

Designing a solution architecture for monitoring credit scoring analytic models

Citation for published version (APA):

Cordero Cruz, J. A. (2021). *Designing a solution architecture for monitoring credit scoring analytic models*. Technische Universiteit Eindhoven.

Document status and date:

Published: 01/09/2021

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.



PDEng THESIS REPORT

Designing a solution architecture for monitoring credit scoring analytic models

Jorge Alberto Cordero Cruz
September 2021
Department of Mathematics & Computer Science

PDEng SOFTWARE TECHNOLOGY

Designing a solution architecture for monitoring credit scoring analytic models



Jorge Alberto Cordero Cruz

September 2021

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

PDEng Report: 2021/082

*Confidentiality Status:
Public Report*

Partners		
	Rabobank	Eindhoven University of Technology
Steering Group	Prof.dr. Mark van den Brand (TU/e)	
	ir. Harold Weffers PDEng (TU/e)	
	Rik Bosman MSc. (Rabobank)	
	ing. Jaqualine Gooijer (Rabobank)	
Date	September 2021	

Composition of the Thesis Evaluation Committee:

Chair: Prof.dr. Mark van den Brand

Members: Prof.dr. Mark van den Brand

ir. Harold Weffers PDEng

Rik Bosman MSc.

Joost Landman

Dr. Vlado Menkovski

The design that is described in this report has been carried out in accordance
with the rules of the TU/e Code of Scientific Conduct.

Contact Address	Eindhoven University of Technology Department of Mathematics and Computer Science MF 5.080A, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands +31 402743908
Partnership	This project was supported by Eindhoven University of Technology and Rabobank.
Published by	Eindhoven University of Technology Stan Ackermans Institute
PDEng-report	2021/082
Preferred reference	<u>Designing a solution architecture for credit scoring analytic models</u> . Eindhoven University of Technology, PDEng Report 2021/082, September 2021
Abstract	A Rabobank innovation project aims to breach the disconnection between smallholder farmers without financial records that need access to funding and credit providers by generating credit scores using alternative data. These credit scores can be used by credit providers to analyze risk profiles of smallholder farmers requesting loans. This work describes a solution architecture for monitoring credit scoring analytic models. A monitoring engine was designed to identify changes in the input data and predictions that could result in inaccurate or biased credit scores. In addition, a deployment pipeline and a model registry were designed to support the monitoring of every deployed analytic model. The monitoring engine, deployment pipeline, and model registry are described in the solution architecture. The proposed solution architecture was validated by architects and data scientists and verified by creating a prototype. The prototype shows that analytic models can be deployed, stored, and monitored as described by the solution architecture.
Keywords	PDEng, Software Technology, TU/e, Rabobank, Monitoring analytic models, Credit scores, Alternative data
Disclaimer Endorsement	Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or Rabobank. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or Rabobank, and shall not be used for advertising or product endorsement purposes.
Disclaimer Liability	While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.
Trademarks	Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright

Copyright © 2021. Eindhoven University of Technology. All rights reserved.

No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and **Rabobank**.

Foreword

North Star, the innovation department of Rabobank Wholesale and Rural especially focuses on connecting: bridging gaps between organizations or between people. Rabobank emerged from small agricultural cooperative banks, and the cooperative foundation and the cooperative philosophy have remained our guiding principle throughout our history.

Rabobank and the TU Eindhoven are also connected: through Jorge Cordero. Rabobank provided the assignment in North Star and various colleagues represented Rabobank during the project. The university was responsible for the education of the Jorge, provided supervision to guard the educational aspects for him. Both the bank and the university provided expert knowledge on relevant aspects of the problem domain.

Jorge worked during his assignment in Credit Connect, one of the North Star solution spaces where we aim to connect smallholders and credit providers. A loan provided by the credit provider leads to prosperity for the farmer, food for the society and often also focuses on sustainable practices. However, assessing a farmers credit worthiness with no financial history, no collateral and in high risk sector is very difficult. That's why Rabobank developed a credit scoring model that works with so-called alternative data.

During the assignment Jorge researched an interesting subject: the verification of a credit scoring analytic model through a monitoring engine: is the outcome of the model in line with expectations. Or in other words: are the input data and the result "connected".

The analysis and thought process of Jorge was not an easy journey. Especially because of COVID-19, most of the work was done remotely. Rabobank is very happy with the results, and we hope that the monitoring engine will show its added value in the upcoming time.

Rik Bosman
September 21, 2021

Preface

This report contains results of the ten-month graduation project of the Jorge Alberto Cordero Cruz conducted at Rabobank. Prior to this project, the trainee acquired knowledge and skills about software design and architecture as part of the Professional Doctorate in Engineering (PDEng) program in Software Technology (ST). The PDEng ST offered by the Eindhoven University of Technology trains T-shaped professional designers capable of working in high-tech industry.

This project was initiated and funded by Rabobank to enhance their model development process that is used for creating and maintaining analytic models for several applications. In particular, in this project we focused on analytic models used for generating credit scores using alternative (non-financial) data. The results of this project include an architecture design and a prototype for deploying and monitoring credit scoring analytic models. During this project, the PDEng trainee was supervised by Rik Bosman MSc., Prof. dr. Mark van den Brand, and ir. Harold Weffers PDEng.

This report is useful for both technical and non-technical readers. Anyone interested in the motivation behind this project and its requirements can read Section 1, 2, and 4. Technical readers who are interested in the design and prototype of the proposed monitoring solution can read from Section 3 to Section 9. Anyone interested in the working process during this project can read Section 2 and Section 10. For readers who want a quick understanding of the project, the executive summary and Section 11 are recommended. The last section of this report, Section 12, provides the author's self-reflection on the project.

Jorge Alberto Cordero Cruz
September 2021

Acknowledgements

This project would not have been possible without the support of many people who during the last ten months provided me professional guidance as well as personal support.

I want to thank Yanja Dajsuren who ensured I was surrounded by professional mentors and colleagues from whom I could learn a great deal about being a software designer.

I also want to thank Desiree for her continuous help arranging educational activities and social events that made the PDEng an enjoyable experience.

To my TU/e supervisors, Harold Weffers and Mark van den Brand, I say thank you for your critical questions and your original ideas that helped me to push the boundaries of my work.

Similarly, I appreciate the support of my company supervisor, Rik Bosman, from whom I learned several things about software architecture but also about public speaking. His teachings helped me to become better at explaining my ideas.

I appreciate the sessions with Assel and Ederlyn, my co-supervisors at Rabobank. Their alternative points of view helped me to improve my designs and deliveries. In addition to work, they also provided personal guidance that helped me engage better with my Rabobank colleagues.

I could have not been part of Rabobank without the trust of Jaqualine Gooijer. I am grateful for the opportunity she gave me and for trusting my work when the results were not going as expected.

I could not have written this document without the help of Judith Strother. Her teachings allowed me to become better at writing and to understand that there is still more to be learnt.

Working with the innovation team was a wonderful experience full of learning and challenges. They were always willing to discuss work ideas and to talk about some of the most interesting things in life.

In the most difficult moments, when the corona lockdown seemed not to have an end, my friends were there for me. I want to thank you all because without you the last months would have felt like a cold winter. In special, I am highly grateful to Jessica, Dennis, Evi, Kaylee, Marrit, Manouk, Annita, Anna, and Luis.

In particular, I want to thank my parents for their kind words of encouragement and advice. I am proud of you and every day try to follow your footsteps. In addition, I want to say thanks to my sister and brother, your hard work serves as inspiration when I feel I cannot continue.

There are many more people with whom I have worked or shared some quality time in the last months. To them I say, thank you for being part of my professional and personal life. I am looking forward to more challenging discussions or quiet tea conversations.

September 2021

Executive summary

An innovation project at Rabobank aims at building bridges between smallholder farmers without financial records and credit providers. To achieve this goal, a credit scoring application leverages alternative (non-financial) data, such as historical weather conditions, field characteristics, and yield production, to calculate credit scores that credit providers can use for assessing the risk profile of their potential clients. These credit scores are predicted by analytic models created based on using a set of variables identified by Rabobank data scientists together with scientists from the Wageningen University.

In accordance with Rabobank policies, analytic models must follow a model development process to ensure the quality of their predictions and minimize potential risks. Some of these potential risks lead to reputation damage and fines due to wrong or biased predictions. In this report, we present a solution architecture that contains a monitoring engine designed to identify changes in the input data and predictions of analytic models that could lead to these risks.

The proposed solution supports monitoring changes in the input data and predicted credit scores in the absence of repayment behavior data that can be used as reference to measure the quality of the predictions. In addition, the accuracy of the analytic models can also be monitored when truth values related to farmer repayment behavior are available. Furthermore, analytic models can be monitored across different versions.

In addition to monitoring, the solution architecture contains other elements that support the monitoring engine:

- A model deployment pipeline that ensures deployed analytic models pass at least a set of unit tests that validate such models can be executed and monitored.
- A model registry that describes how analytic models should be stored in order to enable tracking and monitoring models across different versions.

Based on the solution architecture and its implementation design, we created a prototype to validate that analytic models can be deployed, executed, activated and deactivated, as well as monitored. This prototype was implemented using a modified version of the credit scoring application. Even though the implementation is not production-ready, it shows how the proposed solution architecture can be implemented in the innovation project.

Based on the results of this project, we recommend a gradual implementation of the proposed monitoring engine and its supportive elements. A good starting point is to modify the current structure of the analytic models to support versioning. After that, the model deployment pipeline and the model registry can be implemented. Finally, the model monitoring engine can be included. In this way, the innovation team can ensure that every deployed model can be monitored as soon as the model monitoring engine is implemented.

Table of contents

Foreword.....	i
Preface.....	iii
Acknowledgements	v
Executive summary.....	vii
Table of contents	ix
List of figures.....	x
List of tables.....	xii
1. Introduction	1
2. Stakeholder analysis	5
3. Domain analysis	7
4. Requirements elicitation	13
5. Feasibility analysis.....	19
6. Solution architecture	23
7. Implementation design	27
8. Implementation	33
9. Verification & validation	37
10. Project management	41
11. Conclusions	43
12. Project retrospective	46
Glossary	47
Bibliography	49
Appendix A.....	51
About the Author	53

List of figures

Figure 1: Business process model describing a simplified version of the current loan application process.	8
Figure 2: Business process model describing a simplified version of the loan application process including CSA.....	8
Figure 3: Domain analysis of analytic models used by CSA.....	10
Figure 4: Changes in analytic models that must be supported by the solution architecture, part 1.	11
Figure 5: Changes in analytic models that must be supported by the solution architecture, part 2.	12
Figure 6: Rabobank model development lifecycle containing four processes: model data development, model development, model implementation, and model monitoring. This diagram shows the activities and roles related to each process. The roles shown in this diagram are: software engine (SE), data engineer (DE), solution architect (SA), data scientist (DS), and business stakeholder.	23
Figure 7: Proposed solution architecture for monitoring credit scoring analytic models.	24
Figure 8: Class diagram for the prediction engine. A general analytic model interface is defined that all types of analytic models must implement such that the prediction engine can use any specific analytic model implementation.	27
Figure 9: state machine representing the different status that analytic models can have.....	30
Figure 10: Sensitivity analysis applied to analytic models for credit scoring.....	31
Figure 11: Identifying changes in the distributions of observations over a specific period.....	32
Figure 12: Computing the accuracy of predicted credit scores based on repayment behavior.....	32
Figure 13: Webapp tab that shows the model monitoring engine functionality of the prototype. The name of the active model being monitored is hidden behind a gray rectangle due to privacy policies.	34
Figure 14: Webapp tab that shows the model activation functionality of the prototype.....	35
Figure 15: CSA (<i>credit_scoring_app</i>) code containing the implementation of the solution architecture prototype. Notice how some of the elements of the solution architecture are mapped directly in the modules of the <i>credit_scoring_app</i> code.	36
Figure 16: Monitoring the distribution of predictions of an analytic model and the distribution of input variable age. The monitoring is performed using a window size of 100 observations.	39
Figure 17: Monitoring the effects that the modification of the value of an input variable have in the predictions of an analytic model.	40
Figure 18: Main elements of the work-breakdown structure used during the project.	42
Figure 19: Monitoring of analytic models enabled by the solution architecture.	51
Figure 20: Deploy a new model to the model registry.....	52
Figure 21: Replace a deployed analytic model for a different version stored in the model registry	52

List of tables

Table 1: A sample of methods for identifying changes in observations. These methods are evaluated according to four properties. Based on these properties, the KS statistical test is preferred over the other methods.....	20
Table 2: Encoding formats for analytic models that should be deployed in production.....	21
Table 3: Relationship between the solution architecture components and their role on the monitoring of analytic models.	24
Table 4: Relationship between the solution architecture and the requirements.	26

1. Introduction

1.1 Context

1.1.1. Smallholder farmers situation

Rabobank is a cooperative bank that started 132 years ago by financing farmers that nobody else wanted to finance. In the global market, Rabobank provides financial services for the Food and Agri sector. According to their vision and in response to new trends, Rabobank has identified several innovation initiatives that will ensure its position as a financial institution in the future. One of these initiatives, composed of several projects, aims to help to reduce the number unbanked farmers in emerging countries.

Nowadays, millions of smallholder farmers, who own farms of size between 2 to 50 acres (approximately 4,047 square meters), require capital to plant, grow or sell their crops but have limited options for requesting loans or using other types of financial services. The main issue is that these farmers do not have financial history records. In contrast, financial institutions such as banks, agriculture fintech companies, microfinance institutions, and wholesale companies, also known as credit providers, have the capital required to finance these farmers. However, these credit providers cannot assess the risk profile of the smallholder farmers.

Risk profiles are used to predict which farmers are likely to pay back their loans and how much interest to charge for each loan. In the absence of such risk profiles, the credit providers typically deny the loans requested by smallholder farmers. Smallholder farmers who cannot obtain loans via credit providers, typically obtain them with high interest from unregistered organizations. Such a loan application process does not generate financial history records and impact the farmers' expected revenue, as the repayment interest are higher than with credit providers.

An example of an interaction between smallholder farmers, credit providers, and unregistered organizations is as follows:

- First, smallholder farmers apply for a loan to a credit provider.
- Then, the loan application is rejected due to lack of financial history records.
- Next, smallholder farmers get their loan via unregistered organizations.
- From this loan the farmers do not generate financial history records while the loan interest rate is extremely high.
- The next time smallholder farmers require a loan, they apply to credit providers because they offer lower interest rate and other advantages.
- Once again, the smallholder farmers are very likely to be rejected and the cycle continues.

Clearly, from the example shown above, the lack of financial history records works against the smallholder farmers; it impacts their revenue and interferes with the farmers ability to improve their life quality.

1.1.2. Credit scoring application

One innovation project from Rabobank, a credit scoring application (CSA), aims to breach the existing barrier between smallholder farmers without financial history records and credit providers that have no way to assess risk profiles without such a financial history. Leveraging the advances in technology, CSA offers a way to compute credit scores using alternative (non-financial) data such as historical weather conditions, field characteristics, yield production, and personal information. The resulting credit scores can be used by credit providers to evaluate the risk profile of their potential clients.

CSA is designed to be a platform that offers credit scoring as a service for credit providers. These credit scores are computed using analytic models created based on alternative variables identified by Rabobank data scientists and scientists from the Wageningen university. As these models can impact the decisions of credit providers, they are developed and maintained following an analytic model development process standardized by Rabobank, which has been designed to minimize the adverse effects that could result from using the output of analytic models in financial services.

1.1.3. Project focus

Rabobank's model development process (Gallo, Heil, & Bosman, Credit Scoring Project Start Architecture, 2020), which is based on Rabobank's MLOps process (Rooijen van, 2021), includes general guidelines and requirements for the following tasks (elements): data collection, model development, model deployment, and model monitoring. For models being used in a production environment, model monitoring is used to corroborate the models generate high quality predictions and to identify biased predictions. Due to the type of models used by CSA, many off-the-shelf monitoring techniques, such as measuring the precision, recall, and ROC curves of the predictions, cannot be directly used in the software of CSA. Using these techniques implies that the truth or expected values are known at the moment predictions are generated, which is not the case for CSA models. A predicted credit score can only be evaluated after the farmer has paid back or defaulted on his loan.

The focus of this project is on designing a solution for monitoring credit scoring analytic models, using CSA as a reference project. This solution must adhere to Rabobank's existing architecture standards and must also aid data scientists and data engineers to perform their work. This solution must ensure that biased models can be identified in the production environment in order to improve them or replace them as soon as possible.

