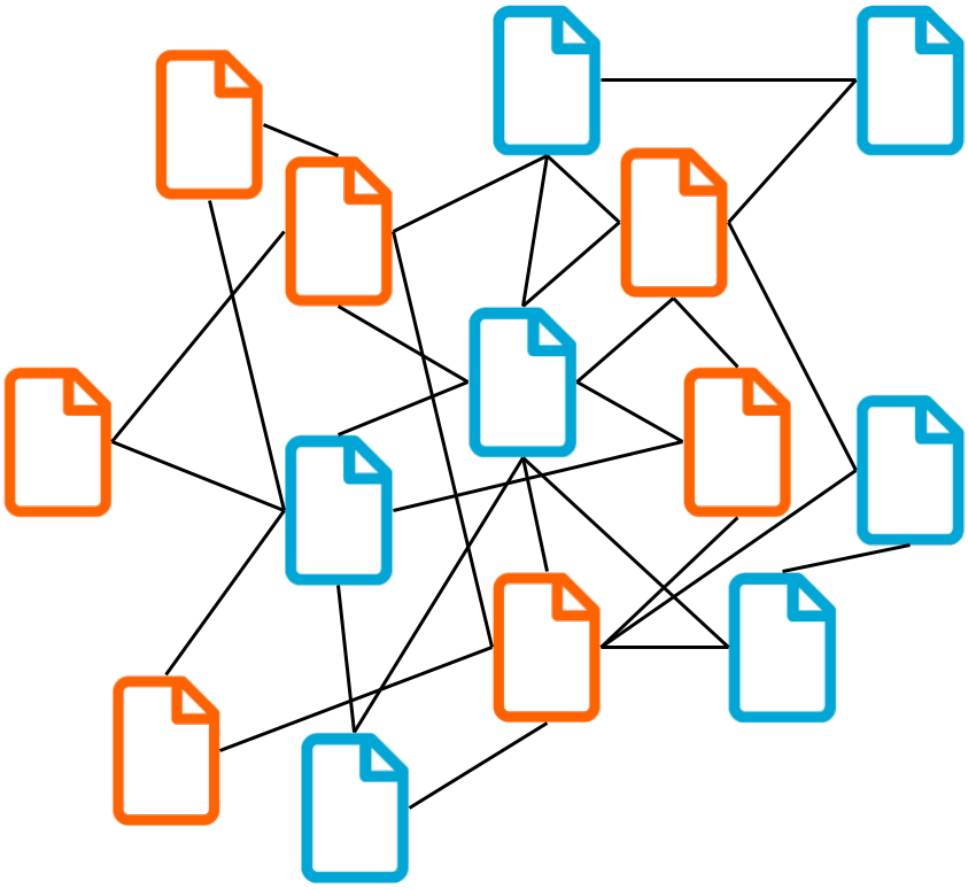
Predicting Delays in Software Deliveries using Networked Classification at ING

Version of August 22, 2022



Pravesh Moelchand

Predicting Delays in Software Deliveries using Networked Classification at ING

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK: DATA SCIENCE AND TECHNOLOGY

by

Pravesh Moelchand

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS
Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

AI for Fintech Research Lab
ING Analytics
Bijlmerdreef 106
Amsterdam, the Netherlands
https://se.ewi.tudelft.nl/ai4fintech/

Cover picture: Story network with delayed and non-delayed stories

Predicting Delays in Software Deliveries using Networked Classification at ING

Author: Pravesh Moelchand
Student id: 4570294

Abstract

Delays in the delivery of software projects and the corresponding cost and schedule overruns have been common problems in the software industry for years. A challenge within software project management is to make accurate effort estimations during planning. Software projects are complex networks, with multiple dependencies between software tasks.

This study aims to combine the field of effort estimation and networked classification to utilise network information for delay prediction in industry. We conducted a case study at ING, resulting in a number of insights with regards to networked classification in an industry setting.

There is a difference in the organisational structure of open-source and industry projects. This constitutes to a difference in available information, but there is also an opportunity to leverage the organisational structure of ING to improve delay prediction performance.

Using weights in networked classification has shown no improvement compared to not using them, but relational models do benefit from larger datasets as the used network contains more relational information.

Based on the insights we recommend ING to: keep track of more information, improve data quality by educating their teams and create models for specific domains or teams to leverage their organisational structure.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
Company supervisor:	MSc. Elvan Kula, ING - AI for Fintech Research
Committee Member:	Dr. Jie Yang, Faculty EEMS, TU Delft
Committee Member:	Prof. Dr. ir. Rini van Solingen, Faculty EEMCS, TU Delft

Preface

It's done!

The road leading up to this thesis has been one with many obstacles, which we have luckily been able to overcome. *We*, you read that correctly. I conducted this research project by myself, but I have never had the feeling I was on my own. This is thanks to all the people who supported me in these past months.

Firstly, I would like to thank my (ex-)supervisors Rini, Arie and Elvan for guiding me through the process of conducting this study. I have always had the feeling that you genuinely put effort into supervising me and felt we were in this together.

Doing a thesis project on your own for so many months could have been lonely, if it wasn't for my friends. Thank you Brian and Martin, for giving me valuable insights. Thank you Emiel and Thijmen, for supporting me when the need was high. And of course thank you to all the heroes of the fourth floor, my final months in Delft were a blast because of you.

Studying is however not everything. I want to thank all my friends and family, who were there to hang out when I needed time to relax, who understood that I sometimes had to cancel our plans, who kept checking in on me for all these months and who have always had the faith that I could do this.

Finally, coming home to a cooked dinner on busy days, being provided with loads of opportunities and always getting all the support to follow my dreams have been crucial in obtaining this degree. Thank you Akash, Rashvin, Mom and Dad for creating such an environment for me.

Pravesh Moelchand
Delft, the Netherlands
August 22, 2022

Contents

List of Figures

Chapter 1

Introduction

Imagine being a software developer working in a development team in industry and being in the planning meeting at the start of a new iteration. You are deciding which tasks you want to tackle and relying on the expertise of the team, you put together a decent list of tasks which you think will all be completed on time.

As the week progresses, you and your team work hard to complete all the planned tasks. Some are completed, some take more time than expected but are finished in time and some are delayed to later iterations. Even though you tried to plan the iteration to the best of your abilities, not all tasks were finished in time.

The research area of effort estimation is all about making this kind of planning more accurate. Leveraging Machine Learning has recently gained a lot of popularity [30], but many approaches only look at locally available information of a software task, not considering relational information such as dependencies between tasks. This relational information could however be used in relational classifiers to achieve better results.

In 2015, Choetkiertikul et al. combined relational information with effort estimation. They developed an approach for *Predicting Delays in Software Deliveries using Networked Classification* [10]. The approach made use of relational information to create a network of software tasks and employed relational classifiers to predict delays in an open-source software development setting.

At ING, a Dutch internationally operating bank, more than 15,000 developers are employed worldwide. The company is organised following the Agile *Squads, Tribes and Chapters* model of Spotify [27], meaning most of the development teams make use of user stories for their project planning. The environment of ING enabled us to assess the effectiveness of the approach presented by Choetkiertikul et al. in an industrial setting and expand upon it.

1.1 Research Questions

Using relational classification for effort estimation has already proven to work well on open-source data [10]. The aim of this study is to apply relational classification for delay prediction in an industry setting. ING is used as case company to perform this study.

We investigate different aspects of relational classification for effort estimation in industry. At first, we look at the performance of relational classification in an industry setting and which information is useful to include for delay prediction. Then, we tweak the relational classifiers by adding weights to the relations. Finally, we investigate the effect of different sizes of sliding windows on prediction performance. These aspects are covered by the following three research questions:

RQ 1 What is the performance of delay prediction using networked classification in an industry setting compared to open-source data?

a) What is the performance of delay prediction in an industry setting using features that work well on open-source data?

b) Which local features, available at ING, improve delay prediction in an industry setting?

c) Which relational features, available at ING, improve delay prediction in an industry setting?

d) Which features are most important for delay prediction on ING data?

RQ 2 What is the impact of adding weights to relations on delay prediction using networked classification?

a) What is the prediction performance when determining weights based on the time interval between stories?

b) What is the prediction performance when determining weights using the assortativity coefficient of the network?

RQ 3 What is the effect of using different sizes of sliding windows when performing delay prediction on ING data?

1.2 Contributions

The contributions of this study consist of six insights about delay prediction using networked classification in an industry setting:

1. There is a difference in the information that is available between open-source and industry settings.
2. Record-keeping is treated differently across teams.
3. Using networked information improves delay prediction.
4. Using a general approach for ING as a whole performs worse than a team-specific approach.
5. Using weights in networked classification does not improve prediction performance at ING.
6. Larger dataset sizes improve prediction performance for relational models.

Based on the insights, we derive three recommendations to ING for implementing delay prediction using networked classification:

1. Keep track of more information.
2. Improve data quality.
3. Create models for specific domains or organisational units.

1.3 Report Outline

Chapter 2 presents the broader context of this study and provides the reader with background knowledge of related topics and work. The research method of this study is described in Chapter 3. The results discussion for each research question are presented in Chapters 4, 5 and 6. The threats to validity are discussed in Chapter 7 and the conclusion and future work are given in Chapter 8.

Chapter 2

Background

This chapter provides the reader with the background knowledge that is required for understanding the contents of this report. Agile development methods will be covered, but also academic work related to effort estimation and networked classification. We finally describe the context of this study at ING.

2.1 Agile Development

2.1.1 Overview

Agile software development, or Agile development in general, is an iterative approach to managing software projects according to the twelve principles of the Agile Manifesto [2]. Over the past two decades, Agile development methods have grown in popularity amongst organisations to manage software projects [13]. In an Agile project, the work is split into smaller chunks of work at different levels and software is developed through short iterations to enable organisations to react to changes in the market and customer needs. Agile teams require a high level of self-organisation and intense collaboration [13], [44], something which is to be stimulated through the structure and values of the organisation [39].

In large scale software companies, the user requirements are commonly expressed using the hierarchy introduced by Leffingwell [31], which is visually represented in Figure 2.1. The work is divided into different levels: *themes*, *epics*, *features*, *stories* and *tasks*. At the highest level are the strategic themes and epics. These provide high-level functional goals for the product(s) [15]. Epics usually span three to twelve months and can further be subdivided into features, which can further be subdivided into user stories. User stories represent specific end user requirements and span only one iteration or sprint [14], which is one to four weeks at ING, depending on the team. Stories can further be subdivided into tasks, which are the technical work that needs to be done to complete a user story. In table 2.1 we give

an example of the different levels. For our study, we will be focusing on the story level, as these are most widely-used to plan the work across sprints within ING.

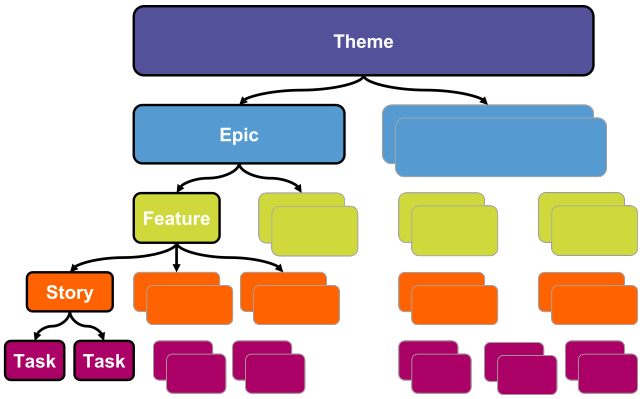


Figure 2.1: Agile work breakdown structure. Diagram based on Kula et al. [30]

Table 2.1: Example of agile building blocks

Level	Example
Theme	Helping clients get funding for their homes
Epic	Apply remarketing across the website
Feature	Emailing clients about opportunities
Story	Follow-up emails related to remarketing are working at the end of the sprint
Task	Email client is ready

2.1.2 Agile development in industry and open-source settings

While developers within large software companies are usually organised along software teams responsible for specific products or software domains [7], this is however different for developers within open-source projects. Open-source projects are often lead by a core team of developers, supported by up to hundreds of developers who join the project voluntarily [16]. The consequence is that instead of being organised along software teams, each developer can be viewed as a self-organised team, following the guidelines and goals set by the core team [28]. This means in industry there is an extra level of detail, as stories can be related to not only developers but also to teams.

