Strongstep, o SCRAIM, para que este possa receber as previsões de impacto e probabilidade de risco que são obtidas após a criação de um novo projeto nessa ferramenta de gestão de projetos.

As tarefas de *machine learning* foram realizadas com o uso das bibliotecas Scikit-learn, Numpy, e Pandas para a linguagem Python. As bibliotecas Numpy e Pandas forneceram métodos úteis para tarefas comuns que eram necessárias, tais como carregar o conjunto de dados e extrair as variáveis necessárias, enquanto a biblioetca Scikit-learn foi usada para criar os modelos de *machine learning* e realizar as tarefas de preparação dos dados antes do treino dos modelos, tais como *encoding* e normalização dos dados. Outras bibliotecas relacionadas com *machine learning* foram utilizadas, como foi o caso das bibliotecas ELI5 e Yellowbrick, com o objetivo de analisar diferentes aspetos do processo de treino dos modelos, tais como perceber a que atributos os modelos atribuíam maior importância, ou a evolução da precisão dos modelos à medida que a quantidade de dados utlizada para treino aumentava, entre outros.

O processo de hiperparameterização foi utilizado para obter o melhor conjunto de parâmetros para treino de diferentes algoritmos de classificação, algoritmos esses que foram posteriormente testados através do recurso a *cross-validation*, e cujos resultados foram comparados e analisados em maior detalhe. Outra abordagem utilizada para tentar maximizar os resultados dos algoritmos testados foi a conversão do problema a partir do problema original de classificação multi-classe para um problema de classificação binária, seguindo uma abordagem comum nesta área de investigação.

Os resultados obtidos nos testes com os diferentes algoritmos utilizando ambas as abordagens de classificação multi-classe e conversão praa classificação binária mostraram que a classificação multi-classe utilizando os algoritmos *Support Vector Machine* e *Naive Bayes* eram as melhores escolhas para este problema, e foram consequentemente escolhidos para os modelos de previsão de impacto e probabilidade de risco, respetivamente.

Os testes realizados mostraram que o algorimo *Support Vector Machine* tinha uma *accuracy* de 69%, *AUC score* de 0.69, *F-Measure* de 0.64, *precision* de 0.60, e *recall* de 0.69, sendo estes os valores mais altos encontrados para todas as métricas no modelo de previsão de impacto de risco, exceto no caso de *AUC score*, cujo valor está 0.03 abaixo do melhor algoritmo nesta métrica (*Neural Network*). Adicionalmente, o algoritmo *Naive Bayes* treinado com *oversampling* aplicado aos dados de treino obteve os melhores resultados para *accuracy* (63%), *AUC score* (0.62, obtido também por *Random Forest*), *F-Measure* (0.59), *precision* (0.62), e recall (0.63) no modelo de previsão de probabilidade de risco.

Keywords: Gestão de Riscos, Avaliação de Riscos, Projetos de *Software*, Aprendizagem Computacional, Classificação

Acknowledgements

First, I would like to thank my dissertation supervisors, Prof. João Carlos Pascoal Faria and Prof. João Pedro Carvalho Leal Mendes Moreira, for their continued support and guidance during the course of this dissertation.

Secondly, I would like to thank Duarte Gomes of Strongstep and the rest of the PROMESSA project team for the help and incredible availability during the development of this project.

A word of gratitude also goes out to all my professors and colleagues at FEUP who, in one way or another, had an impact on the completion of this master's degree.

Lastly, a very special thanks to all family and friends who were always there for me in these past two years, without whom it would have been impossible to achieve this lifelong goal of mine. In particular, I would like to thank my aunt Eduarda for her ever-growing interest, support and motivation from day one of this journey.

André Oliveira Sousa

“How does a large software project get to be one year late?
One day at a time!”

Frederick Phillips Brooks Jr.

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Problem . . . 2
1.3 Objectives and Contributions . . . 3
1.4 Methodology . . . 4
1.5 Document Structure . . . 5

2 Background and State of the Art **7**
2.1 Risk in Software Projects . . . 7
2.2 Risk Management in Software Projects . . . 11
2.3 Machine Learning . . . 15
2.3.1 Neural Network . . . 16
2.3.2 Support Vector Machine . . . 17
2.3.3 Naive Bayes / Bayesian Network . . . 18
2.3.4 Logistic Regression . . . 19
2.3.5 Decision Tree . . . 19
2.3.6 Random Forest . . . 20
2.3.7 Evaluation Metrics . . . 20
2.3.8 Oversampling and Undersampling . . . 24
2.4 Machine Learning Approaches in Risk Management . . . 24
2.4.1 Machine Learning Practices in Risk Management . . . 25
2.4.2 Literature Review of Machine Learning Approaches in Risk Management 26
2.5 Summary . . . 31

3 Data Preparation **33**
3.1 Data Set . . . 33
3.2 Data Preprocessing . . . 35
3.3 Summary . . . 40

4 Solution Development **41**
4.1 Technologies . . . 41
4.2 Hyperparameter Optimization . . . 43
4.3 Solution Workflow . . . 45
4.4 Converting a Multi-Class Classification Problem to a Binary Classification Problem 51
4.5 Experimental Setup . . . 53
4.6 Summary . . . 53

5 Results and Discussion **55**
5.1 Multi-class Classification Results 55
5.1.1 Risk Impact 55
5.1.2 Risk Likelihood 59
5.2 Binary Classification Results 62
5.2.1 Risk Impact 62
5.2.2 Risk Likelihood 65
5.3 Discussion 68
5.4 Summary 69

6 Conclusions and Future Work **71**
6.1 Conclusions 71
6.2 Future Work 72
6.2.1 Provide More Specific Information 72
6.2.2 Risk Treatment and Mitigation Action Classifier 73
6.2.3 Facilitate Integration with More Data Sources 73
6.2.4 Workflow Automation 74

References **75**

List of Figures

1.1 CRISP-DM phases and respective tasks 4

2.1 SEI's Project Risk Management Paradigm [63] 11
2.2 Mind map of concepts related to risk and risk management 15
2.3 Example of a neural network with three layers 17
2.4 Graphical illustration of a Support Vector Machine [26] 18
2.5 Example of how a simple decision tree works 19
2.6 Example of an AUC graph 22
2.7 Mind map of machine learning approaches for risk management 30

3.1 Risk impact class distribution 37
3.2 Risk impact level distribution per risk category 37
3.3 Risk likelihood class distribution 38
3.4 Risk likelihood level distribution per risk category 38
3.5 Project manager profile (left) and number of team members distribution (right) . 39

4.1 Feature ranking for the risk impact model showing the most and least important features during training 42
4.2 Learning curve for the risk impact model showing the evolution of training and cross-validation scores as the number of training instances grows 43
4.3 Grid search parameterization script output example 45
4.4 Randomized search parameterization script output example 45
4.5 Standalone REST API documentation example 46
4.6 Deployment diagram explaining the PROMESSA project architecture 47
4.7 Example of an estimator tree from the Random Forest algorithm 50
4.8 Risk impact class distribution comparison: original (left), class distribution used for the first classifier (centre), class distribution used for the second classifier (right) 52

5.1 Example of a confusion matrix of predictions made by the risk impact model (Support Vector Machine) 56
5.2 Example of a confusion matrix of predictions made by the risk impact model (Support Vector Machine) with oversampling applied to the training data 57
5.3 Risk impact class distribution without (left) and with oversampling applied to the training data (right) 58
5.4 Example of a confusion matrix of predictions made by the risk likelihood model (Naive Bayes) 59
5.5 Example of a confusion matrix of predictions made by the risk likelihood model (Naive Bayes) with oversampling applied to the training data 60

5.6 Risk likelihood class distribution without (left) and with oversampling applied to the training data (right) . . . 61
5.7 Example of a confusion matrix of predictions made by the risk impact model using binary classification conversion (K-Nearest Neighbours) . . . 62
5.8 Example of a confusion matrix of predictions made by the risk impact model (K-Nearest Neighbours) with oversampling applied to the training data and using binary classification . . . 64
5.9 Risk impact class distribution without (left) and with oversampling applied to the training data (right) after the conversion to a binary distribution is performed . . . 64
5.10 Example of a confusion matrix of predictions made by the risk likelihood model using binary classification conversion (Random Forest) . . . 66
5.11 Example of a confusion matrix of predictions made by the risk likelihood model (Support Vector Machine) with oversampling applied to the training data and using binary classification conversion . . . 67
5.12 Risk likelihood class distribution without (left) and with oversampling applied to the training data (right) after the conversion to a binary distribution is performed . 67

List of Tables

1.1 Results of The Standish Group's CHAOS Report [31] 2

2.1 Boehm's top ten software risk items [18] 9
2.2 Wallace's classification of software risk factors in different project dimensions [59] 10
2.3 Risk priorities according to their probability and impact 10
2.4 Risk management steps and commonly used techniques [18] 14
2.5 Confusion Matrix Example 20
2.6 Studies and experiments on the use of machine learning techniques for risk assessment 28

3.1 Example of the Project Type attribute using one-hot encoding 36

4.1 Evaluation metrics results for the different algorithms tested (I = Impact, L = Likelihood) 49

5.1 Results of the risk impact model evaluation with different algorithms 56
5.2 Comparison of the risk impact model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data 57
5.3 Results of the risk likelihood model evaluation with different algorithms 59
5.4 Comparison of the risk likelihood model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data 60
5.5 Results of the risk impact model evaluation with different algorithms using binary classification conversion 62
5.6 Comparison of the risk impact model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data and using binary classification conversion 63
5.7 Results of the risk likelihood model evaluation with different algorithms using binary classification conversion 65
5.8 Comparison of the risk likelihood model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data and using binary classification conversion 66
5.9 Results and parameters used of the algorithms selected for the risk impact and risk likelihood models 69

Abbreviations

API	Application Programming Interface
AUC	Area Under the ROC Curve
BBN	Bayesian Belief Network
BMMRE	Balanced Mean Magnitude of Relative Error
CMM	Capability Maturity Model
CRISP-DM	Cross Industry Standard Process for Data Mining
CV	Cross-Validation
DAG	Directed Acyclic Graph
FNR	False Negative Rate
FPR	False Positive Rate
MAE	Mean Absolute Error
ML	Machine Learning
MMRE	Mean Magnitude of Relative Error
MSE	Mean Squared Error
NN	Neural Network
PMBOK	Project Management Body of Knowledge
RE	Risk Exposure
REST	Representational State Transfer
ROC	Receiver Operating Characteristic
SDLC	Software Development Life Cycle
SEI	Software Engineering Institute
SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machine
TNR	True Negative Rate
TPR	True Positive Rate
TRM	Team Risk Management
WSGI	Web Service Gateway Interface

Chapter 1

Introduction

This chapter presents the contextualization of the work that was done by listing a few details regarding the success and failure rate of software development projects in section 1.1, the problem which the work developed targets in section 1.2, the objectives related to the work developed in section 1.3, and the methodology chosen for the development of the software in section 1.4. Lastly, the structure of the remaining document is presented in section 1.5.

1.1 Context

According to the Project Management Body of Knowledge, a project risk is "an uncertain event which, if it occurs, has a positive or negative effect on one or more project objectives" [51]. Software projects are notoriously complex development activities, and thus the concept of risk cannot be ignored when considering this type of projects. Software projects can encounter various types of risks, such as technical, management, financial, contractual/legal, personnel, and related to other resources [62].

In 2015, the Standish Group International's CHAOS Report [31], a study of the success of software projects throughout the year, reported a 29% success rate for the roughly 5000 projects investigated. A project is considered successful if it is completed within its allocated budget, its original delivery deadline, and with all the features that were planned at the start of its development life cycle [34]. The same study reported a 19% failure rate for the set of projects investigated, meaning the projects suffered from cost overruns, time overruns, or lacking content that was initially specified.

However, it is in the category of challenged projects that we can find the largest percentage of projects. As can be seen in table 1.1, from 2011 to 2015, 49%, 56%, 50%, 55%, and 52% of the software projects, respectively, were considered challenged, meaning they were completed but either over-budget, over the allocated time estimates, or offering fewer features than originally planned. While there has always been a larger percentage of successful projects compared to the percentage of failed ones in this time period, the extremely high percentage of challenged projects in the same time period indicates that there is definitely a possibility to improve the success rate of a large amount of these projects. This data highlights the notorious complexity for which software development projects are known, and why it is important to increase the number of successful projects in the software development industry. This is where the concept of risk, and more importantly, risk management is important to consider.

Table 1.1: Results of The Standish Group's CHAOS Report [31]

	2011	**2012**	**2013**	**2014**	**2015**
Challenged projects	49%	56%	50%	55%	52%
Successful projects	29%	27%	31%	28%	29%
Failed projects	22%	17%	19%	17%	19%

Risk management is a process used for early identification, analysis, planning, and control of risks in a project [19], with the goal of minimizing negative risks and maximizing positive risks [23], also referred to as opportunities. By identifying the risks and creating mitigation plans to deal with them if or when they occur before the project development starts rather than dealing with these problems on-the-fly and coming up with strategies to deal with risks in the moment they materialize, project managers and development teams are better prepared to deal with the risks and their effects on a project, which in turn can lead to more projects being completed on time and within their allocated budgets. Risk management is typically the first activity to be removed from the project management activities when a project falls behind schedule [43] and is seen by many professionals in the industry (developers and managers alike) as an inhibitor to creativity [43].

In recent years, there has been an increasing use of machine learning algorithms and techniques for risk assessment, particularly supervised learning ones where the model is trained using a data set, and then the same model is used to predict information on a new set of data (in this area, it could be to predict possible risk factors of a project based on its characteristics, such as time and allocated budget). Commonly used algorithms for supervised learning tasks include Decision Trees, Naive Bayes classifiers (NB), Neural Networks, and Support Vector Machines (SVM). This is a field that can have a great effect on changing the previously mentioned opinion of professionals in the area regarding risk management, as it provides a completely different way of predicting, assessing, and managing risks in a software project when compared to traditional risk management models and processes.

1.2 Problem

Identifying and preparing mitigation strategies to deal with the possible risks in a software project is an important task to reduce the chances of failure during the development of a software project. More often than not this is done when the development of the project is already underway, when it should instead be performed before development begins. This way, the team gains more time to prepare mitigation plans to deal with risks that might happen as the project is being developed and does not become overwhelmed with the risks if they occur.

However, the process of identifying risks, their probability of occurrence, and their possible impact before development begins is not an easy task. After all, there is already so much to consider before developing a new software project in an organizational environment, and some factors will inevitably be overlooked.

Regardless, the process of risk management is one of the fundamental knowledge areas for project management and should not be overlooked. Traditional models and processes to implement risk management already exist, but with the shift in project management to use more and more software-based management tools, the process of integrating risk management activities into these tools is not always an easy one.

1.3 Objectives and Contributions

Considering the problem described in the previous section, the main objective of this project was to develop a software module aimed at predicting the impact and likelihood of risks of a new software project before development begins through the use of machine learning techniques. The module is part of a larger project: the PROject ManagEment Intelligent aSSistAnt, or PROMESSA for short, which will aid in tasks such as project prioritization, optimization and recommendation of resources, delivery date estimates, and analysis of risk and their related management actions through the use of machine learning techniques. The software module described in this document tackles the risk management tasks of this project.

The module should present the possible impact and likelihood of occurrence of risks in various risk categories, so that the data presented can be analysed and used for the preparation of risk treatment actions and risk mitigation plans to deal with the common risks in the categories which are predicted to have higher levels of impact and probability of occurrence.

The machine learning models in this software module are trained using historical data of past projects from the project's partner company's project management tool: SCRAIM, from Strongstep. The prediction of the impact and likelihood of risks in different risk categories is done based on characteristics of a new project that is created in SCRAIM, such as the number of team members, duration of the project, allocated budget, type of project, among others.

The software module is integrated with the main PROMESSA module so that it can receive the information of a new project when one is created on SCRAIM, predict the impact and likelihood of risks in each category based on the information received, and return the results back to SCRAIM so that they can be analysed by the project manager.

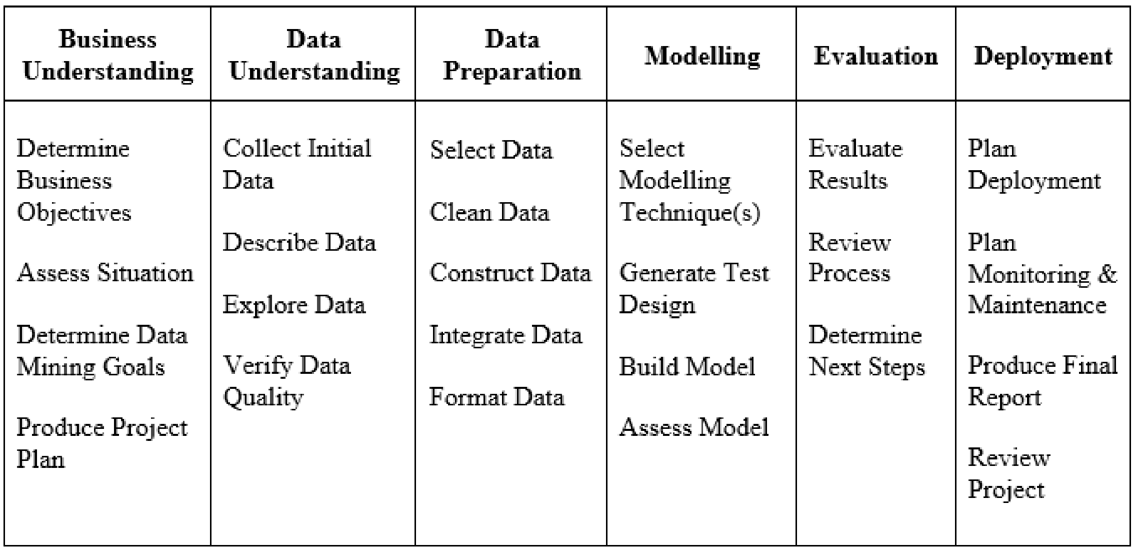
Lastly, the work developed resulted in two articles for software engineering conferences. The first one focuses on the state of the art of this research area, and was published on The 33rd International Conference on Software Engineering & Knowledge Engineering [56]. The second article created presents the results obtained in this project, and is currently under review in the 1st International Workshop on Machine Learning in Software Engineering.

1.4 Methodology

Data mining methodologies are commonly used in this type of projects to encourage the use of best practices in the industry with the goal of obtaining the best results possible. For this project, **CRISP-DM**, which stands for Cross-Industry Standard Process for Data Mining, was used with those same goals in mind.

CRISP-DM presents various phases that should be covered to improve the results obtained in the project, where each phase is made up of several tasks, as can be seen in figure 1.1. CRISP-DM provides a logical order for these phases and their respective tasks, but they can be swapped or performed multiple times if necessary. The CRISP-DM methodology assigns six phases to a data mining process [64]:

- **Business Understanding** - here the goal is to understand the business requirements and objectives for the project, before creating an initial plan to achieve those objectives. For this work, business understanding includes understanding and analysing the domain to which it belongs. This was done by reviewing the state-of-the-art of this research area, which is presented in Chapter 2, specifically sections 2.1 and 2.2.
- **Data Understanding** - in the Data Understanding phase, data collection tasks are started and activities to understand the data are performed, such as the data description and exploration that is presented in Chapter 3. This is a phase that is tied closely to Business Understanding, as detailing the objectives of the project should be done with some knowledge of the data in order to identify what can and cannot be done, or what information is easier or more difficult to extract from the data available.



Business Understanding	Data Understanding	Data Preparation	Modelling	Evaluation	Deployment
Determine Business Objectives Assess Situation Determine Data Mining Goals Produce Project Plan	Collect Initial Data Describe Data Explore Data Verify Data Quality	Select Data Clean Data Construct Data Integrate Data Format Data	Select Modelling Technique(s) Generate Test Design Build Model Assess Model	Evaluate Results Review Process Determine Next Steps	Plan Deployment Plan Monitoring & Maintenance Produce Final Report Review Project

Figure 1.1: CRISP-DM phases and respective tasks

- **Data Preparation** - in this phase, tasks to prepare the data to train the models are performed. This comes after verifying the quality of the data in the Data Understanding phase, where problematic factors such as missing or duplicate data can be identified. This phase includes tasks such as data cleaning, construction, or even integration in case more data becomes available. Tasks performed for this phase in the project are described in section 3.

- **Modelling** - in the Modelling phase, different models are created and tested with the processed data set until an optimal set of parameters is found. As previously mentioned, the phases in CRISP-DM can be performed multiple times if necessary, and the previous phase of Data Preparation will likely be repeated after the Modelling phase begins, as there might be a need for more tuning in the data set after the models are tested. Tasks performed for this phase in this project are described in section 4.3.
- **Evaluation** - in the Evaluation phase, a set of high quality models should be available for use, but the performance of the models should be evaluated more closely in order to assure that it matches the objectives identified in the Business Understanding phase. If a key objective has not been targeted by the models, then there is still the option to either return to the modelling phase to perform the changes needed to fix this problem or re-evaluate the list of objectives identified in the first phase. Tasks performed for this phase in the project are described in Chapter 5.
- **Deployment** - lastly, in the Deployment phase, the knowledge extracted from the models is prepared and delivered to the customer. In this case, the software module developed is prepared for integration with the main PROMESSA module. Tasks performed for this phase in this project are described in section 4.3.

1.5 Document Structure

This document begins with a review of the state-of-the-art in the key research areas related to the dissertation theme: risk management and machine learning in Chapter 2, and the application of machine learning techniques for risk assessment in Section 2.4. Chapter 3 lists the data used in this project and explains the steps taken to process and improve the quality of the data before it was ready for use. Chapter 4 details the creation and functionalities of the solution that was developed for the project, and Chapter 5 presents the results of the models created. Lastly, Chapter 6 presents the conclusions extracted based on the work developed for the Dissertation and the results obtained, and lists the next steps for future work in this project.

Chapter 2

Background and State of the Art

This chapter provides a review of the state-of-the-art of risk management and its application in software projects. Section 2.1 presents some definitions of risk by researchers in this area, various types of risks that can impact a software development project, and techniques that can be used to handle them. Section 2.2 presents the concept of risk management and frequently used traditional risk management models and processes. Section 2.3 presents various machine learning techniques and algorithms. Lastly, Section 2.4 presents a literature review of experiments and studies performed in the research area of machine learning techniques applied to risk assessment.

2.1 Risk in Software Projects

There are several definitions of risk and its relationship with software projects in the literature. The aforementioned Project Management Body of Knowledge defines a project risk as “an uncertain event which, if it occurs, has a positive or negative effect on one or more project objectives” [51]. Other definitions look at risk in software projects as:

- “The potential that a chosen action or activity will lead to a loss or an undesirable outcome” [23];
- “A set of factors or conditions that can pose a serious threat to the successful completion of a software project” [59];
- “The probability and impact of an event on a project” [18];
- “The possibility of suffering loss” [63].

From these definitions, it is possible to identify a few common concepts or themes, such as a risk possibly leading to a loss. In a software project, a loss can manifest itself through lower quality of the final product, increased costs, changes to the release date of the product, or, in a worst-case scenario, failure and cancellation [63]. While risk can be seen as an occurrence that negatively impacts the project, there is a positive counterpart to risk: opportunities. After identifying the events that can impact a project, one should assess the benefits and drawbacks of these events, and adequately mark the event as a risk to the project, or an opportunity that can lead to further gains on the project if it is explored. Essentially, a balance must be found between the negative consequences of risks and the potential gains from opportunities.

Risks as a whole can be categorized as known (identified after assessing the project plan and the market), unknown (difficult to identify in advance), and predicted (can be inferred from previous development or management experience) [23]. Software projects in particular can be impacted by various specific types of risks [62]:

- **Technical risks** - problems with the programming languages and frameworks of choice, project size, functionality, or processes. This type of risk can occur as a consequence of lack of experience, excessive constraints, or lack of maturity of the technologies being used.
- **Management risks** - these risks can occur as a result of problems with communication with top management and customers, lack of planning, or lack of project management experience.
- **Financial risks** - problems regarding budget, cash flow, or doubts about the return on investment of the project.
- **Contractual and legal risks** - problems regarding adjusting the schedule or the requirements to fit the market, government regulations, or health and safety problems.
- **Personnel risks** - these can be due to conflicts among staff, lack of experience and training, ethical and moral issues, or productivity issues resulting from a combination of the aforementioned risks.
- **Other resource risks** - these occur due to situations that are generally not a responsibility of the project team, such as unavailability of computer resources, unavailability of equipment and supplies, and delayed resolution of these problems by the responsible teams.

When it comes to the most frequent specific risks in software projects, Boehm, regarded by many as one of the most important authors in this research area, gathered the information presented in table 2.1 through a survey of experienced project managers.

The high percentage of challenged projects seen in the Standish Group International's CHAOS reports [31] is consistent with the information presented by Boehm [18], as the first and second risk items listed are directly related to the concept of challenged projects, and often come as a consequence of the majority of the remaining risk items listed occurring during the development of a project.