1.2 *Outline of the report*

At the beginning of this project, the scope of the project was unclear because there were several elements of Rabobank's model development process that could be adapted for CSA. In addition, on some occasions, the requirements of CSA as a whole were updated in respond to changes in the business roadmap. Some of these changes impacted the usage of the model development process, which in turn affected the scope of this project.

To tackle this uncertainty, this project was executed following an evolutionary process with the following lifecycle stages: exploration, design, and implementation.

The results obtained from the exploration stage are presented in Section 2, 3, and 4. Section 2 describe the main stakeholders involved in this project and their concerns. In addition, it describes the procedure followed to interact with these stakeholders. Section 3 explains the domain analysis performed to understand the elements of analytic models, how they can be designed, and how they should work in CSA. Moreover, this analysis describes how analytic models can change over time and how such changes could be tracked. Finally, Section 4 presents the requirements and use cases used to design the solution architecture presented in Section 6. These requirements are presented as business requirements, functional requirements, and non-functional requirements. The latter two types are used to enable the business requirements.

The outcomes of the design stage are presented in Section 5, 6, and 7. Section 5 presents the main challenges considered before the creation of the solution architecture. Additionally, it describes the design decisions that influenced the solution architecture and implementation design. Section 6 presents the solution architecture created for analytic model monitoring. It also explains supporting elements included to ensure model monitoring works according to the requirements. Section 7 contains the implementation design that describes elements of the solution architecture in more detail. Several elements of this section are used for the prototype implementation.

The implementation stage was performed at the end of the project. During this stage we created the prototype described in Section 8. In the next section, we describe the methods used to validate and verify the solution architecture and implementation design.

The last elements of the report are general information describing processes, results, and personal views of the project. Section 10 presents the project management activities followed during this project. The next section contains the conclusions and recommendations of the project. On Section 12, the author presents a personal reflection during his work on this project.

Last but not least, the Glossary contains relevant definitions of terms and the Appendix A describe some diagrams related to the solution architecture.

2. Stakeholder analysis

This section describes the main stakeholders with whom the trainee interacted to identify and map their needs and concerns into the proposed monitoring solution. These stakeholders belong to three categories: TU/e stakeholders, Rabobank stakeholders related to the credit scoring application (CSA), and credit providers that are users of CSA. In this project, most of the requirements were derived from the needs of Rabobank stakeholders related to analytic model monitoring. Concerns related to CSA as a product were only considered if they were related to model monitoring.

2.1 Introduction

This project was performed mostly online due to the Corona regulations placed by the Dutch government. Nonetheless, I collaborated with several members of the CSA team and Rabobank during standups and weekly meetings. The collaboration with TU/e members was mainly during reviewing sessions and monthly Project Steering Group (PSG) meetings, where the trainee supervisors evaluate the trainee's progress. In addition, the trainee collaborated with stakeholders from South American credit providers during special meetings where a translator was required.

The stakeholder registry described below was used to keep track of the main stakeholders and to understand their needs. Even though this is a simple procedure, it helped to identify how to interact with them in order to identify their needs and concerns.

2.1.1. Rabobank stakeholders

These stakeholders were crucial to understand how CSA works and what were the priorities of the CSA team in the short, medium, and long term.

Stakeholder role: Business architect and company supervisor

Concerns:

- Ensure that the proposed monitoring solution adheres to the business goals determined for CSA.
- Ensure that the proposed monitoring solution stays relevant as CSA grows.
- Introduce the trainee to Rabobank standards for architecture design.

Involvement: every week during discussion sessions and during the monthly PSG meetings

Stakeholder role: Solution architect and company co-supervisor

Concerns:

- Ensure that the proposed solution helps the team during the development and maintenance of CSA.
- Explore model development approaches that were not initially considered for the development of CSA.

Involvement: every week during discussion sessions

Stakeholder role: Digital project manager

Concerns:

- Ensure that the CSA team delivers a useful product.
- Explore several promising ideas that could result in new functionality added to CSA.

Involvement: during daily standups and in some discussion sessions

Stakeholder role: Product owner

Concerns:

- Ensure that CSA follows the business roadmap.
- Onboard new CSA users.
- Identify the needs of users to improve the functionality provided by CSA.

Involvement: every week during team meetings and sometimes during the discussion sessions

Stakeholder role: Data scientist

Concerns:

- Ensure that the analytic models can be deployed in CSA.
- Identify the variables required for developing analytic models.

Involvement: during daily standup sessions, team meetings, and pair programming sessions

Stakeholder role: Data engineer

Concerns:

- Ensure that CSA can ingest alternative data required from external sources.
- Implement a data processing pipeline that fit the needs of the deployed analytic models.
- Safeguarding the quality of the data ingested by analytic models.

Involvement: during daily standup sessions, team meetings, and pair programming sessions

2.1.2. TU/e stakeholders

These stakeholders were mainly consulted to review the scope of the project, evaluate the progress of the project, and verify the design decisions made for the solution architecture.

Stakeholder role: University supervisor

Concerns:

- Guide the trainee during the PDEng final project.
- Discuss the designs proposed by the trainee to spot areas of improvement.
- Ensure that the proposed solution and final report adhere to the PDEng quality standards.

Involvement: during biweekly discussion meetings and monthly PSG meetings

2.1.3. Credit provider stakeholders

These stakeholders were initially not considered. However, interacting with these stakeholders provided useful information about the data collection processes used by CSA users, some of their needs related to CSA, and their goals related to financing smallholder farmers.

Stakeholder role: Credit provider staff

Concerns:

- Ensure that CSA fits the needs of his/her organization.
- Ensure that CSA analytic models do not unethically discriminate against smallholder farmers. with whom they work or want to work.
- Ensure that Rabobank has a data integrity process in place to keep their data safe.

Involvement: via email and during meetings when a translator was required

3. Domain analysis

The previous section describes the most important stakeholders involved in this project and the frequency with which they were consulted. Based on the information collected from these interactions, this section focuses on exploring some of the main concepts that drive the design of the solution architecture described in Section 6. In particular, in this section, the elements of analytic models for credit scoring and their use in the credit scoring application (CSA) are explored in detail.

3.1 Context

As described in Section 1, Rabobank wants to help to reduce the number of smallholder farmers in developing countries who have no access to financial services. In particular, Rabobank wants to solve the following problem: smallholder farmers without financial history want to obtain loans but credit providers require financial history records to perform a risk assessment of the loan applications. If the risk assessment could be performed without the need of financial history records, then the smallholder farmers could have a chance to obtain loans from credit providers.

Based on this assumption, Rabobank started an innovation project, which for privacy reasons we will call credit scoring application (CSA) in this report, that aims to generate credit scores of smallholder farmers from alternative data. Using analytic models such as rule-based models and statistical models, CSA will generate credit scores related to smallholder farmers. Then, these credit scores can be used by credit providers to evaluate applicants without financial history. In addition, they could also be used to complement the existing credit risk analysis process. A business process model indicating the current loan approval process is shown in Figure 1. An alternative process involving CSA during the loan request evaluation is shown in Figure 2.

In the loan approval process, CSA offers credit scoring as a service to credit providers but is not involved in the loan approval decisions. In addition, CSA is not used by Rabobank to provide loans to Rabobank clients. Whether Rabobank will one day use CSA for its own services is a question outside the scope of this project. Thus, we cannot answer it.

At the moment of writing this report, the role of CSA is to provide a credit scores based on alternative data that credit providers can use to determine the credit risk of loan applications made by smallholder farmers. Therefore, CSA can be considered a credit scoring as a service application.

As an innovation project, CSA's roadmap involves starting operations with a small number of credit providers (CSA's users) and growing the number of supported credit providers over time. At the moment of writing this report, CSA is about to go live providing support for a small number of credit providers.

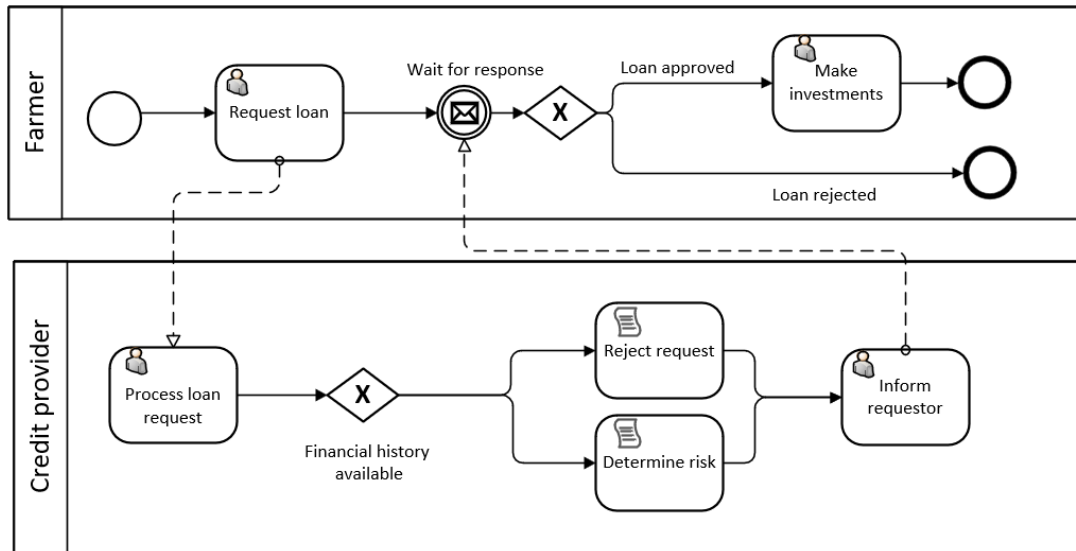


Figure 1: Business process model describing a simplified version of the current loan application process.

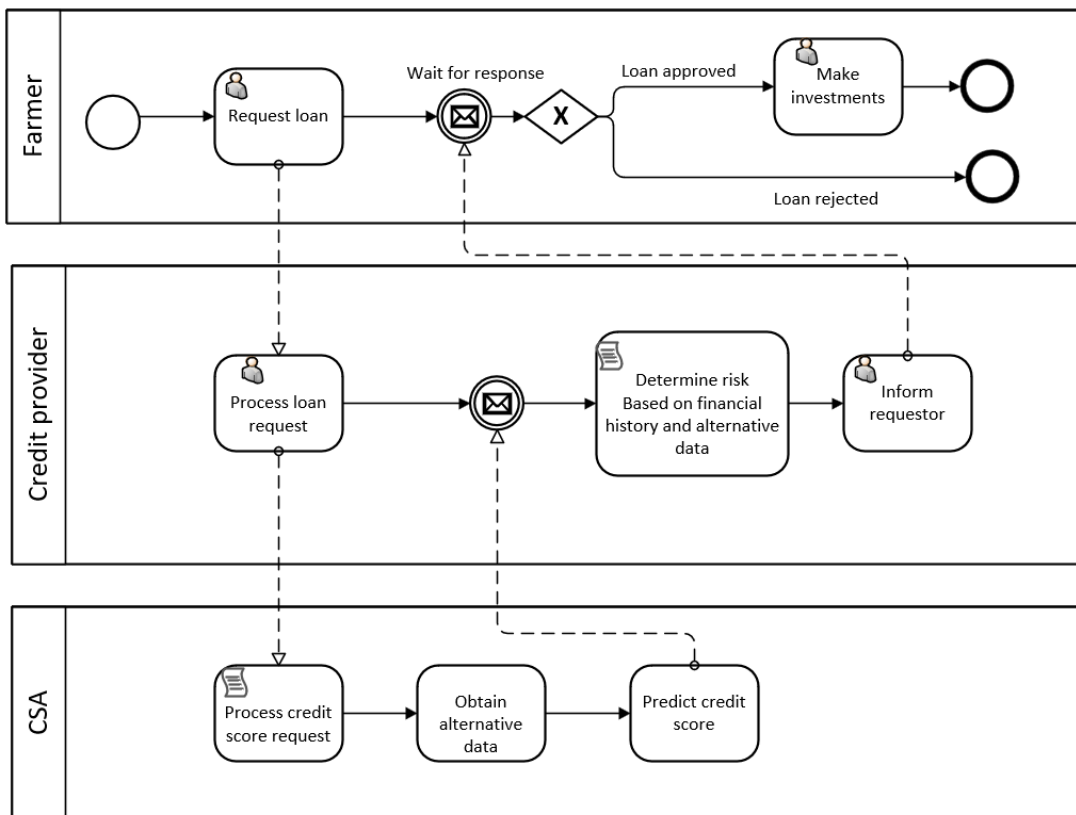


Figure 2: Business process model describing a simplified version of the loan application process including CSA.

3.2 *Analytic models used by the credit scoring application*

The credit providers that the CSA team wants to onboard require analytic models that generate credit scores that are related to their smallholder farmers. These credit providers serve smallholder farmers that live in certain regions and cultivate certain crops. For instance, credit providers in South America might work with farmers who grow cocoa, corn, or coffee while credit providers in Asia might work with farmers who grow rice and other crops.

Regarding the desired alternative input data, Rabobank data scientists in collaboration with researchers from the Wageningen University¹ have defined more than 100 variables that can be used to create credit scores. These variables can be grouped as follows:

- Environmental data describing historical weather conditions
- Field characteristics that can be obtained by analyzing satellite images from a given GPS location
- Crop management information describing the process followed to cultivate crops
- Historical yield information indicating average yields obtained in the past years on a specific farm and a specific region
- Personal information including details such as age and health

The actual alternative input data ingested by analytic models depends on the information that can be gathered from the farmers associated with each credit provider: type of crops being cultivated, cultivation techniques required by the crops, weather conditions of the region, regional crop prices, and personal information of the farmers. Additionally, sometimes credit providers cannot provide a required input variable but can provide similar information.

As we can see, analytic models must be specifically created to process the input data provided by specific credit providers. Each unique analytic model can contain a combination of alternative data variables. Additionally, the analytic models should contain elements that allow their monitoring and traceability.

Figure 3 shows a diagram that models the elements of analytic models for credit scoring and how they can be used by CSA. This diagram was created based on CSA's documentation and the answers obtained from interviewing the stakeholders described in the previous section. In addition, the definition of an analytic model provided in Figure 3 was used to perform the feasibility analysis in Section 5 and create the solution architecture and its implementation design in Section 6.

The following are some examples of how the diagram in Figure 3 should be read:

- A credit provider uses CSA.
- CSA can have one or more analytic models.
- An analytic model ingests input data and generates predictions.
- An analytic model is an implementation of an analytic model design.

¹ <https://www.wur.nl/en.htm>

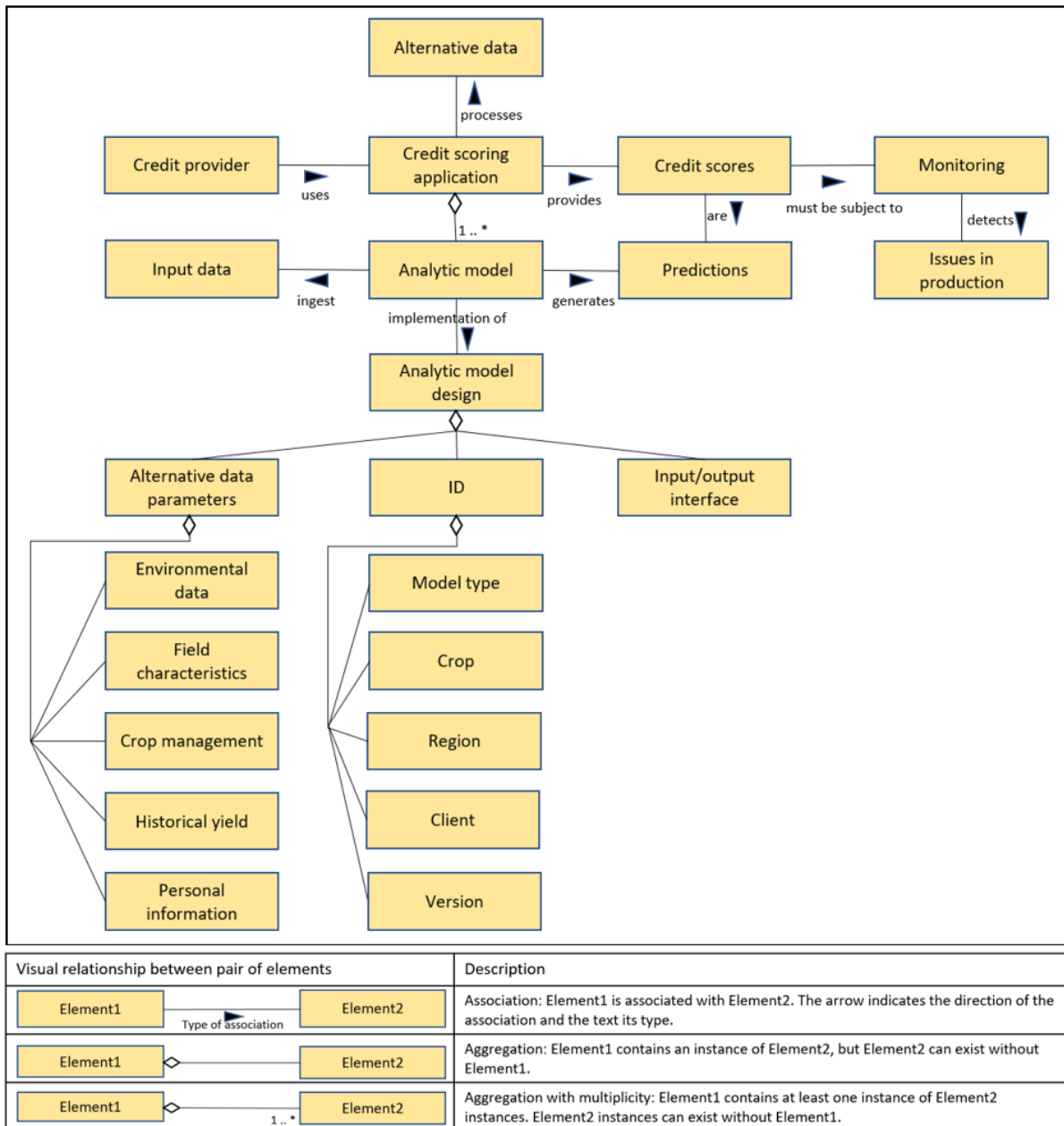


Figure 3: Domain analysis of analytic models used by CSA.