2.2 Related Work on Delay Prediction

2.2.1 Effort estimation

Effort estimation is at the core of the short-term iterative planning in Agile software development [15]. A large number of works have been published on effort estimation methods [26]. Agile teams mostly rely on subjective expert judgement for estimating the amount of work that can be completed within an iteration [17, 47]. In order to provide automated, contextual support for estimating effort in software projects, a number of recent works have introduced the use of Machine Learning for effort estimation. There have been promising results for effort estimation in story completion [12, 40, 43] and predicting bug-fix duration [4, 21, 38, 49]. Mining source code and historical data is key to creating a dataset that can be leveraged by Machine Learning approaches [24, 35]. In the context of delay prediction Choetkiertikul et al. present the most recent studies [9, 11]. These approaches employ Machine Learning techniques to classify software tasks and consider them as independent data points, ignoring dependencies and relationships between tasks.

2.2.2 Networked classification

Networked data is present in many different areas of our daily lives, such as social networks, biological networks and communication networks. These networks have been utilised in various studies, from web page classification [33] to scientific research paper network building [20, 46]. Networked data allows for collective inferencing, meaning that related nodes in the network are classified simultaneously [32]. The principle of a node's neighbours' class probability estimate influencing the node's own estimate has been described early on by Markov chains [3, 18, 19]. A first-order Markov model assumes the state of a node depends only on the states of its direct neighbours.

Networked classification has been applied in software-related studies, such as predicting software defects [22] and software quality [25, 36, 37, 48]. Using networked classification for effort estimation has most recently been studied by Choetkiertikul et al. [10]. Their research was carried out on open-source data and forms the basis of our study, where we apply networked classification on industry data to predict delays.

2.3 Case Study at ING

Our study is conducted at ING, a large Dutch internationally operating bank with more than 15,000 developers worldwide working on projects across a variety of domains. ING has reinvented its organisational structure to a completely Agile structure based on Spotify's

Squads, Tribes and Chapters model [27]. A squad consists of 5 to 9 members and focuses on a specific software functionality. Squads are grouped within tribes, which focus on a specific business area, such as *mobile*. Using this model allows for coordination of hundreds of Agile development teams. Each development team at ING use Scrum [42] as Agile methodology [30]. Each team plans their work along user stories such as described in Section 2.1. Story points are assigned to stories using expert judgement during a sprint planning meeting at the start of the sprint. ING lends itself perfectly for studying delays, as about a quarter of its user stories are delayed. Historical data of these stories is available in a data warehouse, which gives us access to a significant amount of data for our study.

Chapter 3

Research Method

In this chapter we describe the general steps taken for answering our research questions. We first explain how the data is collected and preprocessed. Then, we discuss how the features are engineered and how the data points, which are used in the models, are constructed. Finally, the local and relational model used for delay prediction are presented.

3.1 Data Collection

For this study, historical story data from ING is used. The majority of teams at ING make use of the backlog management tool *ServiceNow*. Since 2016, all versions of all stories from this system are stored in a data warehouse, which contains over 20 million historical logs[1] as of 2022 and this number is growing as the database is updated continuously. In this section, we give an overview of the data selection process and the characteristics of the resulting dataset.

3.1.1 Dataset requirements

Not all data available is of use to the project and as the dataset is continuously updated with new information, we have to make a selection of which data to include in our dataset. We set the following requirements for the dataset.

Time frame This study started in October 2021 and we have started working with the data in 2022. To ensure that our dataset is not affected by new updates to the data warehouse, we only consider logs from before 2022.

[1] A log is a specific version of a story in the data warehouse

State The state of a story is stored as a numerical value in ServiceNow. Stories can have one of the following states: (-6) *draft*, (1) *ready*, (2) *work in progress*, (3) *complete* or (4) *cancelled*. For our study, we only include logs with state -6, 1, 2 or 3 of stories which have been completed. Logs which belong to stories which have been cancelled at some point are thus not included in the dataset.

Dataset size In Table 3.1 the number of logs and stories have been listed for each year. We see that teams have increasingly been using the backlog management tool over time and that 2021 therefore contains the most logs. As more teams start to use the system and teams get more experience with using the system, the amount and quality of the data will get better. Taking this into consideration, we choose to use the 2021 part of the dataset. This provides us with a reasonable amount of data which is also the most recent data.

Table 3.1: Number of logs and stories for each year

Year	Logs	Stories
2016	232,706	26,392
2017	729,763	96,397
2018	1,624,060	241,826
2019	2,448,701	414,026
2020	3,240,871	598,868
2021	4,126,954	849,069
Total	12,403,055	2,226,578

3.1.2 Dataset characteristics

The dataset resulting from the selection process consists of 4,126,954 logs of 849,069 stories in 2021, spread over 60,954 sprints. The stories span across 1,949 themes and 20,409 epics. The stories are distributed over 2,563 squads and 17,373 developers worldwide.

Every story has 44 fields, of which 30 fields contain information about the story and the other 14 fields serve as trackers for data warehouse. In Table 3.2 a description of these fields can be found.

The availability of information varies however for each field. The following fields have a value for each log: *Id*, *Updated On*, *Active*, *Blocked*, *Description*, *Epic Reference*, *Number*, *State*, *Story Points*, *Created By*, *Created On*, *Updated By*, *Theme Reference*, *Tribe Reference*, *Request*, *Secure* and *Template Story*. The availability for the remaining fields are shown in Table 3.3.

Table 3.2: Fields of a story

Field	Description
Id	Unique story ID
Updated On	The date and time when the story was updated to this version
Acceptance Criteria	Description of acceptance criteria
Active	Whether the story is currently active
Assigned To	To whom the story is assigned
Blocked	Whether the story is blocked
Blocked Reason	The reason why the story is blocked
Component	Reference to component ID
Defect Reference	Reference to defect ID
Description	Description of the story
Enhancement	Reference to enhancement ID
Epic Reference	Reference to epic ID
Number	Unique number of story used as ID in ServiceNow
Squad Reference	Reference to squad ID
Short Description	Short description of the story
Sprint Reference	Reference to sprint ID
State	The current state of the story
Story Points	The number of story points estimated for this story
Created By	ID of employee who created the story
Created On	The date and time when the story was created
Updated By	ID of the employee who updated the story
Theme Reference	Reference to theme ID
Feature Reference	Reference to feature ID
Tribe Reference	Reference to tribe ID
Request	Whether the story is a request
Secure	Whether the story is secure
Closed On	The date and time when the story is closed
Closed By	Reference to the employee who closed the story
Actual Start	The date and time when work on this story started
Template Story	Whether the story is a template story or not
Change Reference	Reference to change ID

Table 3.3: Availability of story data. Numbers indicate the fraction of logs or stories that have a value for the corresponding field

Attribute	Logs	Stories	Ratio logs	Ratio stories
acceptance_criteria	1369062	249135	0.331737	0.293421
assigned_to	2687129	670249	0.651117	0.789393
blocked_reason	98484	4648	0.023864	0.005474
component	60326	12615	0.014618	0.014857
defect_reference	12	12	0.000003	0.000014
description	4110087	845428	0.995913	0.995712
enhancement	0	0	0.000000	0.000000
epic_reference	4126936	849065	0.999996	0.999995
short_description	4112955	846022	0.996608	0.996411
sprint_reference	3223824	781879	0.781163	0.920866
story_points	3642002	819506	0.882492	0.965182
theme_reference	4126937	849065	0.999996	0.999995
feature_reference	2042235	398846	0.494853	0.469745
tribe_reference	4126934	849065	0.999995	0.999995
closed_on	1031037	752314	0.249830	0.886046
closed_by	1031034	752309	0.249829	0.886040
actual_start	1947590	767498	0.471919	0.903929
change_reference	80919	36077	0.019607	0.042490

3.2 Data Preprocessing

We give an overview of the steps taken to prepare the data for the feature extraction and experiments. Each feature requires different sets of information, so there is no general data cleaning process apart from the data selection step as described in the previous section. Furthermore, we discuss data compression, type conversion and delay classification as preprocessing step.

3.2.1 Classifying ground-truth delay status

Stories do not contain explicit information about their delay status. We therefore have to derive this from the available fields. Stories in open-source projects contain explicit information about their *planned to complete* date, but the only planning-related field for the stories in our dataset is the *Closed On* field. From this, we can not infer whether a story was completed on time or over time.

We therefore decide to look at the number of sprints a story has been assigned for classifying its ground-truth delay status. A story can only be assigned to one sprint at a time.

When a story has been assigned to multiple sprints, this means the story was not completed within the first sprint that it was assigned to, meaning it has been delayed to a later sprint. In our study, the delay status is thus classified as either *not delayed* or *delayed*.

To establish a ground-truth for every story, we count the number of sprints for each story based on the historical data. The ground-truth is then assigned to all logs of the story. After the delay classification, we see that 73% of the stories are not delayed and 27% are delayed.

3.2.2 Dealing with missing information

As discussed in subsection 3.1.2, it is possible that logs do not have values for all fields. We distinguish between local fields, which contain information that is available for each story independently, and relational fields, which contain information that can be used for creating a network of story.

Logs which have missing values for local fields are removed from the dataset. For local features, data points with missing values are removed from the dataset. For relational fields, we can however not simply remove logs with missing values because we are dealing with a networked information. The relational fields are used for establishing links between nodes in the network (see subsection 3.6.1). Removing a node from the network will thus affect the information available to its neighbours. We therefore choose to replace the missing values with `-100` as the standard range of values for relational features is 0 to 1. This is a common technique for dealing with missing information in Machine Learning [45].

3.3 Feature Engineering

Not all information available in the dataset can be directly used by the models use of in our study. Therefore, we need to engineer features from this information, that we can use in our models. We can subdivide the features into different sets, which we will describe below. Similar to local and relational fields, we distinguish between local and relational features. Local and relational feature sets are respectively indicated by L and R. We use a non-capitalised subscript to indicate a specific feature set. An example of this is L_o, which are local open-source features.

3.3.1 Local open-source (L_o)

[Changing of description — Developer's workload (stories) — Delayed ratio of developer workload — Story repetitions — Waiting time]

These are local features which have been used in delay prediction on open-source data [10]. Together with feature set R_o these will form the baseline for our evaluation. Below, we describe how these features have been engineered at ING. The descriptions are formatted as follows:

[Feature name] [*"Description from Choetkiertikul et al."*] [10]. [Implementation using ING data]

Changing of description *"The number of times in which the task description was changed"*. We can implement this directly using the available data at ING.

Delayed ratio of developer workload *"The percentage of delayed tasks in all of the tasks which have been assigned to a developer"*. The percentage of stories that have been delayed of the total number of stories that have been assigned to a developer within a sprint.

Developer workload (stories) *"The number of opened tasks that have been assigned to a developer at a time"*. The number of stories that have been assigned to a developer at the start of a sprint.

Story repetition *"The number of times that a task is reopened"*. The number of times the story is set to state ready or state WIP after is has been set to completed.

Waiting time *"The time when a task is waiting for being acted upon"*. This can be interpreted as: (1) when the story is assigned to a sprint or (2) when a story is acted upon within a sprint and thus set to state WIP. As we are predicting the risk of delay at the start of a sprint, we define the waiting time as follows: the time between *a story being ready and not assigned to a sprint* and *a story being ready and assigned to a sprint*.