To reduce the high percentage of challenged projects in the software industry, project managers have to consider a wide variety of knowledge areas in order to manage their projects towards successful completion. One of those areas is risk management, as risks can be identified in various

Table 2.1: Boehm's top ten software risk items [18]

Risk item	**Risk management techniques**
Personnel shortfalls	Staffing with top talent, job matching
Unrealistic schedules and budgets	Incremental development, software reuse
Developing the wrong functions and properties	Prototyping, mission analysis
Developing the wrong user interface	Prototyping, scenarios
Gold-plating	Requirements scrubbing, prototyping
Continuing stream of requirements changes	Information hiding, incremental development
Shortfalls in externally furnished components	Benchmarking, inspections
Shortfalls in externally performed tasks	Reference checking, team-building
Real-time performance shortfalls	Simulation, benchmarking
Straining computer-science capabilities	Technical analysis, prototyping

areas of a software project. In a software development project, risks can be influenced by the business domain, the business style, culture of the organization, and characteristics of the members involved in the project [46], so it is important to identify risks according to the environment in which the project is being developed. To facilitate this process, risk factor and item classifications found in the literature can be used. These classifications usually list the most frequent risk items that can affect a project's path towards success, and teams can use these to evaluate if there is a possibility of any of those risks occurring during the development of their own projects, and if so, what could be their impact on the project. Essentially, those are the 2 parts that make up a risk: the likelihood of the risk happening, and the degree of impact it has on the project if it does occur.

Wallace's categorization [59] of risk items according to six risk dimensions, which is presented in Table 2.2, is still widely used in this research area to this day. Not only does it present common risk items in software projects, but by grouping them according to a specific dimension within the areas that the project managers have to consider in the development of a software project, it makes it so they can identify what areas are more likely to be problematic throughout the course of the development of the project and prepare their risk management strategies accordingly.

Those are just some examples of risks that can affect specific areas and have a negative impact on a software project. In reality, there are a lot more risks that can be identified in any project, so identifying these risks at an early stage of the project (ideally before development starts) is crucial for a successful development life cycle, as it means the project manager as well as the development team can start to think and plan actions to take if these risks materialize during the project development.

A good way of classifying the identified risks according to their priority is through the use of a portfolio chart, such as the one presented by Dr. Ernest Wallmüller [60], an example of which can be seen in table 2.3.

Table 2.2: Wallace's classification of software risk factors in different project dimensions [59]

Risk dimension	**Items**
Team	Conflict between developers Frequent turnover Team members unfamiliar with automated tasks Team members lacking required specialized skills Inadequately trained developers Lack of commitment to the project among developers Inexperienced members
Organizational Environment	Lack of top management support Change in management during the project Organization restructuring during the project Unstable organizational environment Organizational politics having a negative effect Reduction in resources available to the project
Requirements	Wrong, unclear, or conflicting system requirements Users lacking an understanding of the system's functionalities and limitations Undefined or unclear project success criteria Difficulty defining the inputs and outputs of the system
Planning and Control	Project milestones not clear Project progress not monitored well enough Inexperienced manager Poor project planning and management Poor communication Inadequate estimation of required resources and project schedule
User	Users not committed to the project or resistant to change Lack of user participation and cooperation Conflict among users Users resistant to changes in the project
Complexity	Project involves the use of technology that is new to the developers or the use of immature technology High level of complexity Large amount of external suppliers involved Complex tasks being automated

Risks in the A region are high priority risks and are those where their probability of occurrence is between medium and high, and their impact on the project is also between medium and high. Risks in the B region are considered medium priority risks and include risks with low probability but high degree of impact on the project, risks with both medium probability and impact, and risks with high probability of occurrence but rather low impact. Lastly, risks in the C region are low priority ones, and are those with a low to medium probability of occurrence and a low to medium level of impact.

Table 2.3: Risk priorities according to their probability and impact

Impact				
High	B	A	A	
Medium	C	B	A	
Low	C	C	B	
	Low	Medium	High	**Probability**

Identifying, classifying, and prioritizing actions for risks according to their priority are just some of the phases of a process called risk management, which is explained in greater detail in the next section.

2.2 Risk Management in Software Projects

Standard ISO/IEC/IEEE 24765:2010 defines risk management (in systems and software engineering) as "an organized process for" [27]:

- Identifying and handling risk factors;
- Assessing and quantifying the identified risks;
- Developing and, if necessary, implementing a plan to deal with the identified causes of risks.

This is consistent with various risk management models found in the literature, the majority of which include similar steps to identify, analyse, plan, and control risks in a software project. One example of this is the Software Engineering Institute's (SEI) Risk Management Paradigm [63], which defines six key steps for a successful risk management process: identification, analysis, planning, tracking, control, and communication, shown below in figure 2.1.

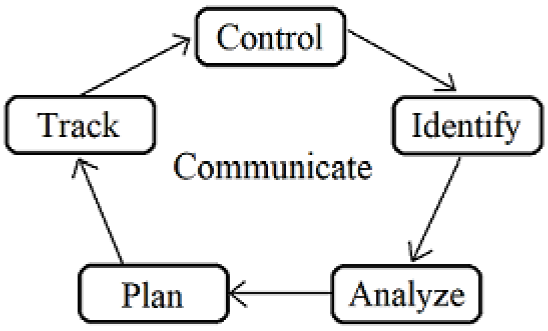


Figure 2.1: SEI's Project Risk Management Paradigm [63]

In the identification phase, the goal is to identify and make all project risks identified explicitly before the development of the project begins. Analysing the risks will make use of the risk information identified in the previous phase to start thinking about possible decisions to be taken if the risks occur. These decisions are then used in the planning phase, where risks should be prioritized and a risk management plan should be created, containing the strategies and mitigation plans to deal with the risks if they occur. Tracking a risk involves monitoring its status and deciding what action to take against the risk in order to mitigate it. There is no one-size-fits-all mitigation strategy, but rather a combination of risk mitigation strategies that will fit the risks identified. Boehm [19] identified several fundamental risk mitigation strategies, including:

- **Understanding** - a strategy where the focus is on gaining more insight into the problem. This can be done through prototyping to learn more about the requirements or buying commercial off-the-shelf products for interoperability.

- **Avoidance** - this is where actions are taken to remove the risk from the critical path of the project or to remove it from the project altogether.
- **Transfer** - this involves moving a risk from one party to another or sharing the risk exposure between more than one party.
- **Reduction** - here, actions are taken to reduce the risk exposure by either lowering the likelihood of occurrence or minimizing the degree of impact of the risk if it occurs.
- **Acceptance** - this is a mitigation strategy that is often used when the impact and probability of an identified risk is low enough that the project development can go ahead smoothly even if the risk occurs [19]. This is not the same as ignoring a risk (which should rarely, if ever, happen in any risk management plan), as an accepted risk still needs to at least be managed to guarantee that adequate responses will be prepared if its probability of occurrence or impact change in the future.

The control phase relies on the project manager's ability to control the risk action plans developed, respond to events as they occur, and continuously improve the risk management process as a whole. Risk management is not a process that takes place before the start of development and is over the moment development starts. It should be a part of every step of the Software Development Life Cycle (SDLC), as risks can very often be found in any of the SDLC phases. Lastly, communication is a big part of not only the SEI's Project Risk Management Paradigm, but ideally a big part of any risk management strategy that is employed. Effective communication makes it so the information obtained in all of the phases described is shared between the developers, as well as the project manager, making it so everyone is always aware of what risks have been identified and what plans exist to deal with them if they occur.

The SEI's Project Risk Management Paradigm is one frequently mentioned process, but there are several other traditional risk management models and processes that can be found in the literature. The authors in [49] analysed several risk management models and processes, such as:

- **Team Risk Management (TRM)** [33] - this is a technique that manages risks in the full software development life cycle, and involves all members and stakeholders (as opposed to the SEI risk management process, which only involves team members and not all stakeholders), improving the efficiency of the decision-making process. TRM frequently ensures continuous risk management through regular reviews and monitoring of the implemented processes.
- **Softrisk management technique** [53] - this technique is constructed on the basis of documentation and gives special focus to extreme risks by focusing on what can be leading to those risks. In this technique, both "general risks that can occur on any project", and "specific risks that can occur in software development projects are identified". Re-estimation, re-prioritization, re-assessment, and re-documentation are performed to also guarantee continuous risk management.

- **Riskit model** [41] - in this model, risks are ranked according to historical data, using visual representations to document them. As with TRM and the Softrisk management technique, this involves all the project's stakeholders, with the goal of reducing the causes and chance of failure of the project. Frequent risk monitoring and tracking are also used for continuous risk management purposes.

- **Project Risk Management framework** [44] - this framework goes through two key phases: risk assessment and risk control. In the assessment phase, analysis and prioritization steps are taken to categorize and rank the risks. These risks are then monitored in the risk control phase for continuous risk assessment.

- **Capability Maturity Model-based model** [49] - this model follows the Capability Maturity Model (CMM) framework for risk mitigation, keeping a risk repository for assessment and control purposes. It also goes through the risk assessment and risk control phases, with risk assessment being done first, specifically through the activities of risk identification and risk prioritization. Afterwards, in the risk control phase, mitigation plans are created, and risk monitoring is performed frequently through the software development life cycle, updating risks, and tracking new ones in the future.

- **Wallmüller's Risk Management Process** [60] - in this process, the risk management activities are conducted by the project team at the same points where the cost, time, quality, and requirement management activities are performed. It is a sequence-oriented process consisting of two parts: risk planning, which starts in the project planning and before development starts, and risk control, where risk avoidance, mitigation, and overall precautionary measures are implemented. A major point of difference in this approach compared to the ones that were previously mentioned is the introduction of risk management roles which are assigned to different members of the team, thus making sure the entire team contributes to the risk management tasks and is up-to-date on the status of risks in the project.

As was previously mentioned, risk can be split into two attributes: the probability that the risk occurs, and the impact that the risk has on the software project if it does occur, which can be used to calculate something that is commonly referred to as risk exposure (RE) [19].

$$RE = Probability(Loss) * Impact(Loss) \tag{2.1}$$

As seen in the previous section, risks can be classified according to their likelihood and impact if they happen to materialize during the project's development, usually expressed in monetary values. As an example, using a high priority risk with an 80% probability of occurrence and an impact of €250,000, the exposure for this risk is equal to:

$$RE = 0.80 * €250,000 = €200,000 \tag{2.2}$$

In a very simplified way, the goal of risk management is then to increase the probability of positive events on a software project, while at the same time decreasing the probability of negative events on the same project [19]. To do that, risk management is often divided into two key activities: risk assessment and risk control, which are composed of more specific steps.

In [18], Boehm splits risk assessment into three phases: identification, analysis, and prioritization. In the same publication, he also splits risk control into three phases: management planning, resolution, and monitoring. Other researchers may change the categories to which these steps belong to, such as in [38], where risk assessment is made up of phases for identification, analysis, prioritization, planning, and resolution, and risk control is made up of only one phase, which is monitoring, but the key concepts and phases remain the same, which are shown in table 2.4.

Table 2.4: Risk management steps and commonly used techniques [18]

Risk management step	**Commonly used techniques**
Risk identification	Checklists, decision-driven analysis, comparison with experience, decomposition
Risk analysis	Performance models, cost models, network analysis, statistical decision analysis, quality-factor analysis
Risk prioritization	Risk exposure analysis, risk reduction leverage analysis, group-consensus techniques
Risk management planning	Checklists of risk-resolution techniques, cost-benefit analysis, risk management plan outlines
Risk resolution	Prototypes, simulations, benchmarks, mission analysis, incremental development
Risk monitoring	Milestone tracking, risk assessment, corrective action

The risk assessment activity starts with the risk identification phase, where a list of risk items related to that project that have a higher chance of affecting the success of the project is created. Common techniques in this phase include checklists, decision-driven analysis, and comparing the identified risk items with the experience of the members involved in the process to try to identify common risks and risk management strategies used to deal with them in the past. Afterwards, the loss probability and impact of each identified risk item is assessed in the risk analysis step of the process. Analysis of factors such as quality, reliability, and availability is a common task in this phase. However, there is usually some uncertainty when it comes to estimating the losses that are a result of the occurrence of a risk [18], so the assessments done are very often subjective, and often the result of interviewing domain experts [18]. In the risk prioritization step, the identified risks are ordered through the use of techniques such as the analysis of risk exposure and risk reduction.

With the first three phases of risk management (according to Boehm) completed, the risk control activities can begin. These start with the risk management planning step, which addresses the risk items identified through steps such as buying information (*e.g.*, investing in a prototype to better understand the specific risk in question), risk avoidance, risk reduction, risk transfer, and risk plan integration. Common techniques used in this step are checklists of risk-resolution techniques, cost-benefit analysis, and risk management plan outlines. Afterwards, in the risk resolution phase, the identified risk items are eliminated or resolved through techniques such as risk avoidance by reducing the demands in requirements.

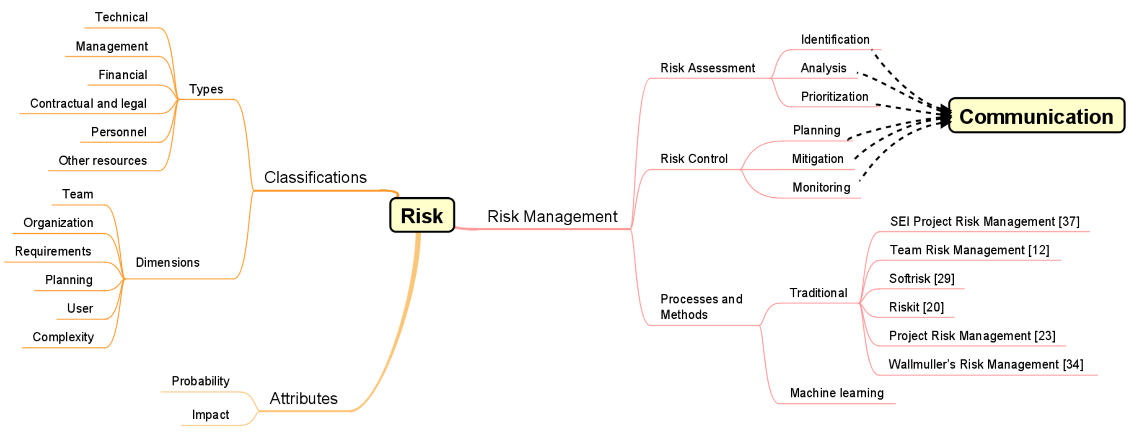


Figure 2.2: Mind map of concepts related to risk and risk management

Lastly, in the risk monitoring phase, the project's progress is tracked towards completion by resolving the previously identified risk items and taking corrective action whenever necessary through the use of techniques such as milestone tracking and risk assessment.

As can be seen by the information provided in this section, there are a lot of different aspects to consider in the process of risk management in software projects, which have been placed on a mind map that can be seen above in figure 2.2 to connect the key concepts that have been presented in the last two sections.

There is another area that has been gaining a tremendous amount of attention, particularly in recent years, with the goal of improving risk management processes in software projects, and that is the application of machine learning techniques and methods to improve the risk management workflow in software development companies. Key concepts related to machine learning are presented in the next section, and specific examples of the application of these concepts in this research area are presented in greater detail in the Chapter 2.4.

2.3 Machine Learning

Machine learning is a process where a system is trained with the ability to learn from experience over time. Machine learning algorithms build a model based on a data set with the goal of predicting some piece of information or deciding between possible outcomes. These algorithms can be classified according to three categories [55]:

- **Supervised Learning** - in this type of algorithms, the labels in the data set act as a teacher to train the model. Afterwards, the model can be used to predict information or to decide an outcome based on new data that comes in. Supervised learning problems can be marked as either classification or regression problems. In the former, the model's output is a discrete value (*e.g.*, true or false, 1 or 0), whereas in the latter, the model's output is a continuous value (*e.g.*, weight, height). Commonly used algorithms in this type of machine learning problems include Neural Networks, Support Vector Machines, Linear Regression, Decision Trees, Naive Bayes, among others.

- **Unsupervised Learning** - here, training is done without guidance, that is, the information that is sent to the model during the training phase is not labelled, categorized, or classified. The model tries to identify similarities in the data set and create a structure that is present in the data that is received. The main unsupervised learning problems can be divided into clustering and association problems. In the former, the goal is to discover the natural groupings between elements in the data set such that those elements are similar among themselves according to one or more relevant characteristics or attributes, whereas in the latter, the goal is to find frequent patterns or generate rules which are valid for large portions of the data set. Commonly used algorithms in this type of machine learning problems are k-means for clustering or the Apriori algorithm for frequent pattern mining, among others.
- **Reinforcement Learning** - lastly, reinforcement learning algorithms operate as a type of dynamic learning approach which educates the algorithm using arrangements of punishment and payoff/reward. There is no visible solution in these algorithms, but the reinforcement agent tries to maximize the output through the feedback received from the environment.

The use of machine learning approaches for risk assessment in software projects is achieved primarily through supervised learning techniques. When creating a machine learning model for problems that are tackled through supervised learning approaches, it is important to consider the type of information we want it to predict or return, as well as the data that will be used to train the model. The developers of the model need to make sure that the data used is clean, that is, that there is no missing information, that it does not contain errors, duplicate values, or outliers, and that it is consistent between itself. Noisy and unreliable data can impact the predictive accuracy of machine learning models and poor quality of data (*e.g.*, missing values, outliers) can result in inconsistent and unreliable predictions. Consequently, data preparation is a critical step to perform before building a machine learning model [52].

As seen in the categorization of machine learning problems and algorithms, there are a lot of different algorithms that can be used to create a machine learning model. The next subsections present an overview of the most commonly used algorithms for risk management and risk assessment purposes using machine learning approaches.

2.3.1 Neural Network

Neural Network (or Artificial Neural Network) is a supervised learning model with a large number of interconnected information processing elements called neurons, which are interconnected among each other through weighted links [55]. During the training phase, the network iterates through the training data set and adjusts the weights to reduce the error on each observation [40]. The neurons learn the relationships between the inputs and outputs by continuously adjusting the weights through the use of an activation function, which can be Linear, Sigmoid, Gaussian, or the Hyperbolical Tangent (tanh).

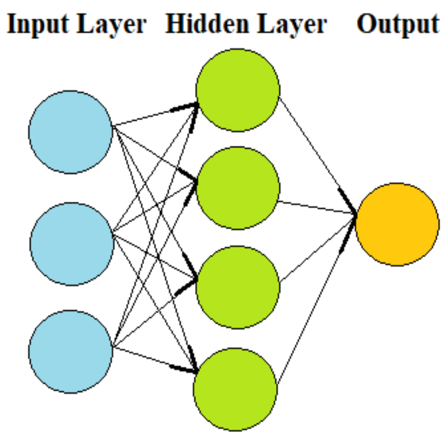


Figure 2.3: Example of a neural network with three layers

Feedforward Neural Networks were the first type of neural networks created and are still one of the most used types. The neurons are arranged in layers (input, hidden, and output) and are connected among each other through weighted links. Neurons in the input layer provide the data for the neural network, whereas the hidden layer (or layers) sits between the input and output layers and is where all the computation work is performed. Lastly, the output layer returns the relevant information that we are trying to obtain from the model. An example of this can be seen in figure 2.3.

Neural networks can be advantageous in software risk management problems as they are flexible and robust when it comes to quality of data, can deal with errors and noisy data in general, which is particularly important in situations where historical information about a company's projects is missing or has to be obtained from memory, meaning there will inevitably be a loss of quality in the information used [35]. However, they are slow in training, require considerable amounts of data for successful training, and a lot of effort must be expended in parameter tuning to improve the model's performance.

2.3.2 Support Vector Machine

A Support Vector Machine maps non-linear data in a high dimensional space through the use of kernels (*e.g.*, Polynomial, Radial Basis Function, Sigmoid) and then tries to find the hyperplane that best separates data into different domains with the maximum margin [66], as represented in figure 2.4. It is used in supervised learning problems, and can be used for classification or regression tasks, although it is more commonly used for the former [57].

Support Vector Machine algorithms are very effective in high dimension spaces, can handle nonlinear data efficiently and are scalable for high dimensional data [55]. However, they require a lot of time to process large data sets and, similarly to Neural Networks, require a considerable amount of effort to be expended in parameter tuning to improve the model's performance, as different kernels have different parameters that need to be configured.

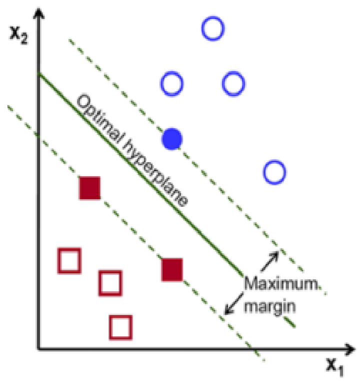


Figure 2.4: Graphical illustration of a Support Vector Machine [26]

2.3.3 Naive Bayes / Bayesian Network

The Naive Bayes classifier maps a set of samples to a number of classes, then calculates sample probabilities in the available classes and finds the most likely of the given classes through the application of the Bayes theorem [16], defined mathematically by the following equation:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{2.3}$$

In the equation above, P(A | B) is the probability of A being true when B is true, P(B | A) is the probability of B being true when A is true, P(A) is the probability of A being true, and P(B) is the probability of B being true.

It is a supervised learning technique used mainly for classification tasks. Naive Bayes is a fast algorithm both in training and in classification and can handle reasonable amounts of noise in the data. However, the Bayes theorem assumes the conditional independence of the variables, which in practice is not often the case.

Bayesian Belief Networks (BBNs) model the relationships (causal or otherwise) of a system or a data set and provide a graphical visualization of the structure of the data set by using Directed Acyclic Graphs (DAGs). In these graphs, the nodes represent random variables with probability distributions, while the edges show the weighted relationships between the nodes [42]. Bayesian networks have attracted a lot of attention due to their great ability to represent and deal with uncertainties in data sets [36].

BBNs can be developed quickly and with small data sets, and are easy to update whenever new data becomes available [42]. They are also robust to reasonable levels of noise in the data.

2.3.4 Logistic Regression

Logistic Regression is primarily used for binary classification tasks. It is used to predict between one of two possible values (*e.g.*, true/false, 0/1) of an outcome (or dependent variable) based on the values of the independent variables of the data set. In the context of risk management, this can be used to return a basic classification of risk level of a project as either “risky” or “not risky”.

It is a simple and effective algorithm, and does not require feature scaling, but its performance is not very good on non-linear data or data sets with highly correlated attributes.

2.3.5 Decision Tree

Decision Tree is a supervised learning algorithm used frequently in classification problems. As the name implies, it uses a model of decisions based on trees, with the root at the top, and then splits into conditions through the use of different branches (or edges). When a branch does not split anymore, the decision (or leaf) has been reached.

A simple example of how a decision tree works can be seen in figure 2.5. The root is the first decision, in this case regarding the remaining days until project deadline. This splits into two branches, one if the deadline is within 30 days, and the other if it is more than 30 days away. In the latter, there are no more branches, and a decision is reached (keep deadline). In the former, two more branches are formed, this time depending on the number of key features missing, with each of the branches leading to a decision (keep deadline or extend it by 2 weeks).

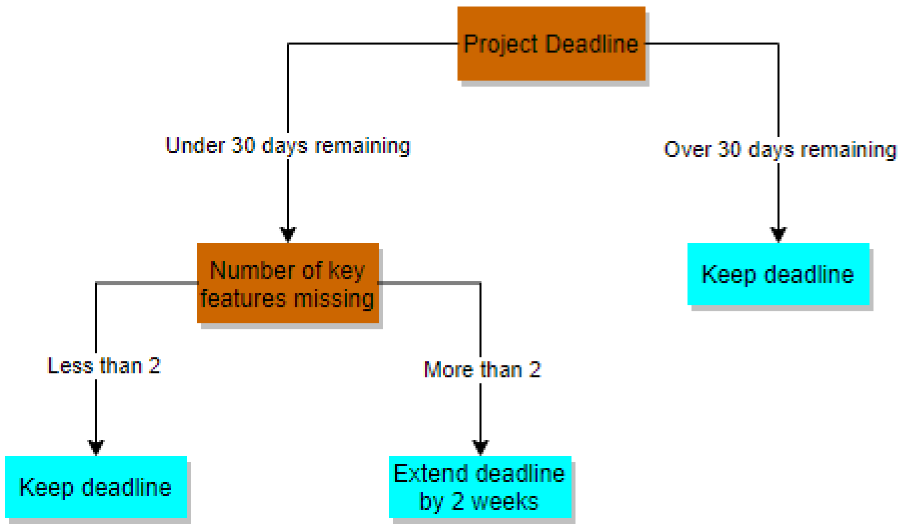


Figure 2.5: Example of how a simple decision tree works

This algorithm is simple and easily interpretable. It is possible to understand the impact of each branch on the final outcome. However, the trees can become quite large and prone to overfitting. In these situations, a technique called pruning should be used, which optimizes the decision tree by removing branches of the tree that are not crucial or are otherwise redundant to the classification.

2.3.6 Random Forest

Random Forest is a method that combines the results of multiple classifiers to increase its own accuracy (known as ensemble methods) which uses decision trees. This method is an option for both regression and classification tasks, although it is more commonly used in the latter. In a Random Forest model, various decision trees are generated, and the individual decision is associated with each tree.

Random forest can handle large data sets and missing data and are less prone to overfitting when compared to decision trees. However, compared to decision trees, it is harder to understand and visualize what decisions are being taken in the trees.

2.3.7 Evaluation Metrics

After the model has been developed and trained using any of the algorithms mentioned in the previous subsections, it is necessary to validate its predictions. After all, there is nothing to gain by using a model that cannot predict or generate useful information. The most basic metric used to evaluate a model is accuracy, and it is the result of the division between the number of correct predictions by the total number of predictions [14].

$$Accuracy = \frac{\text{Number of correct results}}{\text{Total number of predictions}} \tag{2.4}$$

Accuracy is only a useful metric for evaluation if there is an equal number of samples per class in the problem [14]. As an example, if there are three possible classes to choose from in the data set's dependent variable, and 80% of the samples are of class A, 10% are of class B, and the final 10% are of class C, the model can easily obtain a high accuracy by just predicting the majority of the samples to be of class A. Associating this to the research area of focus of this document, namely risk management in software projects, if the goal is to predict the risk level of a project with the possible options being "low", "medium", and "high", and 80% of the samples are of class "low", 10% are of class "medium", and the remaining 10% are of class "high", the model could obtain a high accuracy in testing by just predicting the majority of the projects to be of low risk, but the use of a model like this will lead to problems, particularly in a scenario like this example, as the cost of classifying a high risk project as "low" risk is much higher than the cost of classifying a low risk project as "high" risk.