At the time of writing, the innovation team uses rule-based models, known as expert judgement (EJM) models, to compute credit scores due to the lack of enough representative data to train statistical models. However, once enough representative input data is available, statistical models will be designed, trained, and deployed in CSA. Considering one step further in the roadmap, machine learning models could also be explored once statistical models have become the default option for analytic models.

Overall, the EJMs work as follows:

- First, for a given input data observation, the values of each variable are mapped to a predefined weight set by the data scientists during the creation of the analytic model.
- Then, both the value of the variables and their corresponding weights are used to compute intermediate scores.
- Next, using the intermediate scores (one for every variable), a final score is computed.
- Finally, the final score is mapped to a credit score in a valid range.

In contrast to rule-based models, the weights of statistical models are not defined by data scientists. Instead, these weights are learned from training data. These types of models generate predictions by mapping the values of input variables to credit scores using the learned weights and specific computations associated to the type of model being used (Hastie, Tibshirani, & Friedman, 2001).

At the time of writing, it is still unknown what type of statistical models could be used in CSA. However, in this project, we based our domain analysis considering that linear regression models could be used for predicting credit scores. Therefore, the analytic models described in Figure 3 can be EJMs or linear models.

3.3 Evolution of analytic models

Considering the needs of credit providers and the number of users that the innovation team wants to onboard, CSA will eventually need to support several analytic models. These models will be created, updated, or removed over time. For auditing purposes, the evolution of such models must be traced over time.

The ID parameter described in Figure 3 enables the traceability of analytic models. For instance, consider {model type}-{crop}-{region}-{client(s)}-{version} as the ID format. Then, the analytic model EJM-coffee-mexico-bk-v0.2 can be seen as an update of EJM-coffee-mexico-bk-v0.1. Moreover, both analytic models generate credit scores for the same credit provider b1, which serves smallholder farmers that cultivate coffee in Mexico. Notice that in this example, both analytic models are EJMs.

The previous example describes an analytic model update where only internal elements change from one version to another. This type of model evolution is depicted in Figure 4 in cases (1) and (2). Case (1) corresponds to an analytic model whose internal weights have been updated. Here, the type of model input and output interface stay the same but the internals of the model change. In contrast, case (2) corresponds to an analytic model whose input interface has been modified to include a new input variable. For this model, the input interface and internal weights can be modified but the type of model stays the same.

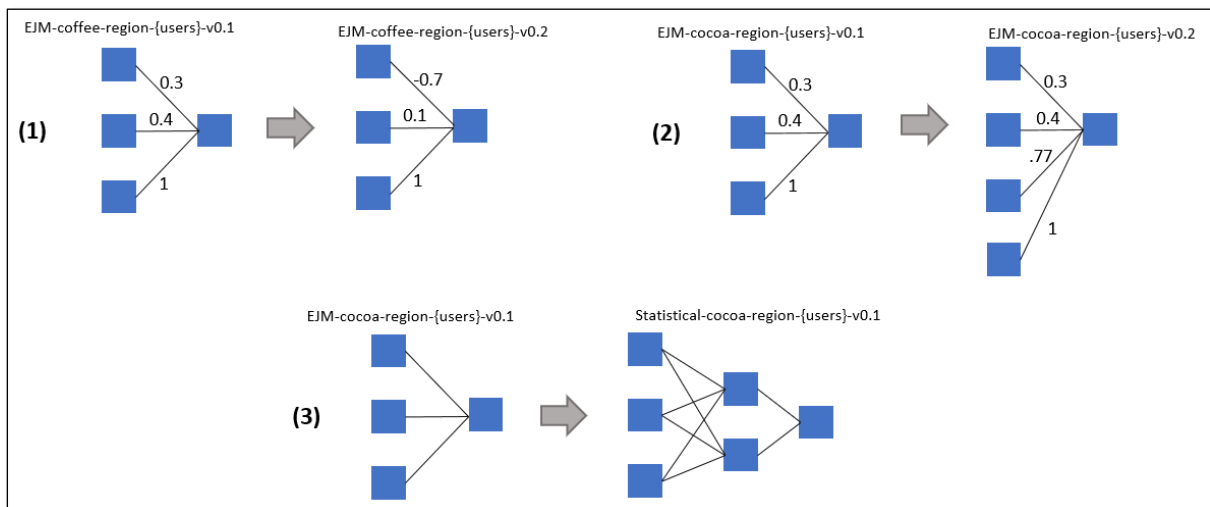


Figure 4: Changes in analytic models that must be supported by the solution architecture, part 1.

Analytic models can also be updated by changing its type. For instance, an EJM (EJM-coffee-mexico-bk-v0.2) can be used for a period while enough data is collected to train a corresponding statistical (ST) model. In this case, the model ST-coffee-mexico-bk-v0.1 corresponds to the update of EJM-coffee-mexico-bk-v0.2. This situation is depicted in case (3) of Figure 4. In this case, all the changes in case (2) can apply as well.

The last two cases shown in Figure 5, (4) and (5), involve the replacement of analytic models that are used to generate credit scores for more than one client on the same type of crop and region. Case (4) can happen when one analytic model that is used to serve two different users is split into two new models because the alternative data related to the farmers of both users can no longer be analyzed with a common model. Here, all changes in case (3) can apply and the resulting models are identified with an ID that indicates an initial version for each new analytic model.

In case (5), multiple analytic models could be merged into one because the alternative data related to farmers served by different credit providers can be analyzed using a common model. In this case, all changes in case (3) apply and the resulting ID indicates the initial version of model used for several users.

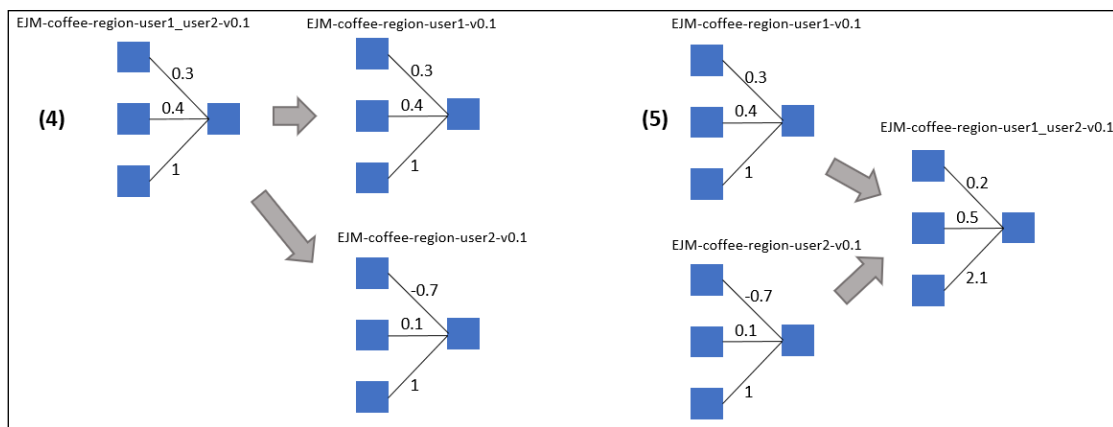


Figure 5: Changes in analytic models that must be supported by the solution architecture, part 2.

Notice that updating a model requires that the number of users served by the model, the type of crop, and the region stay constant. In contrast, replacing a model allows the number of users to change but the type of crop and region must stay the same. In both cases, the evolution of models for a certain client can be traced over time using the ID parameter.

4. Requirements elicitation

This section adds on the preliminary analysis described in the previous section by describing the requirements of the proposed solution architecture and how these requirements were obtained.

4.1 Requirements elicitation process

The obtained requirements reflect some of the main concerns of several actors involved in the design, development, and usage of the credit scoring application (CSA). In particular, these requirements focus on enabling the monitoring of the predictions made by CSA's analytic models and not on making CSA a suitable product for the credit providers.

In this work, three types of requirements were considered: business requirements, functional requirements, and non-functional requirements. The business requirements identify the concerns and wishes of the model owner, who is the person responsible for the analytic models deployed in production. The functional requirements describe the main functions that should be provided by the proposed monitoring solution. Furthermore, the non-functional requirements describe main characteristics of the solution that enable the business and functional requirements.

The requirements were collected using three main strategies:

- Interviews: All the stakeholders mentioned in Section 2 (Stakeholder Analysis) were interviewed using questionnaires. Most of the business requirements were obtained from these interviews.
- Software development: During this project, CSA was still under development. To understand better the concerns of the data scientists, data engineers, and the solution architect, the trainee participated in some software development activities. This first-hand experience resulted in some technical and non-technical requirements.
- Meetings with CSA users: Some of the CSA users (credit providers) reside in South America and speak Spanish. Due to this situation, the trainee helped to translate information during meetings and via email. Directly observing the needs of the credit providers helped to improve some technical and non-technical requirements and to obtain a better understanding of how CSA is intended to be used by the credit providers.

Upon collection, the requirements were presented to the business architect, the solution architect, members of the CSA team, and Rabobank's data scientists. Their feedback was used to update, remove, or include requirements.

4.2 Business requirements

These requirements are composed of an ID, a title, a description, and a rationale explaining the importance of a such requirement. The description is provided in the format of a user story: *as a [role], I want to [activity] so I can [reason behind the activity]*. This format is used by Rabobank architects to document business requirements in the architecture documents (Gallo, Heil, & Bosman, Credit Scoring Project Start Architecture, 2020).

Id: BR1

Title: Monitor input data distribution

Description: As a model owner, I want to monitor the distribution of the input data over time so I can identify deviations in the expected distributions that could impact the quality of predicted credit scores.

Rationale: The predictions of analytic models depend on their input data. As soon as the input data presents changes in its expected distribution, the predictions generated by analytic models could be affected and their accuracy could significantly decrease (Hulten, 2018). In data intensive applications, the properties of processed input data tend to change over time (data drift) and eventually such changes make existing analytic models obsolete. Therefore, to identify data drift and minimize its impact in analytic models, the contents of input data must be constantly monitored.

Id: BR2

Title: Monitor responsiveness of prediction engine

Description: As a model owner, I want to monitor the responsiveness of the prediction engine over time so I can identify responsiveness issues that could damage the user experience.

Rationale: In order to provide a good user experience, CSA must react to user actions within a certain time. Generating credit scores is the main functionality offered by CSA and it should not be slow. The responsiveness of the prediction engine has a high impact on the time required to generate credit scores. Therefore, changes in the responsiveness of the prediction engine should be monitored to identify issues that can affect the user experience.

Id: BR3

Title: Monitor accuracy of analytic models

Description: As a model owner, I want to monitor the accuracy of analytic models to identify when models decay in performance and the corresponding cause of this decay.

Rationale: Deployed analytic models are expected to generate predictions with accuracy above certain threshold. If the accuracy of such predictions falls below such a threshold, the corresponding analytic models must be properly analyzed to identify the cause of such decay in performance. Once the cause is identified, the models can be updated, replaced, or even kept as they are when the cause of decay does not merit any change. By constantly monitoring the accuracy of the predictions, underperforming models and their respective causes can be identified to be handled properly.

Id: BR4

Title: Monitor biased analytic models

Description: As a model owner, I want to monitor the fairness of analytic models' predictions so I can identify models that generate biased predictions.

Rationale: Even when financial analytic models generate accurate predictions, such predictions can benefit certain populations and affect vulnerable populations. Historical data used to train analytic models can contain bias against unrepresented groups. In order to provide accurate credit scores for all smallholder farmers, the predictions generated by analytic models should be unbiased. Therefore, analytic models must be monitored to identify biased predictions as soon as they appear and take corresponding actions to minimize such biases.

Id: BR5

Title: Upgrade/downgrade version of an executed analytic model

Description: As a model owner, I want to be able to replace the version of an analytic model that is being used to generate credit scores (active analytic model) so I can select the most appropriate analytic models for our users according to their needs.

Rationale: Sometimes deployed models present critical issues in production and fixing such problem requires a significant amount of time. In such cases, a temporary solution to the problem is to (re)deploy the previous version of the problematic model while the data scientists fix or create a replacement for this analytic model being replaced. In addition, temporary changes in the input data provided by a credit provider might require to temporarily use a different version of the current analytic model. In this case, the version of the active analytic models should be modified accordingly.

Id: BR6

Title: Ensure testing for deployed analytic models

Description: As a model owner, I want to be sure that every analytic model deployed in production has been properly tested to minimize the risk of having model-related errors in production.

Rationale: CSA is expected to eventually use many analytic models to support their users. Each of these models could be the source of errors in production if they are not properly tested before they are deployed. Therefore, there must be a mechanism that ensure that every model that is deployed has been properly tested. This mechanism should be automated to avoid human errors that can occur when performing a repeated task several times.

Id: BR7

Title: Receive alerts

Description: As a model owner, I want to receive alerts when the analytic models in production present critical issues so that I can request an immediate intervention by the team to solve such an issue.

Rationale: Any software system is susceptible to unexpected errors that can affect their users. When these errors occur, immediate action must be taken to fix such a problem as soon as possible. By implementing an alerting system that works based on how the analytic models perform, the CSA team will know which model-related issues are critical and require immediate intervention and which ones can be considered normal issues.

4.3 *Functional requirements*

The following requirements describe the functionality supported by the monitoring solution. These requirements use the MoSCoW prioritization method (Cline, 2015) for indicating their importance: must have > should have > could have > will not have. However, the word *have* is replaced for another verb to put an emphasis on actions enabled or disabled by the system.

Id: FR1

Title: Select period for monitoring

Description: The system **must allow** monitoring input data and predictions over specified timeframes.

Rationale: The users of the monitoring engine will benefit if they can filter the monitoring functions by specific time frames (1 week, 1 month, or 1 year).

Id: FR2

Title: Select input data parameters for monitoring

Description: The system **must allow** detecting changes in the distribution of selected input data parameters.

Rationale: Some input data variables are more relevant than others. Therefore, the monitoring engine must be able to monitor the distribution of input data parameters specified by its users.

Id: FR3

Title: Detect biased predictions

Description: The system **must allow** the detection of biased predictions based on input data values selected by the users.

Rationale: One way to identify biased prediction is by comparing the predictions resulting from different input data values. By allowing the users to select such input data values, the engine can provide monitoring results according to the characteristics of each analytic model and its input data.

Id: FR4

Title: Compare predictions of models

Description: The system **should enable** users to compare the predictions of different analytic models.

Rationale: In some cases, the predictions of different analytic models must be compared to answer questions such as: what kind of input data results in good predictions for most of the models and what

kind of input data results in biased predictions for most of the models. Thus, the monitoring engine should allow this type of comparison as long as the models can process input data with the same properties.

Id: FR5

Title: Compute accuracy using historical repayment behavior

Description: The system **must allow** users to observe the accuracy of historical predictions using data related to repayment behavior.