3.3.2 Local ING (L_i)

[Blocked — Developer workload (story points) — Request — Secure — State at start sprint — Story Points]

These are local features which can be engineered from the information available in the ING dataset. The *Blocked*, *Request*, *Secure* and *Story Points* fields can directly be used as

input for our models. The other two features require some engineering. The steps taken and reasoning behind using these features are explained below:

Blocked The *Blocked* field can be used by developers to indicate whether a story is blocked. Developers can also give a textual reason for the blockage in the *Blocked Reason* field. When a story is blocked, work cannot continue on this story, possibly causing a delay.

Developer workload (story points) Similar to the *Developer workload (stories)*, but instead the number of story points are summed up for each sprint.

Request When a story is requested by a customer, it likely has a higher priority due to the commercial nature of the industry. This means that it is more likely to be completed in time in order to satisfy the customer's needs.

Secure Secure stories are related to security issues or developments. These type of stories often deal with complex dependencies as they have a high-risk status. Stories which are secure are therefore more likely to be delayed.

State at start sprint The status of the story when it is assigned to a sprint. Developers are responsible for setting the status of a story. It is customary that the status of a story is *ready* when it is assigned to a sprint. In the case that it is not, the story might not be completely refined yet and this can be an indicator of delay.

Story Points Stories with more story points are more complex and require more effort. The number of story points can therefore be an indicator of delay.

3.3.3 Relational open-source (R_o)

[Assigned To — Created By]

These are relations which have been used in delay prediction on open-source data [10]. When two stories share the same value for a feature, this means there is a relationship based on that feature. Here, *Assigned To* corresponds to the *Developer* relationship of Choetkiertikul et al. and *Created By* corresponds to the *Reporter* relationship of Choetkiertikul et al. [10].

Note that these two relational features are also part of the relational resource-based feature set (R_r).

3.3.4 Relational attribute-based (R_a)

[Epic Reference — Theme Reference]

These are relations based on the attributes of a story, more specifically the strategic or architectural area they belong to. It might be the case that there is a lot of delay within a specific epic or theme due to a variety of reasons, therefore this might be an indicator of delay. When two stories belong to the same epic or theme, there is a relationship between them.

3.3.5 Relational attribute-based features with low availability (R_{a-low})

[Change Reference — Component — Feature Reference]

These are relations based on the same principle as in R_a, however, there are less stories in the dataset with this information available as can be seen in Table 3.3. This means we can only use a part of the dataset, so we decide to not include these in the standard relational attribute-based feature set. We do evaluate their performance separately.

Note that the *Defect Reference* and *Enhancement* can also be used as relational attribute-based features, but this is infeasible due to their almost non-availability in the dataset (see Table 3.3.

3.3.6 Relational resource-based (R_r)

[Assigned To, Created By, Squad Reference, Tribe Reference]

These are relations based on the resources that are used by the story. Developers and squads can be seen as resources as their hours of work are not unlimited [10]. If a developer or the squad or tribe they belong to has a history with delayed stories, this might be an indicator for new delays. When two stories are assigned to the same developer, have been created by the same developer, are belonging to the same squad or belonging to the same tribe, there is a relationship between them.

3.3.7 Descriptive (D)

[Acceptance Criteria — Blocked Reason — Description — Short Description]

These are features based on textual columns. Textual information can often not directly be used as input for machine learning models as the semantics of text are not directly clear to a machine [23]. To make use of textual information, the semantics should be extracted using NLP techniques such as topic modelling. The topics then serve as features for the textual information. In our study we have explored applying topic modelling using Latent Dirichlet Allocation [5] on the *Blocked Reason* field in the data. We decided however to not

continue in this direction as we concluded that manual and qualitative analysis is required to verify the resulting topic clusters. This is outside the scope of our study.

3.4 Transforming Logs to Data Points

After having engineered the features, our dataset still contains all logs of all stories. The models used in our study will however be trained on the available information at a given moment in time. This means that for each story, we will have to select one log to be used as data point for our models. We illustrate this process using Figure 3.1.

Suppose we pick the 1st of July 2021 as the given moment in time (3.1a). We then denote all logs before that moment as known information and all logs after that moment as unknown information (3.1b). Then, for the known information, we select the latest log as the version to be considered for the model and for the unknown information, we select the first log as the version to be considered. From these two sets of information, we create a training set and test set. We see that for Story C in the example, there are also logs in the unknown information set. It is however customary to exclude training data from the test set, so for all stories that are in the known set, we remove their logs from the unknown set (3.1c). This leaves us with a training set and a test set as shown in (3.1d).

3.5 Building the Local Model

To avoid confusion with regards to the meaning of classifiers and models, we start this section with a definition of these terms in our study:

Definition 1 (Classifiers and models)**.** *We define the following terms:*

- *Local model: Uses a local classifier to predict delayed stories.*
- *Local classifier: Random Forests Classifier with local features as input.*
- *Relational model: Uses a local classifier to compute initial probabilities. Then uses multiple relational classifiers stacked on top of each other using Stacked Graphical Learning (subsection 3.6.2) and Stacked Inference (subsection 2) to utilize relational information.*
- *Relational classifier: Random Forests Classifier with relational features as input.*

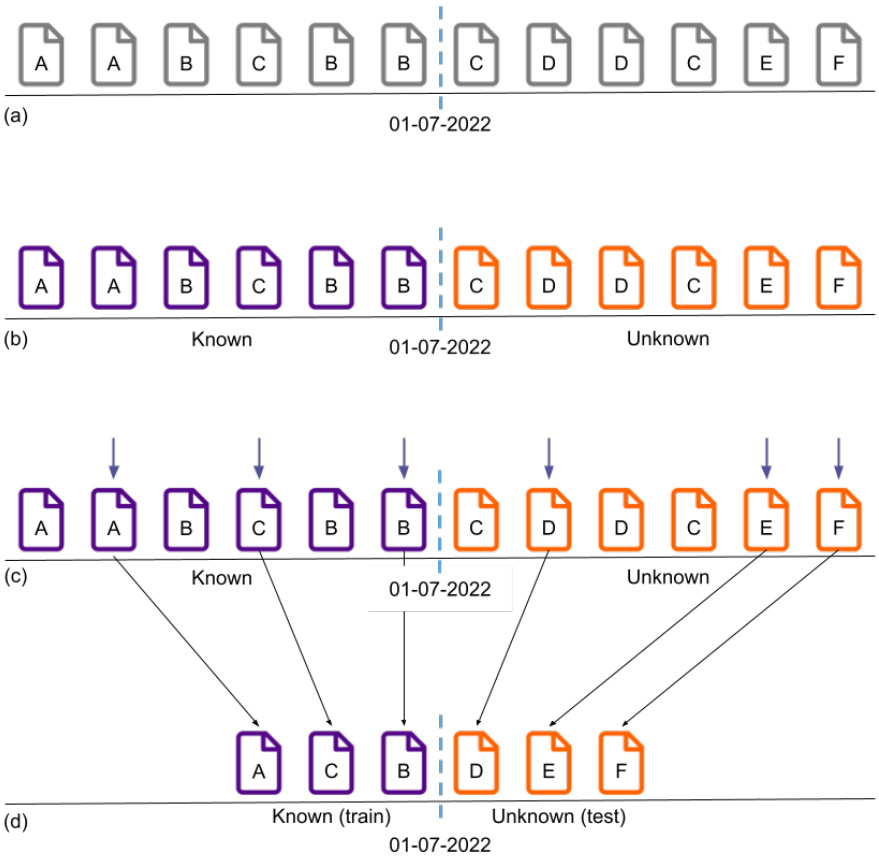


Figure 3.1: Transforming logs to data points

The local model is a Random Forests classifier [6] implemented using the scikit-learn library[2]. The number of trees is set to `100`, which is the default, and we fix the random state to `seed=2000` to ensure consistency across our experiments. This classifier is also used as the basis for the relational model as described in Section 3.6.

The Random Forests classifier fits its model to the given training set based on a number of features. Consequently it estimates the probability of being delayed for the stories in the test set. We use the default threshold of `0.5` to predict the delay status of a story. If the probability is higher than 0.5, the prediction is that the story will be delayed. If it is lower than or equal to 0.5, the prediction is that the story will not be delayed.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

3.6 Building the Relational Model

The relational model is based on the approach of Choetkiertikul et al. [10] and Kou and Cohen [29] and consists of two algorithms: Stacked Graphical Learning for training and Stacked Inference for the prediction phase. Both algorithms make use of the local classifier as described in Section 3.5 and require a story network as additional input. Below we describe the story network construction and the two algorithms used.

3.6.1 Story network construction

The relational model makes use of relations coming from a network of stories. This network is constructed based on the different relational feature sets (R_o, R_a, R_{a-low}, R_r). The story network is based on the task network presented by Choetkiertikul et al. [10] and is defined as follows:

Definition 2 (Story network). *A story network is an undirected graph $G = (V, E)$ where:*

- *Each vertex $v \in V$ represents a story in the form of $\langle id, attrs, p \rangle$, where id is the unique identifier of the story, $attrs$ is a set of attribute-value pairs of the story in the form of $(attr_i, val_i)$ representing the local features and p is the probability estimation of the story being delayed.*
- *Each edge $e \in E$ represents a link between two stories u and v in the form of $\langle u, v, type, weight \rangle$, where $type$ is the relationship type of the link, representing a relational feature, and $weight$ is the weight of the link.*

In Figure 3.2 an example is given of a story network. The stories have different probabilities of being delayed and are linked by different types of relationships. In this example, all weights are set to 1 and are not shown in the diagram. For RQ 1 and 3, all weights are set to 1.

3.6.2 Stacked Graphical Learning

Utilizing network information in a classifier is non-trivial as there are multiple ways of transforming the network information into features that can be used by a classifier. Stacked Graphical Learning [29] provides a way to incorporate relational information in delay prediction. The Stacked Graphical Learning algorithm is applied to the training set created using the method described in Section 3.4 and is described in Algorithm 1.

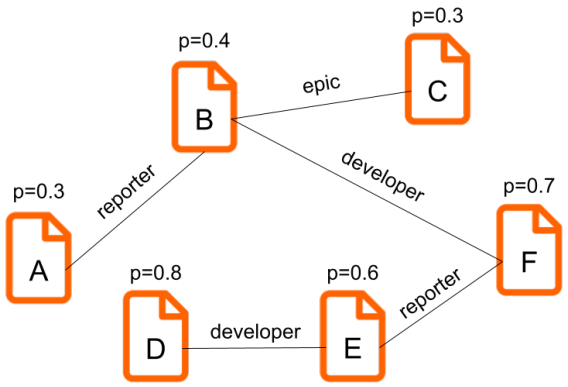


Figure 3.2: An example of a story network with story A to F. The labels on the edges indicate the type of relationship, i.e. story B and C share the same epic.

The idea is to use a local classifier in combination with multiple relational classifiers stacked on top of each other. The local classifier is used to initialise the probability estimations of all the stories in the network. Then, for each node, the probabilities of the neighbours for each type of relationship are averaged to form relational features. Consequently, the local features and relational features are used to train a Random Forests classifier. We then proceed to the next step.

In the next step, the trained classifier from the previous step is used to predict new probability estimations for all stories in the network. The algorithm eventually returns T classifiers for T steps which are then used in the Stacked Inference algorithm described in subsection 3.6.3. In our approach $T = 3$, based on the findings of Kou and Cohen [29].

3.6.3 Stacked Inference

To predict the delay status for stories in the test set, Stacked Inference is used. The Stacked Inference algorithm used in our approach is based on the Stacked Inference algorithm used by Choetkiertikul et al. [10].

The idea is to use the T classifiers created by the Stacked Graphical Learning algorithm to predict the delay status for stories in the test set. The network used in our Stacked Inference algorithm is constructed using the stories in the training set and test set. This is to resemble a real-world scenario as close as possible. By incorporating the training set in the network, we make use of known information as all the stories in the training set exist before the stories in the test set.