In binary classification problems, a confusion matrix is used to visualize the results of a model's predictions compared to the actual results. An example can be seen in table 2.5.

Table 2.5: Confusion Matrix Example

		Actual class	
		Yes	**No**
Predicted class	**Yes**	TP	FP
	No	FN	TN

Confusion matrices focus around four basic evaluation metrics, which are the basis of other commonly used evaluation metrics:

- **True Positive (TP)** - number of predictions made where the model correctly predicted the positive class.
- **False Negative (FN)** - number of predictions made where the model incorrectly predicted the negative class.
- **False Positive (FP)** - number of predictions made where the model incorrectly predicted the positive class.
- **True Negative (TN)** - number of predictions made where the model correctly predicted the negative class.

Other commonly used evaluation metrics are obtained through the use of these four basic concepts in a confusion matrix [32]:

- **Accuracy (AC)** - the percentage of correct predictions made [32].

$$\text{AC} = \frac{TP+TN}{TP+FP+FN+TN} \tag{2.5}$$

- **Precision (P)** - the percentage of the predicted positive cases that were correct. Higher values of precision are better for a useful model.

$$\text{P} = \frac{TP}{TP+FP} \tag{2.6}$$

- **True Positive Rate (TPR)** - the percentage of positive cases that were correctly identified. Project managers who use predictive models for risk management usually give a lot of attention to this evaluation measure, as it provides the accuracy of the model when it comes to the correct identification of high risk projects [32].

$$\text{TPR} = \frac{TP}{TP+FN} \tag{2.7}$$

- **True Negative Rate (TNR)** - the percentage of negative cases that were classified correctly [32].

$$\text{TNR} = \frac{TN}{TN+FP} \tag{2.8}$$

- **False Positive Rate (FPR)** - the percentage of cases that were incorrectly classified as being of the positive class. A smaller value of FPR indicates better performance of the model.

$$\text{FPR} = \frac{FP}{FP+TN} \tag{2.9}$$

- **False Negative Rate (FNR)** - the percentage of cases that were incorrectly classified as being of the negative class.

$$\text{FNR} = \frac{FN}{FN+TP} \tag{2.10}$$

- **F-measure** - as the use of Precision or TPR on their own might not be sufficient to obtain enough information about the performance of the model developed, F-measure provides a way to express both metrics through a single score. It is expressed as the harmonic mean of TPR and Precision, and has a value between 0.0 and 1.0, where 0.0 is a poor score, and 1.0 is a perfect score.

$$\text{F-measure} = \frac{2*P*TPR}{P+TPR} \tag{2.11}$$

- **Area Under the ROC Curve (AUC)** - the probability of the model ranking a random positive example higher than a random negative one. The Receiver Operating Characteristic (ROC) curve measures the results of a classification model in different FPR and TPR thresholds, with the FPR at the X-axis and the TPR at the Y-axis of the AUC graph. AUC returns a value between 0 and 1, where the higher the AUC, the better the model is at distinguishing positive and negative classes. Figure 2.6 shows a typically good AUC graph.

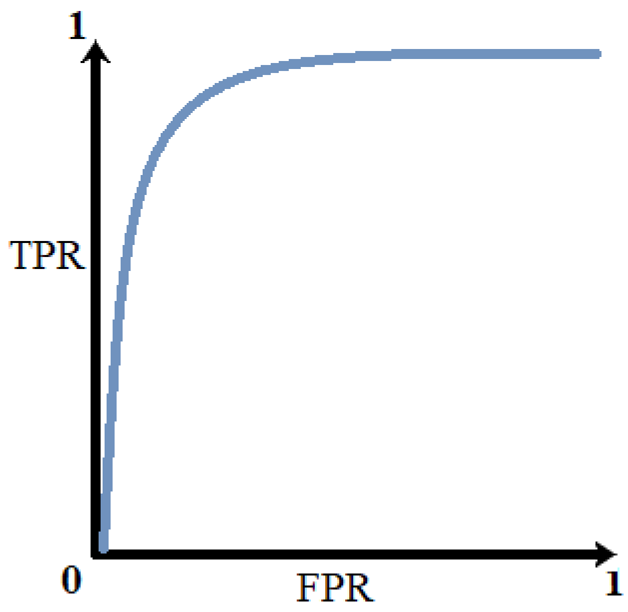


Figure 2.6: Example of an AUC graph

AUC is a validation metric that provides a better interpretability of the model's performance, as an AUC value close to 1 means the model is extremely good at distinguishing positive and negative classes, whereas an AUC value of around 0.5 means the model cannot distinguish between the positive and negative classes. When it nears 0, the model is reciprocating the classes, meaning it is predicting positive classes as negative, and negative classes as positive. This is important to consider due to the two types of classification errors that a model can commit in a binary classification problem, such as one where the model is classifying a project as being of low or high risk:

- A low risk project classified as a high risk one, generally referred to as a type-1 error
- A high risk project classified as a low risk one, generally referred to as a type-2 error

If the model cannot accurately distinguish between low and high risk projects, it can result in a lot of problems, particularly if more type-2 errors occur. The occurrence of a type-1 error means more attention will be given to low risk projects, which is not necessarily bad, as risks should still be controlled even if their impact and probability of occurrence is low, but ultimately it still results in extra effort that could have been used elsewhere. However, the occurrence of a type-2 error can mean extending the project deadline or dealing with budget overruns.

Generally speaking, the metrics above are used for classification problems, whereas regression problems can use the following evaluation metrics to assess the model's performance:

- **Mean Absolute Error (MAE)** - this is the average of the difference between the predicted values and the actual ones. It can be used to check how far the predictions made were from the real results [17]. A smaller value means better performance.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|Yi - \widehat{Yi}\right| \tag{2.12}$$

- **Mean Squared Error (MSE)** - this is similar to MAE, but MSE instead uses the average of the square of the difference between the predicted values and the original ones [14]. With MSE, the impact of larger errors is more pronounced than that of smaller ones, providing a point of focus for both the developers of the model and the project manager. As with MAE, a smaller value of MSE indicates better results.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Yi - \widehat{Yi})^2 \tag{2.13}$$

- **Mean Magnitude of Relative Error (MMRE)** - mean of the absolute percentage errors [42]. A lower value of MMRE indicates better predictive performance.

$$\text{MMRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|Yi - \overline{Yi}\right|}{Yi} \tag{2.14}$$

- **Balanced Mean Magnitude of Relative Error (BMMRE)** - due to the fact that MMRE penalizes overestimates [42], the balanced mean magnitude of relative error measure should also be taken into account. As with MMRE, a lower value of BMMRE indicates better predictive performance.

$$\text{BMMRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|Yi - \overline{Yi}\right|}{min(Yi, \overline{Yi})} \tag{2.15}$$

Two important concepts to consider in the evaluation of the performance of a machine learning model are overfitting and underfitting. The former occurs when the model created corresponds too closely (or even exactly) to the set of data used for training and may therefore fail to predict future observations reliably. In this situation, the model has learned too much from the training set. Underfitting, on the other hand, happens when a model cannot understand the relationships in the data set. Here, the model has not learned enough from the training data, resulting in poor performance and inaccurate predictions.

2.3.8 Oversampling and Undersampling

If an unbalanced data set is used to train the model, it will tend to predict the classes that occur the most and generally ignore the least occurring ones [61]. To tackle this problem, sampling methods can be used to change the distribution of unbalanced classes in the data set [61], with oversampling and undersampling being the most frequently used ones.

Oversampling increases the number of samples of the minority classes either through data replication or by generating new data to improve the imbalance detected in the data set, whereas undersampling removes random instances from the predominant class. There are trade-offs involved in both techniques, so it is important to keep them in mind when choosing which one to apply. For instance, since undersampling can remove data instances at random, useful data might be removed without us knowing [61]. Oversampling does not have this problem, but since more data is added to the data set, training and execution times will be longer. Additionally, since oversampling replicates instances that are already found in the data set, it will increase the chances of overfitting occurring [39].

2.4 Machine Learning Approaches in Risk Management

The previous sections of this report have presented important elements such as the concept of risk in software projects, popular risk management techniques and processes found in the literature, and various machine learning techniques and algorithms. The first subsection presents how project managers can further improve their risk management implementations in software projects through the use of machine learning. Subsection 2.4.2 presents a literature review of experiments and studies performed in the research area of machine learning techniques used for risk assessment, detailing the inputs used, outputs extracted from the model, algorithms used, and metrics used to evaluate the performance of the models developed.

2.4.1 Machine Learning Practices in Risk Management

Initially, the project manager, together with the development team, should identify what type of data they want to obtain by using a machine learning model. This is a decision that will have an impact on the number of machine learning algorithms and methods that are adequate for the problem at hand (i.e., whether the team wants to identify the likelihood of different risk factors through a multi-layer Neural Network or simply assess the overall risk level of the project - low, medium, high - through the use of a Bayesian classifier such as the Naive-Bayes classifier).

Machine learning models need data. After an agreement between the developers and the manager(s) of the project regarding the type of data that will be predicted by the model is found, they now need to create a data set that will be used to train the model. This can be built from a variety of sources, such as a historical record of projects in a company (if there is one available), or from the project manager and the team's recollection of past projects (if the company does not keep track of past projects), or even historical data sets that can be obtained online. All of these approaches have their disadvantages though, so it is necessary to keep them in mind when choosing how to build the data set.

Obtaining information from the project manager and the development team's recollection of past projects inevitably means some information will be missing. This information can either be removed completely from the data set, or it can be handled through various strategies to deal with missing data. Common techniques for this purpose include replacing the missing values with the attribute's mean or its most probable value, which may not be entirely accurate according to what actually happened in those projects, but it is still a better approach compared to leaving missing values in the data set, which has a negative effect on the training of the models and, consequently, their predictive performance. Additionally, depending on the size of the company, the number of projects might not be very high to begin with, resulting in a small data set which might not be enough for a successful training of the machine learning model.

Alternatively, historical data sets are available online. It becomes the responsibility of the project manager and the development team (especially members that will be responsible for building the machine learning model) to assess whether or not the type of information that is available in these data sets matches the information that they want to use as inputs to the model as well as the information that they want to obtain from the model - the model's outputs (*e.g.*, software module quality, or overall risk level of the project - low, medium, or high).

Additionally, in a situation where the historic data sets do not quite match the team's ideas of inputs and outputs to and from the model, but they contain a large amount of information that can be used to train the model, the team can decide whether it is worth it instead to change their ideas of what should be sent to the model and what should be retrieved from it in order to match what is available on their historical data sets of choice. This is often used mostly in worst-case scenarios as it requires a major shift in the objectives of the team, but it is an option that should not be completely discarded nonetheless.

Throughout this report, it has already been mentioned why risk management is a process that should start before development does and then frequently returned to during the different stages of the SDLC, but in situations where machine learning techniques are being used in the risk management process, the decisions regarding the structure of the data set that will be used to train the model should be very clear and close to final (if not entirely final) by the time the project development starts, otherwise that can lead to its own risk and extra effort being spent on re-training, re-evaluating, and in some situations, completely reworking the model in order to match the new project objectives.

Afterwards, the team needs to decide where the machine learning model will be created and used, or where it will be integrated. Programs such as RapidMiner [7], WEKA [11] and MATLAB [5] are often seen being used in the literature regarding applications and evaluations of risk management classification through the use of machine learning models. Alternatively, machine learning models can be developed with the help of machine learning libraries that are available in a wide variety of popular programming languages (*e.g.*, Java-ML [2] package and the WEKA API [11] in Java, TensorFlow [9] and scikit-learn [8] packages in Python), providing more flexibility in terms of the options available to the developers. What this also adds is the ability to integrate the model created with popular project management applications such as JIRA [3] or Asana [4] through the use of their APIs for the same programming languages, creating a software module containing the machine learning model that can be fully integrated with the project management tools of choice in those situations.

Compared to traditional risk management techniques and models, the introduction of machine learning methods will increase the amount of effort required initially to develop a risk management strategy that includes the usage of the machine learning model. This is because there might still be the need to identify, assess, categorize, and potentially prioritize risks (depending on the information that will be used as inputs to the model) in order to build the training data set. Additionally, there is the work required to develop the model and fine-tune its parameters in order to get the model to be as accurate as possible, and the effort required to test and evaluate the model until it surpasses the predetermined acceptable levels of results in the metrics used for evaluation. However, that time is recovered in the form of an implementation of a risk management strategy that provides continuous risk assessment and review. By the time the first iteration of risk assessment phases are completed, the model should be ready for use, which allows to save some time through its usage during the next iteration of risk assessment instead of manually going through those steps again, assuming the model created has been correctly evaluated and provides a high degree of confidence through its predictions.

2.4.2 Literature Review of Machine Learning Approaches in Risk Management

Machine learning in risk management has obtained increasing popularity in recent years, and a lot of different approaches have been used. For the purposes of the analysis of the state-of-the-art, the focus was on finding practical applications of machine learning to predict possible project risks or an overall risk level of a project. From there, it was possible to identify not only some of the most frequently used algorithms and evaluation metrics, but also frequently used characteristics which served as inputs to train the models.

Throughout the creation of this state-of-the-art review, bibliographic databases such as Scopus and DBLP were used to search for various articles, scientific papers, and surveys related to the topic in review. Searches were performed using keywords such as “risk assessment”, “risk classification”, “risk management”, “machine learning”, and “software projects”, together with the appropriate query modifiers to reduce the scope of the results to the application of machine learning for risk assessment specifically in the software development industry. By reading the abstracts and briefly analysing the contents of the search results, the ones that were considered more relevant were read and analysed in greater detail. Some examples of relevant studies and experiments done in this research area are described below and can be seen in greater detail in terms of inputs and outputs used in table 2.6, or grouped by use cases and by algorithms used in the mind map in figure 2.7.

In [22], an Artificial Neural Network model was created to predict deviations in new software projects. The inputs to the model were the risk factors detected in the projects, and the outputs were the differences found in time, budget, and number of personnel, number of completed work packages, and success of the project under investigation. This experiment showed the applicability of Neural Networks when the intended information spans more than one category (in this case, the deviations in five attributes related to the project), as well as the fact that the model can have a great performance and accuracy, as seen in its results.

A Neural Network model was also created in [35], together with a Support Vector Machine model to compare both approaches and their accuracy in evaluating the risk level of software projects. The input used was a vector of software risk factors collected after various interviews with experts in the industry, which were then grouped according to six different risk categories (Environment Complexity, Project Requirement Complexity, Cooperation, Team, Project Management, and Engineering). The output was the predicted outcome of the project (“successful”, “failed”, or “challenged”). The Support Vector Machine model had a higher accuracy compared to the Neural Network method (80% vs 70%, respectively) due to Neural Network’s tendency in finding a local optima [35], but after changes were made to the Neural Network method by optimizing it with a Genetic Algorithm (GA), this made it so the Neural Network-Genetic Algorithm method surpassed Support Vector Machine in accuracy (85% vs 80%, respectively) by reducing the search for a local optima.

In [32], the author proposes a Neural Network architecture with a back propagation algorithm to learn the patterns of a data set of projects completed in the past, which also includes 22 project risk factors of areas such as estimations, requirements (*e.g.*, ambiguous requirements, frequent changes to requirements), and team organization (*e.g.*, low morale, lack of skills or experience). The output of the model was a classification of the risk level of the project: “risky” or “not risky”. The model developed was found to have a higher accuracy and sensitivity when compared to a Logistic Regression model developed from and applied on the same data set.

Table 2.6: Studies and experiments on the use of machine learning techniques for risk assessment

Reference	**Inputs**	**Outputs**	**Algorithm(s)**	**Evaluation metric(s)**
A Novel Model for Risk Estimation in Software Projects using Artificial Neural Network [22]	45 risk factors of 20 software projects (70% of data used for training, 30% for testing)	Deviations in project duration, cost, number of personnel, completed work packages, project success	Neural Network	Training R = 0.9978 Testing R = 0.9935 Validation R = 0.996 MSE = 0.001
Software Project Risk Management Modelling with Neural Network and Support Vector Machine Approaches [35]	Data of 120 software projects collected through questionnaires distributed in cities in China (83.3% of data used for training, 16.7% for testing)	Classification of projects as either “successful”, “challenged”, or “failed”	Neural Network	Accuracy = 70%
			Genetic Algorithm NN	Accuracy = 85%
			Support Vector Machine	Accuracy = 80%

Discriminating Risky Software Project Using Neural Networks [32]	22 attributes of 40 projects in the OMRON database (80% of data used for training, 20% for testing)	Risk level of the project - “risky” or “not risky”	Neural Network	Accuracy = 82.2%
				Precision = 81.82%
				TPR = 81.82%
				TNR = 82.61%

			Logistic Regression	Accuracy = 87.5%
				Precision = 100%
				TPR = 66.7%
				TNR = 100%

An Empirical Evaluation of Predicting Runaway Software Projects Using Bayesian Classification [46]	Responses on a 4 point Likert scale to a questionnaire focusing on 5 viewpoints of key risk factors in 40 SSBC projects (10-fold cross-validation used for testing)	Project classification as either “runaway” or “success”	Bayesian classifiers	Accuracy = 82.5%

A Probabilistic Software Risk Assessment and Estimation Model for Software Projects [42]	Assessment of 27 risk factors (low, medium or high) in 12 software projects	Probability of the project being of low, medium, or high risk	Bayesian classifiers	MMRE = 0.03842
				BMMRE = 0.03911

Software Project Risk Analysis using Bayesian Networks with Causality Constraints [36]	Software project data from 302 projects collected through questionnaires (10-fold cross-validation used for testing)	Classification of project’s performance based on risks identified as “low” or “high”	Bayesian network with causality constraints	Accuracy = 75.15%
			Decision Trees	Accuracy = 70.86%
			Naive Bayes	Accuracy = 72.85%
			Bayesian classifiers	Accuracy = 74.17%

Classification of Risk in Software Development Projects using Support Vector Machine [66]	530 samples of a data set created from information of software development projects (70% of data used for training and 30% for testing)	Project risk classification as either “low risk” or “high risk”	Support Vector Machine	Accuracy = 99.51%
				AUC = 98%

An Intelligent Model for Software Project Risk Prediction [37]	64 risk factors of data from 120 projects (83.3% of data used for training, 16.7% used for testing)	Classification of projects as either “successful”, “challenged”, or “failure”	Neural Network	Accuracy = 75%
			Support Vector Machine	Accuracy = 85%
Prediction of Risk Factors of Software Development Project by Using Multiple Logistic Regression [24]	Data obtained from questionnaires regarding the risk level of 70 software projects	Classification of characteristics of a software project as “risk” or “non risk”	Multiple Logistic Regression	Accuracy = 90%
An Empirical Approach to Characterizing Risky Software Projects Based on Logistic Regression Analysis [58]	Responses on a 4 point Likert scale to a questionnaire focusing on 5 viewpoints of key risk factors in 40 SSBC projects	Classification of projects as either “risky” or “not risky”	Logistic Regression	Accuracy = 87.5%

The authors in [46] developed an approach to predict runaway projects (that is, ones that greatly exceed budget and deadlines and have also failed to produce an acceptable deliverable) in an organization through the use of a questionnaire to identify the characteristics of projects and then classify them into "runaway" or "success" projects through the use of a Naive Bayes classifier. These characteristics are classified according to five different categories: requirements (*e.g.*, ambiguity of requirements), estimations (*e.g.*, lack of stakeholders present for estimation process), planning (*e.g.*, unspecified milestones), team organization (*e.g.*, lack of skills or experience), and project management (*e.g.*, inadequate project monitoring). All of the characteristics used in the questionnaire have a negative connotation to them, and the more characteristics the survey participants agree with in regards to their project, the more likely that project is to be classified as a "runaway" project. 10-fold cross validation was used to evaluate the effectiveness of their solution, showing a predictive accuracy of 82.5%, with 33 out of 40 projects classified correctly.

Bayesian classifiers were also used in [42] and [36]. In the former, a Bayesian Belief Network (BBN) was used to build a software risk estimation model that was used for the main software risk indicators for risk assessment in software projects, whereas in the latter a model was created using a Bayesian network with causality constraints to not only identify but also analyse risks in software development projects through data collected from 302 software projects. The authors found that it had an accuracy of 1% to 7% higher than the other models tested (Logistic Regression, Decision Tree, and Naive Bayes), which they attributed to the incorporation of expert domain knowledge and causality discovery into the BBN.

In [66], the authors used a Support Vector Machine to model risk classification in software projects. The model classified projects as either high risk or low risk projects.

A Support Vector Machine is also used in [37] to predict the risk level of different projects as either "low", "medium", or "high". A Neural Network was used for comparison, and the authors found that the Support Vector Machine was more accurate (85% accuracy of Support Vector Machine compared to 75% of the Neural Network).

Multiple Logistic Regression was used in [16] to classify different characteristics of software projects as either a "risk" or a "non risk". The input data was obtained through questionnaires sent to experts in the software project development and management fields, which asked them to classify risk factors from 8 categories (User, Requirements, Estimations, Cost, Schedule, Planning and Control, Team, Software) according to their risk level on a scale from 1 to 5.

Lastly, the authors in [58] used Logistic Regression to classify projects as either "risky" or "not risky". Responses to a questionnaire focusing on 5 viewpoints of key risk factors (Requirements, Estimations, Planning, Organization, Management) were used as the input data, and the model developed classified 35 out of 40 projects correctly.

After performing this literature review of studies and experiments on the use of machine learning for risk assessment, the contributions that the solution developed for this work can provide to this research area became clear:

- **Integration with project management tools** - in table 2.6, very little reference is made to the continuous use of the models developed in the studies and experiments presented, as most of those models were created using software that can't be integrated with project management tools (*e.g.*, RapidMiner, MATLAB, WEKA). The integration with tools used for daily project management tasks would make it so risk management becomes just another step in the project management cycle, rather than something that requires a large overhaul in an organization's workflow to integrate it in their processes.

- **Predictions per risk category** - in the studies seen in table 2.6, these generally classify projects or characteristics of projects as a type of global evaluation of the project. The application developed for this project provides information about impact and likelihood of risks in different categories instead, which allows project managers to focus on more specific areas of their project in order to improve the chances of a successful development.
- **Understanding the predictions made** - some of the articles presented in table 2.6 compare different machine learning algorithms (*e.g.*, [35] and [37]) with the goal of comparing their predictive performance in the context of a specific problem. However, a greater focus should be placed in also comparing them in terms of interpretability and the performance trade-offs involved in more interpretable algorithms, such as Decision Trees or Naive Bayes.
- **Use of organizational project data to train, test, and continuously use in the model** - all of the studies listed in table 2.6 either create a data set based on responses to questionnaires or surveys, or use a historical data set (*e.g.*, OMRON, NASA) both to train the model and then test it. The use of data of projects from the organization implementing the model makes it so the data that is being used to train the model and to predict outcomes on a new project data is always relevant to that organization (compared to a historical data set which might not be entirely relevant to the organization, depending on what type of projects the organization focuses on). This data is also relevant from a temporal point of view, as the data that will be used in this situation is likely to be more recent, thus matching the needs and demands (and corresponding risks associated with them) from the market in the current day, rather than those of the market in the years where the historical data sets were created.

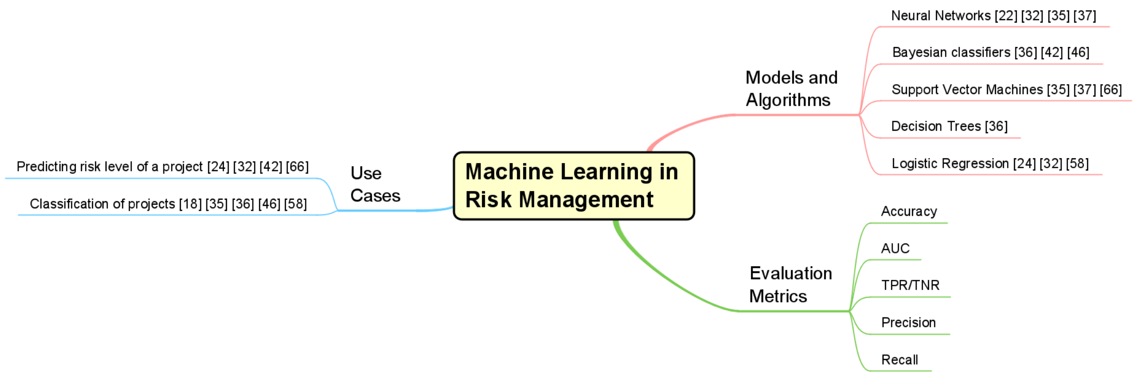

Machine Learning in Risk Management
Use Cases
Predicting risk level of a project [24] [32] [42] [66]
Classification of projects [18] [35] [36] [46] [58]
Models and Algorithms
Neural Networks [22] [32] [35] [37]
Bayesian classifiers [36] [42] [46]
Support Vector Machines [35] [37] [66]
Decision Trees [36]
Logistic Regression [24] [32] [58]
Evaluation Metrics
Accuracy
AUC
TPR/TNR
Precision
Recall

Figure 2.7: Mind map of machine learning approaches for risk management

2.5 Summary

With the information provided in this state-of-the-art review, it is clear that the concept of risk is very important to consider during the development of software projects. Risks can be grouped according to several categories [62]: technical, management, financial, contractual and legal, personnel, and related to other resources. But identifying risks and the category that they belong to is only a fraction of the work that makes up the process of risk management, which is focused on not only identifying the risks, but also assess their likelihood and possible impact on a software project in order to prepare strategies to deal with the identified risks before they materialize themselves during the development of the project.