Rationale: Data related to repayment behavior indicates whether provided loans were repaid by the farmers. Such data is not available at the moment of the predictions, but it will be provided over time by the credit providers. Using this data, the accuracy of the models can be computed.

Id: FR6

Title: Prevent invalid model upgrades and downgrades

Description: The model deployment pipeline **must prevent** users from upgrading or downgrading analytic models in execution with analytic models not supported in the backend.

Rationale: The input interface of model executed by the prediction engine must be supported by the existing feature extractor code. In some cases, deploying a new version of a model will require updating the code of the feature extractor code. In this case, analytic models should not be upgraded or downgraded before the feature extractor code is updated to support such a change in the analytic model version. This kind of behavior should be enforced by the model deployment pipeline.

Id: FR7

Title: Alerting

Description: The system **could alert** users when the monitored properties of input data or model predictions fall below a certain threshold.

Rationale: This requirement is related to BR7. By having thresholds for the different input data and model prediction properties to measure, the system should be able to generate alerts that can inform its users about critical or unexpected situations.

4.4 ***Non-functional requirements***

The following requirements indicate the main properties the proposed solution should have in order to support the implementation of the functional requirements. Such properties are traceability, security, data integrity, among others. Here, the MoSCoW prioritization strategy also indicates the relevance of these requirements.

Id: NF1

Title: Deployment of different types of analytic models

Description: The system **must support** the monitoring of all deployed analytic models.

Rationale: Over time, CSA aims to serve several credit providers, which will have specific needs. Each of these models must be monitored to ensure they work as expected or identify issues. Thus, the system must support the monitoring of every deployed model.

Id: NF2

Title: Analytic model versioning

Description: The system **must support** versioning of analytic models.

Rationale: Over time, the needs of some users can change, and therefore, the corresponding analytic models should be updated. Versioning can help to keep track of the evolution of these analytic models. The proposed monitoring solution must be able to use versioning information to enable tracking the evolution of models over time.

Id: NF3

Title: Test analytic models before deployment

Description: The system **must monitor** only analytic models that have been properly tested before deployment.

Rationale: In order to prevent deploying malfunctioning analytic models, all deployed models must be tested to ensure that at least, their input interface and output interfaces are supported by the monitoring and model execution code, they do not make trivial prediction mistakes, and they do not generate trivial biased predictions, among others. Users of the monitoring solution must be certain that all monitored models have been previously tested, and if this is not the case, they should know that.

Id: NF4

Title: Prevent usage of identifiable personal data

Description: The system **must prevent** the storage or usage of identifiable personal data for monitoring purposes.

Rationale: Identifiable personal data should not be used by the model for prediction. Moreover, by combining several variables it is sometimes possible to relate input data to groups of people or even individuals. As the system can combine variables in the input data for monitoring (FR2), the system should also ensure that such combinations do not result in identifiable personal data. Otherwise, monitoring results could (in theory) be used for unethical purposes.

4.5 *Use cases*

Use cases indicate activities that the users of the system should be able to accomplish. Most of these use cases are related to a model owner, who is the person responsible for the analytic models once they are deployed. The following are use cases derived from the requirements:

- UC1: a model owner generates a report analyzing input data and model predictions for a specific model over a period.
- UC2: a model owner generates a report comparing the analytic model predictions for different versions of the same model.
- UC3: a model owner generates a report containing the accuracy of analytic model predictions based on data related to repayment behavior.
- UC4: a model owner receives an alert when an analytic model has generated biased predictions for a vulnerable population sector during a specified period.
- UC5: a model owner downgrades a deployed analytic model.
- UC6: a data scientist submits a new analytic model for deployment.

5. Feasibility analysis

The domain analysis and requirements described in the previous sections provided the foundation for the solution architecture. However, there were several possibilities that needed to be considered for creating a solution that fits the needs of the credit scoring application (CSA), and Rabobank. This section describes some of the main challenges encountered while considering how to map the requirements and use cases into a sound solution architecture.

5.1 Challenges

The following challenges were considered in detail before the creation of the architecture:

1. Data that describes the repayment behavior of smallholder farmers will not always be available during monitoring activities. How can analytic model predictions be monitored in this case?
2. Analytic models can be manually monitored for a small number of deployed models. However, how should the monitoring tasks be automated?
3. The team involved in CSA have their own way of working. How should the proposed solution architecture be designed to have a minimum impact and help the team to do their work?
4. Validation data was not available at the time the solution architecture was created; however, this architecture must be evaluated. How should the architecture and its implementation design be evaluated?

The first challenge focuses on measuring the quality of the credit scores generated by analytic models. Commonly, monitoring of analytic models is performed by computing statistics such as precision, recall, accuracy and ROC curves (James, Witten, Hastie, & Tibshirani, 2021). Such statistics require a source of truth that can be used to evaluate the predictions of analytic models. In the case of CSA, the predictions (credit scores) do not have a source of truth immediately after they are generated. Such source of truth can be obtained long after (months or perhaps years) the predictions are generated. Therefore, monitoring must be performed considering that such a source of truth will not be available for a long period.

The second challenge focuses on automating the monitoring of analytic models. It is possible to manually monitor the input data and predictions of deployed analytic models, but this is a cumbersome and error prone task when several analytic models must be frequently monitored. In order to automate the monitoring of analytic models, these models must adhere to a specific interface, such that a monitoring software can load them, execute them, and if required explore their internal elements. Therefore, before implementing a monitoring solution, an input/output interface must be defined for all the analytic models that will be monitored.

The third challenge focuses on the tasks performed by different actors such as data engineers, data scientists, solution architects, and model owners. Each of these actors can be involved in the creation, deployment, and monitoring of analytic models. These actors have a way of working according to their expertise and the tasks they must perform. The proposed solution architecture must have a minimum impact in these ways of working while enabling model monitoring according to the requirements.

The fourth challenge impacts the decisions taken while tackling the first three challenges. In the absence of data to validate the elements of the proposed monitoring solution for CSA analytic models, other validation methods must be considered to evaluate the outcomes of the design decisions resulting from the first three challenges.

5.2 Design decisions

5.2.1. Measuring the quality of predicted credit scores

In machine learning, unsupervised methods are used when the data to be analyzed does not have a source of truth (labels) that can be used to evaluate the results of the analysis (James, Witten, Hastie, & Tibshirani, 2021). Some of these methods focus on anomaly detection (Chandola, Banerjee, & Kumar, 2009), whose purpose is to identify observations that, based on certain metrics, appear to be outliers from the rest of the observations. Anomaly detection can be used for detecting spam, invalid credit card payments, among others.

Similar to anomaly detection, by identifying significant changes in the input data or predicted credit scores, model owners and data scientists can know when deployed analytic models perform different than expected. Table 1 shows some of the methods considered for measuring the quality of predicted credit scores (and input data) in the absence of a source of truth.

The main factors for selecting a change detection method were:

- Given that the distribution of the input data and predictions is unknown (fourth challenge), the selected method must not make assumptions in the distribution of the observations under analysis.
- Considering that the CSA team focuses on creating appropriate models for each of their users, the selected method must not be more complex than any of the models under development.
- Methods that are already implemented in well-known public software are preferred over those that must be implemented based on their description.
- Methods that do not require training are preferred.

Table 1: A sample of methods for identifying changes in observations. These methods are evaluated according to four properties. Based on these properties, the KS statistical test is preferred over the other methods.

Method	Does not assume distribution of observations	Low complexity level	Implemented in well-known public software	No training required
Quartile based method	✗	✓	✓	✓
<i>KS statistical test</i>	✓	✓	✓	✓
Gaussian methods for anomaly detection	✗	✗	✓	✗
Deep learning methods for anomaly detection	✓	✗	✓	✗

Based on the criteria mentioned before, the Kolmogorov-Smirnoff (KS) test (Kolmogorov-Smirnov equality-of-distributions test, 2018) was selected. In short, this statistical test computes and compares the distributions of two sets of observations. The output of this method is a value that can be used to evaluate the similarity of both sets of observations. Based on a threshold value, if both distributions are different, then the observations can be considered to be different.

Using the KS test as a method for measuring similarity, subsets of input data and credit scores can be compared to detect unexpected changes. Such subsets can be created based on time frames or specific values. The usage of the KS test for monitoring is explained in more detail in Section 7.5.

5.2.2. Automating the monitoring of analytic models

Analytic models deployed to production can be encoded as source code, data files, or a combination of both. Analytic models encoded as source code can be included as part of the application software that executes them. They allow the easiest integration level because problems in the input/output interfaces of the models are detected as soon as they appear due to errors in the overall software. Nonetheless, deploying new models require changes to the software that uses them. In this way, their use is not advised when new models must be deployed frequently to a software whose functionality rarely changes.

On the contrary, analytic models encoded as data files can be used when models must be deployed frequently but the software executing such models does not need to change for every newly deployed model. However, these data files must be properly tested to ensure that they can be loaded and used by the corresponding application software. Otherwise, newly deployed models can lead to unexpected errors in production.

A third encoding approach is to create models as source code and save this code in files that can be loaded and executed by the corresponding application software. Once loaded, such files can be converted back into the original code and used by the software. As the source code offers input/output interfaces, this strategy facilitates testing analytic models before they are used, and it also offers the same advantages as encoding models as data files. However, only trusted models must be executed by the application software to avoid loading malicious code that could result in security issues.

Table 2 shows the characteristics used to decide the encoding format for analytic models that facilitates analytic model monitoring.

Table 2: Encoding formats for analytic models that should be deployed in production

Model encoding	Simple implementation	Minimum integration testing required before deployment	Support frequent deployment	Allows tracking the evolution of models
As code	✓	✓	✗	✗
As data files	✗	✗	✓	✓
<i>As source code stored in data files</i>	✗	✗	✓	✓

The most important characteristics for this decision were: supporting of frequent deployment without modifying the application software and allowing tracking the evolution of models (i.e., evaluate the quality of the predicted credit scores over time for different versions of a model).

Given that the CSA team uses tools that support the development of analytic models as code, source code stored in data files was selected as the model encoding approach for the solution architecture. In this case, such data files contain the code of analytic models and also metadata that can be used for their monitoring.

5.2.3. Enabling the collaboration between different actors

By working together with the CSA team, we obtained a better understanding of the working process followed by data scientists, data engineers, and a DevOps engineer. In short, the data scientists and data engineers agree on the format of the input data and the input/output interfaces of the analytic models. Then they work on their corresponding tasks while ensuring that the application software works as expected after changes are made to the data processing pipeline and prediction engine. The DevOps

engineer is responsible for creating and maintaining the deployment pipeline that runs the required tests to ensure changes to the application software (and models) can be deployed to production.

Considering the proposed encoding approach for analytic models, the way of working of the data scientists is the most affected. To help the data scientist in creating (or updating) and deploying analytic models, a series of unit tests for analytic models are designed based on the works of (Breck, Cai, Nielsen, Salib, & Sculley, 2016) and (Ribeiro, Wu, Guestrin, & Singh, 2020). These unit tests ensure that:

- The input and output interface of analytic models is already supported in the application software and monitoring solution.
- Considering that a testing dataset is available, the analytic models generate credit scores with the required accuracy.
- The analytic model does not generate biased credit scores that target specific groups of people.

6. Solution architecture

This section presents the solution architecture created for monitoring credit scoring analytic models. The elements of this architecture were created following the constraint and ideas discussed in the domain analysis, requirements elicitation, and feasibility analysis sections.

6.1 Description

The Rabobank model development lifecycle (Gallo, Heil, & Bosman, Credit Scoring Project Start Architecture, 2020) depicted in Figure 6 is a general guideline used by the credit scoring application (CSA) team to create and maintain credit scoring analytic models. This diagram indicates the relationship between activities that belong to the following processes: model data development, model development, model implementation, and model monitoring. These processes are composed of activities; for instance, model implementation is composed of model deployment and model execution.

The processes and activities in this model development lifecycle are adapted to the needs of the team. For instance, due to the type of data required to create credit scoring analytic models, it is unlikely that the whole data acquisition activity can be reused for other teams working in similar product. However, some of elements of this activity can be reused by other teams.

In this project, we focused on providing a solution architecture for the model monitoring activity that fits the needs of the CSA team. Even though this solution architecture was designed for this particular team, most of its elements can be reused in other projects that make use of analytic models. In addition to the model monitoring activity, this solution architecture covers model deployment and model execution to ensure that model monitoring can be performed according to the requirements.

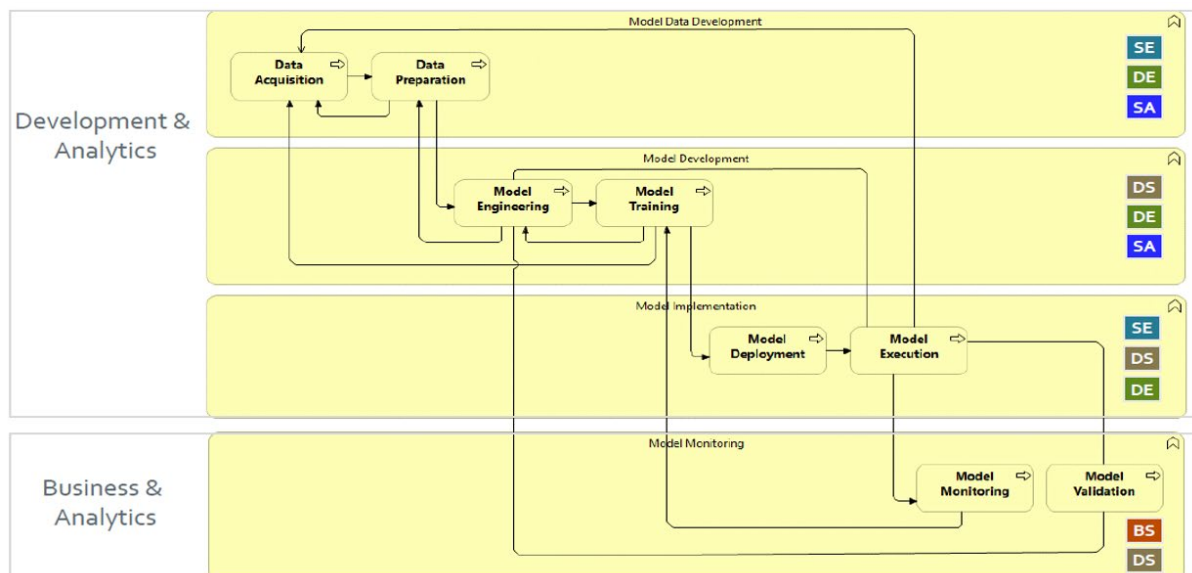


Figure 6: Rabobank model development lifecycle containing four processes: model data development, model development, model implementation, and model monitoring. This diagram shows the activities and roles related to each process. The roles shown in this diagram are software engine (SE), data engineer (DE), solution architect (SA), data scientist (DS), and business stakeholder (BS).

Figure 7 shows the proposed solution architecture for the monitoring of credit scoring analytic models. As shown in Table 3, this diagram contains three types of components: components already existing in CSA, components required to support model monitoring, and components for model monitoring.