The Stacked Inference algorithm is described in Algorithm 2 and is similar to the Stacked Graphical Learning algorithm. The algorithm initialises the probabilities for non-delayed sto-

ries in the training set to 0 and for delayed stories in the training set to 1. The probability estimations for the test stories are initialised using the classifiers from the Stacked Graphical Learning algorithm. Then, the relational features are created in the same way as in the Stacked Graphical Learning algorithm. The last classifier T is finally used to compute the probability estimations of the test stories which predicted as delayed if their probability p is over 0.5.

Algorithm 1: Stacked Graphical Learning algorithm (adapted from [10] and [29])

```
T ← 3 Classifier 1 ← local classifier trained using local features;
for step t = 2,3,...,T do
    Compute probability estimations for all stories in the network using classifier t − 1;
    for each story s do
        for each relation type r do
            Rel. feature r ← average of all prob. of the neighbours with relation r to story s;
        end for
        Create new feature set by concatenating all rel. features with the loc. features;
    end for
    Classifier t ← relational classifier trained on new feature set;
end for
Return T classifiers (1 local, T − 1 relational);
```

Algorithm 2: Stacked Inference algorithm (adapted from [10])

```
Set probability estimation for non-delayed training stories to 0;
Set probability estimation for delayed training stories to 1;
Compute probability estimations for test stories in the network using classifier 1;
for step t = 2,3,...,T do
    Compute probability estimations for test stories in the network using classifier t − 1;
    for each story s do
        for each relation type r do
            Rel. feature r ← average of all prob. of the neighbours with relation r to story s;
        end for
        Create new feature set by concatenating all rel. features with the loc. features;
    end for
end for
 Compute probability estimations p for test stories using classifier T;
for each story s do
    if p_s > 0.5 then
        Classify story s as delayed
    else
        Classify story s as not delayed
    end if
end for
```

Chapter 4

Networked Classification in an Open-Source and Industry Setting (RQ1)

This section focuses on research question 1: *What is the performance of delay prediction using networked classification in an industry setting compared to open-source data?*

We first discuss the general evaluation setup and then present the approach, results and discussion for each subquestion **RQ 1a** to **RQ 1d** separately.

4.1 Evaluation Setup

4.1.1 Creating train/test splits

For every experiment, the input set of features is modified. Every experiment is evaluated using *10-fold time series cross-validation*. Time series cross-validation takes into account that data points in the training set should occur before data points in the test set.

Every month, approximately 30,000 stories are active within the dataset. We pick our fold size in such a way that both the training and test set span roughly one-third of a month. To generate 10 folds, we select the first day of the months February until November as *splitting points*. For each fold, we select 10,000 stories before the splitting point as training set and 10,000 after the splitting point as test set using the method described in Section 3.4. The resulting folds are visualised in Figure 4.1.

4.1.2 Performance metrics

To evaluate the performance of the models, a confusion matrix is used. The results are stored in the confusion matrix as follows: if a story is predicted as delayed, when it actually

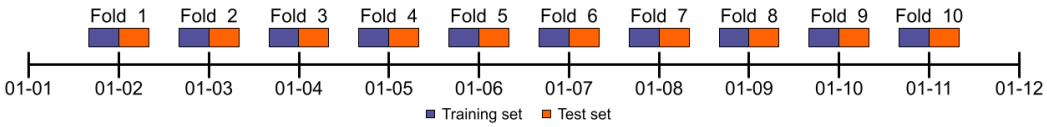


Figure 4.1: Visual representation of the 10 generated folds

is delayed, the classification is a True Positive (TP). If a story is predicted as delayed, when it actually is not delayed, the classification is a False Positive (FP). If a story is predicted as not delayed, when it actually is delayed, the classification is a False Negative. Finally, if a story is predicted as not delayed, when it actually is not delayed, the classification is a True Negative (TN). This is visually presented in Figure 4.2.

We then calculate the following metrics, commonly used in machine learning models for each fold: (a) Precision, the percentage of correctly predicted delays of all stories predicted as delayed. (b) Recall, the percentage of delayed stories predicted correctly of all actually delayes stories. (c) F-measure, the harmonic mean of precision and recall. (d) Area under (ROC) Curve, the ability of the model to distinguish between classes. The exact computation of each metric is shown in Figure 4.3. For each metric, the average over the folds is computed, which gives us the performance measure for each experiment. As we are not focusing on optimising precision or recall specifically, we regard the F-measure and AUC scores as leading for our design decisions.

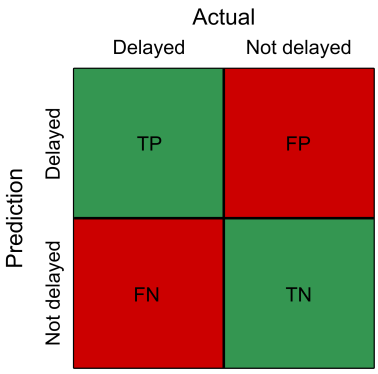


Figure 4.2: Confusion matrix

(a) Precision: $\frac{TP}{TP+FP}$

(b) Recall: $\frac{TP}{TP+FN}$

(c) F-measure: $\frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$

(d) Area under (ROC) Curve: $\frac{TP+TN}{TP+TN+FP+FN}$

Figure 4.3: Performance measures

4.2 Open-Source Features in an Industry Setting (RQ 1a)

RQ 1a: *What is the performance of delay prediction in an industry setting using features that work well on open-source data?*

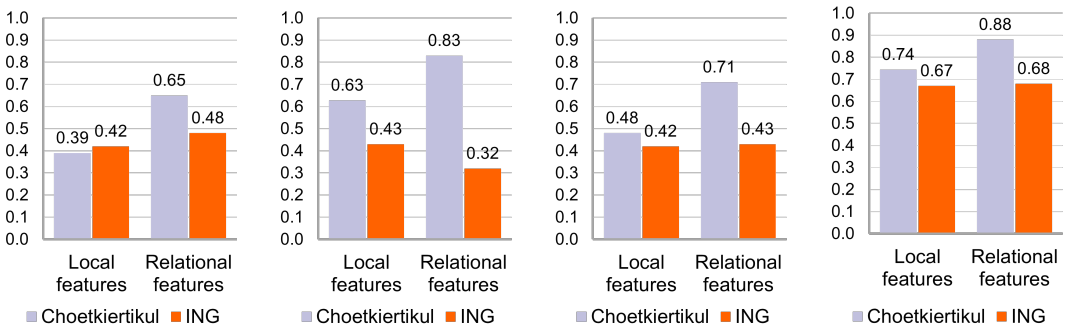
4.2.1 Approach

To answer this research question, we consider the local model and relational model separately. First, we use the local open-source features which are available at ING (L_o) in the local model. Then, we use these features together with the relational open-source features which are available at ING (R_o) in the relational model.

The performance is then compared against the reported performance of Choetkiertikul et al. [10]. Note that Choetkiertikul et al. report the performance scores for their models on five different projects, the average of these performance scores is used as baseline to compare our results to.

4.2.2 Results

Figure 4.4 shows the precision, recall, F-measure and AUC achieved by the local model and the relational model. The performance scores are compared against the reported results of Choetkiertikul et al. [10].



(a) Precision (b) Recall (c) F-measure (d) AUC

Figure 4.4: Evaluation results for baseline and using feature sets L_o and R_o on ING data

It is directly clear that in both experiments, local and relational, the performance of the models is worse at ING than in open-source projects. Only the precision in the local model is better at ING. Also note that the difference in performance is greatest for the relational model.

When comparing the local model to the relational model at ING, we see that the relational model performs better in terms of precision, whilst the local model performs better in terms of recall. The scores of the F-measure and AUC are very close to each other for both models.

4.2.3 Discussion

It is not surprising that the performance at ING is worse than the performance reported by Choetkiertikul et al. on open-source data as less features are available within the ING context. Choetkiertikul et al. make use of 15 local features and 6 relational features. At ING, only 5 of the local features are available (resulting in feature set L_o) and only 2 of the relational features are available (resulting in feature set R_o).

This discrepancy can be explained by the organisational structure of the two different settings. In an open-source project, the main communication channel is the project management system as developers are usually joining the project from all over the world. This means that it is highly important to keep track of a lot of information in the project management system. In an industry setting, developers usually work in teams which are focused on a specific product or part of the software system. Developers make use of project management systems, but this is not their main communication channel. They meet daily and often work from the same location, which means that a lot of information can be shared outside of a project management system. Information like the *number of votes*, *number of watches* or *discussion time* is often not useful for industry developers to keep track of.

One of the relational features that is missing at ING, is the explicit *blocked by* dependency (i.e. story A is blocked by story B). This dependency provides very valuable relational information when explicitly kept track of [10].

In the subsequent research questions, we therefore explore which information is available within the industry setting that can be used for delay prediction.

Insight 1 (Difference in availability of information between open-source and industry)**.** *There is a difference in which information is kept track of within open-source projects and within industry projects. This is due to the organisational structure of the two settings.*

Furthermore, the dataset used at ING comprises all stories of all software teams worldwide. This means that multiple projects are included in the dataset and it is possible that the model is trained on data from project A, while its performance is tested on project B. This can also explain the worse performance at ING compared to the open-source setting.

4.3 Local Features from Industry (RQ 1b)

RQ 1b: *Which local features, available at ING, improve delay prediction in an industry setting?*

4.3.1 Approach

For answering this research question, we are going to extend the local open-source feature set L_o with the local ING feature set L_i in two steps:

Firstly, we add the *Blocked*, *Request*, *Secure* and *Story Points* fields as features. This is because these fields can be directly fed into the Random Forests classifier and do not need an engineering step. This set of features is therefore indicated by *non-engineered ING*. We add each feature separately to the model and measure its performance. Based on the scores for each feature, we select the best combination of features and measure the model performance again. The result is compared to the results presented by Choetkiertikul et al. [10] and the result of only using feature set L_o. The best set of features is then used as the basis for step 2.

Secondly, we engineer the *Developer workload (story points)* and *State at start sprint* features. This set is indicated by *engineered ING*. These features are added separately and in combination to the best set of features from the *non-engineered ING* feature set. Based on the performance, we select which features should finally be included in the set of local features which improve delay prediction in an industry setting. This best set of features is the answer to RQ 1b.

4.3.2 Results

Figure 4.5 shows the precision, recall, F-measure and AUC achieved by adding the *non-engineered ING* and *engineered ING* to the local open-source feature set L_o. The performance scores are compared to the results presented by Choetkiertikul et al. [10] and to the performance of local model using only local open-source feature set L_o.

For each added set of features, there is a clear increase of precision. F-measure and AUC also show some improvement, while recall shows a decrease. Compared to the results of Choetkiertikul et al., the local model is still not able to perform better on ING data with regards to the recall, F-measure and AUC, but it does approach the same performance in terms of F-measure and AUC.

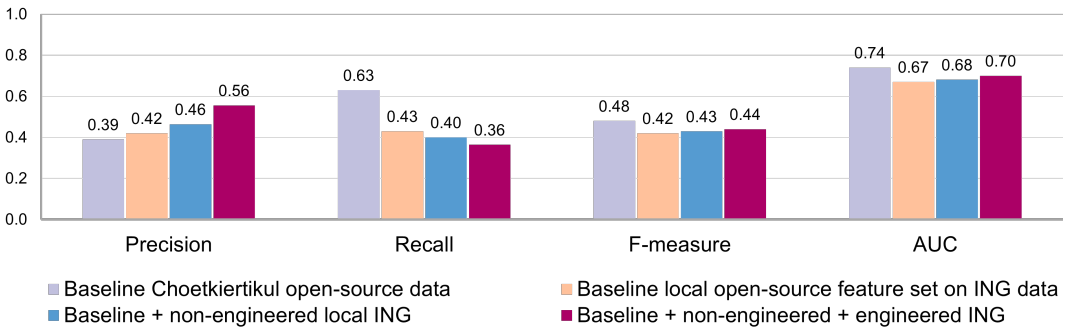


Figure 4.5: Evaluation results for baseline and using feature sets L_o and L_i on ING data

In Table 4.1 the performance measures of adding different combinations of features to the local open-source feature set L_o are shown. These results are discussed below.