Traditional risk management models focus on different aspects of the process. For instance, SEI's Project Risk Management Paradigm places a large importance on communication between the teams responsible for each phase of the process, whereas the Softrisk management technique focuses more on the use of complete documentation.

The information presented in this chapter shows the quantity and the depth of not only the machine learning algorithms that can be used to train a model used for classification tasks (such as risk prediction), but also of the evaluation metrics that are available to verify the model's results and performance with the data that is available to it. Metrics such as Accuracy, AUC, and TPR are key to decide if the model is ready for use or needs to be improved a bit further. Accuracy on its own might not be enough, as it not only requires an equal number of samples per class in order to be more useful, but also due to the fact that it only tells us how accurate the model is in its predictions, with no distinction regarding the classes being predicted. This is where the usage of AUC and TPR is important, as it provides greater interpretation of the model's results, making it so the strengths and weaknesses of the model become clearer.

There are a lot of possibilities when it comes to machine learning models and methods that can be used to predict risks in software projects, or the risk level of software projects. Neural Networks, Support Vector Machines, and Bayesian classifiers are some of the most frequently used algorithms for these tasks and are present in the list of algorithms selected to be tested for the project to which this paper refers to.

The information presented in this chapter also led to the identification of possible innovation factors in this research area, or simply areas that can be further explored to improve the use of machine learning for risk assessment, such as: integration with project management tools, analysing the interpretability of algorithms and their respective performance trade-offs, and the use of organizational project data to train, test, and continuously use in the model.

Chapter 3

Data Preparation

As mentioned in the previous chapter, machine learning models require data to be trained before using that information to make predictions on a new set of data. However, the data that is used to train the models must be verified in terms of its consistency and quality before it can be used to maximize the quality of the models in their predictions.

This chapter presents the characteristics of the data set that was used to train the machine learning models used for this application in section 3.1, as well as the data preprocessing tasks that are performed in the application prior to using the data to train the models in section 3.2.

3.1 Data Set

The data used for this application was provided by Strongstep, one of the project's partner companies, in the form of various spreadsheets with different information related to projects in their project management tool SCRAIM. One spreadsheet listed the risks related to the projects, which were added through SCRAIM, and other spreadsheets listed characteristics related to the projects themselves such as the project type, number of team members involved in the project, budget allocated, among others. The information in the latter is especially important during training, as this is the type of information that will be sent to the application when a new project is created on SCRAIM and will be used to predict the impact and likelihood of risks in different categories before project development begins.

However, not all of the fields are required to be filled when adding this information to SCRAIM. For example, the list of risks had a large number of variables, but due to the fact that most of the fields did not need to be filled when creating a new risk, there were a lot of empty cells in the risk spreadsheets. To improve the quality of the data set that was going to be used to train the models, a new spreadsheet was created using the information available in the various spreadsheets provided by Strongstep, focusing on the information that was not only considered relevant to train the models, but also of mandatory filling when adding such information to SCRAIM in order to avoid missing information in the training data set.

This new data set consists of 140 risk items from 18 projects and 13 variables, where the first three variables are used only to identify the projects and respective risks in the data set and are not used when training the models. The data available was divided into two different sets: one set for validation of hyperparameter tuning (40 instances); the other set (100 instances) was added up to the first set for training and testing the models in a process that will be explained later.

The complete set of variables is as follows:

- **Project ID** - unique ID of the project. This is an incrementing quantitative value used as an identifier of the project and is not used for training.
- **Project Name** - name of the project. As with Project ID, this is a qualitative field that is not used for training, and only serves as an identifier of the project.
- **Risk Description** - description of the risk. Similarly to the previous two variables, this is a field that is not used for training, and only serves as an identifier of the risk.
- **Project Type** - this is a categorical variable that represents the type of project in the platform, and includes values such as CMMI, ISO 27001, RGPD, among others.
- **Project Manager Profile** - this variable indicates the profile of the manager of the project. It is also a categorical variable with possible values being Junior and Senior.
- **Number of Team Members** - quantitative value indicating the number of team members involved in the project.
- **Project Duration** - quantitative value indicating the duration of the project in hours.
- **Budget** - budget allocated to the project. This is a categorical variable with values that describe the budget allocated to the project based on the numeric amount. Possible values are Very Low, Low, Medium, or High.
- **Risk Category** - this is a categorical variable indicating the category of a risk that was added for a given project. The categories in this variable are very close to the most frequent risk categorizations that are found in the literature (*e.g.*, table 2.2), and includes risk categories such as Project Complexity, Management, Planning and Control, among others.
- **Risk Treatment** - this is a categorical variable indicating the treatment action that should be taken for a risk that was added for a given project. As with the Risk Category variable, the categories in this variable are very close to the most frequent risk treatment actions that are found in the literature, and include actions such as Accept, Track, Transfer, among others.
- **Risk Mitigation Action** - this is a description of the mitigation action taken to deal with the specified risk. The information is obtained through the SCRAIM knowledge base, which provides a list of the most common mitigation actions used to deal with risks identified.

- **Risk Impact** - impact of the risk on a scale from 1 to 3, where 1 indicates a risk with low impact, 2 indicates a risk with medium impact, and 3 indicates a risk with high impact on the project. The first model created focuses on predicting this information when a new project is created on SCRAIM.

- **Risk Likelihood** - likelihood of the risk occurring during project development on a scale from 1 to 3, where 1 indicates a risk with low likelihood of occurrence, 2 indicates a risk with medium likelihood of occurrence, and 3 indicates a risk with high likelihood of occurrence. The second model created focuses on predicting this information when a new project is created on SCRAIM.

The Risk Treatment and Risk Mitigation Action variables are also not used to train the models developed for this module specifically, as it is information added only during the development of the project, and not beforehand. They were added to the data set as they were some of the variables with most information available in the spreadsheets shared by Strongstep and were considered useful for possible future developments in the project.

This new data set provides a more organized structure, one which is more applicable to use when training the machine learning models in the application. However, the process described above only tackles the organization and structure of the data, not the quality and preparation of the actual data in the new set. That is where data preparation comes in.

3.2 Data Preprocessing

Data preprocessing is a crucial process for a successful resolution of any data mining and machine learning-oriented problem. This is due to three reasons:

- Data preprocessing removes impurities in data. Impurities in data can come in the form of incomplete data (data which lacks values or attributes of interest), noisy data (data which contains errors or outliers), and inconsistent data (data which contains discrepancies in values) [67].

- Data preprocessing can generate a smaller data set, with the goal of improving the efficiency of the models. This is done by selecting relevant data (remove anomalies or duplicate records) or by reducing the data (*e.g.*, sampling) [67].

- Data preprocessing improves the quality of the data. By scaling the range of the quantitative features, and removing outliers or features with very low variance from the data set [67], this improves the quality of the data that will be used to train the models and improve their predictive performance as a result.

In the work for this dissertation, data preprocessing was handled at two levels: the data set level and the application level. The former deals with tasks performed directly in the data set (in this case, the spreadsheet described in the previous section), whereas the latter refers to data preprocessing tasks performed in the application prior to using the data loaded from the spreadsheet to train the models.

At the data set level, the first point and part of the second were tackled with the process of creating the new spreadsheet that was described in the previous section. This way, only attributes that are interesting and relevant to use to train the models were selected and, as the new spreadsheet was continuously updated as more data became available, noisy, or inconsistent data could be detected and fixed before being used to train the models.

Lastly, some of the techniques to improve the quality of the data mentioned in the third point were performed after analysing the class distribution of the attributes of the data set. This was done by creating a new spreadsheet using the complete data set to create pivot tables containing the class distribution of each attribute in the data set, and, in some cases, the class distribution of each attribute per risk impact level and risk likelihood level.

Various charts were created through these pivot tables with different goals in mind. The class distribution charts were created to identify possible class imbalance problems (*e.g.*, one risk impact class being listed much more frequently than the others) and, consequently, evaluate the need for different preprocessing tasks at the application level that could deal with a class imbalance problem, such as oversampling and undersampling. The class distribution per risk impact and risk likelihood level charts were used to try to identify useful patterns in the data (*e.g.*, a risk category having consistently higher levels of risk impact and likelihood), which could then be compared with the predictions made by the models. By comparing not only their predictions but also the class probability estimates for each risk impact and likelihood class, the model's strengths and weaknesses become more apparent. For instance, if it is attributing a relatively high probability of low risk impact to a risk category which has no low risk impact risks, then it is clear that there are improvements to be made to the model. Examples of the risk impact and risk likelihood class distribution charts can be seen in figures 3.1 and 3.3, and examples of class distribution per risk impact and likelihood can be seen in figures 3.2 and 3.4.

As for the application level tasks, categorical variables were encoded through one-hot encoding, which represents each class through a new attribute with either a 0 or 1, where 0 indicates the absence of that class, and 1 indicates its presence. This was used in the Project Type, Budget, and Risk Category variables. An example of what this process does using Project Type as the categorical variable can be seen in table 3.1.

Table 3.1: Example of the Project Type attribute using one-hot encoding

Project Type	**CMMI**	**ISO 27001**	**RGPD**
CMMI	1	0	0
ISO 27001	0	1	0
RGPD	0	0	1

Additionally, feature scaling was used to scale the range of the values of the numeric independent variables. This is an important step because if one attribute's range of values varies signifi-

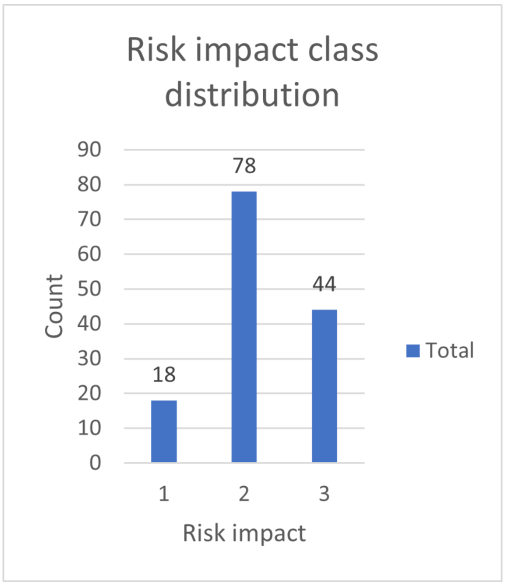


Figure 3.1: Risk impact class distribution

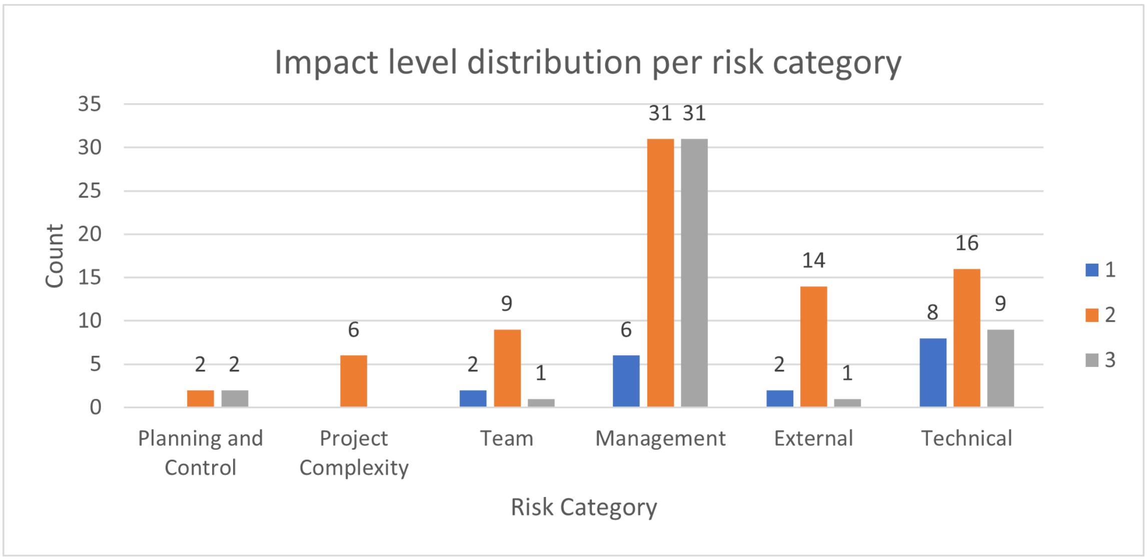


Figure 3.2: Risk impact level distribution per risk category

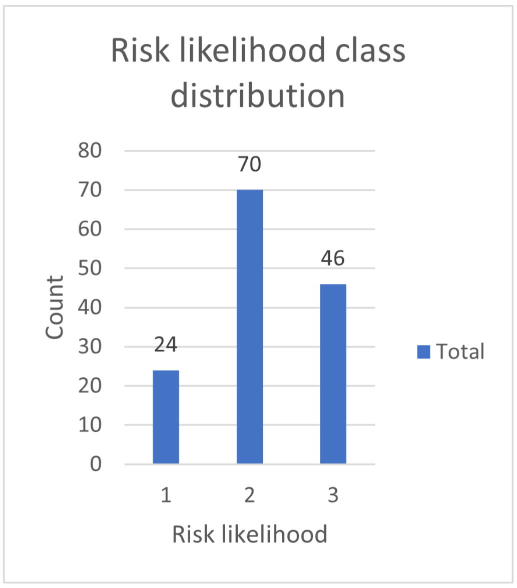


Figure 3.3: Risk likelihood class distribution

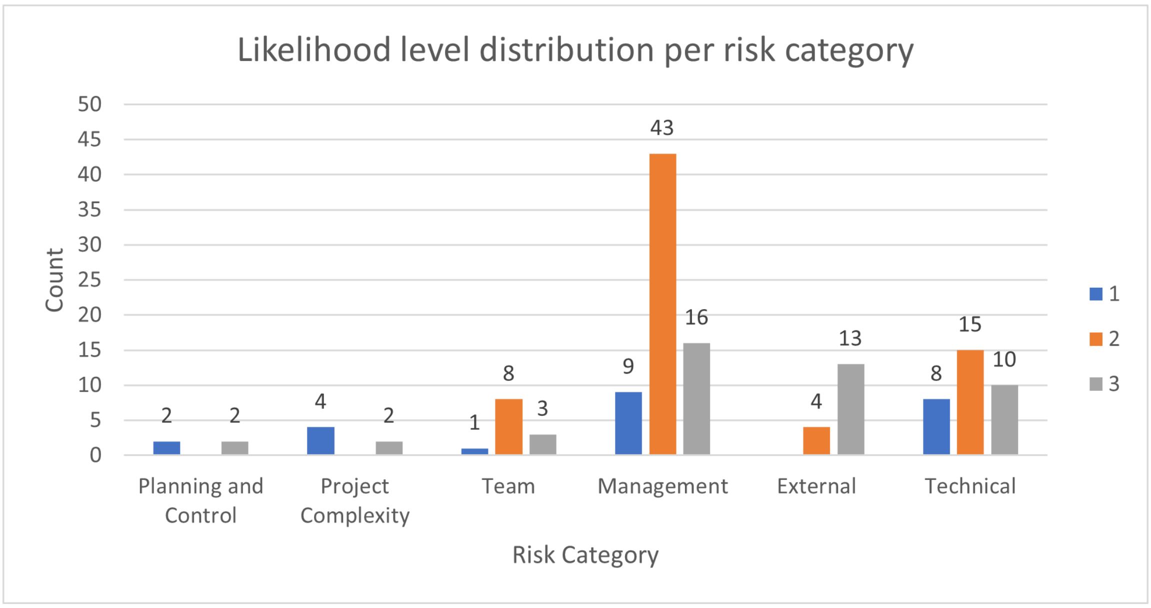


Figure 3.4: Risk likelihood level distribution per risk category

cantly more than the range of values of another attribute, the former will become dominant in the data set, resulting in an anomaly and a decrease in the predictive performance of the models. For the purposes of this application, min-max scaling was used through Scikit-learn's MinMaxScaler.

Lastly, feature selection was performed to select which of the remaining 8 independent features would be used to train the models. The Risk Treatment and Risk Mitigation Action variables were discarded as while they can be used for different types of predictions in the future, they are not useful for risk impact and likelihood level predictions when creating a new project. This is due to the fact that this information is added when a new risk is created for a given project on SCRAIM, and is not applicable in this use case where the information is generated prior to the creation of a new project.

Out of the remaining 6 independent features, low variance features were removed from the training set. For the purposes of this project, features with 30% or below variance were considered to be low variance ones, and were thus removed from the training set. This was the case with the Project Manager Profile and Number of Team Members attributes, with Project Manager Profile value "Senior" and Number of Team Members value "2" found in over 70% of the data set risks. The distribution of the values of these 2 attributes over the complete data set can also be seen in figure 3.5.

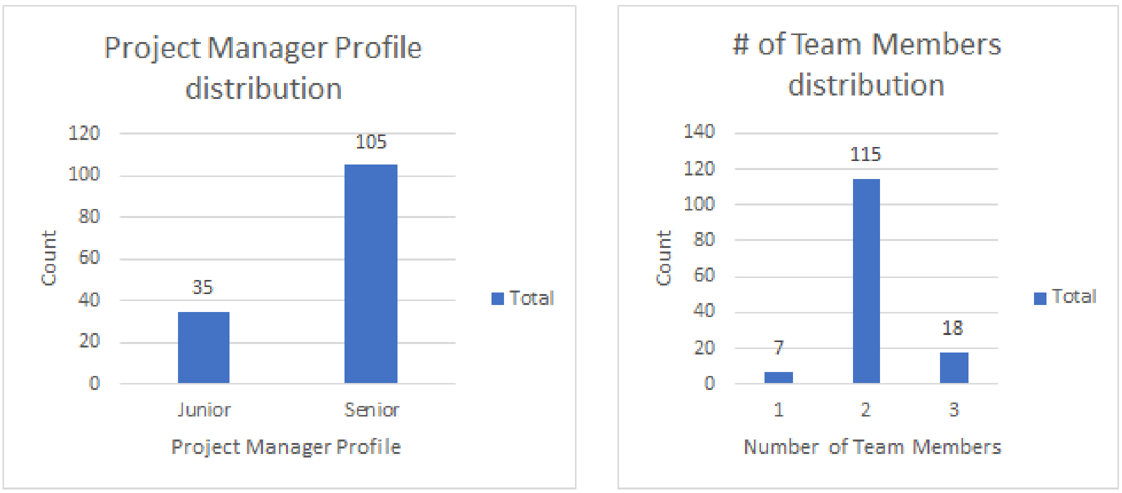


Figure 3.5: Project manager profile (left) and number of team members distribution (right)

Class imbalance was previously mentioned in this section, and it is an important problem to detect prior to training a machine learning model, as explained in section 2.4.1. An example of an unbalanced target variable in the data set for this application can be seen in figure 3.1, where risks with impact level 2 are almost twice as frequent as risks with impact level 3, and four times as much when compared to risks with impact level 1.

In order to try to overcome this problem in the data set, oversampling was used to try to provide a better balance to the risk impact and likelihood distributions. Undersampling was discarded due to the low number of records available in the data set, and this technique would reduce that number even further, not providing the models with sufficient data to provide trustworthy results. Details about the implementation of oversampling at the application level are presented in Chapter 4 and the evaluation results of the model with the oversampled data set is presented in Chapter 5, in comparison to those with the original data set.

3.3 Summary

The data used for this project was provided by one of the project's partner companies: Strongstep. However, since the information was provided through several spreadsheets, its organization and structure needed to be improved before using the data to train the machine learning models. This was done by creating a new spreadsheet with only the most relevant information from the various spreadsheets sent by Strongstep. In total, this spreadsheet had 140 samples and 13 variables, 3 of which were not used in training and were only added to identify the different projects and risks for which there was data available.

To improve the quality of this data before it was used to train the models, data preprocessing tasks were performed both in the data set itself and in the application prior to fitting the data into a classifier. One-hot encoding was used for the categorical independent attributes in the data set, and feature scaling through min-max scaling was used to scale the range of the values of the numeric independent attributes. Lastly, feature selection was performed to remove features with very low variance (such as Project Manager Profile and Number of Team Members) and could thus be having a negative impact on the predictions of the models.

Chapter 4

Solution Development

This chapter presents the application developed to tackle the problem mentioned in the opening chapter of this dissertation. Languages, frameworks and other tools used during the development of the application are presented in Section 4.1. Steps taken for hyperparameter optimization, a crucial task to maximize the performance of the machine learning models, are described in Section 4.2. The workflow of the application is presented in Section 4.3, describing how the application first trains the model, stores it, and is later loaded for prediction purposes when information about a new project is sent from SCRAIM to the application. Another approach taken to try to improve the predictive performance of the algorithms tested, namely the conversion from a multi-class classification problem to a binary classification one, is described in Section 4.4. Lastly, the setup used for the final tests is described in Section 4.5.

4.1 Technologies

Machine learning models can be created in numerous ways. Programs such as RapidMiner and WEKA provide a graphical user interface for users to train and test models, whereas machine learning libraries available in programming languages such as Java and Python allow for more flexibility and integration options for the developers.

For this application, three machine learning libraries were considered: Java-ML (Java), WEKA API (Java), and scikit-learn (Python). Java and Python were chosen as the preferential programming languages for this application due to the experience of the author, and the variety of machine learning packages available for these languages.

Java-ML was quickly discarded. Despite having a lot of alternatives for classifiers and methods for data preprocessing, the documentation available is not very good, and it has not been updated for over 7 years. WEKA was seen as a promising choice, as it was mentioned very often in the literature, but more often than not this was the graphical user interface version rather than the WEKA API. The documentation and examples available for the graphical user interface version are very good but using this would limit the integration options compared to the usage of its API, which had fewer examples and less useful documentation available. Finally, the scikit-learn package was chosen, as it solved the problems found in the previous packages. Scikit-learn provides great documentation and examples, a large number of algorithms for both supervised and unsupervised learning tasks, and is in active development with a great community.

As was previously mentioned in Chapter 3, oversampling was tested to try to resolve any class imbalance problems found in the data set. This was done through the imbalanced-learn [1] Python library, which provides various methods for both oversampling and undersampling tasks. Various oversampling methods were tested, with SVMSMOTE [48] being selected in the end as it was the consistently better performing oversampling method. A comparison between the performance of the models with and without oversampling techniques is presented in Chapter 5.

Furthermore, the eli5 [6] and Yellowbrick [13] packages were used to obtain a better understanding of the behavior of the models and their predictions. Eli5 was used to verify the importance of the features in the model, whereas the Yellowbrick package provided various methods for visualizing the metrics used for evaluating the model's predictions, such as multi-output AUC and precision-recall curve plots, as well as both one-dimensional and two-dimensional feature importance rankings. This information could then be used to explain the models' predictions to Strongstep and the rest of the PROMESSA project team, and identify possible points to improve on before deploying the software module developed. Examples of the information generated by these tools are shown in figures 4.1 and 4.2 further below in this section.

Lastly, to integrate with the project's main module, which enables the different services developed to be the called by the partner companies' own tools (such as Strongstep's SCRAIM), a REST API had to be developed, and this was done using the Flask [12] web framework for Python. However, as the developers of Flask do not recommend using its built-in development server in production due to stability and security reasons, Waitress [10], a Python Web Service Gateway Interface (or WSGI), was used to run the Flask application.

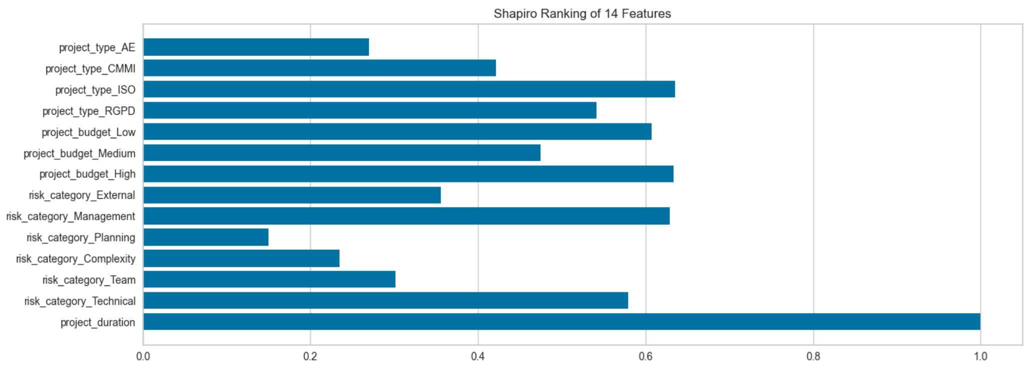


Figure 4.1: Feature ranking for the risk impact model showing the most and least important features during training

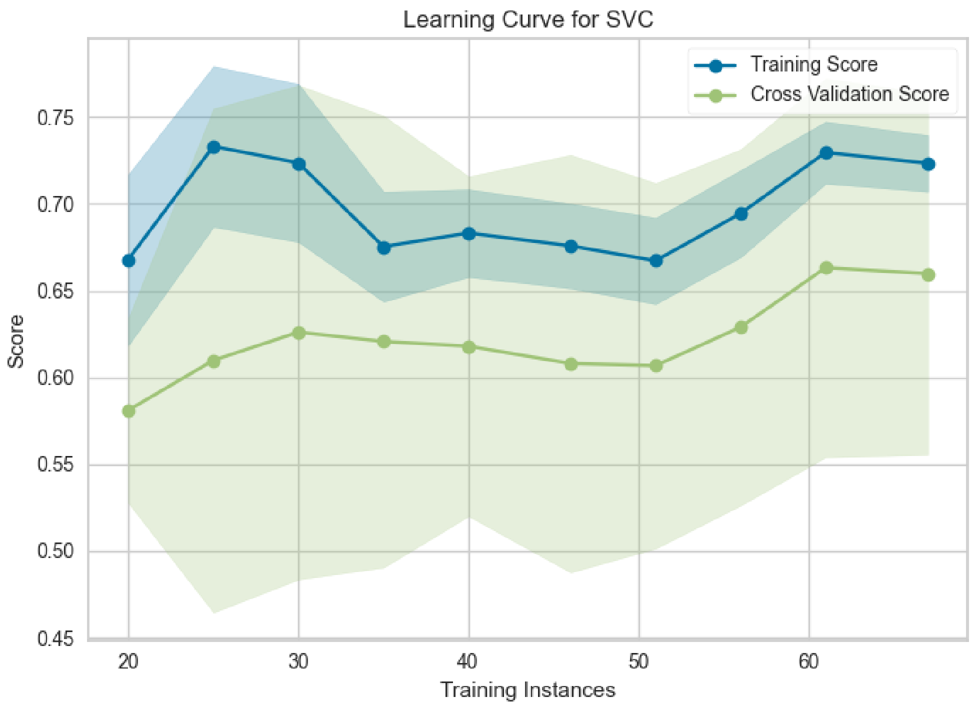


Figure 4.2: Learning curve for the risk impact model showing the evolution of training and cross-validation scores as the number of training instances grows

4.2 Hyperparameter Optimization

Hyperparameters are used in machine learning models to improve the learning process and differ from an algorithm's internal parameters in that they cannot be learned from the data during the training phase [65]. Using a Neural Network as an example, the weights of its hidden layers are internal parameters, whereas the activation function is a hyperparameter.