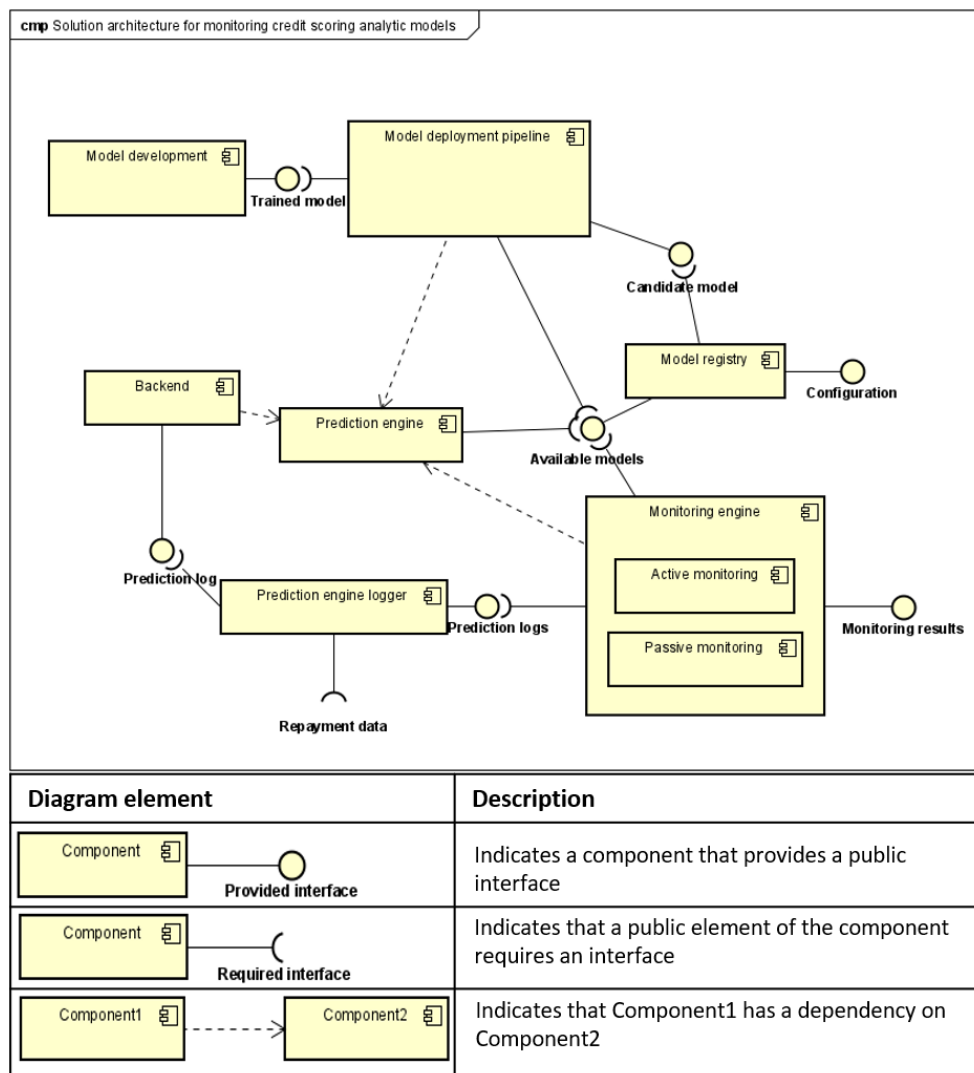


Figure 7: Proposed solution architecture for monitoring credit scoring analytic models.

Table 3: Relationship between the solution architecture components and their role on the monitoring of analytic models.

Component	Already exists in CSA	Supports analytic model monitoring	Used for analytic model monitoring
Model development	✓		
Backend	✓		
Model deployment pipeline		✓	
Model registry			✓
Prediction engine			✓
Prediction engine logger			✓
Monitoring engine			✓

The proposed solution architecture can be read as follows:

- The model development component represents the process followed by data scientists to create (train) analytic models.
- The model deployment pipeline uses an instance of the prediction engine to validate trained analytic models and models available in the model registry. This pipeline provides candidate models.
- The model registry stores candidate models that were accepted by the model deployment pipeline. Its stored models are available to the model deployment pipeline, model deployment engine, and monitoring engine. The contents of the model registry can be modified using a configuration interface.
- The backend component uses an instance of the prediction engine to generate predictions. These predictions are stored in the prediction engine logger. The interaction between the backend and the frontend application has been omitted in this diagram.
- The prediction engine logger accepts a prediction log composed of input data and predictions. It also accepts repayment data that can be provided by a third party. The prediction engine logger provides prediction logs that can be composed of input data, predictions, and repayment data.
- The monitoring engine uses an instance of the prediction engine and passive monitoring and active monitoring tasks. This engine monitors the contents of the prediction logs and the models available in the model registry.

6.2 *Relationship with requirements and use cases*

The relationship between the solution architecture and the requirements described in Section 4 are shown in Table 4. Requirements are considered supported (S), not validated (NV), or not supported (NS).

Not validated requirements are those that solution architecture supports in theory but could not be validated with our prototype described in Section 8. The two not validated requirements (BR3 and FR5) are related to monitoring the accuracy of prediction. They were not validated because we did not have truth predictions (labels) to validate that the accuracy of the predictions can be monitored. However, the architecture has been designed to support the monitoring of statistical metrics that require truth labels.

Only three requirements are not supported by the solution architecture. These requirements are:

- BR7: Receive alerts
- FR7: Alerting
- NF4: Prevent usage of identifiable personal data

The first two not supported requirements were not included in the solution architecture because the focus of the project shifted from generating alerts to ensuring that every deployed analytic model can be monitored. Regarding identifiable personal data, we consider that this requirement should be handled by the backend itself. Every combination of input data used for prediction should not contain personal identifiable information. As this input data should be directly stored in the prediction engine logger, this logger should also not contain such type of information.

Table 4: Relationship between the solution architecture and the requirements.

Requirement	Components	Status
BR1: Monitoring input data distribution	Prediction engine logger Monitoring engine	S
BR2: Monitor responsiveness of prediction engine	Prediction engine logger Monitoring engine	S
BR3: Monitor accuracy of analytic models	Prediction engine logger Monitoring engine	NV
BR4: Monitor biased analytic models	Prediction engine logger Monitoring engine	S
BR5: Upgrade/downgrade version of an executed analytic model	Model deployment pipeline Model registry	S
BR6: Ensure testing for deployed analytic models	Model deployment pipeline	S
BR7: Receive alerts		NS
FR1: Select period for monitoring	Monitoring engine	S
FR2: Select input data parameters for monitoring	Monitoring engine	S
FR3: Detect biased predictions	Monitoring engine Prediction engine logger Model registry	S
FR4: Compare predictions of models	Monitoring engine Prediction engine logger Model registry	S
FR5: Compute accuracy using historical repayment behavior	Prediction engine logger Monitoring engine	NV
FR6: Prevent invalid model upgrades and downgrades	Model deployment pipeline Model registry	S
FR7: Alerting		NS
NF1: Deployment of different types of analytic models	Model deployment pipeline	S
NF2: Analytic model versioning	Model registry	S
NF3: Test analytic models before deployment	Model deployment pipeline	S
NF4: Prevent usage of identifiable personal data		NS

The proposed solution architecture enables three main activities related to the use cases described in Section 4.5:

1. Deploying analytic models (UC6)
2. Replacing models being executed by the prediction engine with models stored in the model registry (UC5)
3. Monitoring analytic models (UC1, UC2, and UC3)

These activities are described in more detail in Appendix A.

7. Implementation design

This section describes the implementation design of the solution architecture described in the previous section. The focus is on the following components: model deployment pipeline, model registry, prediction engine, prediction engine logger, and monitoring engine. These are the elements that enable monitoring credit scoring analytic models according to the requirements described in Section 4.

7.1 Prediction engine

For this component, the focus is on designing an interface implemented by all types of analytic models such that these models can be loaded and executed. By implement this interface, the prediction engine can be used to:

- Test candidate models in the model deployment pipeline
- Execute the models and generate predictions in the backend
- Monitor the models using active monitoring tasks in the monitoring engine

The interface of analytic models is depicted in the class diagram shown in Figure 8. According to this diagram, each analytic model must implement a method called *predict*, which takes input data as a dictionary and generates credit scores as a float value. In this diagram, only statistical models and expert judgement (EJM) models have been modeled. Nonetheless, if required, this diagram can be modified to include other types of analytic models as long as they adhere to the methods required by the interface AnalyticModel.

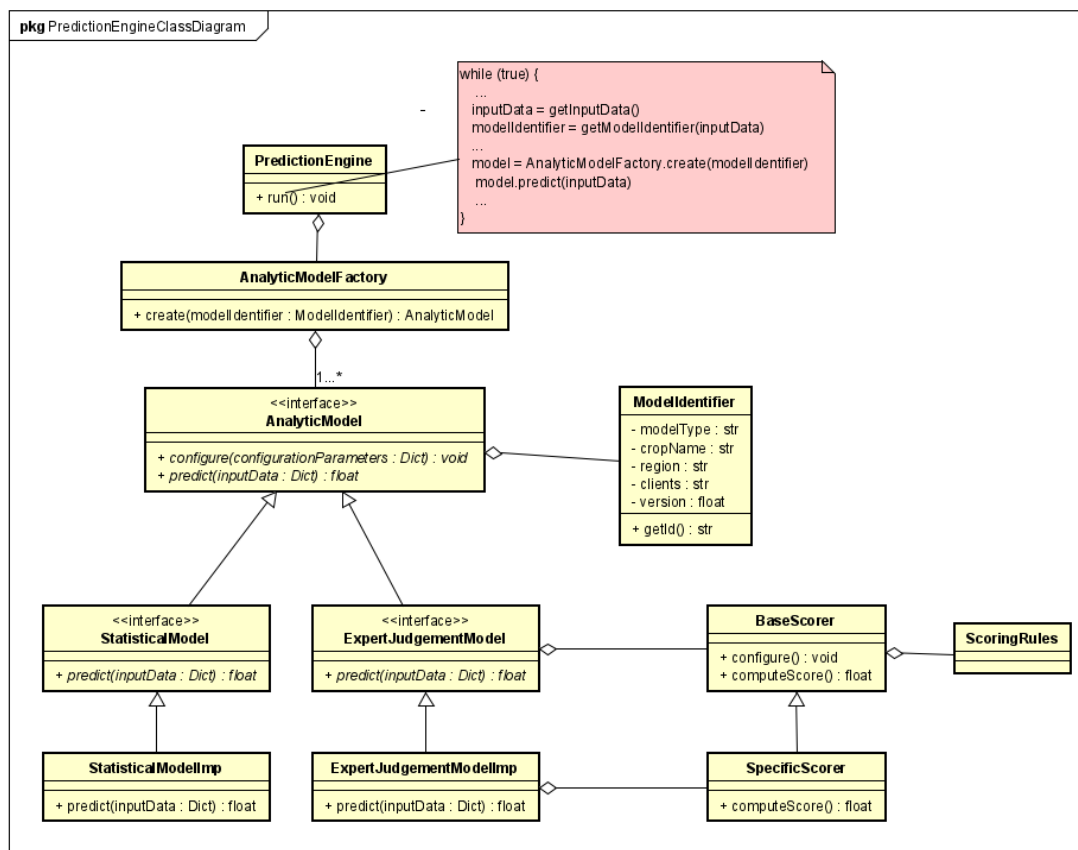


Figure 8: Class diagram for the prediction engine. A general analytic model interface is defined that all types of analytic models must implement such that the prediction engine can use any specific analytic model implementation.

The main difference between statistical models and EJM models is how their weights are set and used to compute predictions. The statistical model class (`StatisticalModelImp`) that implements the statistical model interface (`StatisticalModel`) does not require a class to model its weights. `StatisticalModelImp` is designed this way because it is considered a wrapper around existing statistical model implementations from well-known libraries such as TensorFlow² or scikit-learn³. Such libraries have their own mechanisms for handling model weights.

In contrast, implementations of the EJM model interface (`ExpertJudgementModel`) require a subclass of the `BaseScorer` class to compute intermediate scores based on their corresponding scoring rules. These scoring rules are related to the specific input variables that the EJM models must ingest. The logic of these rules is implemented as code and can vary for each instance of the EJM model implementation class (`ExpertJudgementModelImp`). The general logic required for all EJM models is handled by the `BaseScorer` while the specific rules are handled by the `SpecificScorer`.

The `PredictionEngine` class uses an `AnalyticModelFactory` class to create instances of the analytic models that should be executed. Each of these instances is associated with a model ID (`ModelIdentifier`). The annotation provided in the `PredictionEngine` class shows a piece of pseudocode that describes how analytic models can be created and used to generate predictions.

Based on the prediction engine class diagram, we can observe that the prediction engine enables the following requirements: deployment of different types of analytic models (NF1) and analytic model versioning (NF2).

7.2 *Model deployment pipeline*

The model deployment pipeline focuses on testing all deployed analytic models such that models that pass the tests can be executed and monitored afterwards. In this project, we identified four types of tests that should be implemented in the model deployment pipeline:

- A test to validate the model input interface
- A test to validate the model output interface
- A test to validate the accuracy of the predictions of a model on a testing dataset
- A test to validate that a deployed model does not perform poorly on sensitive groups of data

Each of these tests is explained below.

ID: MDPT1

Name: Validate the model input interface

Rationale: Ensures that deployed analytic model can ingest input data passed by the backend to the prediction engine. Deploying an analytic model with a wrong input interface will lead to errors when the prediction engine executes the model to generate predictions.

Procedure:

- Generate input data according to specifications from the interface of the analytic model being tested.
- Pass this input data to the model.
- If the model generates a prediction for every input data observation, this test succeeds.

ID: MDPT2

Name: Validate the model output interface

Rationale: The backend and prediction engine logger expect the generated predictions in a certain format. A wrong output format will lead to errors when processing and storing the predictions.

Procedure:

² <https://www.tensorflow.org/>

³ <https://scikit-learn.org/stable/index.html>

- Select the predictions generated in MDPT1.
- Compare the format of these outputs to the one the model is expected to generate.
- If both formats match, this test succeeds.

ID: MDPT3

Name: Validate the accuracy of a model on a testing dataset

Rationale: In case testing data is available, all deployed models should be tested using this data to ensure their predictions achieve a certain level of accuracy. Otherwise, we might deploy models from the beginning have a low performance.

Procedure:

- Selected a proper testing dataset according to the characteristic of the model being tested.
- Generate predictions using this dataset.
- Compute the accuracy of using these predictions.
- Compare this accuracy against a threshold value.
- The test succeeds if the accuracy is above the threshold value.

ID: MDP4

Name: Validate that a model does not perform poorly on sensitive groups of data

Rationale: Besides having good accuracy on a test dataset, deployed models should provide quality predictions for all relevant subsets of data. In particular, deployed models should not obtain good testing accuracy at the expense of sensitive groups of data. By preventing this situation, we reduce the chance of deploying a biased analytic model.

Procedure:

- Identify sensitive subsets in the testing data for which the model generates predictions with low accuracy.
- For each subset, generate the corresponding predictions and compute the associated accuracy.
- The test succeeds only when all the accuracy values are above a required threshold.

The previously describe model deployment tests help us achieve the following requirements:

- BR6: ensure testing for deployed analytic models
- NFT3: test analytic models before deployment

7.3 *Model registry*

The model registry focuses on storing analytic models and their metadata such that they can be used by the prediction engine, the monitoring engine, and sometimes the model deployment pipeline.

To track and monitor models over time, the model registry requires at least the following information items:

- Model ID
- Serialized code of the model
- Description of the model
- Upload date
- Status of the model
- Input interface format
- Output interface format

The model registry must store different versions of the same model. Some of these models can be currently used by the prediction engine while others are waiting to be activated. Also, some stored analytic models might no longer used but are kept for traceability and monitoring purposes.

The state machine shown in Figure 9 illustrates a proposal for managing the status of models stored in the model registry. This diagram enables the following simple analytic model execution lifecycle:

- All analytic models uploaded to the model registry are initially considered *candidate* models.
- Candidate models can be deployed in parallel to existing active models that should be replaced.
- When a candidate model generates better predictions, its model status can be updated to *active*, and the prediction engine should use this model for the new predictions being shown to the users.
- Active models replaced by candidate ones should be considered *inactive* and the prediction engine should no longer use them.
- In case candidate models need to be downgraded, inactive models that can be loaded and executed by the prediction engine can be considered candidate models.
- In the case that a candidate model does not perform better than the currently deployed model that it should replace, this model should have a status called *inactivated* indicating that it passed the model deployment pipeline, but it was never used to provide predictions to the user.

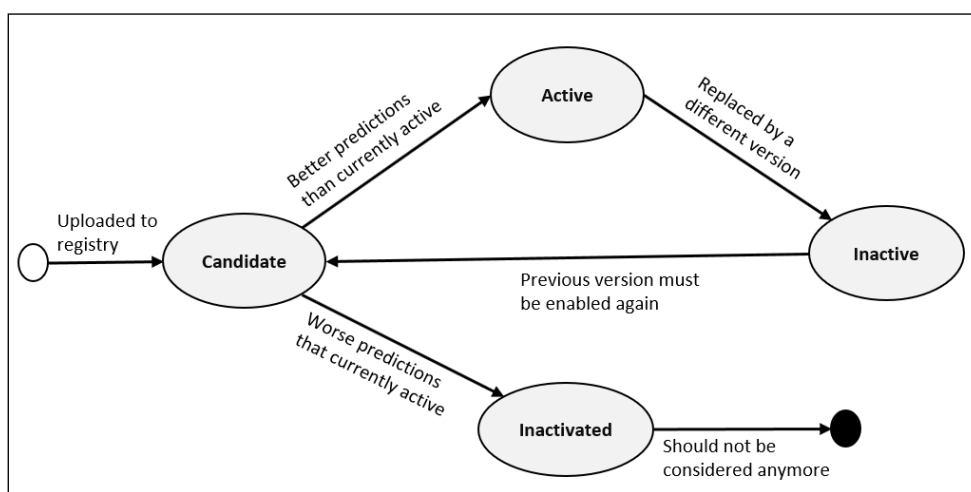


Figure 9: state machine representing the different status that analytic models can have.