Table 4.1: Performance scores for adding local ING features to L_o

Features	Precision	Recall	F-measure	AUC
Baseline (Choetkiertikul)	0.392	0.434	0.482	0.743
Baseline (L_o on ING data)	0.420	0.425	0.422	0.667
Extending baseline with:				
Blocked	0.421	0.421	0.421	0.668
Request	0.420	0.424	0.422	0.667
Secure	0.418	0.425	0.421	0.666
Story Points	0.458	0.409	0.432	0.682
L_o + all non-engineered ING	0.464	0.400	0.429	0.681
Extending L_o and non-engineered ING with:				
Developer workload (story points)	0.501	0.394	0.441	0.693
State at start sprint	0.517	0.326	0.400	0.692
Both of the above	0.556	0.364	0.440	0.702

Adding *Blocked*, *Request* and *Secure* to L_o shows almost no difference in performance, whilst adding *Story Points* does show an increase in precision, F-measure and AUC. When all non-engineered ING features are added to L_o, we see the greatest improvement in terms of precision, F-measure and AUC, but a decline in terms of recall.

Adding *Developer workload (story points)* shows an improvement in precision, F-measure and AUC, while there is a small decline in recall. Adding *State at start sprint* provides an

increase in precision and AUC, but a decrease in recall and F-measure. Adding both of these features shows an even larger improvement in precision and AUC, while also showing an improvement in F-measure. The recall does however decrease compared to the baseline and adding the *non-engineered ING* feature set.

4.3.3 Discussion

When adding the local ING features, the results show a minimal increase in F-measure and AUC, while precision and recall tend to deviate more from the baseline. This means that adding more features from available industry data does increase the performance of the local model. The local model does however not yet achieve the same results as Choetkiertikul et el. report for open-source projects [10]. This is not surprising as Choetkiertikul et al. use 15 local features while the feature set we use (L_o + L_i) consists of 11 features.

It is interesting to see that the precision increases for each added feature set, but the recall decreases (see Figure 4.5). This might be explained by the fact that our experiments are run on many different projects from different teams worldwide such as described in subsection 4.2.3. Every team makes use of ServiceNow differently. For example, team A could assign 2 story points to a story that team B would assign 4 story points to. This means that features of a delayed story can have different values for different teams and learning to characterise these stories is harder.

Insight 2 (Record-keeping is treated differently across teams)**.** *Every team treats record-keeping differently. This means that features of stories can have different values for different teams. Learning to characterise delayed and non-delayed stories when training on data from many different teams will therefore be harder.*

4.4 Relational Features from Industry (RQ 1c)

RQ 1c: *Which relational features, available at ING, improve delay prediction in an industry setting?*

4.4.1 Approach

For answering this research question, we use a local classifier with feature sets L_o and L_i as the basis for the relational model. The relational classifier makes use of the relational features available at ING (R_a, R_{a-low}, R_o) and its performance is compared to the following three baselines:

1. The performance scores of the relational model of Choetkiertikul et al. [10].
2. The performance of using the local open-source and ING feature sets L_o, L_i in the local model.
3. The performance of using the local open-source and ING feature sets L_o, L_i in combination with the relational open-source feature set R_o in the relational model.

Firstly, the relational model is built with relational attribute-based (R_a) and resource-based (R_r) feature sets. We first add each feature set separately and then combine them to evaluate the effect of adding relational features available in the industry setting.

Secondly, we build a relational model using the features from the relational attributes-based features with low availability R_{a-low}. This is done separately from R_a and R_r because the number of data points available with features from R_{a-low} is significantly lower. These smaller datasets will be used to re-create baseline 2 in order to evaluate the effect of these features specifically. The performance scores of adding these features will be compared against baseline 1 and baseline 2 with the correspoding smaller datasets.

4.4.2 Results

Figure 4.6 shows the precision, recall, F-measure and AUC achieved by using the relational attribute-based (R_a) and resource-based (R_r) feature sets. The performance scores are compared to the baselines as described in subsection 4.4.1.

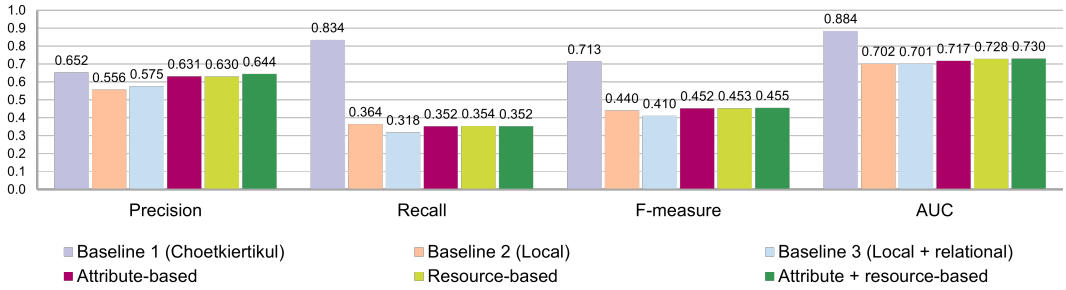


Figure 4.6: Evaluation results for building a relational model with feature sets R_a and R_r on ING data

For both relation sets R_a and R_r and their combination, it is clear that the models do not outperform baseline 1. The models do however show an improvement with regards to baseline 2 and 3. The model with features from both R_a and R_r performs best of the three relational models on ING data, but the difference between the three models is very small.

For the features of the relational attribute-based features with low availability set R_{a-low}, baseline 2 has been recomputed using the smaller datasets available for each of the features. The sizes of these datasets and relevant characteristics are shown in Table 4.2. In terms of number of stories, the *Feature Reference* dataset is the largest, with approximately half the number of stories as in the complete dataset. The *Component* and *Change Reference* datasets are relatively small, respectively containing only 1.2% and 5.5% of the total number of stories. When the three features are combined, we are left with a dataset that is extremely small in comparison with the complete dataset, containing only 0.1% of the total number of stories.

Table 4.2: Data characteristics for availability-low datasets

Number of	Component	Feature Ref.	Change Ref.	Combined	Complete dataset
Stories	3,634 **(1.2%)**	150,839 **(49.6%)**	16,864 **(5.5%)**	262 **(0.1%)**	303,978 **(100.0%)**
Developers	523 **(3.7%)**	9,529 **(67.7%)**	2,965 **(21.1%)**	71 **(0.5%)**	14,076 **(100.0%)**
Reporters	424 **(3.8%)**	7,054 **(62.7%)**	2,572 **(22.9%)**	62 **(0.6%)**	11,254 **(100.0%)**
Squads	127 **(6.4%)**	1,474 **(74.5%)**	705 **(35.6%)**	29 **(1.5%)**	1,979 **(100.0%)**
Tribes	57 **(20.0%)**	230 **(80.7%)**	131 **(46.0%)**	14 **(4.9%)**	285 **(100.0%)**
Epics	429 **(3.1%)**	6,566 **(48.1%)**	2,226 **(16.3%)**	59 **(0.4%)**	13,653 **(100.0%)**
Themes	145 **(9.2%)**	1,083 **(68.8%)**	499 **(31.7%)**	26 **(1.7%)**	1,573 **(100.0%)**

The result of using these smaller datasets to recompute baseline 2 is shown in Figure 4.7. For the datasets corresponding to *Change Reference*, *Component* and all three features from R_{a-low} combined, the recomputed baseline 2 even outperforms the results reported by Choetkiertikul et al.. For the dataset corresponding to *Feature Reference*, the recomputed baseline 2 is similar to the original baseline 2.

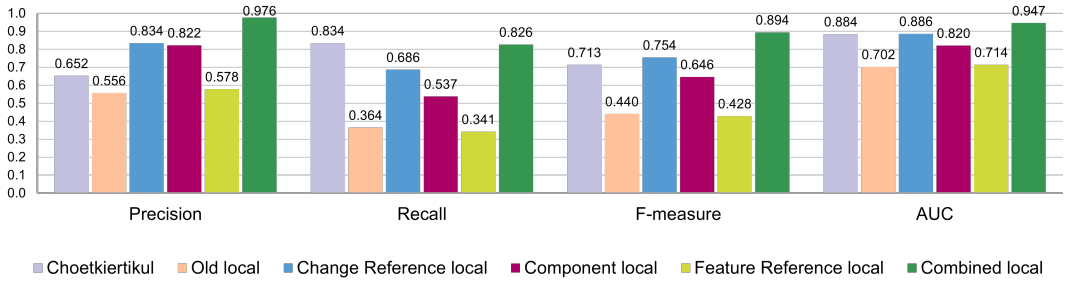
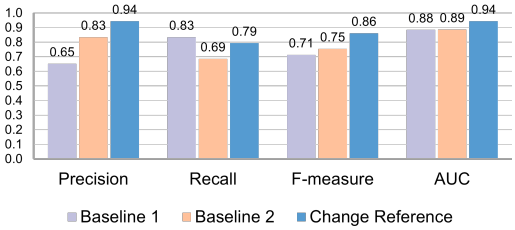


Figure 4.7: Recomputed baseline 2 for datasets corresponding to features from R_{a-low}

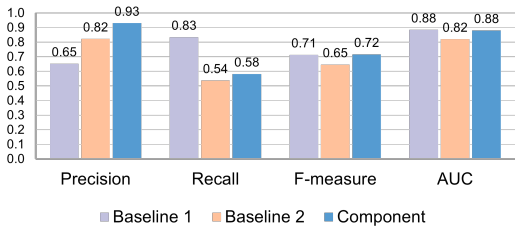
Figure 4.8 shows the precision, recall, F-measure and AUC achieved by adding each of the features of the relational attribute-based features with low availability set R_{a-low}.The ap-

proach using *Change Reference*, *Component* and all three features from R_{a-low} combined, outperforms baseline 1 and 2, with the exception of recall for the individual features. The combined set reaches scores above 90%, with a precision of 100% and AUC of 99%. For *Change Reference* and all features combined, we see that baseline 3 outperforms baseline 1 with regards to precision, F-measure and AUC.

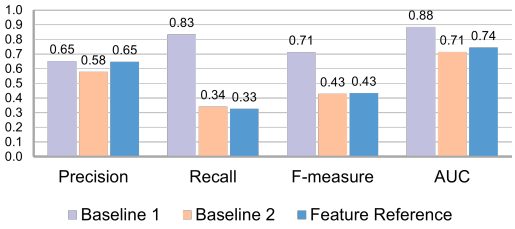
Using *Feature Reference* on its corresponding dataset shows an improvement compared to baseline 2 with regards to precision and AUC, but it does not outperform the results of baseline 1.



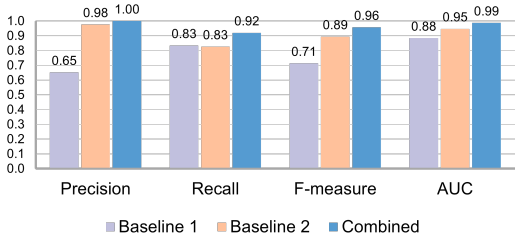
(a) Change Reference



(b) Component



(c) Feature Reference



(d) Features (a), (b) and (c) combined

Figure 4.8: Evaluation results for building a relational model with features from R_{a-low} on ING data

4.4.3 Discussion

When using relational ING features R_a and R_r, the results show an increase in all performance scores compared to baseline 2 (only local features) and baseline 3 (local features and relational open-source features). This shows that adding relational features from ING improves the prediction performance.

There is no significant difference between using attribute- or resource-based features, both sets show similar performance scores. Combining them does however show an increase in performance with regards to precision, F-measure and AUC. From this, we can

conclude that increasing the number of relational features has a positive effect on the prediction performance.