These are important as they can improve or reduce the quality of the model as it learns from the data set. The process to find the best set of hyperparameters for a given machine learning model is called hyperparameter optimization, or hyperparameterization.

Scikit-learn has two implementations of hyperparameter optimization algorithms that can be used for a classifier: GridSearchCV (for the grid search algorithm) and RandomizedSearchCV (for the random search algorithm). The two return the same information, which is the ideal set of parameters from the set that the user has sent to the GridSearchCV or RandomizedSearchCV object, but they do it in different ways. Grid search trains the model with every possible combination of hyperparameter values that were sent to the object, whereas random search reduces the complexity involved in the grid search algorithm by trying random combinations of the set of hyperparameters sent to the object instead [65]. The random search algorithm generally makes a low sacrifice in performance in validation scores for better execution times compared to grid search, but it is unreliable to train more complex models [65].

With this in mind, two Python scripts were created to test and compare both approaches in terms of execution time and validation scores. The scripts run the specified algorithm (grid search or randomized search) for various classification algorithms and save the results in separate text files to improve the readability and interpretability of the results obtained.

In both scripts, hyperparameter optimization was performed on the validation set, together with the use of cross-validation to improve the efficiency of this process. Specifically, scikit-learn's StratifiedKFold cross validator was used, with k = 4. K-fold divides the data set in folds and uses k-1 folds for training, with the final fold reserved for testing. Stratified K-fold does the same, but the folds are created while preserving the percentage of samples for the classes available in the data set, which is particularly useful in a problem such as this one where there is a clear imbalance in the classes, as was shown in the previous chapter.

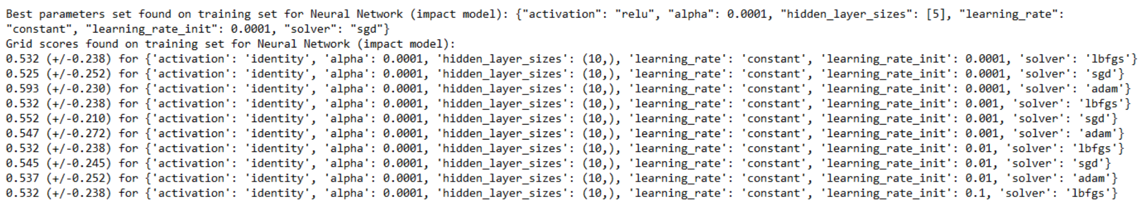
Both scripts were tested in order to compare the performance of both optimization algorithms, with a particular interest in finding out how different the optimal sets of parameters in randomized search would be compared to the more extensive grid search algorithm. In the case of algorithms with fewer parameter sets to test (*e.g.*, Decision Tree), the randomized search algorithm does go through the same complete set of parameter combinations, but it does not test every possible combination for algorithms with more extensive parameter sets (*e.g.*, Neural Network).

After the execution of the scripts, the results saved in the respective text files were analysed. The grid search parameterization script took 43 minutes and 19 seconds to run through the complete set of parameter combinations, while the randomized search script only took 1 minute and 29 seconds to go through a randomized set of parameter combinations.

Since the randomized search algorithm does not always test every possible combination of parameters like the grid search algorithm does, it was interesting to verify how many equal sets of parameters both scripts predicted. The grid search and randomized search scripts returned the same ideal set of parameters in approximately 84.6% of the tests, with 22 out of the 28 scenarios tested having the same set of optimal parameters. The differences detected were in algorithms which had the largest set of parameter combinations to test (*e.g.*, Gradient Boosted Tree, Neural Network), whereas both scripts returned the same ideal set of hyperparameters for most tree and ensemble-based classifiers (*e.g.*, Decision Trees, Random Forest), which had the fewest amount of parameter combinations to test.

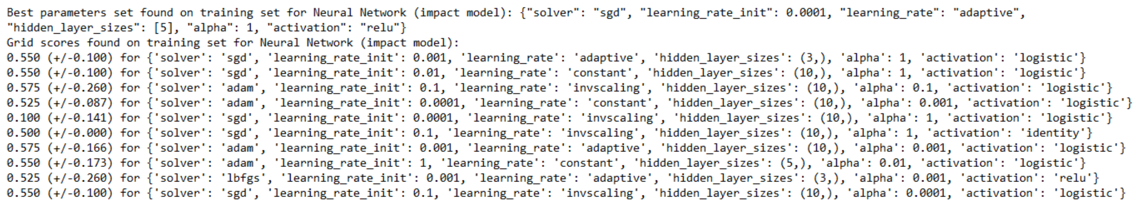
Randomized search showed to be a great option in situations where the list of parameters to test is very big and the time to run through all possible combinations is very high as a result, but for the purposes of this project, the results returned by the grid search algorithm were used to test the different algorithms. Examples of the outputs of these scripts can be seen in figures 4.3 and 4.4 below.

The selected algorithms were then tested on the project's data set with the ideal selection of hyperparameters returned by the grid search parameterization script, and metrics such as accuracy, precision, and recall were used to evaluate the performance of each algorithm on the project's data set, in a process that is described in greater detail in the next section.



```
Best parameters set found on training set for Neural Network (impact model): {"activation": "relu", "alpha": 0.0001, "hidden_layer_sizes": [5], "learning_rate":
"constant", "learning_rate_init": 0.0001, "solver": "sgd"}
Grid scores found on training set for Neural Network (impact model):
0.532 (+/-0.238) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.0001, 'solver': 'lbfgs'}
0.525 (+/-0.252) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.0001, 'solver': 'sgd'}
0.593 (+/-0.230) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.0001, 'solver': 'adam'}
0.532 (+/-0.238) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'solver': 'lbfgs'}
0.552 (+/-0.210) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'solver': 'sgd'}
0.547 (+/-0.272) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'solver': 'adam'}
0.532 (+/-0.238) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.01, 'solver': 'lbfgs'}
0.545 (+/-0.245) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.01, 'solver': 'sgd'}
0.537 (+/-0.252) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.01, 'solver': 'adam'}
0.532 (+/-0.238) for {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1, 'solver': 'lbfgs'}
```

Figure 4.3: Grid search parameterization script output example



```
Best parameters set found on training set for Neural Network (impact model): {"solver": "sgd", "learning_rate_init": 0.0001, "learning_rate": "adaptive",
"hidden_layer_sizes": [5], "alpha": 1, "activation": "relu"}
Grid scores found on training set for Neural Network (impact model):
0.550 (+/-0.100) for {'solver': 'sgd', 'learning_rate_init': 0.001, 'learning_rate': 'adaptive', 'hidden_layer_sizes': (3,), 'alpha': 1, 'activation': 'logistic'}
0.550 (+/-0.100) for {'solver': 'sgd', 'learning_rate_init': 0.01, 'learning_rate': 'constant', 'hidden_layer_sizes': (10,), 'alpha': 1, 'activation': 'logistic'}
0.575 (+/-0.260) for {'solver': 'adam', 'learning_rate_init': 0.1, 'learning_rate': 'invscaling', 'hidden_layer_sizes': (10,), 'alpha': 0.1, 'activation': 'logistic'}
0.525 (+/-0.087) for {'solver': 'adam', 'learning_rate_init': 0.0001, 'learning_rate': 'constant', 'hidden_layer_sizes': (10,), 'alpha': 0.001, 'activation': 'logistic'}
0.100 (+/-0.141) for {'solver': 'sgd', 'learning_rate_init': 0.0001, 'learning_rate': 'invscaling', 'hidden_layer_sizes': (10,), 'alpha': 1, 'activation': 'logistic'}
0.500 (+/-0.000) for {'solver': 'sgd', 'learning_rate_init': 0.1, 'learning_rate': 'invscaling', 'hidden_layer_sizes': (10,), 'alpha': 1, 'activation': 'identity'}
0.575 (+/-0.166) for {'solver': 'adam', 'learning_rate_init': 0.001, 'learning_rate': 'adaptive', 'hidden_layer_sizes': (10,), 'alpha': 0.001, 'activation': 'logistic'}
0.550 (+/-0.173) for {'solver': 'adam', 'learning_rate_init': 1, 'learning_rate': 'constant', 'hidden_layer_sizes': (5,), 'alpha': 0.01, 'activation': 'logistic'}
0.525 (+/-0.260) for {'solver': 'lbfgs', 'learning_rate_init': 0.001, 'learning_rate': 'adaptive', 'hidden_layer_sizes': (3,), 'alpha': 0.001, 'activation': 'relu'}
0.550 (+/-0.100) for {'solver': 'sgd', 'learning_rate_init': 0.1, 'learning_rate': 'invscaling', 'hidden_layer_sizes': (10,), 'alpha': 0.0001, 'activation': 'logistic'}
```

Figure 4.4: Randomized search parameterization script output example

4.3 Solution Workflow

While a considerable amount of the classifiers tested can perform multi-output classification (and Scikit-learn's MultiOutputClassifier allows the ones that natively cannot do it to do so), multi-output classifiers were hard to fit into the sampling methods. It was also difficult to fit them into data visualization and analysis tools like Yellowbrick, which was used to visualize and analyse the importance of the features used to train the models. Because of this, and also through the analysis of the classifiers presented in the previous chapter, a decision was made to create two models instead of one multi-output classification model: one to predict the risk impact, and one to predict the risk likelihood. This way it was easier to use oversampling and verify its effect on the predictive performance of the models, and the Yellowbrick feature importance visualizers could be used to greater effect.

After the training data set was finalized and all the necessary tests were performed to choose the most appropriate algorithms for the models, the risk impact and likelihood models were trained and saved.

The application's entry script, *main.py*, loads the required modules of the application, loads the risk impact and likelihood models, and brings the Flask server online.

Most of the application's logic is stored in different scripts, which isolate each functionality available to the module level and consequently improves maintainability and debugging processes in the application.

The application's file structure consists of 5 folders:

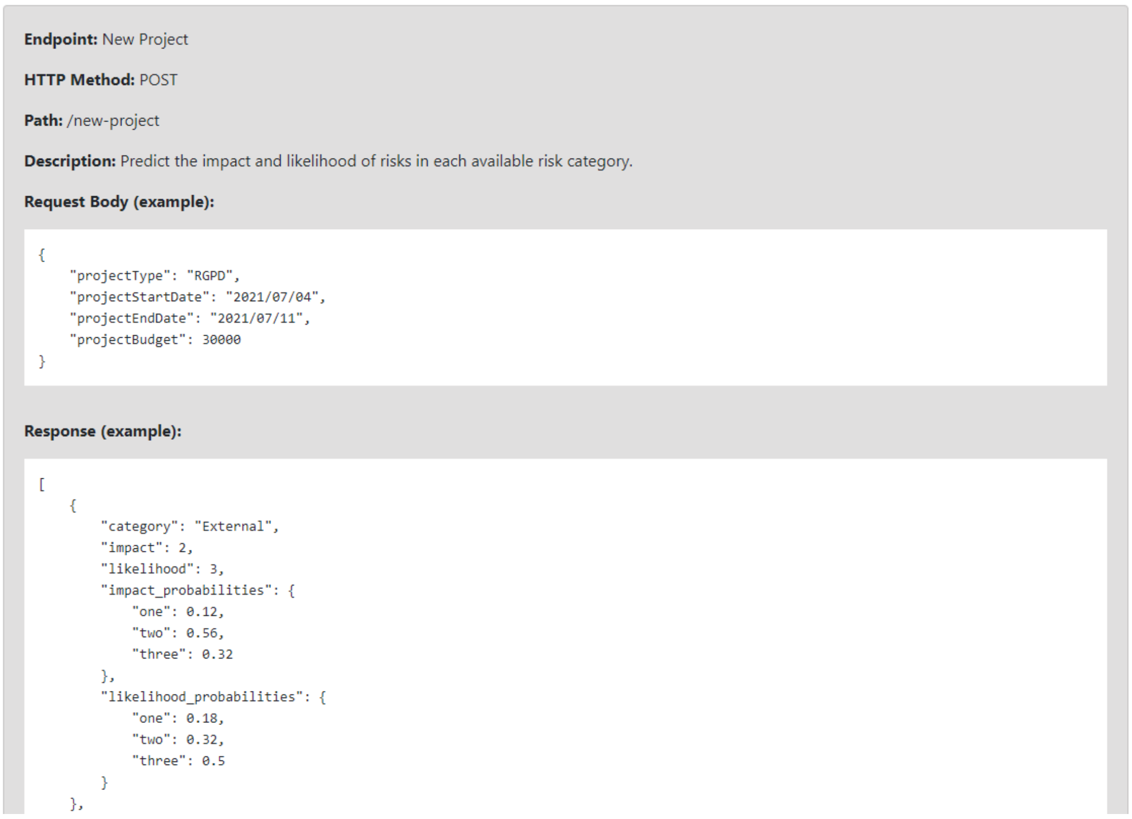
- **data** - folder used to store the spreadsheets used for training the model and analysis of the data used.

- **modules** - this folder isolates the different functionalities of the application to separate modules. It contains a file for the model logic, one for the REST API and Flask server logic, and another to store utility methods and others that are commonly reused throughout the application.

- **scripts** - this folder is similar to the modules one as it also stores various Python scripts, but these are isolated scripts used for things like testing new code logic before it is added to the application (*e.g.*, oversampling code logic). The difference is that the files in the modules folder are directly used in the application, whereas these scripts do not have any impact on the application when it is running.

- **storage** - folder used for persistence of the trained models, encoders, and anything else that needs to be re-used after the models have been trained.

- **web** - folder used to store static content served over the Flask server, such as the REST API documentation.



Endpoint: New Project

HTTP Method: POST

Path: /new-project

Description: Predict the impact and likelihood of risks in each available risk category.

Request Body (example):

```
{
    "projectType": "RGPD",
    "projectStartDate": "2021/07/04",
    "projectEndDate": "2021/07/11",
    "projectBudget": 30000
}
```

Response (example):

```
[
    {
        "category": "External",
        "impact": 2,
        "likelihood": 3,
        "impact_probabilities": {
            "one": 0.12,
            "two": 0.56,
            "three": 0.32
        },
        "likelihood_probabilities": {
            "one": 0.18,
            "two": 0.32,
            "three": 0.5
        }
    },
```

Figure 4.5: Standalone REST API documentation example

While the application can work as a standalone application by simply installing its dependencies listed in the *requirements.txt* file and then running the application's entry script, for the purposes of the PROMESSA project it needs to be integrated with the project's main module, which was developed by other members of the project team. As a result, various steps were taken to make the application available as a Python package: a *setup.py* file was created, which serves as the build script and contains information such as the project name, version, creator details, what Python version is required, among others.

Afterwards, changes were made in the main PROMESSA module to import the risk management module. The PROMESSA *Pipfile* was updated to include the risk management module as a dependency, and the necessary files were updated to register the risk management API endpoints in the main PROMESSA module, as well as create the Swagger API documentation. The API documentation is also available in the Flask server itself, in order to make it available if the application is used in standalone mode rather than in an integration such as the one with the main PROMESSA module, as seen in figure 4.5 above.

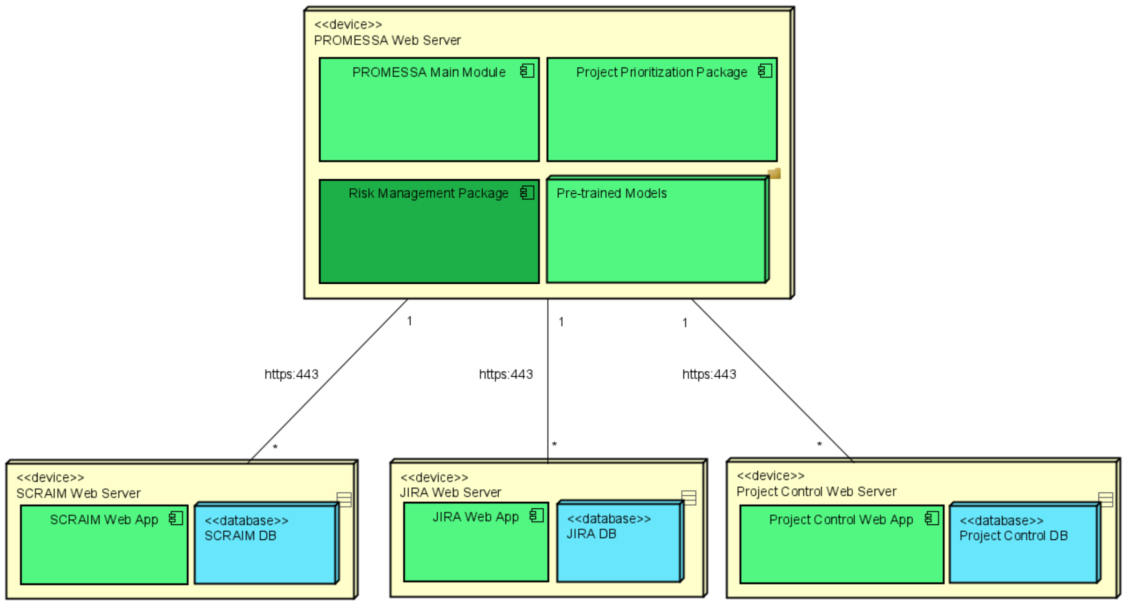


Figure 4.6: Deployment diagram explaining the PROMESSA project architecture

Figure 4.6 above shows the contents of the PROMESSA web server, and the applications that communicate with the PROMESSA main module: SCRAIM, JIRA, and Project Control. The different applications created for the different use cases identified for the PROMESSA project (*e.g.*, Project Prioritization, Risk Management) are stored in the server as dependencies of the main module, together with the pre-trained machine learning models of the different applications. The reason for this is to avoid unnecessarily storing the different data sets in the web server.

As mentioned before, when the application's entry script is run, it loads the models that were trained beforehand using the chosen algorithms. As was presented in Chapter 2 there are various algorithms that are commonly used for classification tasks such as the ones in this problem, so which algorithm to use in the application was an important decision.

After several discussions about this with members of the project's partner company, it was clear that interpretability was an important point to focus on when choosing algorithms to test. Interpretability in machine learning is defined as "the degree to which a human can understand the cause of a decision" [47].

Interpretable machine learning models make it easier to understand not only the prediction made by the model, but more importantly why that prediction was made. In the event that a prediction that does not exactly correspond to what was expected occurs, developers can use this information to identify possible problems in the data set, the model, or possibly both. However, there is a trade-off involved with interpretable machine learning algorithms, namely the fact that predictive performance tends to drop in these algorithms.

With that in mind, and considering the classifiers available in the Scikit-learn package, a short-list of algorithms was created in order to be tested before choosing which one was going to be used for the model. This way it was possible to compare the execution and predictive performance of each algorithm and decide if the trade-offs involved with interpretable machine learning algorithms were too much to consider them for the final models, or if the interpretability provided was worth any possible performance drops.

A Python script was created to test all of the algorithms in the list. The script tests each classifier in the list with a combination of the training and validation data sets used for training, and uses the remaining data available for testing. The script calculates the evaluation metrics used (accuracy, AUC, f-measure, precision, recall) for each classifier and automatically writes the results in an Excel spreadsheet using the openpyxl library. The algorithms tested use only multi-class classification for both impact and likelihood predictions, with 1 (Low), 2 (Medium), 3 (High) being the possible classes of those target variables. No other previously mentioned techniques (*e.g.*, oversampling) are used. The results of the execution of this script can be seen in table 4.1 below.

In the table below, a parameter listed with two values separated by a vertical bar (*e.g.*, learning_rate=0.01 | 0.1) indicates that the value on the left was used in the Impact model, and the value on the right was used in the Likelihood model. If the parameter only has one value, it means that the same parameter was used for both the Impact and Likelihood models (*e.g.*, loss='deviance').

Various tree and estimator-based classifiers were tested: Decision Tree, Random Forest, and Gradient Boosted Trees. Of these, the Decision Tree algorithm performed consistently worse for both the impact and likelihood target variables, only improving the precision score for the likelihood model when compared to the results of Gradient Boosted Trees and matching its accuracy in the same model, but returning worse results elsewhere. As a result, Random Forest and Gradient Boosted Trees were selected for further testing as two of the interpretable algorithms to test, as these offer the possibility to visualize each of the estimators used by the algorithm in a tree-like structure (example shown in figure 4.7) and, more importantly, the ability to estimate the importance of each feature used in training.