The model registry enables the following requirements:

- BR5: Upgrade/downgrade the version of an executed analytic model
- FR6: Prevent invalid model upgrades and downgrades

7.4 Prediction engine logger

The monitoring engine requires historical information about the predictions of analytic models and their corresponding input data. At least, the following information elements should be stored in the prediction engine logger for each generated credit score:

- Model id
- Input data used for prediction
- Input data schema
- Prediction value
- Prediction date
- Repayment behavior data
- Date when the repayment behavior data was received

The first five information items can be stored at the moment the prediction is generated. However, the repayment behavior data is provided by the credit provider once they have such data available. Thus, this information element will be available after its corresponding prediction is made. That is why this

information element has a specific date associated with it, which is different from the date of the prediction.

The model prediction logger enables the following requirements:

- BR1: monitoring input data distribution
- BR2: monitor responsiveness of prediction engine
- BR3: monitor accuracy of analytic models
- BR4: monitor biased analytic models
- FR1: select period for monitoring
- FR2: select input parameters for monitoring
- FR3: detect biased predictions
- FR4: compare predictions of models
- FR5: compute accuracy using historical repayment data

7.5 Monitoring engine

For the monitoring engine, the focus is on the monitoring tasks that should be executed by the active and passive monitoring components. Active monitoring tasks execute analytic models to perform live monitoring while passive monitoring tasks perform analysis using only input data and their corresponding predictions.

In the proposed monitoring solution, active monitoring is mainly used to identify biased predictions using a sensitivity analysis. This analysis, shown in Figure 10, works as follows:

1. Select a set of input data observations over a specific period.
2. Create a copy of this input dataset and modify the values of a variable in the copied dataset.
3. Generate predictions for both the original and modified input data.
4. Compare the distributions of both sets of predictions.
5. Decide whether the predictions are sensitive (not similar enough), or not.

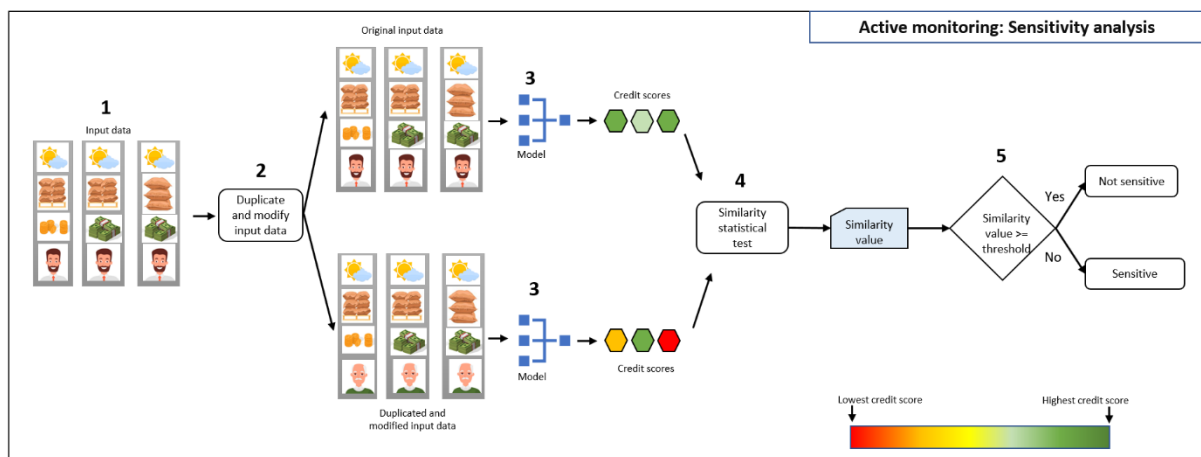


Figure 10: Sensitivity analysis applied to analytic models for credit scoring.

This type of analysis can be used to identify when a model is sensitive to an input variable, i.e., when the model generates biased predictions based on the values of an input variable. For example, for an analytic model that uses *age* as one of its input variables, one could modify the age value of the observations of an input dataset. If certain ranges of age (10-20, 20-30, or 70) are preferred by the analytic model, the sensitivity analysis would indicate whether a specific range of age has a significant impact in the predictions.

Passive monitoring tasks are used to identify changes in the distribution of predictions and the input data over a specified period. In addition, they are also used to compute the accuracy of historical

predictions using true labels that can be computed from repayment behavior data provided by credit providers.

Figure 11 shows a diagram that describes how to identify changes in the distribution of observations (input data or predictions) over time:

- First, observations are selected over a period.
- Then, the selected observations are split into two subsets based on their associated prediction time.
- Next, the similarity distribution of both subsets is compared using a statistical similarity test (in our case, the KS test described in Section 5.2.1).
- Finally, the resulting similarity value is compared against a similarity threshold to identify significant changes.

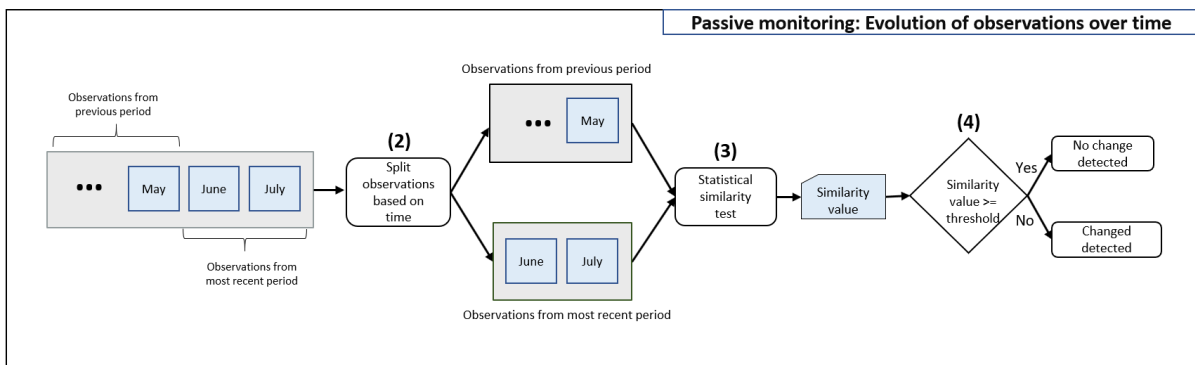


Figure 11: Identifying changes in the distributions of observations over a specific period.

The computation of accuracy for historical predictions is described in Figure 12. This monitoring task works as follows:

- First, reference credit scores should be obtained from data containing repayment behavior.
- Then, the predicted credit scores corresponding to the expected input scores should be selected from the prediction engine logger.
- Next, using both scores, the accuracy of the predictions is computed.
- Finally, the resulting accuracy value can then be compared against a specified threshold to determine whether the predictions correspond to a good or bad performance.

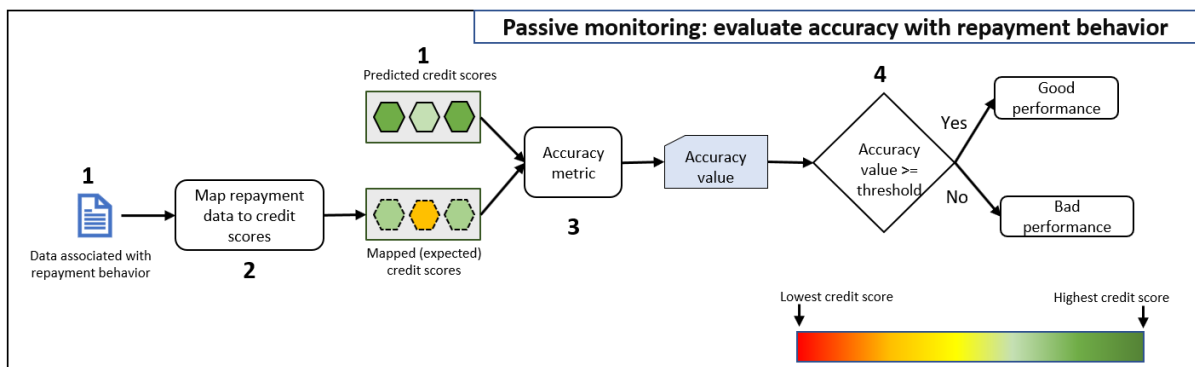


Figure 12: Computing the accuracy of predicted credit scores based on repayment behavior.

The elements of the monitoring engine enable the same requirements that are enabled by the prediction engine logger.

8. Implementation

In this section, we describe the implementation of a prototype that demonstrates the main activities supported by the solution architecture, which are described in Section 6.1.

8.1 Introduction

The prototype was created to provide an example of how the solution architecture can be implemented for the credit scoring application (CSA). For its implementation, two options were considered:

- A standalone implementation that strictly follows the solution architecture but does not use CSA codebase
- An implementation included in CSA that uses simplified versions of some components of the solution architecture

The first option would demonstrate how the solution architecture has been designed to work when CSA handles several credit providers. In this case, automation of model deployment, model execution, and model monitoring tasks can help to create new models and maintain existing ones. Despite the advantages of such an implementation, the current CSA implementation will not benefit much from it. In fact, many changes must be performed to the application codebase while obtaining minimum improvements in the short term.

In contrast, the simplified prototype implemented as part of CSA shows the benefits of automatic model deployment, model execution, and model monitoring while keeping the changes in the codebase to the minimum required. In this case, the prototype can serve as an example to explore the elements that can be implemented in the current code of CSA so that the solution architecture can be implemented in an agile way.

At the moment the prototype was created, the team was building the main functionality of CSA to release their minimum viable product (MVP) to the market. This MVP consisted of a small number of analytic models that can be manually deployed and monitored as well as functionality related to user account management and data ingestion. At this stage, modifications to the code to implement the whole solution architecture could have delay significantly the release of the MVP. Therefore, a prototype of the full solution architecture would likely not be considered until after the MVP was released.

On the contrary, the elements in the simplified prototype could be ported to the credit scoring codebase without significant impact in the schedule of the MVP. Thus, we decided to create the simplified prototype in one of the branches of CSA. Using this implementation, we aim to communicate our solution architecture and implementation design to the team in such a way that they can easily understand their benefits and how to implement its more relevant elements.

8.2 Prototype

The prototype was implemented as a web application that interacted with a backend service. The backend service provided the functionality for the model monitoring engine, model deployment pipeline, and prediction engine. The web application, shown in Figure 13, allowed its users to interact and display the results of the monitoring engine as well as activate candidate models deployed using the model deployment pipeline.



Figure 13: Webapp tab that shows the model monitoring engine functionality of the prototype. The name of the active model being monitored is hidden behind a gray rectangle due to privacy policies.

The monitoring tab of the web application shown in Figure 13 is composed of two elements: user input form and visualization of results. The user input form indicates:

- The active model whose predictions should be monitored
- A variable from the input data that should be monitored
- An integer value used to modify the original values of the monitored input variable in order to perform sensitivity analysis
- The number of records (observations) required for monitoring
- The similarity threshold required by the active and passive monitoring tasks

Notice that based on the number of records (n), two sets of observations are created: the first one corresponding to the latest n observations and the other one corresponding to the previous n observations.

The visualization of results element shows the monitoring results for predictions, input data variables, and a sensitivity test based on the monitored input variable. This functionality corresponds to the third function enabled by the solution architecture: *monitoring of analytic models*. In our prototype, we only monitor one variable at a time instead of all the variables in the input data. However, this implementation can be easily modified to monitor all input variables at once.

In the two graphs shown in Figure 13 we can see that the most recent records are shown as blue circles while the previous records are shown as red circles. We also notice two trend lines that represent the 21 moving average values for both subsets of observations. This means that the first value of the trend lines is the average of the first 21 observations, the second value of the trend lines is the average of subset starting at the second observation up to the 22nd observation, and so on.

The trend lines help to make sense of the monitoring results shown in the text above each graph. For instance, the trend lines of the predictions look similar, and the monitoring result indicates they are similar. Similarly, the monitoring result for the input variable age indicates a change in the distribution of the newest subset of records, which is confirmed by the trend lines.

The results of the sensitivity test are shown in a graph similar to the one created for the input variable. An example of sensitivity results is shown in Section 9.2.3.

The second activity enabled by the solution architecture, *replacing an active model for a candidate model stored in the model registry*, is shown in Figure 14. In this tab, the user can replace an active version of a model for a candidate model. If this operation succeeds, the prediction engine updates the analytic models used for prediction.



Figure 14: Webapp tab that shows the model activation functionality of the prototype.

The remaining activity enabled by the solution architecture, *deploying analytic models*, can be performed using the script `model_deployment_script.py` contained in CSA (backend) code shown in Figure 15. Notice how this backend code resembles the solution architecture and the elements of the prediction engine class diagram shown in Section 6.2.

The model deployment process in our prototype works as follows:

- First, analytic models that are ready to be deployed must be included inside the `trained_models` directory in the CSA code.
- Then, the script `model_deployment_script.py` must be executed.
- After that, the trained models are moved into the `model_deployment_pipeline/models_to_validate` directory.
- Once there, the tests defined in `model_tests.py` are executed using the datasets available in `model_deployment_pipeline/datasets`.
- If the models under validation pass all the tests, they are moved into the `prediction_engine/available_models` directory.
- Finally, the new available models can be activated using the candidate model activation tab of web application, shown in Figure 14.

Every time the status of the models inside the `prediction_engine/available_models` is changed, the prediction engine updates its pool of analytic models used for predicting credit scores. Notice that in our prototype, the model registry is part of the prediction engine. However, in a real implementation the model registry should be separated from the prediction engine as described in the solution architecture.

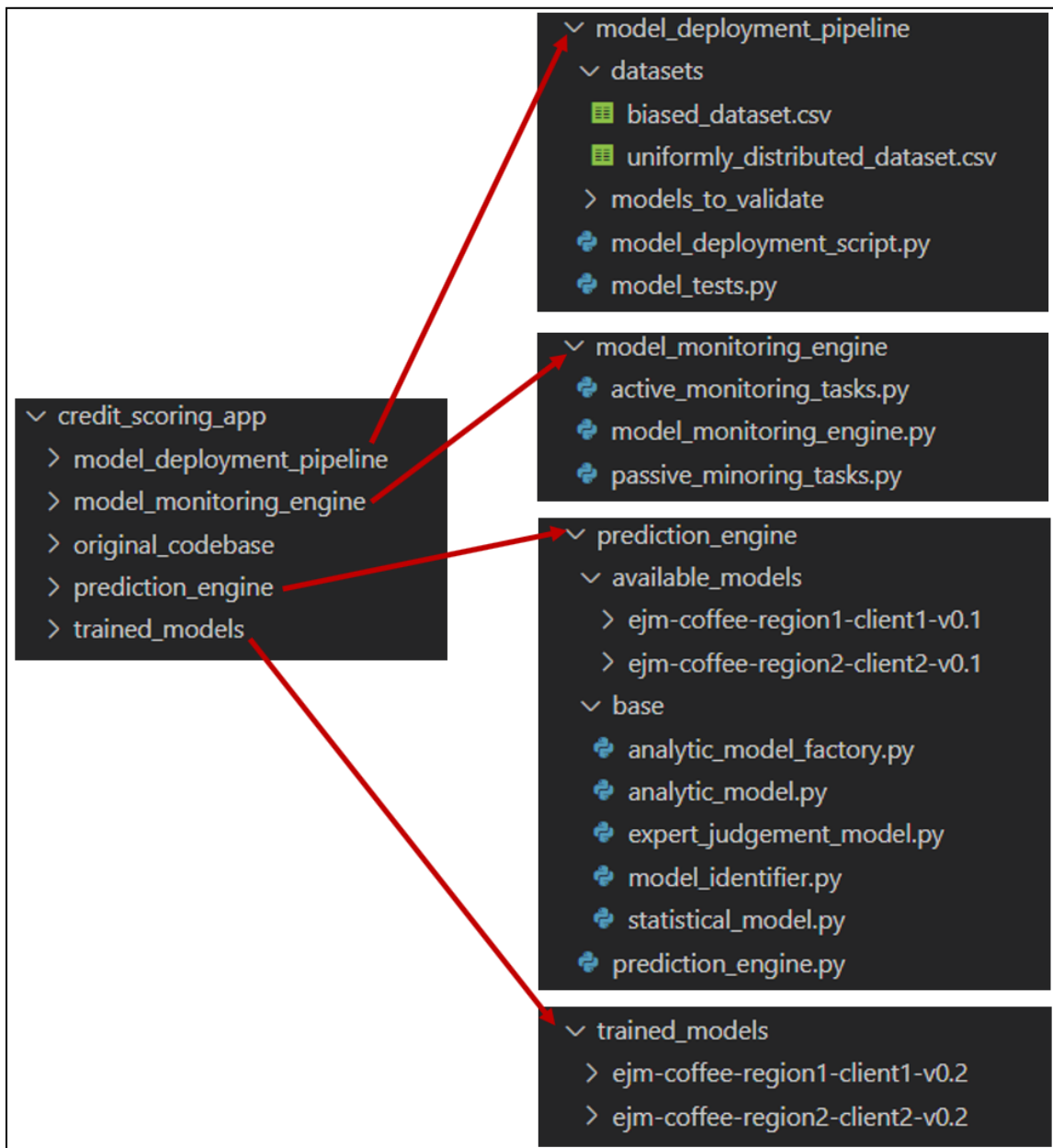


Figure 15: CSA (*credit_scoring_app*) code containing the implementation of the solution architecture prototype. Notice how some of the elements of the solution architecture are mapped directly in the modules of the *credit_scoring_app* code.