Insight 3 (Using networked information improves delay prediction)**.** *Compared to only using local features, adding networked information improves delay prediction. Increasing the number of relational features improves the prediction performance even more.*

Using the features from R_{a-low} and their corresponding datasets shows very interesting results. When using the datasets corresponding to *Change Reference*, *Component* and all three features from the set combined to recompute baseline 2, we see an improvement for all performance measures. This suggests that these datasets are better suited for delay prediction. This can be explained by looking at the characteristics of these datasets in Table 4.2. The unique numbers of developers, reporters, squads and epics is less than a quarter of their unique numbers in the complete dataset. These smaller dataset are thus less diverse with regards to the organisational units compared to the complete dataset.

The recomputed baseline 2 using the *Feature Reference* dataset shows minimal deviation from the original baseline. Looking at the data characteristics, we see that, compared to the other datasets corresponding to features from R_{a-low}, the *Feature Reference* dataset still has a high number of data points. This large subset of the complete dataset is apparently too diverse to show an improvement of the baseline.

Insight 4 (Using a general approach for ING as a whole performs worse than a team-specific approach)**.** *A model that is trained on data from all teams at ING has worse prediction performance than a model that is trained on specific organisational units such as squads and tribes or on specific domains using epics and themes. Training on specific organisational units allows the model to be fit better to the data and consequently improve its prediction performance. This is in line with Insight 2.*

Adding the features from R_{a-low} shows an improvement of all performance measures, which is in line with insight 3.

4.5 Feature Importances (RQ 1d)

RQ 1d: *Which features are most important for delay prediction on ING data?*

4.5.1 Approach

The feature importances of a classifier can be found by using the `feature_importances_` function provided by the `sci-kit learn` library[1]. The feature importances of the baselines and best performing sets of the previous research questions are reported, which are:

- Local open-source (L_o)
- Local open-source + local ING (L_o + L_i)
- Choetkiertikul et al. (L_o + R_o)
- All local + relational attribute + relational open-source (L_o + L_i + R_a + R_o)

4.5.2 Results

Figure 4.9 shows the feature importances across using different sets of features. We see that *Waiting time* and *Delayed ratio* score highly for every feature set. Additionally, *Story Points*, *Developer workload in story points* and *State at start sprint* are amongst the top-most important features for the local ING approach. For the relational ING approach, we see a much more balanced distribution of feature importances across all features.

4.5.3 Discussion

Before discussing the feature importance results, it is important to note that if a feature is important for a certain model, this does not necessarily indicate that it is a good predictor for delays. A feature can be very important to a certain model, but if that model has low performance scores, the feature is apparently a large contributor to poor predictions. Conversely, if the model has very high performance scores, the feature is a contributor to good prediction and thus is a good predictor for delays.

Across all feature sets, *Waiting time* and *Delayed ratio* show a high importance, indicating that these features are important for delay prediction in general. Even in the relational setting, their importances are higher than those of the relational features, emphasising their contribution to delay prediction.

[1] https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

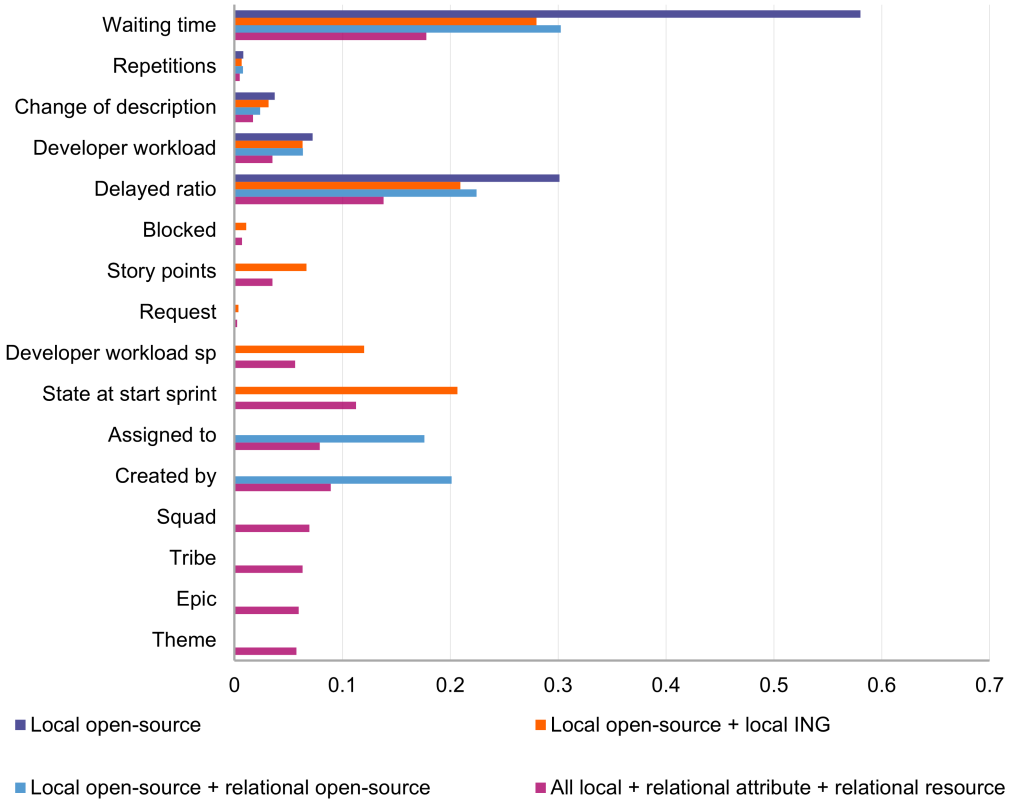


Figure 4.9: Feature importances for different models

For the relational features, the importance scores are relatively similar, suggesting there is not a specific relational feature that is highly effective for delay prediction in networked classification. It is the combination of the features that contributes to the performance of the model.

Chapter 5

Using Weights in Networked Classification (RQ 2)

This section focuses on research question 2: *What is the impact of adding weights to relations on delay prediction using networked information?*

5.1 Evaluation Setup

For both RQ 2a and RQ 2b, the best performing relational model of RQ 1 which uses the complete dataset is chosen as the baseline. This is the relational ING model which uses feature sets L_o, L_i, R_o, R_a and R_r. This model is then extended and evaluated using the same evaluation setup as in RQ 1 (see Section 4.1).

5.2 Weights Based on Time Interval (RQ 2a)

RQ 2a: *What is the prediction performance when determining weights based on the time interval between stories?*

5.2.1 Approach

Suppose a developer used to cause a lot of delays in the beginning of his career. Some months later, the developer has gotten more experience and the amount of caused delays has greatly decreased. In the relational model, all the stories of this developer are related to each other with equal weight. Predictions for new stories assigned to this developer will thus be influenced as much by the 'old' delayed stories as by the 'new' non-delayed stories. To make use of this historical information and reduce the influence of 'old' stories, we add weights to the relationships in the network based on the time interval between stories.

When creating the story network as described in subsection 3.6.1 every edge is assigned a weight using the formulas below. For each vertex (story) v in the network we have a set of edges (relations) E_v. The weights of the relations are determined for each type of relationship. The set of edges (relations) with a specific type (of relationship) r for a specific vertex (story) is then denoted by $E_{v,r}$. The weight w_e of edge $e \in E_{v,r}$ is computed as follows:

Let

$$\begin{aligned} X &= \text{Set of intervals for edges in } E_{v,r} \\ Z &= \text{Set of intervals normalised between 0 and 1 for edges in } E_{v,r} \\ W &= \text{Set of weights for edges in } E_{v,r} \text{ where } sum(W) = 1 \end{aligned} \tag{5.1}$$

Then for each interval i in the set X we apply the 0,1 normalisation as follows:

$$Z_i = \frac{max(X) + min(X) - X_i}{max(X) - min(X)} \tag{5.2}$$

And transform the normalised intervals to weights as follows:

$$W_i = \frac{Z_i}{sum(Z)} \tag{5.3}$$

The resulting weights have values between 0 and 1 and the sum of the weights equals 1. The largest interval has the smallest weight and the smallest interval has the largest weight. In the relational model these weights are then used to compute the weighted average of the probability estimations of the neighbours instead of the average. As $sum(W) = 1$, the resulting weighted averages range between 0 and 1.

We set the time-dependent weights in the relational ING model and compare it with the same model where all weights are set to 1.

5.2.2 Results

In Figure 5.1 the precision, recall, F-measure and AUC for using time-dependent weights are shown. There is an improvement with regards to the recall, but no improvement with regards to the other performance measures. The F-measure is equal for both models.

5.2.3 Discussion

Adding time-dependent weights to the relational ING model does not improve its prediction performance. The recall is slightly higher, but the F-measure is unchanged and the precision and AUC are lower. It seems that adding time-dependent weights introduces more noise to the model. It might be the case that, within the timespan of our dataset, older stories are

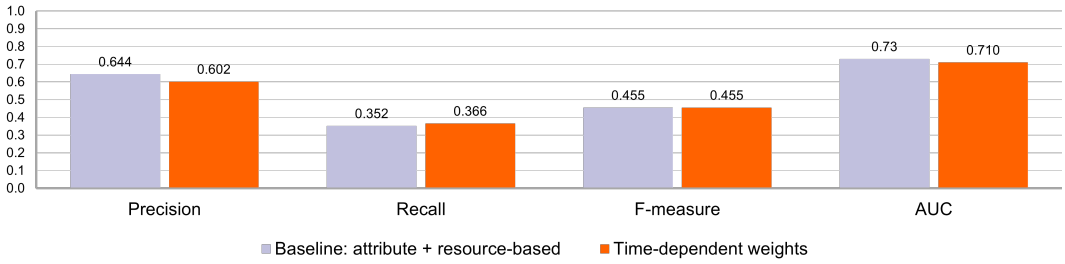


Figure 5.1: Results for using time-dependent weights

equally relevant as more recent stories. The fact that we are applying the model on a general dataset instead of on a more specific subset (Insight 4) can also be a factor.

5.3 Weights Based on Assortativity Coefficient (RQ 2b)

RQ 2b: *What is the prediction performance when determining weights using the assortativity coefficient of the network?*

5.3.1 Approach

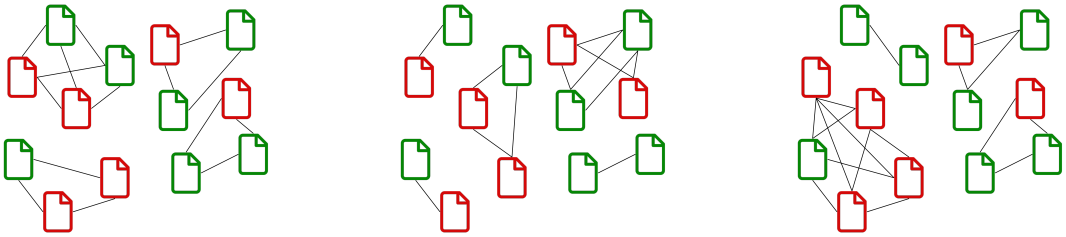
The connectivity of a graph can be expressed by its assortativity coefficient. The assortativity coefficient of a graph can be calculated in various ways and for our study it is most interesting to consider the attribute assortativity coefficient [34]. The attribute assortativity coefficient is defined as: the tendency of nodes in a network to connect with nodes that share similar attributes. In the context of delay prediction using networked classification this is the tendency of stories to have a relationship with stories that have the same delay status.

The relational ING model does not differentiate between the influence of different types of relationships. Each type of relationship is included as a separate feature, which is then fed into the classifier. We can however differentiate between the relational features by combining them into one feature using weights based on the assortativity coefficient for each relational feature to improve prediction performance [32].