Table 4.1: Evaluation metrics results for the different algorithms tested (I = Impact, L = Likelihood)

Algorithm	**Parameters**	**Accuracy**		**AUC**		**F-Measure**		**Precision**		**Recall**	
		I	**L**	**I**	**L**	**I**	**L**	**I**	**L**	**I**	**L**
Decision Tree [20]	criterion='gini' max_features='sqrt' | None max_depth=3 min_samples_leaf=1	0.50	0.52	0.61	0.58	0.44	0.48	0.44	0.50	0.50	0.52
Random Forest [21]	criterion='gini' max_features='sqrt' n_estimators=100 max_depth=3 min_samples_leaf=1	0.63	0.59	0.71	0.62	0.57	0.52	0.56	0.52	0.63	0.59
Bernoulli Naive Bayes [45]	alpha=0.00001 | 1 binarize=1.0 | 0.0 fit_prior=True	0.58	0.53	0.50	0.62	0.43	0.50	0.34	0.48	0.58	0.53
Complement Naive Bayes [54]	alpha=1 norm=True	0.44	0.61	0.68	0.62	0.41	0.56	0.60	0.58	0.44	0.61
Multinomial Naive Bayes [45]	alpha=1 fit_prior=True	0.54	0.55	0.67	0.61	0.50	0.51	0.50	0.50	0.54	0.55
Gradient Boosted Trees [29]	loss='deviance' criterion='friedman_mse' learning_rate=0.01 | 0.1 max_features='sqrt' | None max_depth=3 | 3 min_samples_leaf=1 n_estimators=100 | 100	0.59	0.52	0.69	0.58	0.51	0.49	0.49	0.48	0.59	0.52
Neural Network [30]	hidden_layer_sizes=10 activation='identity' alpha=0.0001 solver='sgd' learning_rate='constant' learning_rate=0.0001 | 1	0.58	0.27	0.60	0.58	0.50	0.13	0.48	0.09	0.58	0.27
	activation='logistic' Remaining parameters are the same	0.58	0.57	0.48	0.62	0.43	0.51	0.34	0.47	0.58	0.57
	activation='tanh' Remaining parameters are the same	0.52	0.55	0.55	0.64	0.46	0.51	0.42	0.52	0.52	0.55
	activation='relu' Remaining parameters are the same	0.52	0.57	0.63	0.59	0.49	0.49	0.54	0.48	0.52	0.57
Support Vector Machine [50]	kernel='linear' C=1 gamma='scale'	0.56	0.56	0.70	0.59	0.52	0.51	0.53	0.49	0.56	0.56
	kernel='poly' Remaining parameters are the same	0.63	0.51	0.64	0.58	0.60	0.48	0.59	0.47	0.63	0.51
	kernel='rbf' Remaining parameters are the same	0.69	0.51	0.67	0.61	0.64	0.48	0.60	0.47	0.69	0.51
	kernel='sigmoid' Remaining parameters are the same	0.59	0.50	0.53	0.45	0.53	0.44	0.52	0.39	0.59	0.50
K-Nearest Neighbours [15]	n_neighbors=8 weights='uniform' algorithm='ball_tree'	0.62	0.56	0.66	0.58	0.58	0.50	0.55	0.50	0.62	0.56
RadiusNeighborsClassifier [15]	radius=0.2 weights='uniform' algorithm='ball_tree'	0.56	0.53	0.66	0.51	0.55	0.48	0.56	0.47	0.56	0.53
LogisticRegression [25]	solver='saga' | 'liblinear' multi_class='ovr' C=1.0 max_iter=100 penalty='l1' | penalty='l2'	0.61	0.55	0.72	0.63	0.56	0.51	0.54	0.52	0.61	0.55

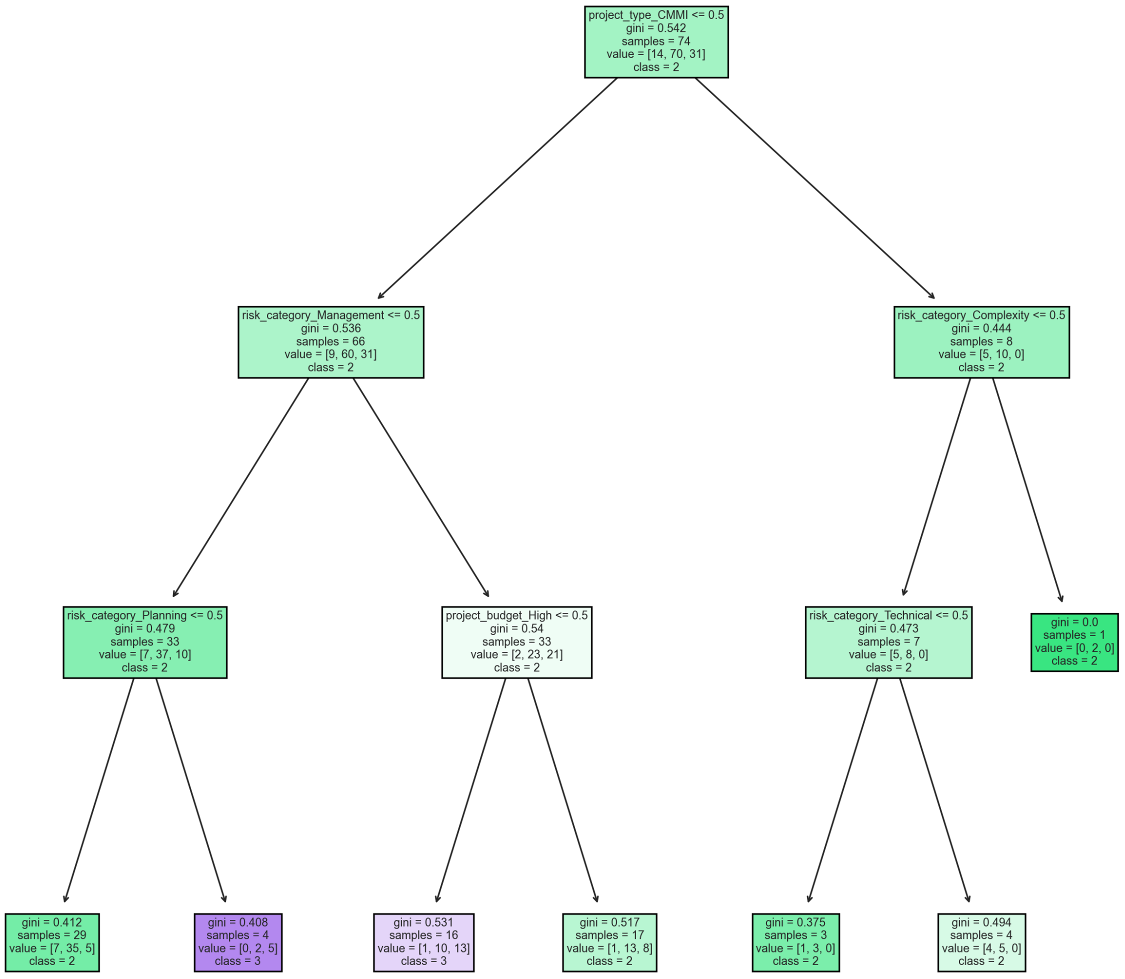


Figure 4.7: Example of an estimator tree from the Random Forest algorithm

As Scikit-learn provides various Naive Bayes classifiers, it was interesting to see how much the results would differ between each other, if at all. From the results in table 4.1, it is clear to see that the Bernoulli and Multinomial Naive Bayes outperform Complement Naive Bayes in the risk impact model, but the Complement Naive Bayes algorithm outperforms the former two algorithms in the risk likelihood model.

Additionally, it is important to compare the results of the Bernoulli and Multinomial Naive Bayes algorithms for the risk impact target variable, as it provides a clear example of an important point that was previously mentioned: accuracy alone is sometimes not enough to assess an algorithm's quality and applicability in a machine learning problem, particularly in ones where there is a clear class imbalance such as this one. The Bernoulli Naive Bayes algorithm has a slightly higher accuracy and recall than Multinomial Naive Bayes, but the latter presents significantly higher results in AUC score (0.17 higher), F-Measure (0.07 higher) and precision (0.16 higher). Naive Bayes was also added to the list of algorithms used for further testing, specifically the Multinomial Naive Bayes and Complement Naive Bayes algorithms, which were used for the risk impact and risk likelihood models, respectively.

Similarly to how various Naive Bayes algorithms were added to this testing script, the Neural Network algorithm was tested with four different activation functions and the Support Vector Machine algorithm was tested with four different kernels while keeping the rest of the hyperparameters obtained through the grid search parameterization script for both algorithms.

It is interesting to note that, in the case of the Neural Network algorithm, all four activation functions perform similarly for the risk impact target variable in terms of accuracy, F-Measure, precision, and recall, but show more noticeable differences in AUC score. Inversely, the four activation functions perform similarly in AUC score for the likelihood target variable but vary significantly in the remaining evaluation metrics. Neural Network with the identity and logistic activation functions showed the most consistent results for the risk impact and risk likelihood target variables, respectively, and were thus selected for further testing.

In the Support Vector Machine's case, this algorithm consistently produced very good results, particularly with the RBF kernel for the impact target variable, and the linear kernel for the likelihood target variable. It was also added to the list of algorithms used in further testing, specifically with the previously mentioned kernels for the risk impact and risk likelihood target variables.

Lastly, the K-Nearest Neighbours, Radius Neighbours, and Logistic Regression algorithms were also tested with the goal of picking another classifier to complete the list of algorithms used in further testing. Based on the results above, K-Nearest Neighbours was the clear choice, as it consistently outperformed the other two algorithms in both target variables save for a few exceptions (*e.g.*, AUC score), and thus completed the list of algorithms selected for further testing.

4.4 Converting a Multi-Class Classification Problem to a Binary Classification Problem

Due to the type of information the models in this application are predicting, this is clearly a classification problem. In this situation, the classes are represented by numeric values ranging from 1 to 3, where 1 represents a low risk impact or likelihood, 2 represents a medium risk impact or likelihood, and 3 represents a high risk impact or likelihood. These values also have a clear order to them: Low < Medium < High. Standard multi-class classification was used since development started, but in cases like this other approaches can also be used to try to maximize the performance of the models developed in this situation.

One of the most common approaches, presented by E. Frank and M. Hall [28], consists of turning the problem from a regular k-class classification problem (k being 3 in this situation) into k-1 binary classification problems, where the outputs are only 0 or 1.

Training is not performed with the data set as it was previously described in Chapter 3. Instead, k-1 binary classifiers are trained with k-1 data sets derived from the original data set, one of each of the k-1 class values [28] which, in this case, means two different data sets for two risk impact classifiers, and two different data sets for two risk likelihood classifiers.

Using risk impact as an example, the first classifier is trained with a new data set that changes the impact level 1 to 0, and impact levels 2 and 3 to 1. The second classifier is then trained with a new data set that changes the impact levels 1 and 2 to 0, and impact level 3 to 1. The same process is then applied for the risk likelihood classifiers. Figure 4.8 shows the original risk impact class distribution, the risk impact class distribution used for the first classifier, and the risk impact class distribution used for the second classifier, respectively, and makes it easier to understand the results of this process.

However, the predictions are still returned as they are in the original multi-class classification problem, but the way in which the results are obtained in multi-class format is different in the binary classification approach. Using the risk impact model as an example again:

- The probability of multi-class risk impact level 1 is equal to 1 minus the probability of the first binary risk impact classifier predicting risk impact level 1: 1 - Pr (Impact > 1 | X);
- The probability of multi-class risk impact level 3 is equal to the probability of the second binary risk impact classifier predicting risk impact level 1: Pr (Impact > 2 | X);
- Lastly, the probability of the middle class (multi-class risk impact level 2) is equal to 1 minus the probability of the second binary risk impact classifier predicting risk impact level 1, multiplied by the probability of the first binary risk impact classifier predicting risk impact level 1: (1 - Pr (Impact > 2 | X)) * Pr (Impact > 1).

After all the probabilities are calculated, the class with the highest probability is selected. This approach was tested and compared to regular *k*-class classification (together with oversampling in both situations) in the various algorithms that were selected for the final testing and evaluation processes as previously mentioned, and the results are shown in section 5.2.

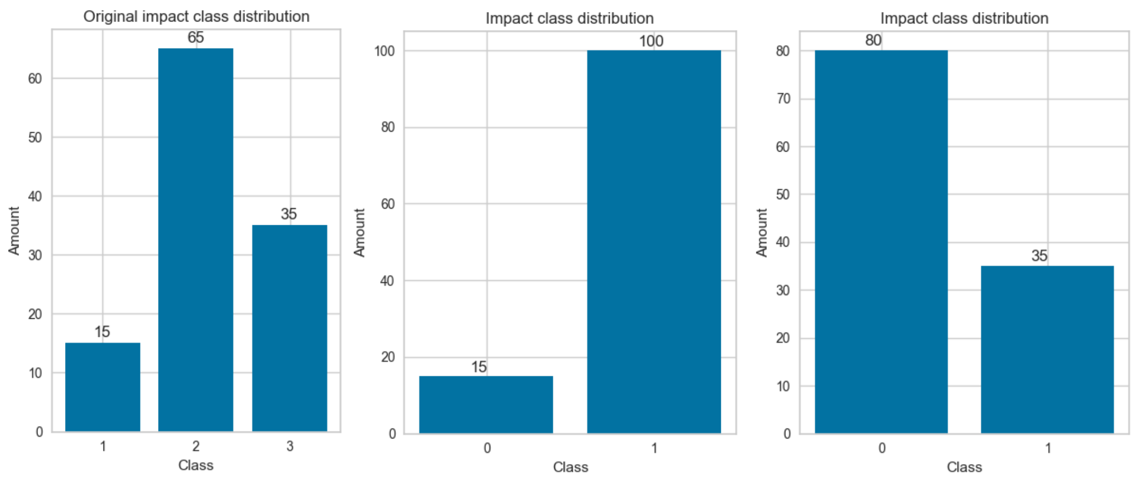


Figure 4.8: Risk impact class distribution comparison: original (left), class distribution used for the first classifier (centre), class distribution used for the second classifier (right)

4.5 Experimental Setup

In order to assess the quality of the models, various evaluation metrics were used: accuracy, AUC, F-measure, precision, and recall. These metrics provide different information so analysing their results can provide a lot of insight into the quality of the predictions made by the models. Additionally, confusion matrices were used to analyse what the models were predicting and, more importantly, where they could be failing in some predictions.

Stratified K-fold cross-validation (with $k = 4$) was used to obtain the results presented. The 40 data samples used for validation are set aside temporarily, and the remaining 100 data samples are used in the StratifiedKFold instance which, with $k = 4$, splits these samples into 4 different groups of 75 training and 25 test samples. As this is a small data set with few samples to train the models, the remaining 40 validation data samples are added to the 75 training samples on each cross-validation group. That way, 115 isolated data samples are fitted to train the model, and tests are performed on the remaining 25 test samples on each group.

The tests were performed on 6 different machine learning algorithms, 3 considered interpretable (Gradient Boosted Trees, Random Forest, and Naive Bayes), and 3 considered non-interpretable (Neural Network, Support Vector Machine, and K-Nearest Neighbours), with the goal of comparing their results and making a decision regarding whether or not the common performance trade-offs in non-interpretable algorithms are worth the reduced amount of information that can be obtained from these.

The tests for binary classification conversion were performed on the same set of machine learning algorithms, with the goal of not only comparing their results and assessing the possible performance trade-offs involved in interpretable algorithms, but to also compare the results of this approach with the ones obtained by the original multi-class classification approach.

Both sets of tests were run 10 times for each algorithm, and the results presented in the next chapter are the average of the results of those runs.

4.6 Summary

A Python application was developed to tackle the previously described problem. It uses the scikit-learn package for the machine learning tasks, and a combination of the Flask and Waitress packages to make a REST API available, whose endpoints are called by SCRAIM when a new project is created in order to obtain the impact and likelihood of risks occurring in various risk categories.

The application's entry script loads the modules required by the application, loads the trained models, and brings the Flask server online. Strongstep's project management system, SCRAIM, calls the required endpoints when a new project is created. Information about the project such as budget, duration, and type are sent in the request, and the endpoint responsible for handling the request uses the machine learning models that were loaded during the application's startup process to make a prediction on that new data. After the endpoint receives the predicted risk impact and likelihood for the different risk categories from the models, it prepares the response and sends it back to SCRAIM.

In an attempt to maximize the accuracy of the results of the models, hyperparameter optimization was performed through grid search using scikit-learn's GridSearchCV to try to find the best set of hyperparameters for the different algorithms that were tested. Another approach tested was the conversion of the problem at hand from a multi-class classification problem to a binary classification one, following a well-known approach in this research area.

A wide variety of classifiers was tested after obtaining the ideal set of hyperparameters through a grid search parameterization script, with the goal of reducing this list in order to perform more extensive testing with the chosen algorithms. As a result of this process, the Random Forest, Gradient Boosted Trees, Naive Bayes, Neural Network, Support Vector Machine, and K-Nearest Neighbours algorithms were selected, resulting in a diverse selection of algorithm types, whose results could then be further analysed in terms of interpretability and performance.

Chapter 5

Results and Discussion

The main objectives of the work done for this dissertation were the study of the state-of-the-art of the research area related with risk management in software projects, particularly with the application of machine learning techniques to improve risk management processes, and the development of a software module capable of predicting the impact and likelihood of different risk categories in a new software project before development begins.

The module developed was described in the previous section, whereas the current chapter presents not only the results of tests made with the different algorithms that were selected, but also a discussion regarding the results obtained and the potential uses cases for the information returned by the application developed, taking into consideration the results obtained. The former is presented in the first two sections - multi-class classification results are presented in section 5.1, whereas binary classification results are presented in section 5.2 - and the latter is presented in section 5.3.

5.1 Multi-class Classification Results

5.1.1 Risk Impact

Table 5.1 shows the results of the tests performed for the risk impact model, which makes predictions on the risk impact target variable, and an example of a confusion matrix of predictions made by one of the algorithms tested for this model is shown in figure 5.1.

As expected, based on the analysis of interpretable algorithms and the trade-offs associated with them, the interpretable algorithms display mostly worse results than their non-interpretable counterparts, with the exception of Random Forest.

While the Random Forest algorithm presents strong results in most metrics (particularly accuracy, AUC, and recall), the overall best results still come from a non-interpretable algorithm: Support Vector Machine, with the highest value for accuracy (69%), F-Measure (0.64), precision (0.60), and recall (0.69), surpassed only by two algorithms in AUC score: Neural Network and

Random Forest. Another non-interpretable algorithm - K-Nearest Neighbours - presents the next highest scores for accuracy (68%), F-Measure (0.63), and recall (0.68).

Table 5.1: Results of the risk impact model evaluation with different algorithms

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.54	0.64	0.52	0.55	0.54
Random Forest	0.63	0.69	0.57	0.55	0.63
Multinomial Naive Bayes	0.54	0.67	0.50	0.50	0.54
Neural Network Identity activation function	0.62	**0.71**	0.58	0.58	0.62
Support Vector Machine RBF kernel	**0.69**	0.68	**0.64**	**0.60**	**0.69**
K-Nearest Neighbours	0.68	0.64	0.63	0.59	0.68

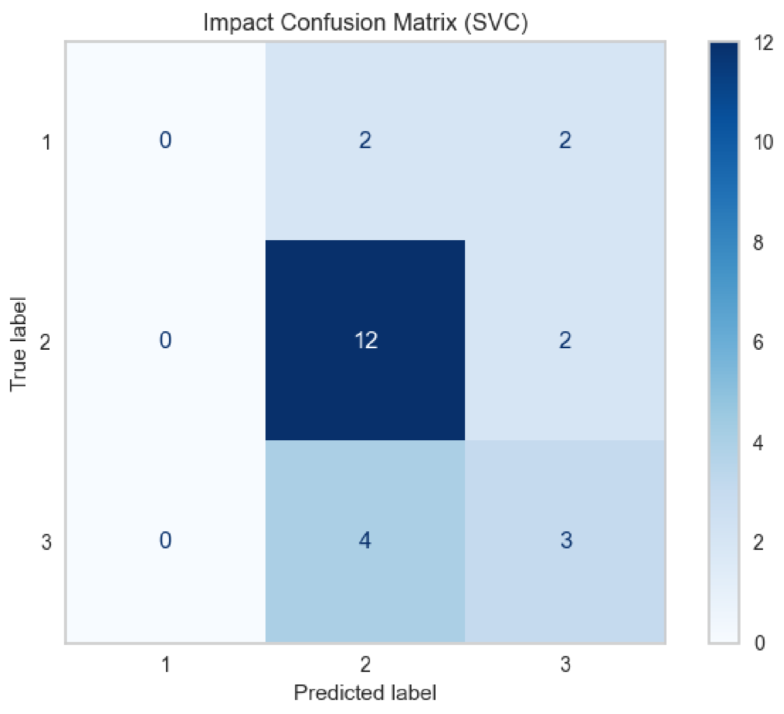


Figure 5.1: Example of a confusion matrix of predictions made by the risk impact model (Support Vector Machine)

As was previously mentioned, oversampling was used to try to improve the model's predictions in this unbalanced data set. SVMSMOTE was applied on the minority class (impact level 1) rather than all classes except the majority one (impact levels 1 and 3), as while there was some discrepancy between the number of risks with impact levels 2 and 3, that problem was considerably worse when comparing risks with impact levels 1 and 2. Figure 5.3 shows the original risk impact class distribution (left) and the risk impact class distribution after the use of SVMSMOTE on the training data (right), respectively.

Unfortunately, as the results in table 5.2 show, oversampling did not manage to improve the results on the evaluation metrics used, returning mostly worse results all around with some exceptions (*e.g.*, minor differences in Random Forest and Support Vector Machine's AUC, 0.03 higher precision in Neural Network). However, by analysing the confusion matrix of the risk impact model trained with the original data set (shown in figure 5.1) and the confusion matrix of the risk impact model trained with oversampling applied to the training data (shown in figure 5.2), it becomes clear where the latter model is failing.

Table 5.2: Comparison of the risk impact model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.51	0.62	0.50	0.53	0.51
Random Forest	**0.54**	**0.70**	0.51	0.53	**0.54**
Multinomial Naive Bayes	0.46	0.66	0.46	0.54	0.46
Neural Network Identity activation function	**0.54**	**0.70**	0.54	**0.61**	**0.54**
Support Vector Machine RBF kernel	**0.54**	0.69	**0.56**	**0.61**	**0.54**
K-Nearest Neighbours	0.50	0.66	0.53	0.60	0.50

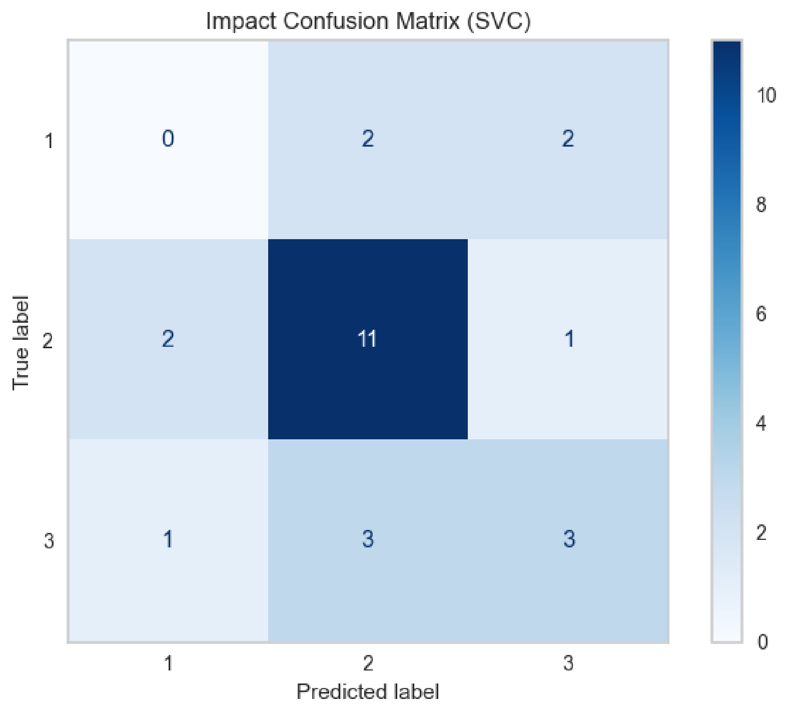


Figure 5.2: Example of a confusion matrix of predictions made by the risk impact model (Support Vector Machine) with oversampling applied to the training data

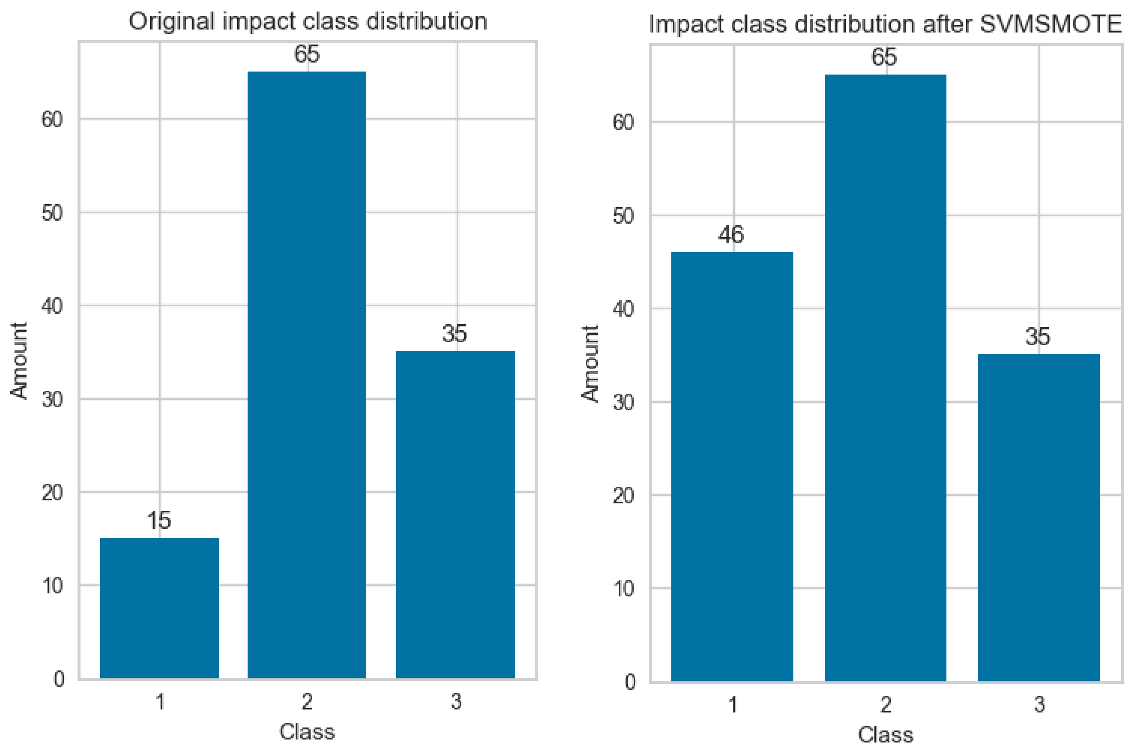


Figure 5.3: Risk impact class distribution without (left) and with oversampling applied to the training data (right)

Risk impact level 3 is predicted 7 times in the tests performed without oversampling, and 6 times in the tests performed with oversampling applied to the training data, but the majority class (risk impact level 2) is no longer the only other class predicted by the model, with 4 predictions of risk impact level 1 when oversampling is applied to the training data. This is the expected behaviour with the more balanced class distribution that is obtained with the use of oversampling, but it did not improve the results for any algorithm tested, aside from the previously mentioned exceptions.

Despite overall worse results compared to the tests performed for the risk impact model (without oversampling), it is interesting to note the proximity when comparing the results of the interpretable algorithms with the results of the non-interpretable algorithms. The results of the interpretable algorithms are now closer to the results of the non-interpretable algorithms, mostly due to the considerable drop in results obtained by the non-interpretable algorithms, particularly Support Vector Machine and K-Nearest Neighbours.

As for the best overall results, it is a mix between interpretable and non-interpretable algorithms: Random Forest matched the best scores in accuracy (54%), AUC score (0.70) and recall (0.54), while Support Vector Machine and Neural Network equalled those results. Additionally, Neural Network and Support Vector Machine obtained the highest score in precision (0.61), while the latter had the best overall score in F-Measure (0.56).

5.1.2 Risk Likelihood

The tests for the risk likelihood model followed the same workflow as the tests for the risk impact model, and the results of these are shown in table 5.3 below, together with an example of a confusion matrix of predictions made by one of the algorithms tested, which is shown in figure 5.4.