9. Verification & validation

This section describes how the solution architecture and implementation design were verified and validated. In this section, we consider that a software architecture is verified to ensure that it fulfills its requirements and there are no incompatibilities in its design. In contrast, the validation of the software architecture can be used to identify issues in the implementation of its design.

9.1 Verification

Our solution architecture was verified to ensure the requirements described in Section 4 were supported. The verification of the solution architecture was performed in collaboration with architects, data scientists, data engineers, and a digital project manager. The process was as follows:

- After each major modification, we presented the solution architecture to one or more reviewers.
- Based on their observations, we improved the solution architecture.

Following this simple procedure, the architecture had two major updates. In the first update, we introduced the model deployment pipeline and model registry to ensure that every deployed model can be monitored. For the second update, we modified the model monitoring engine to support active monitoring tasks such as sensitive analysis.

Based on the latest reviews of the solution architecture and implementation design, we observe that this architecture fulfills most of the requirements described in Section 4. The only requirements not supported were the ones related with alerting (BR7 and FR7). Additionally, requirements related to monitoring the accuracy of predictions (BR3 and FR5) are supported but were not verified or validated.

The details of which requirements are fulfilled by the solution architecture (and by extend the implementation design) are available in Section 6.

9.2 Validation

The validation of the solution architecture was performed using the prototype described in Section 8. Using this prototype, we validated that the three main activities enabled by the solution architecture can be performed:

- Deploying analytic models
- Replacing models being executed by the prediction engine with models stored in the model registry
- Monitoring analytic models

9.2.1. Deploying analytic models

To validate the first operation, we created several analytic models according to the class diagram presented in Section 7 (Figure 8). These models were EJM models with four following input variables. One of these variables corresponds to age values, and the other three are variables related to weather information. The output of these models corresponds to a credit score between 0 and 1000, where 0 indicates a high risk and 1000 indicates a low risk.

The created analytic models were deployed following the instructions provided in Section 8.2. During the deployment procedure, these models were tested using the following tests:

- MDPT1: validate the model input interface
- MDPT2: validate the model output interface (DPT2)
- MDPT4: validate that a model does not perform poorly on sensitive groups of data

For these tests, we created synthetic datasets containing uniformly distributed values for each of the model's input variables. In addition, for MDPT4, some of these datasets were modified to have values that are representative of vulnerable groups. For example, in the case of the variable age, some datasets contain only age values above 50, while the remaining datasets contain age values among the expected age range.

Notice that MDPT3 (validate the accuracy of a model on a testing dataset) was not used for validating the created models. The reason is that because we could not obtain historical predictions of the credit scores obtained from alternative data, we could not identify their statistical properties. Hence, we were not able to generate synthetic truth credit scores to compute accuracy values. Nonetheless, once such truth data is available or can be simulated, these tests could be implemented in a similar way to the other tests.

The results of the model deployment tests indicate that the prototype works as expected regarding the model deployment pipeline. Models with input and output interfaces different than expected are rejected. The same happens for models that are prone to generate biased predictions against input data related to vulnerable groups.

9.2.2. Replacing an active analytic model

The replacement of active analytic models with other models stored in the model registry was tested using the model activation web form shown in section 8.2, Figure 14. This web form requires the id of the candidate model to activate and the id of the active model to be replaced. Once these parameters are filled, clicking on the activate button results in the deactivation of the active model and activation of the candidate model.

The results of this manual operation validated the second operation of the solution architecture, *replacing an active model for a candidate model stored in the model registry*.

9.2.3. Monitoring deployed analytic models

To validate the third operation supported by the solution architecture, we used the web form shown in Figure 13. In addition, we created an input data generator to provide the prediction engine with input data having two types of input variables:

- Input variables with uniformly random generated values
- Input variables with random values that increment slowly over time

Using the input data generator, different models, and the prototype, we manually tested that the monitoring engine can:

1. Monitor predictions based on randomly uniform generated values
2. Monitor predictions based on values that slowly incrementing over time
3. Monitor whether predictions are sensitive (biased) towards certain input variables

From the first two tests we observed that changes that happen over time in input variables can be detected. However, as shown in Figure 16, these changes do not necessarily affect the distributions of the predictions. This result is in line with our expectations because not all input variables have the same effect in the predictions of a model. Thus, it is possible to have large variations in an input variable that have a small effect in the predictions.

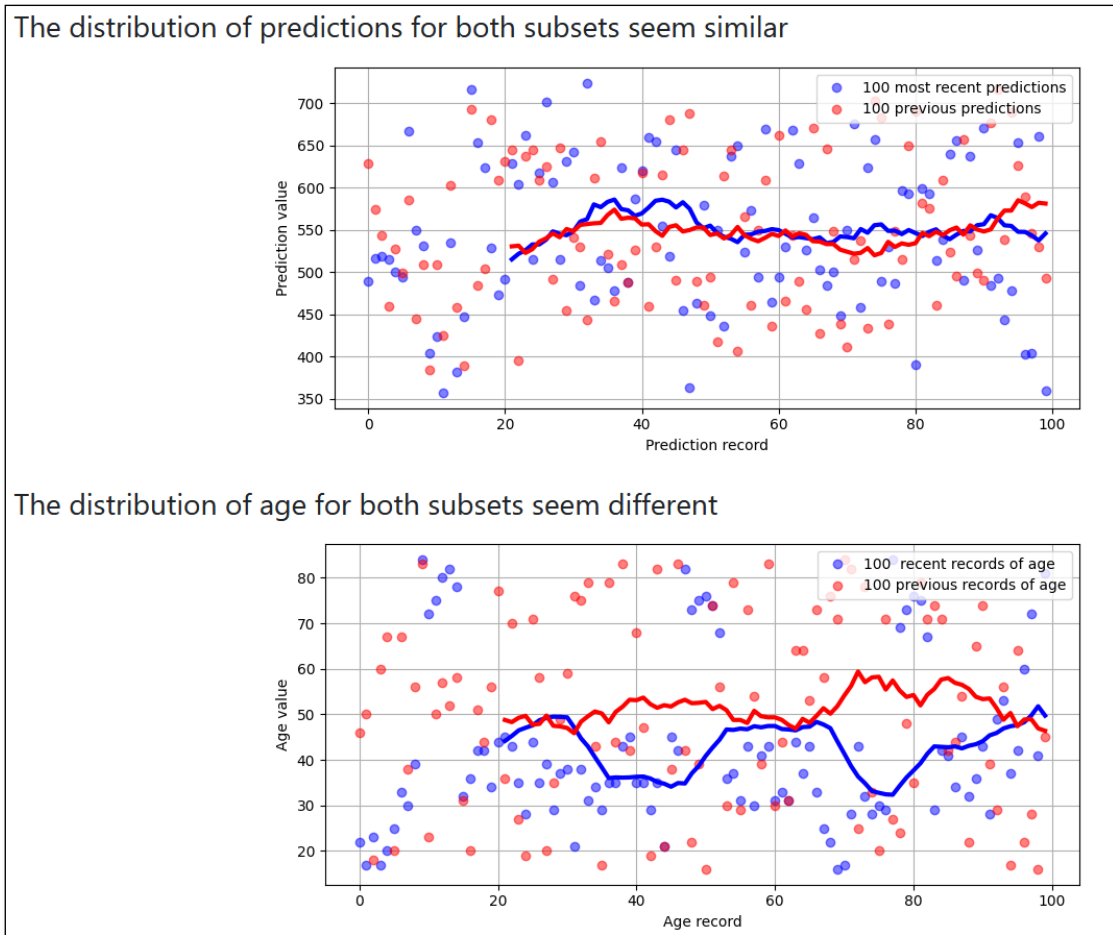


Figure 16: Monitoring the distribution of predictions of an analytic model and the distribution of input variable age. The monitoring is performed using a window size of 100 observations.

The third tests allowed us to validate that we can identify biased models using input data collected over time. In other words, this test validates that our sensitivity analysis explained in Section 7.5 works. Figure 17, shows the predictions generated by an analytic model that was designed to have a small weight for the input variable age. For this model, changes in the age value result in small deviations in the predictions. Therefore, the predictions related to the original age values and the ones related to the modified age values have a similar distribution.

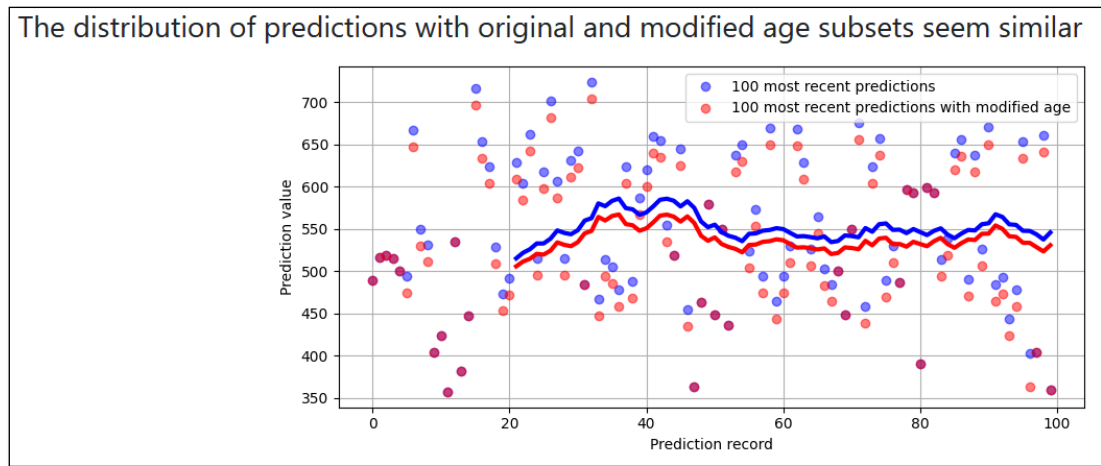


Figure 17: Monitoring the effects that the modification of the value of an input variable have in the predictions of an analytic model.

10. Project management

In the first stage of this project, a simple project management plan was created to describe working approach. This PMP helped to track monthly progress and intermediate deliverables while agile methods were used for weekly activities. The most relevant elements of this plan are the scope definition and schedule.

10.1 *Scope definition*

The scope was defined in the first two months of the project and revisited several times during the project. As the project progressed, the trainee obtained a better understanding of the credit scoring application (CSA), the stakeholder needs, and CSA users' needs.

Usually, once a scope is established, it should not have many changes because the main goals have already been established. However, CSA is an innovation project that was being developed at the same time this project was conducted. Thus, a significant amount of new information was obtained on a weekly basis during the first six months of the project.

Sometimes, this information improved or contradicted previous goals and assumptions, and such cases, the scope definition was revisited. This situation resulted in three versions of the scope definition. The first version was mostly focused on monitoring the input data and predictions of analytic models without truth labels. In the architecture described in Section 6, this functionality is provided by the passive monitoring element of the monitoring engine, shown in Figure 7 (Section 6.1). Additionally, this functionality corresponds to the following requirements: BR1, BR2, BR4, FR1, FR2, and FR3.

The second version of the scope included monitoring the accuracy of analytic models considering that repayment behavior data would eventually be available. This functionality corresponds to the active monitoring element of the monitoring engine in Figure 7. The requirements included in this revision correspond to: BR3 and FR5.

The third version of the scope included monitoring the evolution of analytic models over time. In this scope, downgrading analytic models and computing accuracy of different versions of a model should be possible. This new functionality is enabled by the model deployment engine, the model registry, and the prediction engine history shown in Figure 7. The requirements corresponding to this update are: BR5, BR6, FR4, FR5, FR6, NF1, NF2, NF3, NF4.

10.2 *Schedule*

Initially, the project working schedule was created solely based on the deliverables and deadliness required by TU/e. However, this approach did not work because the working schedule of the main stakeholders was not aligned with the first version of the project schedule. Keeping track of two different schedules proved to be an ineffective strategy.

This problem was solved by adapting the project schedule to the working schedule of the CSA team and other main stakeholders. This new schedule considered both main deliverables and intermediate ones. The deliverables shown in Figure 18 were divided into activities that could be completed during a two-week sprint. Each of these activities corresponds to one of the following phases: exploration, design, and implementation.

During the exploration phase, the goals of this project, main stakeholders, and requirements were identified. This phase lasted around three months; however, during the remaining months some parts of the

project scope and requirements were updated to include new findings discovered during the design phase.

The design phase, which is the longest phase, lasted almost four months. During this phase, several approaches for analytic model monitoring and deployment were explored during the feasibility analysis. The decisions taken during this analysis influenced the solution architecture and implementation design.

After the solution architecture and design were in place, the main focus shifted to implement a prototype to demonstrate the most important elements that are considered the most difficult to realize. This prototype was validated as described in Section 9. Additionally, this report was created while the prototype was finished.

Phase	Activity	Deliverable
Exploration	Stakeholder analysis	Stakeholder analysis document
	Domain analysis	Project scope
	Requirements	Requirements document
Design	Feasibility analysis	Architecture and design document
	Solution architecture and design	Architecture and design document
Implementation	Prototype implementation	Monitoring engine source code
	Verification and validation	Test plan and model deployment pipeline
	Documentation	Technical report

Figure 18: Main elements of the work-breakdown structure used during the project.

11. Conclusions

11.1 Results

This project resulted in a solution architecture for monitoring credit scoring analytic models. In addition, a prototype was created to demonstrate how certain elements of the architecture and its implementation design can be implemented in the code of the credit scoring application (CSA).

11.1.1. Solution architecture

We can identify four major benefits that the solution architecture provides to the current implementation of CSA.

First, the solution architecture introduces a model deployment pipeline to ensure that every deployed model can be executed by the prediction engine. According to the solution architecture diagram, the model deployment pipeline uses an instance of the prediction engine to execute its model deployment tests. For generating predictions, the backend uses an instance of the same prediction engine. Thus, if an analytic model is accepted by the model deployment pipeline, this model will also be executed. Here, we assume that changes in the prediction engine will be handled such that all the relevant deployed models are supported.

The second benefit provided by the solution architecture is the usage of a model registry to keep track of all the deployed models. Active models, candidate models, and models no longer in use can be stored in this registry. Having this registry allows for tracking the evolution of analytic models over time. At any point in time, these models can be monitored.

The third benefit involves creating an engine whose sole purpose is to manage the analytic models that are used for prediction. Having this engine in place allows for updates in the backend without affecting the functionality related to generating credit scoring predictions. More importantly, this prediction engine allows for deploying and activating analytic models without having to modify elements from the backend, unless it is required. Last but not least, this prediction engine can be used in other elements that support the functionality of CSA, such as the model deployment engine and monitoring engine.

The last and main benefit of the proposed solution architecture involves the monitoring engine. This engine was designed to support monitoring tasks that only require the historical input data processed by analytic models and their corresponding predictions. In addition, this engine supports the usage of models from the model registry to perform live monitoring. For instance, we can perform sensitivity analysis to identify whether any of the active models are biased towards changes in input values associated with a specific variable. Supporting these two types of monitoring tasks allow the data scientists to design new monitoring tasks that were not considered during this project but could be eventually required.

11.1.2. Implementation design

From the implementation design, we can identify two major contributions. First, the design of a class diagram that shows a way to create a common interface for analytic models. This interface ensures that all the implementation of analytic models can be tested, executed, and monitored. Having such an interface can help to simplify the maintenance of deployed models.

The second contribution consists in the design of active and passive monitoring tasks that can be used to monitor relevant properties of the models and their input data. In this report we use input data, predictions, and repayment to illustrate how the proposed monitoring tasks can be used. However, minor modifications to the building blocks of these tasks can result in new monitoring tasks. For instance, replacing historical input data with measurements of the time it takes to generate predictions we could monitor the responsiveness of the prediction engine.