For each relational feature, we create a separate network as shown in Figure 5.2. Each network has a corresponding assortativity coefficient, which serves as the weight for that specific relationship type. The assortativity for each relation type is computed using the `attribute_assortativity_coefficient` of the `networkx` library[1]. The weights are de-

[1] https://networkx.org/documentation/stable/reference/algorithms/assortativity.html

termined by scaling all assortativity scores such that the sum of scores is equal to 1. A high assortativity corresponds to a high weight, a low assortativity corresponds to a low weight. To create the combined feature, we take the weighted average of the relational feature and store the result in a new feature with the name *Neighbours*.



(a) Network 1 (b) Network 2 (c) Network 3

Figure 5.2: Example of different networks based on different relational features. Delayed stories are red, non-delayed stories are green

We can include the *Neighbours* feature in the relational model in two ways:

1. Build a relational model with feature sets L_o, L_i as input, extended with the *Neighbours* feature created using R_o, R_a, R_r. This model is indicated by *local + neigbours* (L_N)
2. Build a relational model with feature sets L_o, L_i, R_o, R_a, R_r as input, extended with the *Neighbours* feature created using R_o, R_a, R_r. This model is indicated by *local + relational + neighbours* (LR_N)

The L_N and LR_N models are compared against the relational ING model and the relational ING model using time-dependent weights from RQ 2a.

5.3.2 Results

In Figure 5.3 the precision, recall, F-measure and AUC for using assortativity-based weights in the relational model are shown. With regards to recall, F-measure and AUC, the L_N model shows the highest scores. The performance measures for the LR_N model are very similar to those of the relational model using time-dependent weights. Both L_N and LR_N do not outperform the relational ING model.

5.3.3 Discussion

Adding assortativity-based weights to the relational ING model does not improve its prediction performance. The recall is slightly higher, but the F-measure is unchanged and the

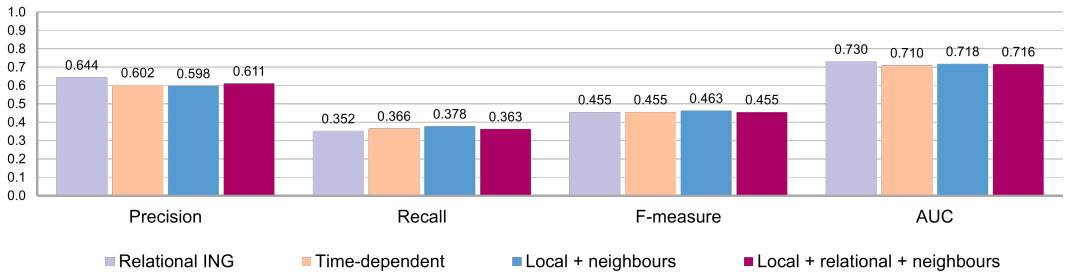


Figure 5.3: Scores for using assortativity-based weights

precision and AUC are lower. It seems that adding assortativity-based weights introduces more noise to the model. Similar to the time-dependent weights model, the fact that we are applying the model on a general dataset instead of on a more specific subset (Insight 4) could explain the non-effectiveness of the assortativity-based weights model.

Excluding or including the relational features separately also does not seem much effect. The precision of LR_N is higher than the precision of L_N, thus more stories are predicted correctly. In terms of the other performance measures however, L_N scores better, especially for the recall, thus more of the delayed stories are identified. This can again be attributed to Insight 4.

Insight 5 (Adding weights to the network does not improve prediction performance)**.** *Adding either time-dependent or assortativity-based weights to the relational ING model does not improve its performance. This can likely be attributed to the diversity of the dataset as described in Insight 4.*

Chapter 6

Using Different Sliding Window Sizes for Delay Prediction (RQ 3)

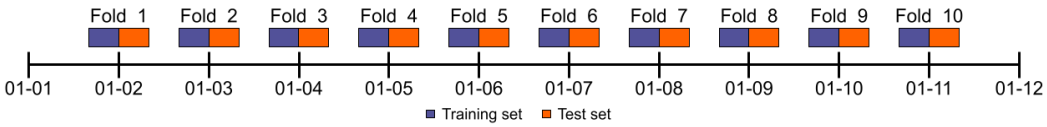
This section focuses on research question 3: *What is the effect of using different sizes of sliding windows when performing delay prediction on ING data?*

6.1 Evaluation Setup and Approach

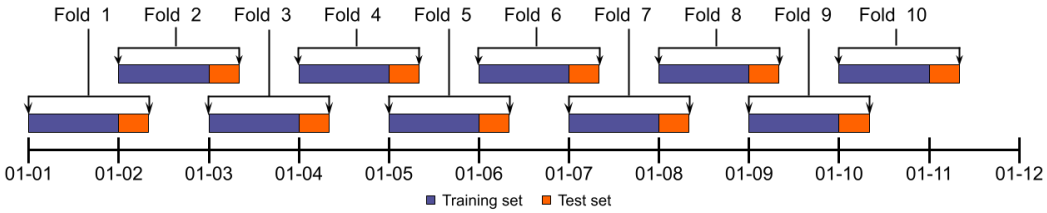
For this research question, models with the following set of features are used:

- Local open-source (L_o)
- Local open-source + local ING (L_o + L_i)
- Choetkiertikul et al. (L_o + R_o)
- All local + relational attribute + relational resource (L_o + L_i + R_a + R_r)

In RQ 1, the prediction performance of these models has been evaluated using a sliding window with a training and test set of 10,000 stories. To answer RQ 3, the models are trained with the following training set sizes: 100, 1000, 5000, 10000, 20000, 30000. The test set size is kept at 10000 for each experiment. In Figures 6.1 the folds are shown for training set sizes 10,000 and 30,000. The prediction performance of using different training set sizes are compared for each model.



(a) Training set size 10,000 and test set size 10,000



(b) Training set size 30,000 and test set size 10,000

Figure 6.1: Visual representation of 10 folds with different training set sizes

6.2 Results

In Figure 6.2 the precision, recall, f-measure and AUC of using different training set sizes for a sliding window are shown. The scores for using a training set size of 100 data points deviate from the other training set sizes.

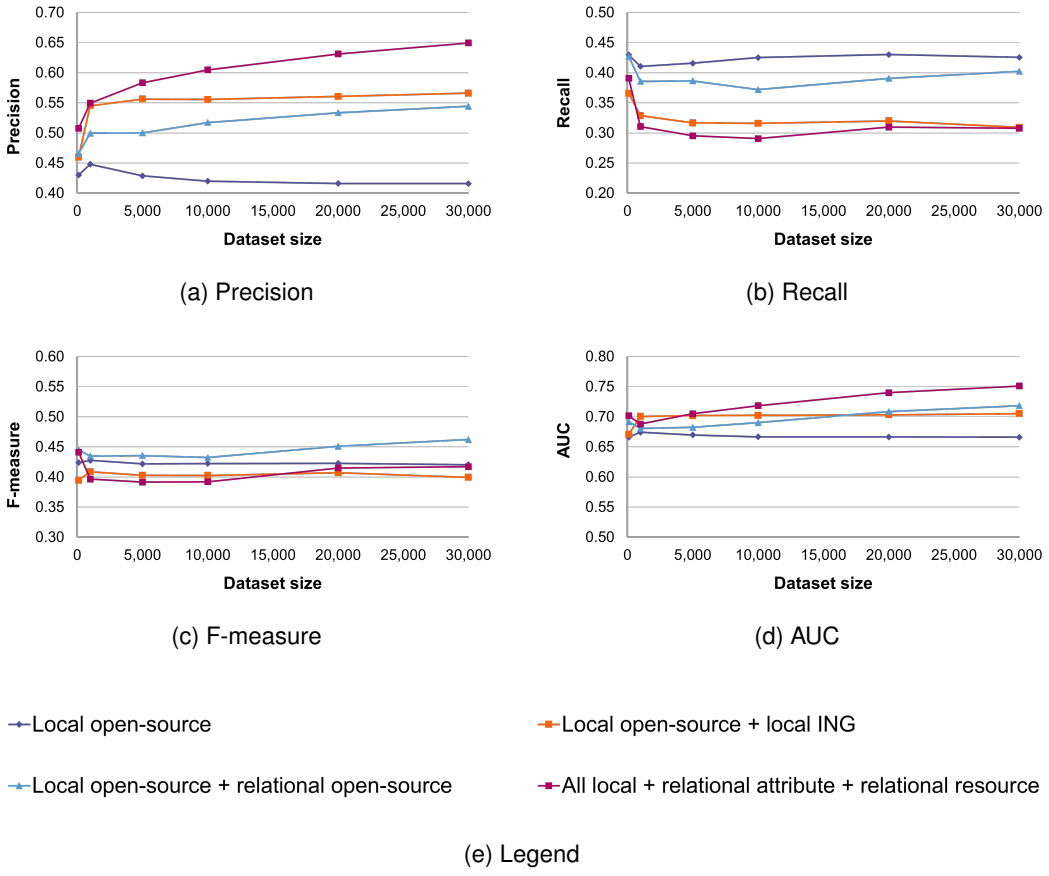
For the two local models, changing the training set size does not have any significant effect. The two relational models do show an improvement with regards to precision, F-measure and AUC when the training set size is increased.

6.3 Discussion

Using a training set size of 100 data points shows significantly different results compared to using other training set sizes. This can be explained by the diversity of the training set (see Insight 4).

Aside from the training set size with 100 data points, the local models show no change in performance across using different training set sizes. The Random Forests classifier is not able to fit a better model when more data points are added. This can possibly be explained by the fact that there is a certain level of noise in the model due to the general approach (Insight 2). Adding more data points does not change the level of noise. Note that for the local open-source model, the precision decreases, so the noise for this model does increase.

The two relational models do show an improvement with regards to precision, F-measure



(a) Precision

(b) Recall

(c) F-measure

(d) AUC

(e) Legend

Figure 6.2: Evaluation results for using different training set sizes. Note that the range of the y-axis varies across the diagrams. The first data point of each diagram is at `dataset size = 100`

and AUC. The recall scores are relatively constant. When more data points are added to the relational model, the constructed network will be larger and there is more relational information available that can be used by the model for its predictions. Creating such a larger network and consequently performing predictions on it does however require more resources (which can be the time available to run the model or the space available to store the data). So using larger datasets is beneficial for relational models, but a trade-off should be made between the size of the dataset and the feasibility of using the model with regards to the available resources. We provide an insight on the next page.

Insight 6 (Larger training set sizes improve prediction performance for relational models)**.** *Using larger training set sizes for the relational model improves its prediction performance. This is because the constructed network is larger and there is more relational information available. There are limits to increasing the training set size as there are computational limits with regards to the time and space (memory) available for the model to run.*

Chapter 7

Threats to Validity

In this chapter, threats to the validity of the study are discussed. We consider construct, internal and external validity.

7.1 Construct Validity

Construct validity defines how effectively an experiment measures up to its claims and deals with whether or not the researcher measures what is intended to be measured [1]. Data variables are considered as constructs that meaningfully measure delay indicators, which introduces possible threats to construct validity [41].We mitigated these threats by collecting real-world data from stories and developers at ING, including all the historical information available.

The ground-truth of the delay status is based on the number of sprints that a story has been assigned to. It might however happen that teams do not properly keep track of their stories within the project management system. Stories might be closed too early or too late, teams might add stories to sprints without being committed to actually finish them within that sprint, or fields like the *Blocked* status of a story are not used. We cannot account for the impact of poor record-keeping by teams on our results.

7.2 Internal Validity

Internal validity defines to what extent the result of an experiment is related to the experimental condition applied [1]. The dataset in our study suffers from the class imbalance problem as 73% of the stories are non-delayed and 27% are delayed. This may affect the ability of the classifier to learn to identify delayed stories. To mitigate the risk of this threat, we focused

on the AUC score, which is insensitive to class imbalance. We do acknowledge that more advanced techniques could also be used, such as statistical over-sampling [8].