Table 5.3: Results of the risk likelihood model evaluation with different algorithms

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.52	0.55	0.49	0.48	0.52
Random Forest	0.58	**0.62**	0.53	0.52	0.58
Complement Naive Bayes	**0.61**	**0.62**	**0.56**	**0.58**	**0.61**
Neural Network Logistic activation function	0.57	0.61	0.50	0.47	0.57
Support Vector Machine Linear kernel	**0.61**	0.59	0.53	0.53	**0.61**
K-Nearest Neighbours	0.56	0.58	0.50	0.50	0.56

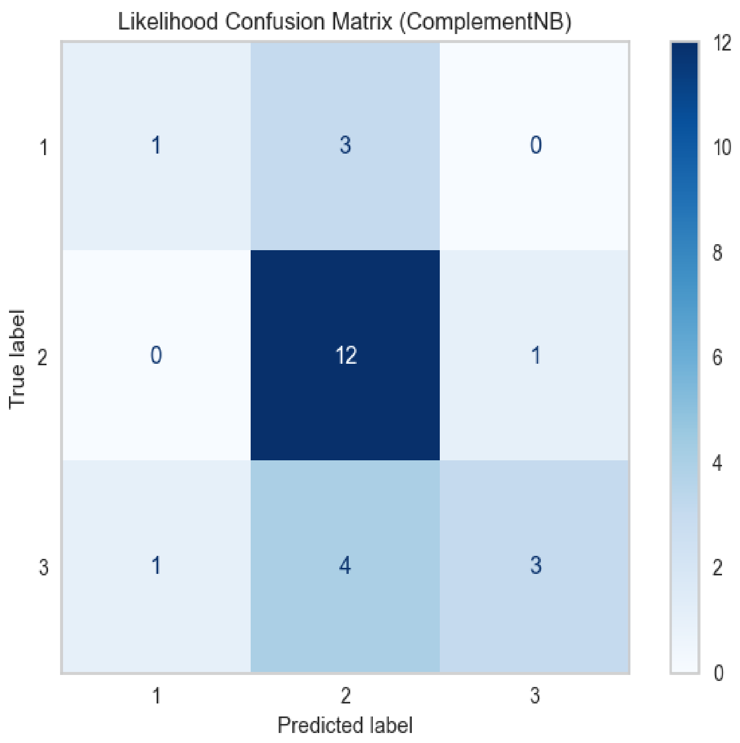


Figure 5.4: Example of a confusion matrix of predictions made by the risk likelihood model (Naive Bayes)

Complement Naive Bayes stands out as the overall best performing algorithm when it comes to predicting risk likelihood levels, with the highest value for accuracy (61%), AUC score (0.62), F-Measure (0.56), precision (0.58) and recall (0.61). A non-interpretable algorithm (Support Vector Machine) matches the best scores in accuracy and recall and presents the next best scores in F-Measure (0.53, also obtained by Random Forest) and Precision (0.53).

Aside from Gradient Boosted Trees, which presents the lowest scores in all metrics except precision, the difference between the results of the interpretable and non-interpretable algorithms is not as obvious in predictions for the risk likelihood target variable compared to what was seen in the predictions for the risk impact target variable. In fact, two interpretable algorithms (Random Forest and Complement Naive Bayes) outperform the remaining non-interpretable algorithms (Neural Network and K-Nearest Neighbours), in a clear contrast to what was seen in the results of the risk impact predictions.

As with the risk impact model, oversampling was also used to try to improve the model's predictions regarding risk likelihood, which also has an imbalanced class distribution. SVMSMOTE was applied only on the minority class (likelihood level 1). Figure 5.6 shows the original risk likelihood class distribution (left) and the risk likelihood class distribution after the use of SVMSMOTE on the training data (right), respectively.

Table 5.4: Comparison of the risk likelihood model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.52	0.45	0.49	0.48	0.52
Random Forest	0.56	0.59	0.53	0.54	0.56
Complement Naive Bayes	**0.63**	**0.61**	**0.59**	**0.62**	**0.63**
Neural Network Logistic activation function	0.51	0.60	0.50	0.51	0.51
Support Vector Machine Linear kernel	0.55	0.54	0.51	0.59	0.55
K-Nearest Neighbours	0.42	0.56	0.41	0.43	0.42

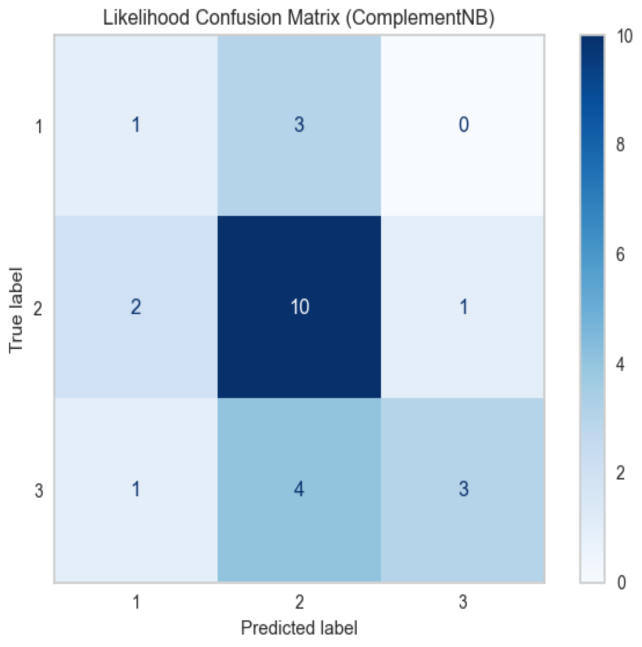


Figure 5.5: Example of a confusion matrix of predictions made by the risk likelihood model (Naive Bayes) with oversampling applied to the training data

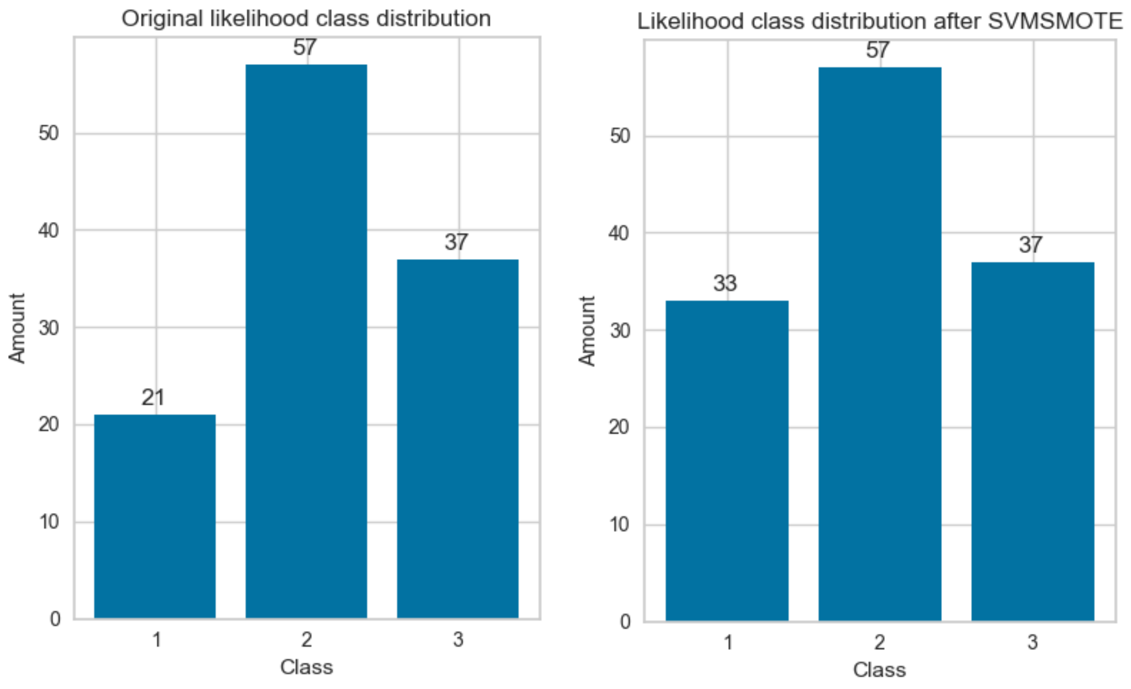


Figure 5.6: Risk likelihood class distribution without (left) and with oversampling applied to the training data (right)

As seen in the tests performed for the risk impact model, the results in table 5.4 show that, in the majority of cases, the use of oversampling did not improve the results of the metrics used to evaluate the risk likelihood model. By analysing the confusion matrix of the risk likelihood model trained with the original data set (shown in figure 5.4) and the confusion matrix of the risk likelihood model trained with oversampling applied to the training data (shown in figure 5.5), it becomes easier to understand this drop in the results obtained with oversampling applied. There are less predictions for the majority class (risk likelihood level 2) and more predictions are made for the minority class (risk likelihood level 1), to which oversampling was applied. It is the expected behaviour and is confirmed by the differences seen in the aforementioned confusion matrices.

Although the overall scores are worse when comparing to the tests performed in the risk likelihood model without oversampling, the differences here are not as high when compared to the differences seen in the risk impact model's results with and without oversampling applied to the training data. The interpretable algorithms mostly keep similar scores while improving on them in a few cases (most noticeably Naive Bayes' results), while the results of the non-interpretable algorithms drop more significantly in comparison to the results without oversampling, particularly in the case of the K-Nearest Neighbours algorithm.

Complement Naive Bayes stands out again as the overall best performing algorithm in the tests for the risk likelihood model, with the highest value for accuracy (63%), AUC score (0.61), F-Measure (0.59), precision (0.62) and recall (0.63). As for the next best scores, it is a mix between both types of algorithms, with Random Forest having the next highest accuracy (56%), F-Measure (0.53) and recall (0.56), and Support Vector Machine obtaining the next highest overall value of AUC score (0.60) and precision (0.59).

5.2 Binary Classification Results

5.2.1 Risk Impact

Table 5.5 shows the results of the tests performed for the risk impact model, and an example of a confusion matrix of predictions made by one of the algorithms tested for this model is shown in figure 5.7. Notice that, as mentioned in Chapter 4, the predictions are still made over the 3 different classes of risk impact and likelihood, but they are obtained differently with the conversion to binary classification.

Table 5.5: Results of the risk impact model evaluation with different algorithms using binary classification conversion

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.52	0.57	0.51	0.52	0.52
Random Forest	0.60	**0.64**	0.51	0.48	0.60
Multinomial Naive Bayes	0.54	0.63	0.49	0.49	0.54
Neural Network Identity activation function	0.61	0.62	0.56	0.54	0.61
Support Vector Machine RBF kernel	0.59	0.53	0.49	0.44	0.59
K-Nearest Neighbours	**0.68**	0.54	**0.63**	**0.58**	**0.68**

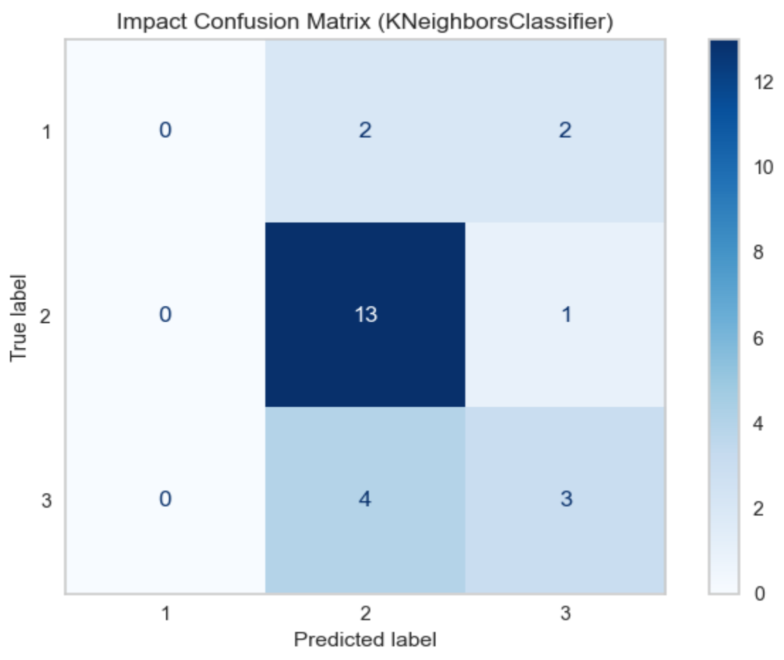


Figure 5.7: Example of a confusion matrix of predictions made by the risk impact model using binary classification conversion (K-Nearest Neighbours)

When it comes to the interpretable algorithms, Random Forest in particular presents good results (even obtaining the highest overall AUC score - 0.64). However, for the most part, these algorithms present worse scores than the non-interpretable ones, except for Support Vector Machine which only presents higher values of accuracy and recall but lower values in the remaining metrics when compared to the other two non-interpretable algorithms.

Despite this, the highest overall scores for accuracy (68%), F-Measure (0.63), precision (0.58) and recall (0.68) still come from a non-interpretable algorithm (K-Nearest Neighbours), as do the next highest overall scores (Neural Network), reflecting the expected performance difference when comparing non-interpretable algorithms and their interpretable counterparts.

In comparison with the multi-class classification results, Naive Bayes, Neural Network, and K-Nearest Neighbours keep similar scores, except in AUC score which is significantly lower using binary classification conversion. The remaining algorithms suffer a more noticeable drop in results using this technique, particularly Support Vector Machine and Random Forest.

As was the case in the multi-class classification tests, oversampling was also used in the binary classification conversion tests to try to improve the model's predictions in this unbalanced data set. With the use of two different data distributions after the conversion is performed (as explained in Chapter 4), SVMSMOTE was applied to both of these data sets on the minority class (0 in one set, 1 in the other). Figure 5.9 shows an example of the original class distribution after the conversion is performed (in this case, the original risk levels 1 and 2 are now 0, and risk impact level 3 is now 1), and the same class distribution after SVMSMOTE is applied on that data set, respectively.

This did not improve the majority of the results found in the tests for the risk impact model without oversampling, as can be seen in table 5.6. The only exceptions are the Support Vector Machine's AUC score and precision, Naive Bayes' precision, Random Forest's precision, and Neural Network's precision. Nonetheless, the interpretable algorithms mostly surpass the results of the non-interpretable algorithms, particularly in the cases of Random Forest and Naive Bayes.

Table 5.6: Comparison of the risk impact model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data and using binary classification conversion

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.46	0.56	0.45	0.50	0.46
Random Forest	**0.52**	0.62	**0.50**	**0.63**	**0.52**
Multinomial Naive Bayes	0.48	0.58	0.47	0.58	0.48
Neural Network Identity activation function	0.44	0.62	0.43	0.59	0.44
Support Vector Machine RBF kernel	0.49	**0.67**	0.46	0.58	0.49
K-Nearest Neighbours	0.27	0.55	0.25	0.52	0.27

Neural Network and K-Nearest Neighbours in particular take a significant hit compared to the values of the tests without oversampling applied to the training data, whereas the drop in results obtained by the interpretable algorithms is not as high. A non-interpretable algorithm - Random Forest - obtained the highest overall scores in this test for accuracy (52%), F-Measure (0.50), precision (0.63), and recall (0.52), surpassed by Support Vector Machine in AUC score (0.67 for Support Vector Machine, 0.62 for Random Forest).

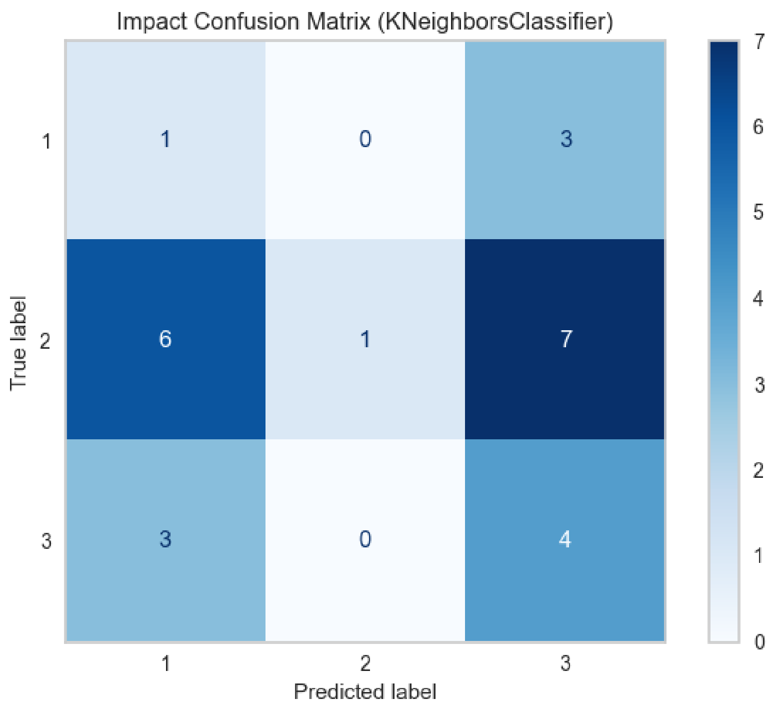


Figure 5.8: Example of a confusion matrix of predictions made by the risk impact model (K-Nearest Neighbours) with oversampling applied to the training data and using binary classification

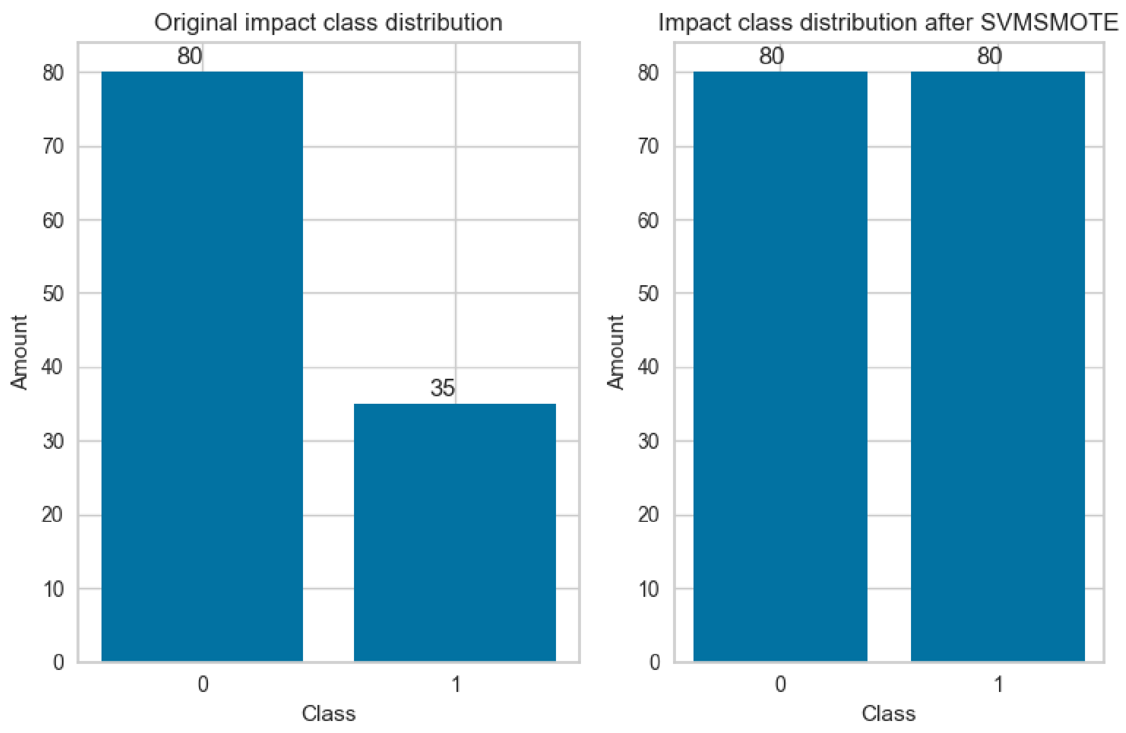


Figure 5.9: Risk impact class distribution without (left) and with oversampling applied to the training data (right) after the conversion to a binary distribution is performed

Similarly to what was observed in the multi-class classification tests with and without oversampling, the effects of oversampling in balancing the class distribution turn out to have the opposite effect in the results of the tests performed. This is noticeable by comparing not only the results in tables 5.5 and 5.6, but also the examples of confusion matrices in figures 5.7 and 5.8.

There are more predictions for non-majority classes when oversampling is applied, which is the expected behaviour with this technique, but it mostly fails to improve on the results obtained in the tests without oversampling applied to the training data.

5.2.2 Risk Likelihood

The tests for the risk likelihood model here followed the same workflow as the tests performed for the risk impact model. The results are shown in table 5.7, and an example of a confusion matrix of predictions made by one of the algorithms tested is shown in figure 5.10.

Table 5.7: Results of the risk likelihood model evaluation with different algorithms using binary classification conversion

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	0.53	0.51	0.50	0.50	0.53
Random Forest	**0.59**	0.60	**0.52**	**0.59**	**0.59**
Complement Naive Bayes	0.27	**0.63**	0.19	0.23	0.27
Neural Network Logistic activation function	0.58	0.59	0.49	0.54	0.58
Support Vector Machine Linear kernel	**0.59**	0.57	0.50	0.54	**0.59**
K-Nearest Neighbours	0.55	0.53	0.45	0.53	0.55

The results obtained by the Naive Bayes algorithm suffer a significant drop compared to the results obtained in the multi-classification tests, except in AUC score which is 0.01 higher here. The remaining interpretable algorithms mostly maintain similar scores compared to the multi-classification tests, with a few more noticeable exceptions such as Gradient Boosted Trees' lower AUC score and Random Forest's higher precision value.

Random Forest presented the highest overall value for accuracy (59%, matched by Support Vector Machine), F-Measure (0.52), precision (0.59) and recall (0.59, matched by Support Vector Machine), only surpassed by another interpretable algorithm - Naive Bayes - in AUC score.

Nonetheless, the conversion to a binary classification problem did not improve on the majority of the results. The example of a confusion matrix shown in figure 5.10 shows the importance of the K-fold cross-validation process, as in the first of four test sets the Random Forest algorithm is correct in 20 out of the 25 predictions it made (80% accuracy), but with the remaining tests with different folds and, consequently, different class distributions, the results eventually average out to the ones seen in table 5.7, presenting a clearer idea of the different algorithms' strengths and weaknesses.

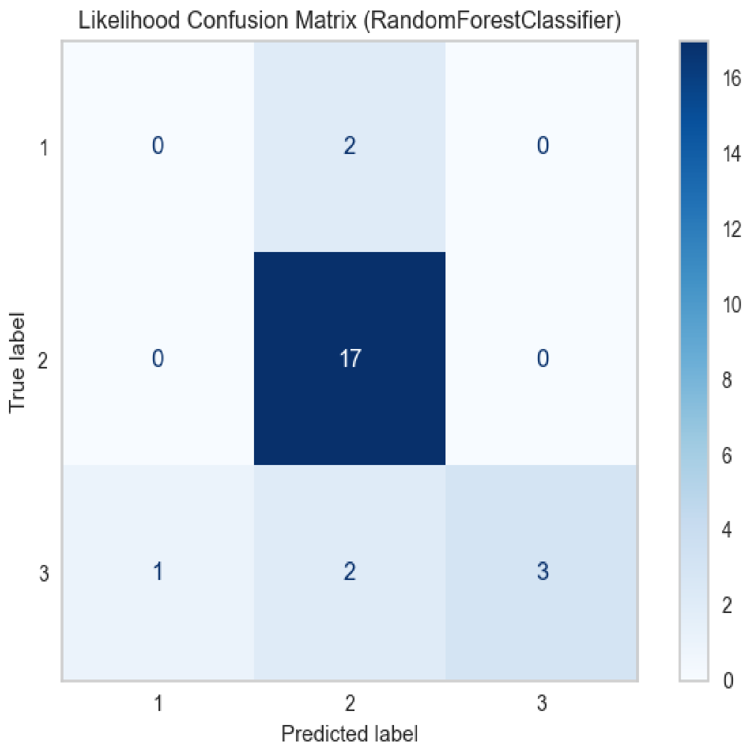


Figure 5.10: Example of a confusion matrix of predictions made by the risk likelihood model using binary classification conversion (Random Forest)

Similarly to the tests performed for the risk impact model, oversampling was also used here to try to improve the model's predictions regarding risk likelihood, which also has an imbalanced class distribution. As with the binary class distribution in the risk impact model, SVMSMOTE was applied to both sets of data distributions in the minority class (0 in one distribution, 1 in the other). Figures 5.12 shows an example of the original class distribution after the conversion to a binary distribution is performed (in this case, the original risk likelihood level 1 is now 0, and risk likelihood levels 2 and 3 are now 1), and the same class distribution after the use of SVMSMOTE on that set of data, respectively.