11.1.3. Prototype

Even though the prototype is not a production-ready implementation, it was implemented in a branch of the source code of CSA. Thus, it shows how model deployment, model execution, and model monitoring can be applied to similar versions of CSA. Moreover, this prototype is another way to communicate our proposed solution. Together with the solution architecture and implementation design, we communicate our solution at different levels of abstraction so architects, engineers, data scientists, and project managers are able to understand how our proposed solution works and its benefits.

The results of the validation use cases show that monitoring tasks are able to identify trends in the input data as well as sensitivity of predictions towards an input variable. In production environments the monitoring parameters should be fine tuned according to the input data being processed by the analytic models. Hence, the monitoring parameters used in our validation procedure should only be considered as part of the demonstration of monitoring tasks.

11.2 *Recommendations and future work*

At the moment of writing, CSA uses a small number of analytic models to serve credit providers. Such a limited number of models is relatively easy to create, deploy, and monitor using the current codebase. However, as more credit providers start using the application, and more analytic models are created, the deployment and execution of models can become difficult and automatic model monitoring will be required.

We recommend preparing the codebase to support model monitoring in three steps:

- First, implement a common interface for all the models to be able to support model deployment, execution, and monitoring tasks. This recommendation is aimed at tackling technical debt before such a constraint starts hurting the development of new functionality.
- Then, implement versioning for the analytic models such that their evolution can be traced over time.
- Finally, implement a prediction engine to separate the code that executes the analytic models from the backend. Having such separation of concerns allows for updating the backend code without having to update the code that generates predictions.

The proposed updates can be implemented without big changes in the existing codebase of CSA and should not affect its current functionality.

Once the previous recommendations are applied, then the model deployment pipeline, model registry, and model monitoring can be implemented. These elements could have a significant impact in the functionality of the application, but the previous modifications would help to reduce such an impact because the codebase have been prepared for such major changes.

In this project, we describe the use of sensitivity tests for performing active monitoring tasks. This type of test was designed to identify deviations from one guideline related to the ethical usage of analytical models: models should not generate biased predictions. As the development of CSA continues, more guidelines or rules related to ethical usage of models and other topics could be required. The design of the model monitoring engine supports the creation of new active and passive monitoring tasks related

to rules and guidelines. Hence, we suggest the team to identify what other types of guidelines or rules the analytic models should follow and create the corresponding monitoring tasks.

Additionally, we propose a simple model execution lifecycle management to handle the status of analytic models to control the versions that are being executed. This lifecycle assumes that analytic models can only have four status. Based on the prototype, this lifecycle seems to be enough for the needs of CSA. However, recommend the team to identify which deployment and execution strategies they plan to use for the analytic models and update the proposed model lifecycle management accordingly.

Finally, in this project we explored the strategies that can be used for model deployment, execution, and monitoring, but not the technology that can be used for these activities. Thus, we suggest the team to research what kind of technology can better fit the needs of CSA. Should they use databases for storing models, or git repositories? This, among others, is one of the questions the team will face when implementing our proposed solution. Thus, they should already plan for this activity.

12. Project retrospective

In this section, I describe some of the insights I gained during this project at Rabobank. The content of this section contains the personal views of the author and by no means reflect the views and opinions of my colleagues, Rabobank and TU/e. The circumstances of this project, my skills and preferred way of working resulted in the experiences shared below.

12.1 *Self-reflection*

I choose to conduct my project at Rabobank for two main reasons: I had no experience working for a financial institution and I had never worked in an innovation project for a large institution. In addition, I decided to learn as much I as could about project management process during this project. Right from the beginning, my goal was to learn as much as possible during this project.

At the start of the project, I tried to plan and execute every action and decision following some of the project management techniques that I learned during the first year at the PDEng. During the first months, trying to deliver results using strict process in an innovation environment resulted in scarce results. Eventually, I realized that the project I worked on is better suited for an agile way of working. When I decided to focus less on the processes and more on the results, I started to produce better deliverables, to understand the project better, and to communicate better with my colleagues and supervisors.

In retrospective, I tackled my graduation project using an approach that was not suitable for the task at hand. I learned a lot from the experience. However, I should have aim to incrementally learn new processes and skills along the whole project, instead of doing it right from the start.

With respect to the innovation project, now, I can see how challenging is to develop a product aimed to help improving the life of people living in developing countries. The type of technology and processes we use in The Netherlands makes it easier to create opportunities for everyone. In contrast, the technology and processes that people use in some developing countries makes it challenging to create opportunities for them.

Considering all the challenges that the innovation team has faced so far to develop their product, I am extremely proud to be able to provide a small contribution to this innovation project. I believe they have what it takes to achieve their vision. The moment their product starts being used by several credit providers, the results of my project will help the team to scale it.

Regarding future endeavors, I want to work on projects that aim to improve people's lives. This time, I will focus on my best skills and slowly improve my weakest points. I believe technology should be used to improve peoples' lives and I think I can contribute to that end.

Overall, I am extremely grateful for the opportunity I had to work on this final project. I learned a lot about innovation projects, how to communicate with colleagues, and how to prioritize my work. I am looking forward to more challenging experiences like this one.

Glossary

Some of the words used in the report and their meaning are shown below.

Model owner	In Rabobank, a model owner is the person responsible for the analytic models deployed to the production environment.
Production environment	A production environment contains the applications that provide a service to users and other applications.
CSA	CSA is the abbreviation of credit scoring application.
KS-test	KS-test is the abbreviation of Kolmogorov-Smirnov test, a statistical test used to compare the similarity of the distributions of two datasets.
Alternative data	Alternative data is considered non-financial data such as historical weather conditions, field characteristics, and yield production.
Credit providers	Credit providers are financial institutions such as banks, agriculture fintech companies, microfinance institutions, and wholesale companies.
MLOps	MLOps is the abbreviation of machine learning operations.
PSG	The Project Steering Group (PSG) are the university and company supervisors that evaluated the trainee progress.
Analytic model	A rule-based model, statistical learning model, or machine learning model used to generate predictions based on input data.
MoSCoW	The MoSCoW method is used to prioritize requirements based on four categories: must have, should have, could have, and would have.
Passive monitoring task	A passive monitoring task only uses the input and outputs of an analytic model for analysis.
Active monitoring task	An active monitoring task uses analytic models, input data, and predictions to perform real-time analysis.

Bibliography

- Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2016). What's your ML test score? A rubric for ML production systems. *Reliable Machine Learning in the Wild - NIPS 2016 Workshop*.
- Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Survey*, 1-58.
- Cline, A. (2015). *Agile Development in the Real World*. Berkely, CA: Apress.
- Gallo, E., Heil, J., & Bosman, R. (2020). *Credit Scoring Project Start Architecture. (Confidential)*
- Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning*. New York, NY, USA: Springer Series in Statistics Springer.
- Hulten, G. (2018). *Building Intelligent Systems*. Berkeley, CA: Apress.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning*. New York, NY: Springer.
- Kolmogorov-Smirnov equality-of-distributions test*. (2018, April 14). Retrieved from stata.com: <https://www.stata.com/manuals15/rksmirnov.pdf>
- Ribeiro, M. T., Wu, T., Guestrin, C., & Singh, S. (2020). Beyond Accuracy: Behavioral Testing of NLP models with CheckList. *ArXiv.org*.
- Rooijen van, X. (2021). *DataOps, MLOps, ModelOps, AIOps. What is it and how does it fit within Rabobank? (Confidential)*
- WBG. (2019). *Credit Scoring Approaches Guidelines*. Retrieved from [www.worldbank.com: https://thedocs.worldbank.org/en/doc/935891585869698451-0130022020/original/CREDITSCORINGAPPROACHESGUIDELINESFINALWEB.pdf](https://thedocs.worldbank.org/en/doc/935891585869698451-0130022020/original/CREDITSCORINGAPPROACHESGUIDELINESFINALWEB.pdf)

Appendix A.

The sequence diagrams shown in Figure 19, 20 and 21, describe the three main functions enabled by the solution architecture described in Section 6, Solution architecture.

Figure 19 shows a sequence diagram describing specific sequence of actions performed during model monitoring. This diagram can be read as follows:

- At any point, analytic models can be stored in the model registry.
- While the monitoring engine is no interrupted by a notification, it will perform monitoring of the input data and predictions as well as sensitivity analysis.
- When the model owner activates a new model, the monitoring engine requests the new active model for monitoring. Using this new active model, the monitoring engine will continue to perform its tasks.
- At any point in time, new input data and predictions can be stored in the prediction engine logger. Additionally, repayment behavior data can be stored in the prediction engine logger.

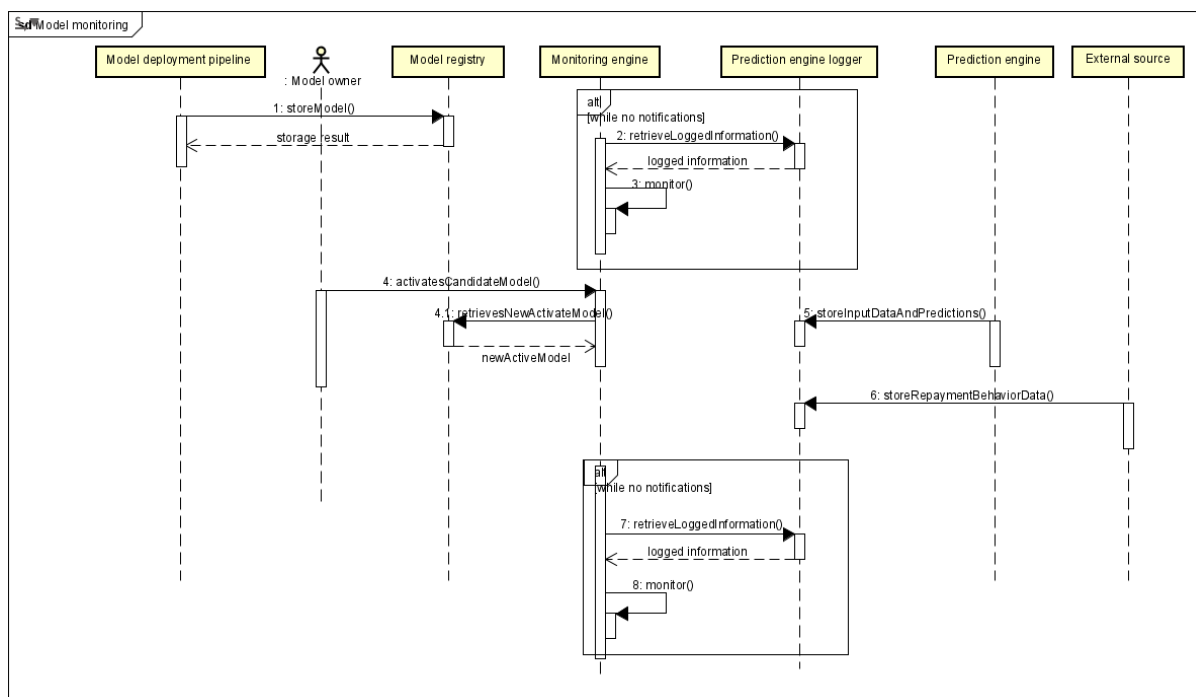


Figure 19: Monitoring of analytic models enabled by the solution architecture.

Figure 20 shows a sequence diagram describing the process of deploying an analytic model. This process works as follows:

- First, a data scientist submits a new trained model to the model deployment pipeline.
- Then, the model deployment pipeline runs a series of tests on the submitted model.
- If the analytic model passes the tests, then it is stored in the model registry.

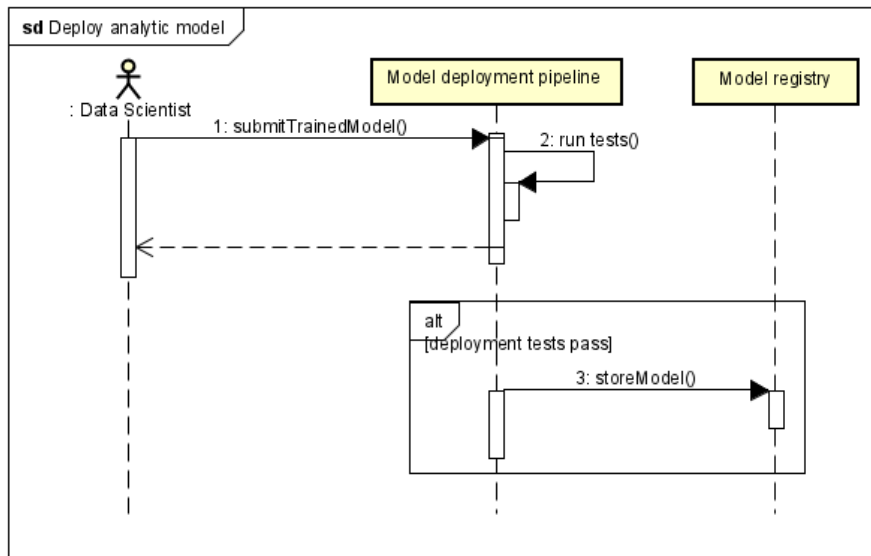


Figure 20: Deploy a new model to the model registry

Figure 21 shows a sequence diagram describing how active models being executed by the prediction engine can be replaced for different versions of the same model. This process works as follows:

- First, the model owner selects the model that will replace the current version of an active model.
- Then, this analytic model is submitted to the model deployment pipeline to ensure it is still a valid model.
- If the tests are passed, the analytic model is activated in the model registry and the active analytic model is deactivate. Additionally, the prediction engine is notified about the changes.

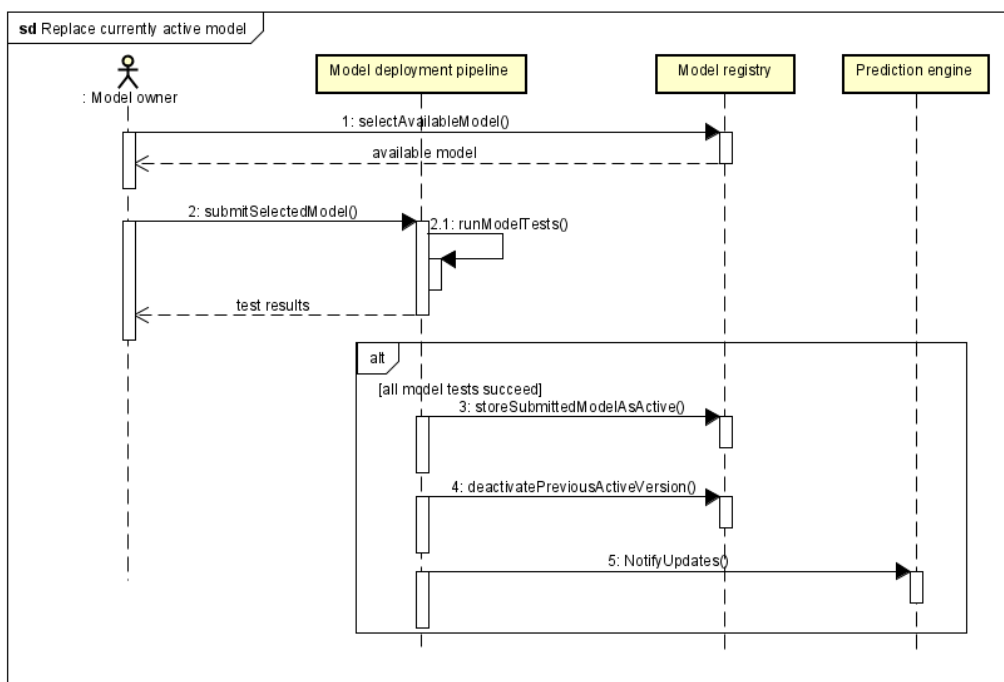


Figure 21: Replace a deployed analytic model for a different version stored in the model registry

About the Author



Jorge Alberto Cordero Cruz received his Master of Data Science from the Eindhoven University of Technology in May 2019. He worked on a method to analyze micro-RNA sequences using convolutional neural networks during his master's thesis. After graduation he worked as a full-stack developer for a public hospital in Mexico. During the PDEng program, he worked on deep learning related projects for Bosch, PixelFarming, and ESA. His PDEng graduation project was conducted at Rabobank. He believes that technology should adapt to people as much as possible. Nowadays, his interests are software development, machine learning, storytelling, and dancing.

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

PDEng SOFTWARE TECHNOLOGY

TU/e EINDHOVEN
UNIVERSITY OF
TECHNOLOGY