Another threat to the internal validity of our study is that the characteristics in the training sets do not resemble those in the test sets. Different (amounts) of teams, tribes and epics might be represented in the sets. To mitigate this threat, we mimicked a real-world scenario by using time series cross-validation.

7.3 External Validity

External validity defines to what extend the result of an experiment can be generalised [1]. Our study is focused on one case, ING, but within this case, we considered 303,978 stories from 1,979 teams. These significantly differ in size, composition and application domain as 13,853 epics from 1,573 themes are considered (Table 4.2). Our data is thus representative of a variety of software projects, but the results might not generalise to other organisations, as the organisational structure and way of working can differ greatly from the ING setting.

Chapter 8

Conclusion and Future Work

In this final chapter, we revisit the insights gained from the different experiments and give recommendations to ING based on these insights. The chapter is concluded by a number of suggestions for future work.

8.1 Insights

In this study, we have investigated the performance of relational models for predicting delays in software deliveries at ING. Different feature sets for delay prediction (RQ 1), using weights in relational models (RQ 2) and different dataset sizes (RQ 3) have been considered. The results yield a number of insights:

1. There is a difference in the information that is available between open-source and industry settings.
2. Record-keeping is treated differently across teams.
3. Using networked information improves delay prediction.
4. Using a general approach for ING as a whole performs worse than a team-specific approach.
5. Using weights in networked classification does not improve prediction performance at ING.
6. Larger dataset sizes improve prediction performance for relational models.

These insights indicate that delay prediction using networked classification within an industry can be beneficial, but that factors regarding the dataset composition and size should be taken into account to achieve better performance.

8.2 Recommendations for ING

Based on the different insights that have been gained throughout the study, there are a number of recommendations that can be given to ING.

Keep track of more information

There is already a lot of information available from the data warehouse at ING, however, only 7 of the 21 features that have proven to work well for delay prediction on open-source data could be engineered. As *Waiting Time* showed to be an important feature for delay prediction, explicitly keeping track of this and possibly other features which track duration could improve prediction performance. In addition, explicit dependencies between stories (i.e.: story A blocks story B) could improve the prediction performance of relational models.

Improve data quality

We have seen that across teams, the project management system is used differently, as not all teams keep track of the same information The *Component* field is a good example of this. Only 127 teams of the total 1,979 teams in dataset make use of the *Component* field, while this has shown to greatly improve prediction performance. Teams might also treat record keeping of stories less seriously, with the consequence that the stories in the project management system do not correctly reflect the actual state of their work. When teams are encouraged to better and more consistently keep track of their stories, this could improve the delay prediction.

Create models for specific domains or organisational units

In our study, we have applied delay prediction on data from a variety of teams working across a large number of different domains. This general approach proved to be less effective at predicting delays than a more domain-specific or team-specific approach. When the prediction models were trained on a subset of the data, originating from a smaller number of unique teams, the prediction performance increased. We therefore recommend to build prediction models at domain-, tribe- or team-level instead of building a model for all the teams in the whole company.

8.3 Future Work

In this work, we investigated the performance of networked classification for delay prediction in an industry setting. We provide the following suggestions for further research in this area:

Use second-order Markov assumptions in the relational model

The relational classifiers in our relational model make use of a first-order Markov assumption, hence, only information about direct neighbours of a story is used. It might however be the case that a delay propagates through the network, so using information about further neighbours could improve the prediction performance of the relational model.

Make rescheduling recommendations based on delay prediction

When the model predicts that stories will be delayed based on the sprint planning of a team, it would be interesting to give a recommendation how this delay could be mitigated. For example by assigning stories to a different developer or lowering the workload by excluding the story from the sprint.

Investigate the performance of using an expanding window

In our study, we investigated the effect of using different window sizes. The results show that using larger window sizes benefit the prediction performance of the relational models. Due to resource limitations, we could only test the models with a maximum training set size of 30,000 stories. It would however be interesting to investigate the performance of using an expanding window where for each window k_i, the stories of the previous windows $k_0 ... k_{i-1}$ are used to train the model.

Investigate optimal granularity for model-training

Our results indicate that training the prediction model on a smaller variety of teams, tribes and domains achieves better prediction performance. It would be interesting to investigate what level of granularity would be optimal for delay prediction. For example training on stories from specific tribes, teams, themes or epics.

Bibliography

[1] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Information and Software Technology, 106:201–230, 2019.

[2] Kent Beck et al. The agile manifesto. Software development, 9(8):28–35, 2001. URL https://agilemanifesto.org/.

[3] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. Journal of the Royal Statistical Society: Series B (Methodological), 36(2):192–225, 1974.

[4] Pamela Bhattacharya and Iulian Neamtiu. Bug-fix time prediction models: can we do better? In Proceedings of the 8th Working Conference on Mining Software Repositories, pages 207–210, 2011.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. Journal of machine Learning research, 3(Jan):993–1022, 2003.

[6] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[7] Richard Brenner and Stefan Wunder. Scaled agile framework: Presentation and real world example. In 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 1–2. IEEE, 2015.

[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16:321–357, 2002.

[9] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Characterization and prediction of issue-related risks in software projects. In 2015 IEEE/ACM 12th

Working Conference on Mining Software Repositories, pages 280–291. IEEE, 2015. URL https://ieeexplore.ieee.org/abstract/document/7180087.

[10] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Predicting delays in software projects using networked classification. In 2015 30th IEEE/ACM international conference on automated software engineering (ASE), pages 353–364. IEEE, 2015. URL https://ieeexplore.ieee.org/abstract/document/7372024.

[11] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Predicting the delay of issues with due dates in software projects. Empirical Software Engineering, 22(3):1223–1263, 2017.

[12] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. A deep learning model for estimating story points. IEEE Transactions on Software Engineering, 45(7):637–656, 2018.

[13] Alistair Cockburn and Jim Highsmith. Agile software development, the people factor. Computer, 34(11):131–133, 2001.

[14] Mike Cohn. User stories applied: For agile software development. Addison-Wesley Professional, 2004.

[15] Mike Cohn. Agile estimating and planning. Pearson Education, 2005.

[16] Kevin Crowston and James Howison. Hierarchy and centralization in free and open source software team communications. Knowledge, Technology & Policy, 18(4):65–85, 2006.

[17] Emanuel Dantas, Mirko Perkusich, Ednaldo Dilorenzo, Danilo FS Santos, Hyggo Almeida, and Angelo Perkusich. Effort estimation in agile software development: an updated review. International Journal of Software Engineering and Knowledge Engineering, 28(11n12):1811–1831, 2018.

[18] PL Dobruschin. The description of a random field by means of conditional probabilities and conditions of its regularity. Theory of Probability & Its Applications, 13(2):197–224, 1968.

[19] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Transactions on pattern analysis and machine intelligence, (6):721–741, 1984.

[20] Lise Getoor. Link-based classification. In Advanced methods for knowledge discovery from complex data, pages 189–207. Springer, 2005.

[21] Emanuel Giger, Martin Pinzger, and Harald Gall. Predicting the fix time of bugs. In Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, pages 52–56, 2010.

[22] Wei Hu and Kenny Wong. Using citation influence to predict software defects. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 419–428. IEEE, 2013.

[23] Vlad-Sebastian Ionescu, Horia Demian, and Istvan-Gergely Czibula. Natural language processing and machine learning methods for software development effort estimation. Studies in Informatics and Control, 26(2):219–228, 2017.

[24] Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), pages 52–61. IEEE, 2008.

[25] Tian Jiang, Lin Tan, and Sunghun Kim. Personalized defect prediction. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 279–289. Ieee, 2013.

[26] Magne Jørgensen. A review of studies on expert estimation of software development effort. Journal of Systems and Software, 70(1-2):37–60, 2004.

[27] Henrik Kniberg and Anders Ivarsson. Scaling Agile @ Spotify. http://www.agileleanhouse.com/lib/lib/People/HenrikKniberg/SpotifyScaling.pdf, 2012. Accessed: 03-08-2022, original upload: http://ucvox.files.wordpress.com/2012/11/113617905-scaling-Agile-spotify-11.pdf.

[28] Stefan Koch. Agile principles and open source software development: A theoretical and empirical discussion. In International Conference on Extreme Programming and Agile Processes in Software Engineering, pages 85–93. Springer, 2004.

[29] Zhenzhen Kou and William W Cohen. Stacked graphical models for efficient inference in markov random fields. In Proceedings of the 2007 SIAM International Conference on Data Mining, pages 533–538. SIAM, 2007.

[30] Elvan Kula, Eric Greuter, Arie Van Deursen, and Gousios Georgios. Factors affecting on-time delivery in large-scale agile software development. IEEE Transactions on

Software Engineering, 2021. URL https://ieeexplore.ieee.org/abstract/document/9503331/.

[31] Dean Leffingwell. Scaling software agility: Best practices for large enterprises (the agile software development series), 2007.

[32] Sofus A Macskassy. Relational classifiers in a non-relational world: Using homophily to create relations. In 2011 10th International Conference on Machine Learning and Applications and Workshops, volume 1, pages 406–411. IEEE, 2011.

[33] Jennifer Neville and David Jensen. Collective classification with relational dependency networks. In Workshop on Multi-Relational Data Mining (MRDM-2003), page 77, 2003.

[34] Mark EJ Newman. Mixing patterns in networks. Physical review E, 67(2):026126, 2003.

[35] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In 2012 Proceedings of the 27th IEEE/ACM international conference on automated software engineering, pages 70–79. IEEE, 2012.

[36] Thanh HD Nguyen, Bram Adams, and Ahmed E Hassan. Studying the impact of dependency network measures on software quality. In 2010 IEEE International Conference on Software Maintenance, pages 1–10. IEEE, 2010.

[37] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. Detecting bad smells in source code using change history information. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 268–278. IEEE, 2013.

[38] Lucas D Panjer. Predicting eclipse bug lifetimes. In Fourth international workshop on mining software repositories (MSR'07: ICSE workshops 2007), pages 29–29. IEEE, 2007.

[39] Rashmi Popli and Naresh Chauhan. Agile estimation using people and project related factors. In 2014 International Conference on Computing for Sustainable Global Development (INDIACom), pages 564–569. IEEE, 2014.

[40] Simone Porru, Alessandro Murgia, Serge Demeyer, Michele Marchesi, and Roberto Tonelli. Estimating story points from issue reports. In Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, pages 1–10, 2016.

[41] Paul Ralph and Ewan Tempero. Construct validity in software engineering research and software metrics. In Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, pages 13–23, 2018.

[42] Ken Schwaber and Jeff Sutherland. The scrum guide. Scrum Alliance, 21(19):1, 2011.

[43] Ezequiel Scott and Dietmar Pfahl. Using developers' features to estimate story points. In Proceedings of the 2018 International Conference on Software and System Process, pages 106–110, 2018.

[44] Helen Sharp and Hugh Robinson. Three 'c's of agile practice: collaboration, coordination and communication. In Agile software development, pages 61–85. Springer, 2010.

[45] Fei Tang and Hemant Ishwaran. Random forest missing data algorithms. Statistical Analysis and Data Mining: The ASA Data Science Journal, 10(6):363–377, 2017.

[46] Ben Taskar, Vassil Chatalbashev, and Daphne Koller. Learning associative markov networks. In Proceedings of the twenty-first international conference on Machine learning, page 102, 2004.

[47] Muhammad Usman, Emilia Mendes, and Jürgen Börstler. Effort estimation in agile software development: a survey on the state of the practice. In Proceedings of the 19th international conference on Evaluation and Assessment in Software Engineering, pages 1–10, 2015.

[48] Timo Wolf, Adrian Schroter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In 2009 IEEE 31st International Conference on Software Engineering, pages 1–11. IEEE, 2009.

[49] Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In 2013 35th International Conference on Software Engineering (ICSE), pages 1042–1051. IEEE, 2013.