Table 5.8: Comparison of the risk likelihood model's evaluation metrics with oversampling applied (SVMSMOTE) to the training data and using binary classification conversion

Algorithm	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Gradient Boosted Trees	**0.51**	0.53	**0.49**	0.49	**0.51**
Random Forest	0.38	0.55	0.39	0.46	0.38
Complement Naive Bayes	0.27	**0.63**	0.20	0.25	0.27
Neural Network Logistic activation function	0.36	0.62	0.37	0.48	0.36
Support Vector Machine Linear kernel	0.42	0.60	0.44	**0.55**	0.42
K-Nearest Neighbours	0.39	0.55	0.42	0.49	0.39

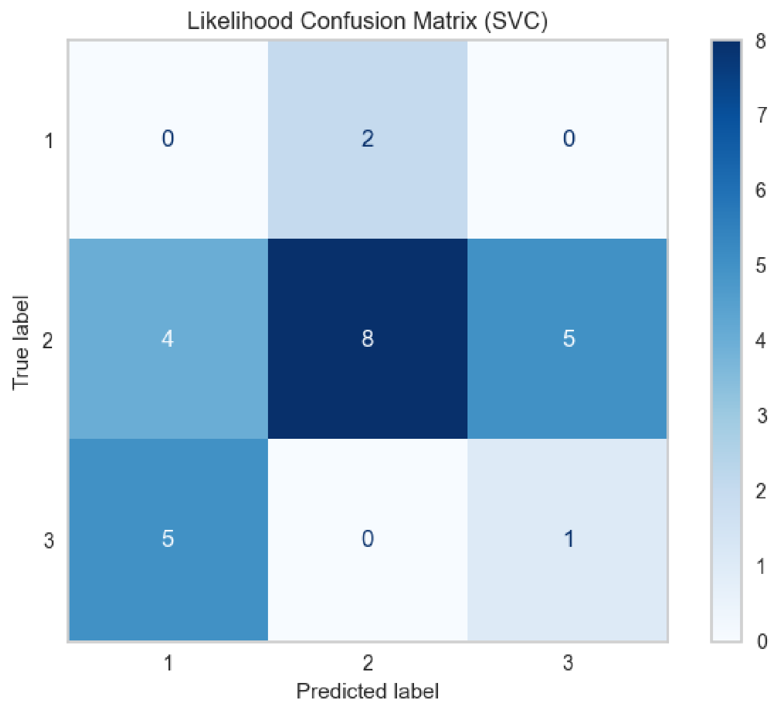


Figure 5.11: Example of a confusion matrix of predictions made by the risk likelihood model (Support Vector Machine) with oversampling applied to the training data and using binary classification conversion

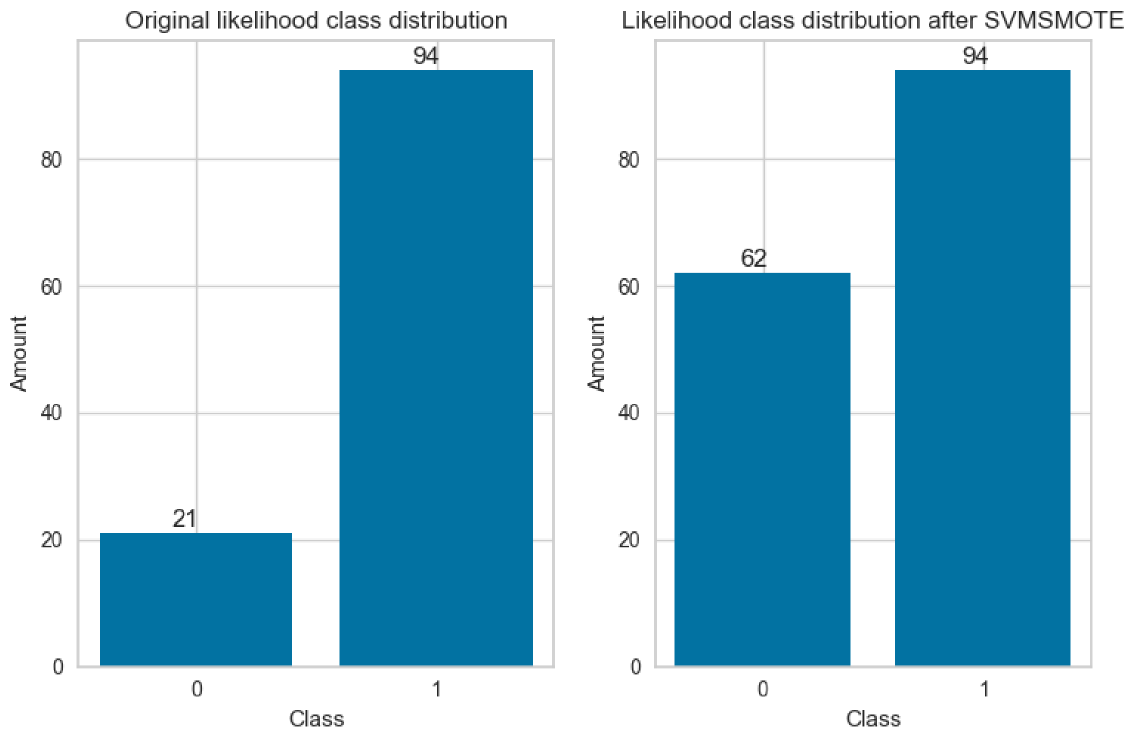


Figure 5.12: Risk likelihood class distribution without (left) and with oversampling applied to the training data (right) after the conversion to a binary distribution is performed

As was the case with the risk impact model, this did not improve the vast majority of the results found in the tests for the risk likelihood model without oversampling applied, save for a few exceptions, as can be seen in table 5.9. Regardless, an interpretable algorithm - Gradient Boosted Trees - had the highest score for accuracy (50%) and recall (0.50), while two non-interpretable ones had the highest scores for the remaining metrics: Support Vector Machine in AUC score (0.64) and precision (0.62), and K-Nearest Neighbours in F-Measure (0.48).

Most algorithms took a significant hit in results, with the results of Neural Network in particular dropping significantly once more when oversampling is applied. The exceptions to this were Gradient Boosted Trees, which kept mostly similar scores compared to the results of the tests performed without oversampling, and K-Nearest Neighbours, which improved in all evaluation metrics except AUC score.

5.3 Discussion

The use of oversampling returned consistently worse results for the most part when compared to the tests performed without oversampling, with a key exception in the Naive Bayes algorithm, whose results were better with the use of oversampling on the training data. By analysing the confusion matrix examples it is clear to see the differences in the predictions made by models trained with data with and without oversampling applied. There is a larger number of predictions made for the non-majority classes, which is the expected behaviour when oversampling is applied to tackle a class imbalance problem such as the one in this problem, but that did not improve the results of the different algorithms tested.

As for the conversion of the problem from a multi-class classification problem to a binary classification one, it was interesting to see the results of the interpretable models match and often surpass those of the non-interpretable models in both the risk impact and risk likelihood tests. Two algorithms in particular suffered a major drop in these tests - Neural Network in the risk impact model tests and Complement Naive Bayes in the risk likelihood model tests.

Nonetheless, as this did not considerably improve the results of the algorithms tested, it was not applied to the final training process of the models. As for oversampling, it was applied to the training data used to train the risk likelihood classifier, but not the risk impact model.

In the case of the risk impact model using multi-class classification and without oversampling, the difference between the best performing interpretable algorithm and the best performing non-interpretable algorithm is not high, which makes it so the decision to go for a slightly better performing non-interpretable model or a slightly worse performing interpretable model with the benefit of different ways of understanding the model's predictions in the future come down to the preference of the client. After discussing the topic with them, the decision was to use the Support Vector Machine algorithm for the predictions of risk impact level.

As for the risk likelihood model (also using multi-class classification and without oversampling), Complement Naive Bayes was the most consistent performing algorithm across both sets of tests for this target variable, and was thus used to train the risk likelihood model. Table 5.9 below summarizes the results and parameters used for the two selected algorithms.

Table 5.9: Results and parameters used of the algorithms selected for the risk impact and risk likelihood models

Algorithm	**Parameters**	**Accuracy**	**AUC**	**F-Measure**	**Precision**	**Recall**
Support Vector Machine	kernel='rbf' C=1 gamma='scale'	0.69	0.68	0.64	0.60	0.69
Complement Naive Bayes	alpha=1 norm=True	0.63	0.61	0.59	0.62	0.63

After presenting the results above and demonstrating the application in action to Strongstep, discussions regarding the use of the application in its current state took place. As it stands, from the company's point of view, the main point of interest regarding the usage of the information returned by the application is in presenting the impact and likelihood levels for each risk category available when a user creates a new project. Another use case is presenting the same information when a user creates a new risk for a given project, considering its characteristics (*e.g.*, budget, type), as a form of suggestion to the user considering the data available from previous projects.

Additionally, conversations regarding what more could be done with the information available took place, and regarding what else could be achieved if different types of information could be obtained in the future. The outcome of these conversations provided the basis to prepare plans for future tasks in this project, which are presented in the next chapter.

5.4 Summary

Different algorithms were tested for both the risk impact and risk likelihood target variables, three which are considered interpretable (Gradient Boosted Trees, Random Forest, Naive Bayes) and three which are not (Neural Network, Support Vector Machine, K-Nearest Neighbours). The goal was to compare their results across different evaluation metrics and make a decision on whether or not the performance trade-offs typically involved in interpretable algorithms are worth the ability to analyse and understand why the models are predicting the classes that they are. Additionally, techniques such as oversampling and the conversion of the problem from a multi-class classification problem to a binary classification one were also tested to try to maximize the performance of the models.

For the risk impact target variable, both non-interpretable and interpretable algorithms presented good results, particularly with multi-class classification and without oversampling applied. As a result, the decision between choosing a slightly worse performing interpretable algorithm while having the option to understand the model's predictions or choosing a slightly better performing non-interpretable algorithm without that option was discussed with the client. In the end, the Support Vector Machine algorithm was chosen for the risk impact model.

As for the risk likelihood target variable, both non-interpretable and interpretable algorithms returned very similar results, with Complement Naive Bayes returning the best results for most evaluation metrics used. As a result, this algorithm was chosen for the risk likelihood model.

In general, the use of oversampling returned worse results compared to the tested performing without oversampling applied to the training data, particularly in the non-interpretable models. This was more noticeable in the tests performed for the risk impact target variable, although it was still noticeable in the tests for the risk likelihood model as well, with a key exception in the results of the Naive Bayes algorithm. As a result, oversampling was not used in the training data used to train the risk impact model, but it was used in the training data used to train the Naive Bayes risk likelihood classifier.

Lastly, the conversion of the problem from a multi-class classification problem to a binary classification problem returned interesting results, though still mostly worse than the original multi-class classification results. As a result, this technique was not used to train the final models.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This report presented an analysis of the state-of-the-art of risk management in software projects, and the application of machine learning techniques to address some limitations in this research area. The implementation of a software module that could accurately classify risk impact and likelihood levels in different risk categories of new software development projects was then detailed, before finally presenting the results of the implementation and integration of this module with the PROMESSA module.

A wide variety of classification algorithms was tested with the complete risk data set. After obtaining their optimal hyperparameters through a grid search hyperparameterization script, they were added to the different algorithms chosen for further testing. Other techniques such as feature selection, oversampling, and converting the problem from a multi-class classification problem to a binary classification one were tested in attempts to continuously improve the results of the models.

Feature selection improved the model results and proved to be a particularly important technique to consider in the feature if more features can be used to train new models. The conversion to a binary classification problem did not often improve on the results of multi-class classification for the risk impact prediction model, but it did show interesting improvements in the risk likelihood prediction model, particularly in non-interpretable algorithms such as Neural Network and Support Vector Machine. Lastly, oversampling was seen as a way to deal with the class imbalance problem that is present in this classification problem, but it consistently returned worse results in the majority of tests performed. One exception was in the Naive Bayes algorithm predicting risk likelihood levels, where oversampling improved the results found in the different evaluation metrics used. As a result, oversampling was used only in the data to train the risk likelihood model.

Different algorithms were tested with the goal of understanding the performance trade-offs typically involved in interpretable algorithms (*e.g.*, Random Forest, Naive Bayes) and whether those trade-offs were worth the different type of information these algorithms can return, such as general information like feature importance or more specific information such as the flow taken by a Random Forest estimator to generate a prediction. In the end, the Support Vector Machine and

Complement Naive Bayes algorithms were chosen for the risk impact and risk likelihood models, respectively. The multi-class logic was used for both models, as it returned the best results for those target variables across multiple algorithms, and the best overall results.

These algorithms obtained 69% and 63% accuracy, respectively, but more importantly, their results in other metrics show that this is not a situation where the algorithms are obtaining a higher accuracy score by simply predicting the majority class (risk impact and likelihood level 2) and can in fact distinguish between the three classes available with a reasonable degree of confidence. Both algorithms obtained the highest result in accuracy, F-Measure, precision, and recall, with only Support Vector Machine being surpassed by Neural Network in AUC score for the risk impact target variable, while Naive Bayes maintained the highest AUC score for the risk likelihood target variable.

The workflow used in this project can be applied for future projects with similar goals in mind. First, a list of algorithms that are more appropriate for the problem should be created. Afterwards, their best set of hyperparameters should be obtained through parameter optimization. Lastly, the combination of chosen algorithms and their respective hyperparameters should be tested, and the results should be analysed through the evaluation metrics that are applicable to the type of problem (*e.g.*, accuracy, AUC, precision, recall in a classification problem; R-Squared, MSE, MAE in a regression problem).

As for future improvements, the focus should be on obtaining more quality data to continuously improve the models' understanding of the relationships between the attributes of a project and the risk impact and likelihood level associated with the different risk categories that can have an effect on the development of a new software project.

Notice the focus on **quality** data, as just integrating with any data source, and increasing the amount of data samples to train the models is sometimes not enough. As was seen, the data set for this project started with 6 training features, but two of them were removed as they had very low variance and were having a negative effect on the predictions of the models instead. It is then important to find more data that can be used, but also analyse the importance and variance of any new features that are added (as well as the features that have been removed if new data is added) to the data set and assess if they have a positive or negative effect on the models' predictions.

6.2 Future Work

As presented in this document, the current state of the project focuses on the accurate classification of impact and likelihood of risks in different categories. However, the information available in the data sets provided by Strongstep can be used for more tasks that can be developed in the future, which are presented in this section.

6.2.1 Provide More Specific Information

In the current state of the project, the application returns predictions of risk impact and likelihood levels based on the characteristics of a new project. After conversations with Strongstep, one of the most consistent points of feedback was the need to provide not only the current set of results the application provides, but also return more specific points of information.

The current risk categories (*e.g.*, Management, External, Technical) combined with the predictions of risk impact and likelihood level for each of them provides useful information about which categories project managers should pay closer attention to, but providing more specific information such as a subcategory to focus on would be very helpful as well.

Alternatively, returning information about specific risks that are common in projects with characteristics that are similar to the one that is being created would also help project managers in identifying specific points to focus on.

6.2.2 Risk Treatment and Mitigation Action Classifier

Due to the way that the models work, the information of risk treatment and mitigation actions available is not used to train the impact and likelihood model. This is because this information is only used when a new risk is created, and the predictions from the model are done when a new project is created instead. The risk treatment and mitigation actions have no impact on the prediction of impact and likelihood, as they are not provided during the creation of a new project on SCRAIM.

Even though Strongstep already has detailed guidelines to specify the risk treatment and mitigation actions when a new risk is added to a project, this information can still be used to train a different classifier that can then be used by new partners who possibly do not have as much experience in the risk management field, and would be interested in obtaining the most appropriate risk treatment and mitigation action for a new risk that they identified in their projects.

The creation of this classifier would follow the same workflow that was established and used during the creation of the risk impact and likelihood classifier described in this document: select the most appropriate models, obtain the optimal set of hyperparameters for them, evaluate their performance through different evaluation metrics, analyse the results and choose the most appropriate algorithm(s) considering the task at hand.

6.2.3 Facilitate Integration with More Data Sources

Since the start of this project, it was known that the data that was going to be used to train the model would be provided by Strongstep and extracted from their own project management tool - SCRAIM. However, as this is a proprietary tool with its own data structure, this can make it difficult to reuse the format in which the data was presented, as not every organization keeps track of risks in the same manner.

The new spreadsheet created from this data, described in section 3.1, was a good starting point to standardize the information needed to train and apply the model to a set of risks in a new project, but there is still more work to be done in order to increase not only the integration with more data sources to provide more data to the model, but also the integration with more partners interested in using the model to predict the impact and likelihood of risks in their own projects.

6.2.4 Workflow Automation

This task is closely tied with the previous suggestion. As it stands, the process of adding new data from the same partner, processing that data, re-training and evaluating the models is a manual task. Additionally, the integration with more data sources can also mean more manual work to prepare the creation of new models to match the new data structures and sources, which, based on the all the information presented up until this point, requires a considerable amount of effort. Between the data preprocessing, hyperparameter optimization, algorithm tests, creation of the machine learning models, and setting up the ways in which those trained models make predictions in the future (*e.g.*, a REST API endpoint), there is a lot to consider whenever a new use case is suggested.

The goal with this task is to automate the majority of this process by receiving as much information as possible from the user sending the training data set to the API, such as the indexes of categorical variables, indexes of quantitative variables, range of independent variables, range of dependent variables, among others. This way previously described steps such as one-hot encoding and feature standardization can be more dynamic and require less manual work. The same goes for hyperparameter optimization and the algorithm tests, as these processes can also become more dynamic with this approach, as long as the data preprocessing beforehand is performed correctly.

References

[1] imbalanced-learn documentation. https://imbalanced-learn.org/stable/. Accessed on June 2021.

[2] Java machine learning library (java-ml). http://java-ml.sourceforge.net/. Accessed on June 2021.

[3] Jira | issue & project tracking software | atlassian. https://www.atlassian.com/software/jira. Accessed on June 2021.

[4] Manage your team's work, projects, & tasks online - asana. https://asana.com/. Accessed on June 2021.

[5] Matlab - mathworks - matlab & simulink. https://www.mathworks.com/products/matlab.html. Accessed on June 2021.

[6] Overview - eli5 0.11.0 documentation. https://eli5.readthedocs.io/en/latest/overview.html. Accessed on June 2021.

[7] Rapidminer | best data science & machine learning platform. https://rapidminer.com/. Accessed on June 2021.

[8] scikit-learn: machine learning in python. https://scikit-learn.org/stable/. Accessed on June 2021.

[9] Tensorflow. https://www.tensorflow.org/. Accessed on June 2021.

[10] Waitress - waitress 2.0.0 documentation. https://docs.pylonsproject.org/projects/waitress/en/latest/. Accessed on June 2021.

[11] Weka 3 - data mining with open source machine learning software in java. https://www.cs.waikato.ac.nz/ml/weka/. Accessed on June 2021.

[12] Welcome to flask - flask documentation (2.0.x). https://flask.palletsprojects.com/en/2.0.x/. Accessed on June 2021.

[13] Yellowbrick: Machine learning visualization. https://www.scikit-yb.org/en/latest/. Accessed on June 2021.

[14] Aditya Mishra. Metrics to evaluate your machine learning algorithm, 2017. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, Accessed on 2020-10-08.

[15] N. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46:175–185, 1992.

[16] Ahmed Banimustafa. Predicting software effort estimation using machine learning techniques. pages 249–256, 07 2018.

[17] Sonam Bhatia and V. Attri. Machine learning techniques in software effort estimation using cocomo dataset. 2015.

[18] B. Boehm. Software risk management: principles and practices. *IEEE Software*, 8:32–41, 1991.

[19] Barry Boehm. Software project risk and opportunity management. *Software Project Management in a Changing World*, pages 107–121, 03 2014.

[20] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. Classification and regression trees. 1983.

[21] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[22] M. Hanefi Calp and M. Ali Akcayol. A novel model for risk estimation in software projects using artificial neural network. In D. Jude Hemanth and Utku Kose, editors, *Artificial Intelligence and Applied Mathematics in Engineering Problems*, pages 295–319, Cham, 2020. Springer International Publishing.

[23] Pramila Chawan, Jijnasa Patil, and Radhika Naik. Software risk management. *International Journal Of Computers & Technology*, 6:60–66, 05 2013.

[24] T. Christiansen, Pongpisit Wuttidittachotti, Prakancharoen Somchai, and Sakda Vallibhakara. Prediction of risk factors of software development project by using multiple logistic regression. *ARPN Journal of Engineering and Applied Sciences*, 10:1324–1331, 01 2015.

[25] J.S. Cramer. The origins of logistic regression. *Tinbergen Institute, Tinbergen Institute Discussion Papers*, 01 2002.

[26] Dr. Lance Eliot. Support vector machines (svm) for ai self-driving cars, 2018. `https://www.aitrends.com/ai-insider/support-vector-machines-svm-ai-self-driving-cars/`, Accessed on 2020-10-15.

[27] Michael Felderer, Florian Auer, and Johannes Bergsmann. Risk management during software development: Results of a survey in software houses from germany, austria and switzerland. pages 143–155, 04 2017.

[28] Eibe Frank and Mark Hall. A simple approach to ordinal classification. volume 2167, pages 145–156, 08 2001.

[29] Jerome Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 10 2001.

[30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[31] The Standish Group. Chaos report 2015, 2015. https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf, Accessed on 2020-10-07.

[32] Wen-Ming Han. Discriminating risky software project using neural networks. *Computer Standards & Interfaces*, 40:15 – 22, 2015.

[33] Ronald Higuera, David Gluch, Audrey Dorofee, Richard Murphy, Julie Walker, and Ray Williams. An introduction to team risk management. (version 1.0). page 55, 05 1994.

[34] Ming-Yuan Hsieh, Yu-Chin Hsu, and Ching-Torng Lin. Risk assessment in new software development projects at the front end: a fuzzy logic approach. *Journal of Ambient Intelligence and Humanized Computing*, 9, 04 2016.

[35] Y. Hu, J. Huang, J. Chen, M. Liu, and K. Xie. Software project risk management modeling with neural network and support vector machine approaches. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 3, pages 358–362, 2007.

[36] Yong Hu, Xiangzhou Zhang, E.W.T. Ngai, Ruichu Cai, and Mei Liu. Software project risk analysis using bayesian networks with causality constraints. *Decision Support Systems*, 56:439 – 449, 2013.

[37] Yong Hu, Xiangzhou Zhang, Xin Sun, Mei Liu, and Jianfeng Du. An intelligent model for software project risk prediction. In *2009 International Conference on Information Management, Innovation Management and Industrial Engineering*, volume 1, pages 629–632, 2009.

[38] Tauqeer Hussain. Risk management in software engineering: What still needs to be done. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Computing*, pages 515–526, Cham, 2019. Springer International Publishing.

[39] Prabhjot Kaur and Anjana Gosain. *Comparing the Behavior of Oversampling and Undersampling Approach of Class Imbalance Learning by Combining Class Imbalance Problem with Noise*, pages 23–30. 01 2018.

[40] T.M. Khoshgoftaar, D.L. Lanning, and Abhijit Pandya. A neural network modeling methodology for the detection of high-risk programs. pages 302 – 309, 12 1993.

[41] J. Kontio. The riskit method for software risk management, version 1.00. 1997.

[42] Chandan Kumar and Dilip Kumar Yadav. A probabilistic software risk assessment and estimation model for software projects. *Procedia Computer Science*, 54:353 – 361, 2015. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India.

[43] Young Kwak and J Stoddard. Project risk management: Lessons learned from software development environment. *Technovation*, 24:915–920, 11 2004.

[44] Jingyue Li, Odd Petter N. Slyngstad, Marco Torchiano, Maurizio Morisio, and Christian Bunse. A state-of-the-practice survey of risk management in development with off-the-shelf software components. *Software Engineering, IEEE Transactions on*, 34:271–286, 04 2008.

[45] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[46] Osamu Mizuno, Takanari Hamasaki, Yasunari Takagi, and Tohru Kikuno. An empirical evaluation of predicting runaway software projects using bayesian classification. In Frank Bomarius and Hajimu Iida, editors, *Product Focused Software Process Improvement*, pages 263–273, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[47] Christoph Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

[48] Hien M. Nguyen, E. Cooper, and K. Kamei. Borderline over-sampling for imbalanced data classification. *Int. J. Knowl. Eng. Soft Data Paradigms*, 3:4–21, 2011.

[49] Maruf Pasha, Ghazia Qaiser, and Urooj Pasha. A critical analysis of software risk management techniques in large scale systems. *IEEE Access*, 6:12412–12424, 2018.

[50] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[51] PMI. *A Guide to the Project Management Body of Knowledge (PMBOK Guide), 4th Edition*. Project Management Institute, 2008.

[52] Przemek Pospieszny, Beata Czarnacka-Chrobot, and Andrzej Kobyliński. An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software*, 137, 11 2017.

[53] Md Forhad Rabbi and Khan Mannan. A review of software risk management for selection of best tools and techniques. pages 773–778, 09 2008.

[54] Jason D. M. Rennie, L. Shih, J. Teevan, and D. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *ICML*, 2003.

[55] Ripu Ranjan Sinha and Rajani Kumari Gora. Software effort estimation using machine learning techniques. In Vishal Goar, Manoj Kuri, Rajesh Kumar, and Tomonobu Senjyu, editors, *Advances in Information Communication Technology and Computing*, pages 65–79, Singapore, 2021. Springer Singapore.

[56] André Sousa, João Pascoal Faria, and João Mendes Moreira. An analysis of the state of the art of machine learning for risk assessment in software projects. In *The 33rd International Conference on Software Engineering and Knowledge Engineering, SEKE 2021, KSIR Virtual Conference Center, USA, July 1-10, 2021*, pages 217–222. KSI Research Inc., 2021.

[57] K. A. Taher, B. Mohammed Yasin Jisan, and M. M. Rahman. Network intrusion detection using supervised machine learning technique with feature selection. In *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*, pages 643–646, 2019.

[58] Yasunari Takagi, Osamu Mizuno, and Tohru Kikuno. An empirical approach to characterizing risky software projects based on logistic regression analysis. *Empirical Software Engineering*, 10:495–515, 10 2005.

[59] Linda Wallace, Mark Keil, and Arun Rai. Understanding software project risk: a cluster analysis. *Information & Management*, 42(1):115 – 125, 2004.

[60] E. Wallmüller. Risk management for it and software projects. pages 165–178, 01 2002.

[61] Gary Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? pages 35–41, 01 2007.

[62] Linda Westfall. Defining software risk management, 2001. http://www.westfallteam.com/sites/default/files/papers/risk_management_paper.pdf, Accessed on 2020-10-12.

[63] R. C. Williams, G. J. Pandelios, and S. Behrens. Software risk evaluation (sre) method description (version 2.0). 2000.

[64] R. Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, 01 2000.

[65] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26 – 40, 2019.

[66] M. Zavvar, A. Yavari, S. M. Mirhassannia, M. R. Nehi, and A. Yanpi. Classification of risk in software development projects using support vector machine. *Journal of Telecommunication, Electronic and Computer Engineering*, 9:1–5, 2017.

[67] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data preparation for data mining. *Applied Artificial Intelligence*, 17(5-6):375–381, 2003.